n-Gram/2L: A Space and Time Efficient Two-Level n-Gram Inverted Index Structure

Min-Soo Kim, Kyu-Young Whang, Jae-Gil Lee, Min-Jae Lee

Department of Computer Science and Advanced Information Technology Research Center (AITrc)
Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea
{mskim, kywhang, jglee, mjlee}@mozart.kaist.ac.kr

Abstract

The n-gram inverted index has two major advantages: language-neutral and error-tolerant. Due to these advantages, it has been widely used in information retrieval or in similar sequence matching for DNA and protein databases. Nevertheless, the n-gram inverted index also has drawbacks: the size tends to be very large, and the performance of queries tends to be bad. In this paper, we propose the *two-level n-gram inverted index* (simply, the *n-gram/2L index*) that significantly reduces the size and improves the query performance while preserving the advantages of the n-gram inverted index. The proposed index eliminates the redundancy of the position information that exists in the n-gram inverted index. The proposed index is constructed in two steps: 1) extracting subsequences of length m from documents and 2) extracting n-grams from those subsequences. We formally prove that this two-step construction is identical to the relational normalization process that removes the redundancy caused by a non-trivial multivalued dependency. The n-gram/2L index has excellent properties: 1) it significantly reduces the size and improves the performance compared with the n-gram inverted index with these improvements becoming more marked as the database size gets larger; 2) the query processing time increases only very slightly as the query length gets longer. Experimental results using databases of 1 GBytes show that the size of the n-gram/2L index is reduced by up to 1.9 $\sim$ 2.7 times and, at the same time, the query performance is improved by up to 13.1 times compared with those of the n-gram inverted index.

**Proceedings of the 31st VLDB Conference,
Trondheim, Norway, 2005**

1 Introduction

Text searching is regarded as an operation of fundamental importance and is widely used in many areas such as information retrieval [18] and similar sequence matching for DNA and protein databases [7]. DNA and protein sequences can be regarded as long texts over specific alphabets (e.g. {A,C,G,T} in DNA) [5]. A number of index structures have been proposed for efficient text searching, and the inverted index is the most actively used one [18].

The inverted index is classified into two categories — the *word-based inverted index* and the *n-gram inverted index* (simply, the *n-gram index*) — depending on the kind of terms [8]. The former uses a word as a term, and the latter an n-gram. A word is a string of variable-length and has a linguistic meaning, but an n-gram is a string of fixed-length n and has no linguistic meaning. The n-grams are extracted as follows: sliding a window of length n by one character in the text and recording a sequence of characters in the window at each time. We call it the *1-sliding technique*. Due to the 1-sliding technique, a very large number of n-grams are extracted in documents compared with the words.

The n-gram index has two major advantages: language-neutral and error-tolerant [18, 9, 3]. The first advantage allows us to disregard the characteristics of the language since the n-grams are extracted using a window of length n. Thus, the n-gram index is widely used for Asian languages, where complicated knowledge about the language is required, or DNA and protein databases, where a clear concept of the word does not exist. The second advantage allows us to retrieve documents with some errors (e.g., typos or miss-recognition by the OCR software) as the query result. It becomes possible by the 1-sliding technique. Thus, the n-gram index is adequate for applications allowing errors such as approximate matching.

Nevertheless, the n-gram index has also drawbacks: the size tends to be large, and the performance of queries — especially, long ones — tends to be bad [18, 9]. These drawbacks stem from the method of extracting terms, that is, the 1-sliding technique. It drastically increases the number of terms extracted, rendering the size of the n-gram index so large. Accordingly, the query performance is also degraded since the number of postings accessed during query processing increases.

There have been a number of efforts to reduce the size of the n-gram index. The compression of the inverted index is widely employed [2, 4, 8]. This scheme, while preserving the index structure, compresses posting lists during

indexing and decompresses them during query processing. It, however, requires additional compression and decompression costs. On the other hand, a few methods have been proposed to reduce the number of terms to be indexed. These methods first extract words from documents, and then, extract n-grams (or a single n-gram) within each word [9, 11]. These methods, however, are difficult to be used for data where a clear concept of the word does not exist.

In this paper, we propose the *two-level n-gram inverted index* (simply, the *n-gram/2L index*) that significantly reduces the size and improves the query performance while preserving the advantages of the n-gram index. We first identify that the large size of the n-gram index is mainly due to the redundancy of the position information. Here, the *position information* represents the document identifier and the offsets within the document where an n-gram occurs. The proposed index eliminates the redundancy of the position information that exists in the n-gram index. The key idea for eliminating redundancy is to construct the index in two steps. For two-step construction, we regard consecutive n-grams as a subsequence and store (1) the positions of the subsequence within the documents and (2) the positions of n-grams within the subsequence. Hereafter, we call the index for subsequences as the *back-end index* and that for n-grams as the *front-end index*.

The n-gram/2L index has three excellent properties. First, the size of the n-gram/2L index is scalable with the database size. The ratio of the size of the n-gram/2L index to that of the conventional n-gram index decreases as the database size gets larger. That is, reduction of the index size becomes more marked in a larger database. Second, query performance of the n-gram/2L index is also scalable with the database size. The ratio of query performance of the n-gram/2L index to that of the conventional n-gram index increases as the database size gets larger, making the improvement of the query performance more marked in a larger database. Third, the query processing time increases only slightly as the query length gets longer, in contrast to an n-gram index where it increases rapidly. We investigate the reasons for these desirable properties in Section 5.

The rest of this paper is organized as follows. Section 2 describes existing work related to the n-gram index. Section 3 presents the motivation of this paper. Section 4 proposes the structure and algorithms of the n-gram/2L index. Section 5 presents the formal model of the n-gram/2L index and analyzes the size and query performance. Section 6 presents the results of performance evaluation performed by using the Odysseus ORDBMS [14]. Section 7 summarizes and concludes the paper.

2 Related Work

In this section, we explain the inverted index [18], and then, the n-gram index and its applications. The *inverted index* is a term-oriented mechanism for quickly searching documents containing a given term. Here, a *document* is a finite sequence of characters, and a *term* a subsequence of a document.

The inverted index consists of two major components: terms and posting lists [8]. A *posting list*, which is related to a specific term, is a list of postings that contain information about the occurrences of the term. A *posting* consists of the identifier of the document that contains the term and the list of the offsets where the term occurs in the document. For each term t, there is a posting list that contains postings $< d, [o_1, ..., o_f] >$, where d is a document identifier, $[o_1, ..., o_f]$ is a list of offsets o, and f is the frequency of occurence of the term t in the document d [4]. Postings in a posting list are usually stored in the increasing order of d, and offsets within a posting in the increasing order of o. Besides, an index such as the B+-tree is created on terms in order to quickly locate a posting list. Figure 1 shows the structure of the inverted index.

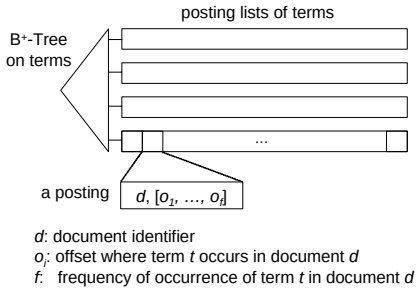


Figure 1: The structure of the inverted index.

The inverted index is classified into two types depending on the method of extracting terms: (1) the word-based inverted index using a word as a term and (2) the n-gram index using an n-gram as a term [8, 11, 3]. We focus on the n-gram index in this paper. The rest of this section is organized as follows. Section 2.1 describes the structure and query processing of the n-gram index. Section 2.2 describes the applications.

2.1 n-Gram Index

The n-gram index uses n-grams as indexing terms. Let us consider a document d as a sequence of characters $c_0, c_1, ..., c_{N-1}$. An n-gram is a subsequence of length n [10]. Extracting n-grams from a document d can be done by using the 1-sliding technique, that is, sliding a window of length n from c_0 to c_{N-n} and storing the characters located in the window. The ith n-gram extracted from d is the sequence $c_i, c_{i+1}, ..., c_{i+n-1}$.

Example 1 Figure 2 shows an example of posting lists of the 3-gram index, which is created on the documents containing the strings "string" and "data". The four 3-grams "str", "tri", "rin", and "ing" are extracted from "string", and the two 3-grams "dat" and "ata" are extracted from "data". We note that to the 1-sliding technique, the differences of the offsets of the 3-grams extracted from the same string become 1. □

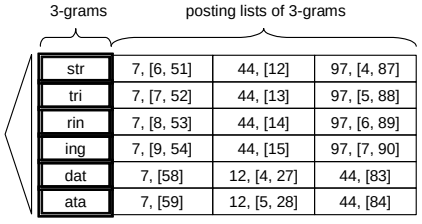


Figure 2: An example of posting lists of the 3-gram index.

Query processing is done in two steps: (1) spliting a given query string into multiple n-grams and searching the posting lists of those n-grams; and (2) performing merge

join between those posting lists using the document identifier as the join attribute [18].

For example, suppose we execute a query "string" by using the n-gram index in Figure 2. In the first step, the query "string" is split into the four 3-grams "str", "tri", "rin", and "ing", and four posting lists of these 3-grams are searched. In the second step, merge join among those four posting lists is performed in order to find the documents where the four 3-grams "str", "tri", "rin", and "ing" occur consecutively — constituting "string". The document identifiers 7, 44, and 97 are returned as the query result since all these documents contain the four 3-grams consecutively.

2.2 Applications of the n-Gram Index

There are many applications of the n-gram index such as string searching, approximate string matching, information retrieval, and similar sequence matching in bioinformatics [10]. We explain information retrieval [18] and similar sequence matching in bioinformatics [7] in more detail.

Information retrieval is the operation that searches for the documents containing a given query string from a text database. The inverted index is widely used to facilitate information retrieval. In general, to construct the inverted index, words are used as indexing terms. However, due to the language-neutral characteristic, the n-grams are often used for Asian languages such as Korean, Chinese, and Japanese, where extraction of words is not simple [15]. Compared with the word-based inverted index, the n-gram index has comparable accuracy in searching, but has a larger index size and poorer performance [9].

Similar sequence matching in bioinformatics is the operation that searches sequences similar to a given query string from a DNA or protein database. Most methods proposed earlier perform sequential scan and do not use indexes. These methods have relatively high accuracy, but have poor performance. In order to improve performance, the index-based methods have been proposed recently. CAFE [7], which uses the n-gram index, is a well-known method. CAFE uses 3-grams for protein sequences and 9-grams for DNA sequences as indexing terms. Compared with earlier methods, CAFE shows comparable accuracy in searching, but shows higher performance by several times to several tens of times. However, it has been pointed out as a problem that the size of the index of CAFE is significantly larger than that of the original database [6].

3 Motivation

In this section, we explain the motivation of our approach: in particular, why the redundancy of the position information exists in an n-gram index, and how we eliminate that redundancy. Figure 3 illustrates a motivating example involving a set of documents, an n-gram index, and a n-gram/2L index.

Figure 3(a) shows an example of a set of N documents. Let the sequence of characters "$a_1a_2a_3a_4$" be the subsequence A occurring at the offsets o_1, o_3, and o_5 in the documents 1, 2, and N. Let the sequence of characters "$b_1b_2b_3b_4$" be the subsequence B occurring at the offsets o_2 and o_4 in the documents 1 and N. Then, the three consecutive 2-grams "a_1a_2", "a_2a_3", and "a_3a_4" occur in the documents 1, 2, and N, and "b_1b_2", "b_2b_3", and "b_3b_4" in the documents 1 and N.

Figure 3(b) shows the n-gram index created on the documents in Figure 3(a). In Figure 3(b), the postings shaded have the redundancy in the position information. In the posting lists of the 2-grams "a_1a_2", "a_2a_3", and "a_3a_4", the fact that the difference between the offset of "a_1a_2" and that of "a_2a_3" (or "a_3a_4") is 1 (or 2) is repeatedly represented not only in the document 1 but also in the documents 2 and N. Such repetition also appears in the posting lists of the 2-grams "b_1b_2", "b_2b_3", and "b_3b_4". That is, if a subsequence is repeated multiple times in documents, the relative offsets (within the subsequences) of the n-grams extracted from that subsequence would also be indexed multiple times.

If the relative offsets of n-grams extracted from a subsequence are indexed only once, the index size would be reduced since such repetition is eliminated. Figure 3(c) shows the n-gram/2L index created on the documents in Figure 3(a). The offsets of the three 2-grams "a_1a_2", "a_2a_3", and "a_3a_4" can be represented in two levels as follows. The first level becomes the offsets of the subsequence A within documents as shown in the right part of Figure 3(c); the second level becomes the offsets of the 2-grams "a_1a_2", "a_2a_3", and "a_3a_4" within the subsequence A as shown in the left part of Figure 3(c).

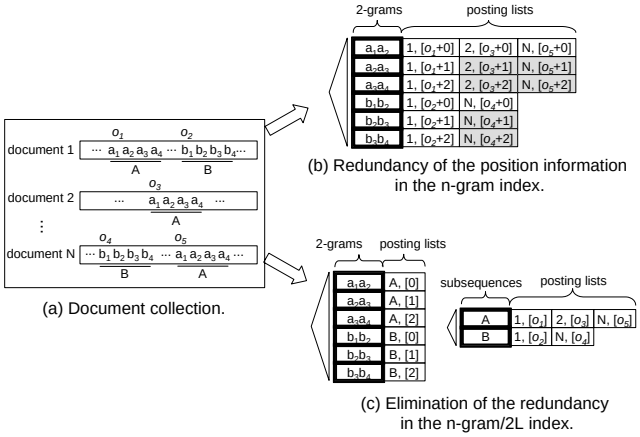


Figure 3: An example of redundancy and its elimination in an n-gram index.

We note that the construction of the n-gram/2L index is identical to the relational normalization process that removes the redundancy caused by a non-trivial multivalued dependency (MVD). We formally prove this proposition in Theorem 2.

4 n-Gram/2L Index

4.1 Index Structure

Figure 4 shows the structure of the n-gram/2L index, which consists of the back-end index and the front-end index. The *back-end index* stores the absolute offsets of subsequences within documents, and the *front-end index* the relative offsets of n-grams within subsequences.

4.2 Index Building Algorithm

The n-gram/2L index is built through the following four steps: (1) extracting subsequences, (2) building the back-end index, (3) extracting n-grams, and (4) building the front-end index. When extracting subsequences, the length of subsequences is fixed to be m, and consecutive subsequences overlap with each other by $n-1$. The purpose for this overlap is to prevent missing or duplicating n-grams,

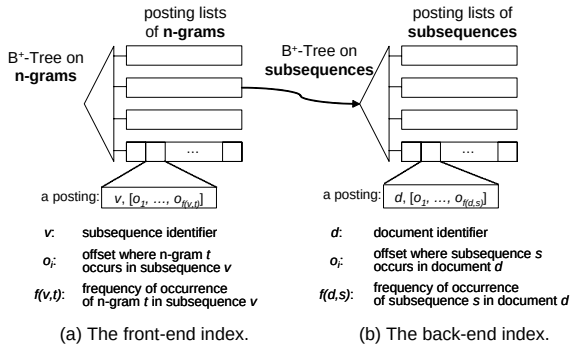


Figure 4: The structure of the n-gram/2L index.

i.e., we extract no more or no less n-grams than is necessary. We formally prove the correctness of this method in Theorem 1. Hereafter, we call the $(n-1)$-overlapping subsequence of length m as the *m-subsequence*. We note that n denotes the length of the n-gram, and m the length of the m-subsequence.

Theorem 1 If m-subsequences are extracted such that consecutive ones overlap with each other by $n-1$, no n-gram is missed or duplicated.
Proof: See Appendix A. □

Figure 5 shows the algorithm for building the n-gram/2L index. We call this algorithm *n-Gram/2L Index Building*. In Step 1, the algorithm extracts m-subsequences from a set of documents such that they overlap with each other by $n-1$. Suppose that a document is the sequence of characters $c_0, c_1, ..., c_{N-1}$. The algorithm extracts m-subsequences starting from the character $c_{i*(m-n+1)}$ for all i where $0 \leq i < \lfloor \frac{N-n+1}{m-n+1} \rfloor$. If the length of the last m-subsequence is less than m, the algorithm pads blank characters to the m-subsequence to guarantee the length of m. In Step 2, the algorithm builds the back-end index using the m-subsequences obtained in Step 1. For each m-subsequence s occurring f times in a document d at offsets $o_1, ..., o_f$, a posting $< d, [o_1, ..., o_f] >$ is appended to the posting list of s. In Step 3, the algorithm extracts n-grams from the set of m-subsequences obtained in Step 1 by using the 1-sliding technique. In Step 4, the algorithm builds the front-end index using the n-grams obtained in Step 3. For each n-gram g occurring f times in an m-subsequence v at offsets $o_1, ..., o_f$, a posting $< v, [o_1, ..., o_f] >$ is appended to the posting list of g.

Example 2 Figure 6 shows an example of building the n-gram/2L index. Suppose that $n = 2$ and $m = 4$. Figure 6(a) shows the set of documents. Figure 6(b) shows the set of the 4-subsequences extracted from the documents. Since 4-subsequences are extracted such that they overlap by 1 (i.e., $n-1$), those extracted from the document 0 are "ABCD", "DDAB", and "BBCD". Figure 6(c) shows the back-end index built from these 4-subsequences. Since the 4-subsequence "ABCD" occurs at the offsets 0, 3, and 6 in the documents 0, 3, and 4, respectively, the postings $< 0, [0] >, < 3, [3] >$, and $< 4, [6] >$ are appended to the posting list of the 4-subsequence "ABCD". Figure 6(d) shows the set of the 4-subsequences and their identifiers. Figure 6(e) shows the set of the 2-grams extracted from the 4-subsequences in Figure 6(d). Since 2-grams are extracted by the 1-sliding technique, those extracted from the 4-subsequence 0 are "AB", "BC", and "CD". Figure 6(f)

Algorithm *n-Gram/2L Index Building*:
Input: (1) The document collection D,
(2) The length m of subsequences ,
(3) The length n of n-grams
Output: The two-level n-gram inverted index
Algorithm:
Step 1. Extraction of m-subsequences: for each document in D
1.1 Suppose that a document d is a sequence of characters $c_0,c_1,...,c_{N-1}$;
extract m-subsequences starting from the character
$c_{i*(m-n+1)}$ $(0 \leq i < \lfloor (N-n+1)/(m-n+1) \rfloor)$ and
record the offsets of the m-subsequences within d.
1.2 If the length of the last m-subsequence is less than m,
pad blank characters to the m-subsequence.
Step 2. Construction of the back-end inverted index:
for each m-subsequence obtained in Step 1
2.1 Suppose that an m-subsequence s occurs in a document d at offsets $o_0,o_1,...,o_f$;
append a posting <d, [$o_0,o_1,...,o_f$]> to the posting list of s.
Step 3. Extraction of n-grams: for each m-subsequence obtained in Step 1
3.1 Suppose that an m-subsequence s is a sequence of characters $c_0,c_1,...,c_{L-1}$;
extract n-grams starting at the character c_i $(0 \leq i < L-n+1)$ and
record the offsets of the n-grams within s.
Step 4. Construction of the front-end inverted index: for each n-gram obtained in Step 3
4.1 Suppose that an n-gram g occurs in an m-subsequence v at offsets $o_0,o_1,...,o_f$;
append a posting <v, [$o_0,o_1,...,o_f$]> to the posting list of g.

Figure 5: The algorithm of building the n-gram/2L index.

shows the front-end index built from these 2-grams. Since the 2-gram "AB" occurs at the offsets 0, 2, 1, and 2 in the 4-subsequences 0, 3, 4, and 5, respectively, the postings $< 0, [0] >, < 3, [2] >, < 4, [1] >$, and $< 5, [2] >$ are appended to the posting list of the 2-gram "AB". □

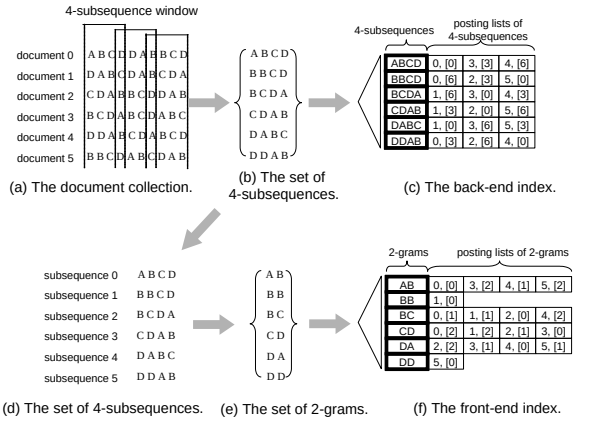


Figure 6: An example of building the n-gram/2L index.

4.3 Query Processing Algorithm

In this section, we present the algorithm for processing queries using the n-gram/2L index. For easy of exposition, we deal with only exact-match queries that consist of a single string. Our algorithm can be extended to accommodate approximate-match queries or boolean queries that consist of multiple strings.

The query processing algorithm consists of the the following two steps: (1) searching the front-end index in order to retrieve candidate results, (2) searching the back-end index in order to refine candidate results. In the first step, we select the m-subsequences that *cover* a query string by searching the front-end index with the n-grams extracted from the query string. The m-subsequences which do not cover the query string are filtered out in this step. In the

second step, we select the documents that have a set of m-subsequences $\{S_i\}$ *containing* the query string by searching the back-end index with the m-subsequences retrieved in the first step. The documents including one or more m-subsequences retrieved in the first step represent a set of candidate results satisfying the necessary condition of (i.e., covering) the query. The final results can be obtained in the second step by doing refinement that removes the false positives.

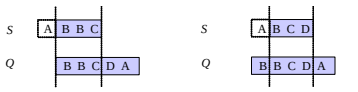
Now, we formally define *cover* in Definition 1 and *contain* in Definition 3.

Definition 1 S *covers* Q if an m-subsequence S and a query string Q satisfy one of the following four conditions: (1) a suffix of S matches a prefix of Q; (2) the whole string of S matches a substring of Q; (3) a prefix of S matches a suffix of Q; (4) a substring of S matches the whole string of Q. □

Definition 2 The *expand* function expands a sequence of overlapping character sequences into one character sequence. (1) For a sequence consisting of two character sequences: Let $S_i = c_i...c_j$ and $S_p = c_p...c_q$. Suppose that a suffix of S_i and a prefix of S_p overlap by k (i.e., $c_{j-k+1}...c_j = c_p...c_{p+k-1}$, where $k \geq 0$.) Then, $expand(S_iS_p) = c_i...c_jc_{p+k}...c_q$. (2) For a sequence consisting of more than two character sequences: $expand(S_lS_{l+1}...S_m) = expand(expand(S_lS_{l+1}), S_{l+2}...S_m)$. □

Definition 3 $\{S_i\}$ *contains* Q if a set of m-subsequences $\{S_i\}$ and a query string Q satisfy the following condition: Let $S_lS_{l+1}...S_m$ be a sequence of m-subsequences overlapping with each other in $\{S_i\}$. A substring of $expand(S_lS_{l+1}...S_m)$ matches the whole string of Q. □

Example 3 Figure 7 shows examples of covering. Here, Q is the query and S is an m-subsequence. In Figure 7(a), S covers Q since a suffix of S matches a prefix of Q. In Figure 7(b), S does not cover Q since "BCD" does not satisfy any of the four conditions in Definition 1. □



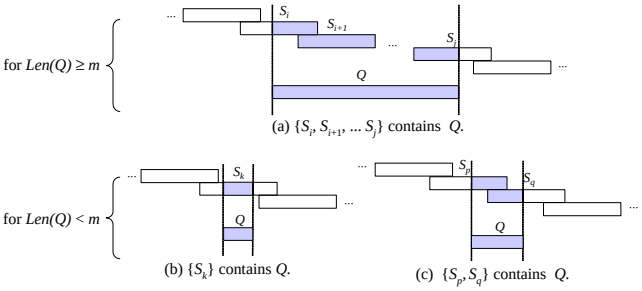
(a) *S* covers *Q*. (b) *S* does not cover *Q*.

Figure 7: Examples of an m-subsequence S covering the query Q.

Lemma 1 A document that has a set of m-subsequences $\{S_i\}$ containing the query string Q includes at least one m-subsequence covering Q.

Proof: We first show the cases that a set of m-subsequences $\{S_i\}$ contains Q in Figure 8. Let $Len(Q)$ be the length of Q. We classify the cases depending on whether $Len(Q) \geq m$ (Figure 8(a)) or $Len(Q) < m$ (Figures 8(b) and (c)). In Figure 8(a), the set $\{S_i, ..., S_j\}$ contains Q. In Figure 8(b), the set $\{S_k\}$ contains Q. In Figure 8(c), the set $\{S_p, S_q\}$ contains Q. From Figure 8 we see that, if the set $\{S_i\}$ contains Q, at least one m-subsequence in $\{S_i\}$ satisfies a condition in Definition 1, covering Q. □

Figure 9 shows the algorithm of processing queries using the n-gram/2L index. We call this algorithm *n-Gram/2L Index Searching*. In Step 1, the algorithm splits the query string Q into multiple n-grams and searches the posting lists of those n-grams in the front-end index. Then, performing merge outer join among those posting lists using the m-subsequence identifier as the join attribute, the algorithm adds the m-subsequences that cover Q by Definition 1 (i.e., m-subsequences satisfying a necessary condition in Lemma 1) into the set S_{cover}. Since an m-subsequence covering Q typically does not have all the n-grams extracted from Q, the algorithm performs merge outer join in Step 1.2. Here, the algorithm uses the offset information in the postings to be merge outer joined in order to check whether the m-subsequence covers Q. In Step 2, the algorithm performs merge outer join among the posting lists of the m-subsequences in S_{cover} using the document identifier as the join attribute. It identifies the set $\{S_i\}$ of the m-subsequences having the same document identifier d_i and performs refinement by checking whether $\{S_i\}$ indeed contains Q according to Definition 3. Since the set $\{S_i\}$ may be a subset of S_{cover}, the algorithm performs merge outer join in Step 2.1. Here, the algorithm uses the offset information in the postings to be merge outer joined in order to check whether $\{S_i\}$ contains Q. If $\{S_i\}$ contains Q, d_i is returned as a query result.



(a) $\{S_i, S_{i+1}, ... S_j\}$ contains *Q*. (b) $\{S_k\}$ contains *Q*. (c) $\{S_p, S_q\}$ contains *Q*.

Figure 8: The cases that a set of m-subsequences contains Q.

Algorithm *n-Gram/2L Index Searching*:
Input: (1) The two-level n-gram inverted index
(2) A query string *Q*
Output: Identifiers of the documents containing *Q*
Algorithm:
Step 1. Searching the front-end inverted index:
1.1 Split *Q* into multiple n-grams and search the posting lists of those n-grams.
1.2 Perform merge outer join among those posting lists
using the m-subsequence identifier as the join attribute;
add the m-subsequences that cover *Q* by Definition 1 into the set S_{cover}.
Step 2. Searching the back-end inverted index:
2.1 Perform merge outer join among the posting lists of m-subsequences in S_{cover}
using the document identifier as the join attribute.
2.1.1 Identify the set $\{S_i\}$ of m-subsequences having the same
document identifier d_i and perform refinement by checking
whether $\{S_i\}$ contains *Q* or not according to Definition 3.
2.1.2 If $\{S_i\}$ contains *Q*, d_i is returned as the query result.

Figure 9: The algorithm of processing queries using the n-gram/2L index.

5 Formal Analysis of the n-Gram/2L Index

In this section, we present a formal analysis of the n-gram/2L index. In Section 5.1, we formally prove that the n-gram/2L index is derived by eliminating the redundancy in the position information that exists in the n-gram index. In Section 5.2, we present the space complexities of these indexes. In Section 5.3, we present their time complexities of searching.

5.1 Formalization

In this section, we observe that the redundancy of the position information existing in the n-gram index is caused by a non-trivial multivalued dependency (MVD) [17, 1] and show that the n-gram/2L index can be derived by eliminating that redundancy through relational decomposition to the Fourth Normal Form (*4NF*).

For the sake of theoretical development, we first consider the relation that is converted from the n-gram index so as to obey the First Normal Form (1NF). We call this relation the *NDO relation*. This relation has three attributes N, D, and O. Here, N indicates n-grams, D document identifiers, and O offsets. Further, we consider the relation obtained by adding the attribute S and by splitting the attribute O into two attributes O_1 and O_2. We call this relation the *SNDO$_1$O$_2$ relation*. This relation has five attributes S, N, D, O_1, and O_2. Here, S indicates m-subsequences, O_1 the offsets of n-grams within m-subsequences, and O_2 the offsets of m-subsequences within documents.

The values of the attributes S, O_1, and O_2 appended to the relation SNDO$_1$O$_2$ are automatically determined by those of the attributes N, D, and O in the relation NDO. The reason is that an n-gram appearing at a specific offset o in the document belongs to only one m-subsequence as shown in Theorem 1. In the tuple (s, n, d, o_1, o_2) thus determined by a tuple (n, d, o) of the relation NDO, s represents the m-subsequence that the n-gram n occurring at the offset o in the document d belongs to. o_1 is the offset where the n-gram n occurs in the m-subsequence s, and o_2 the offset where the m-subsequence s occurs in the document d.

Example 4 Figure 10 shows the 2-gram index built on the documents in Figure 6(a). Figure 11(a) shows the relation NDO converted from this n-gram index. Figure 11(b) shows the relation SNDO$_1$O$_2$ ($m = 4$) derived from the relation NDO in Figure 11(a). Here, the tuples of the relation SNDO$_1$O$_2$ are sorted by the values of S. In Figure 11, the marked tuple of the relation SNDO$_1$O$_2$ is determined by the marked tuple of the relation NDO. Since the 2-gram "BC" at the offset 1 in the document 0 belongs to the 4-subsequence "ABCD" in Figure 6(a), the value of the attribute S of the marked tuple becomes "ABCD". The value of the attribute O_1 becomes 1 because the 2-gram "BC" occurs in the 4-subsequence "ABCD" at the offset 1, and that of the attribute O_2 becomes 0 because the 4-subsequence "ABCD" occurs in the document 0 at the offset 0. □

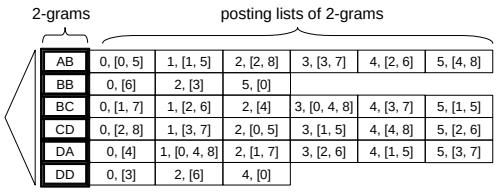


Figure 10: An example of the n-gram index.

We now prove that non-trivial MVDs hold in the relation SNDO$_1$O$_2$ (i.e., the n-gram index) in Lemma 2.

Lemma 2 The non-trivial MVDs S $\rightarrow\!\!\rightarrow$ NO$_1$ and S $\rightarrow\!\!\rightarrow$ DO$_2$ hold in the relation SNDO$_1$O$_2$. Here, S is not a superkey.

Proof: By the definition of the MVD [12, 17, 1], $X \rightarrow\!\!\rightarrow Y$ holds in R, where X and Y are subsets of R, if whenever r is a relation for R and μ and ν are two tuples in r, with

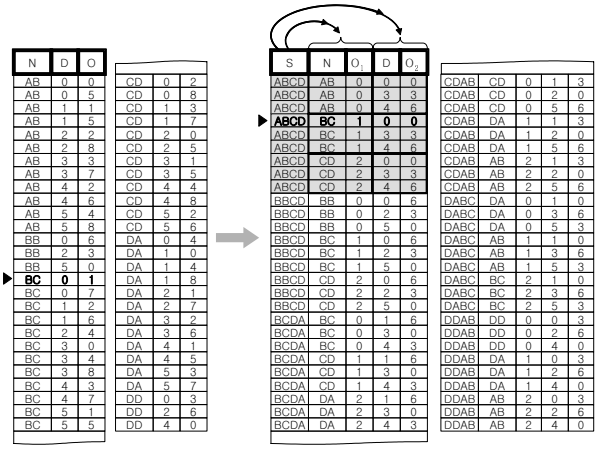


Figure 11: An example showing the existence of a non-trivial MVD in the relation SNDO$_1$O$_2$.

$\mu, \nu \in r, \mu[X] = \nu[X]$ (that is, μ and ν agree on the attributes of X), then r also contains tuples ϕ and ψ, that satisfy three conditions below. $X \rightarrow\!\!\rightarrow Y$ is non-trivial if $Y \not\subseteq X$ and $X \cup Y \neq R$ (i.e., $R - X - Y \neq \emptyset$)(meaning Y and $R - X - Y$ are non-empty sets of attributes) [17, 1]. That is, a non-trivial MVD holds if, for any value of the attribute X, the Y-values and the $(R - X - Y)$-values form a Cartesian product [16].

1. $\phi[X] = \psi[X] = \mu[X] = \nu[X]$
2. $\phi[Y] = \mu[Y]$ and $\phi[R - X - Y] = \nu[R - X - Y]$
3. $\psi[Y] = \nu[Y]$ and $\psi[R - X - Y] = \mu[R - X - Y]$

Let $\{S_1, ..., S_r\}$ be a set of m-subsequences extracted from the document collection, $\{N_{i1}, ..., N_{iq}\}$ a set of n-grams extracted from an m-subsequence S_i, and $\{D_{i1}, ..., D_{ip}\}$ a set of documents where S_i occurs ($1 \leq i \leq r$). Then, the set of n-grams $\{N_{i1}, ..., N_{iq}\}$ is extracted from every document in the set of documents $\{D_{i1}, ..., D_{ip}\}$ since S_i is in these documents. Hence, in the set of tuples whose S-value is S_i, the NO$_1$-values representing the n-grams extracted from S_i and the DO$_2$-values representing the documents containing S_i always form a Cartesian product. Suppose that R = SNDO$_1$O$_2$, X = S, Y = NO$_1$, and R-X-Y = DO$_2$. Then, the three conditions above are satisfied because the Y-values and the (R-X-Y)-values form a Cartesian product in the set of tuples having the same X-values. These conditions above are satisfied also when $Y = DO_2$. We also note that NO$_1 \not\subseteq$ S, DO$_2 \not\subseteq$ S, NO$_1 \cup$ S $\neq$ SNDO$_1$O$_2$, and DO$_2 \cup$ S $\neq$ SNDO$_1$O$_2$. Thus, the non-trivial MVDs S $\rightarrow\!\!\rightarrow$ NO$_1$ and S $\rightarrow\!\!\rightarrow$ DO$_2$ hold in the relation SNDO$_1$O$_2$. S is obviously not a superkey as shown in the counter example of Figure 11(b).

□

Intuitively, non-trivial MVDs hold in the relation SNDO$_1$O$_2$ because the set of documents, where an m-subsequence occurs, and the set of n-grams, which are extracted from that m-subsequence, are independent of each other. If attributes in a relation are independent of each other, non-trivial MVDs hold in that relation [12, 16, 17, 1]. In the relation SNDO$_1$O$_2$, due to the independence between the set of documents and the set of n-grams, there exist the tuples corresponding to all possible combinations of documents and n-grams for a given m-subsequence.

Example 5 Figure 11(b) shows an example showing the existence of the non-trivial MVDs S $\rightarrow\!\!\rightarrow$ NO_1 and S $\rightarrow\!\!\rightarrow$ DO_2 in the relation $SNDO_1O_2$. In the shaded tuples of the relation $SNDO_1O_2$ shown in Figure 11(b), there exists the redundancy that the DO_2-values (0, 0), (3, 3), and (4, 6) repeatedly appear for the NO_1-values ("AB", 0), ("BC", 1), and ("CD", 2). That is, the NO_1-values and the DO_2-values form a Cartesian product in the tuples whose S-value is "ABCD". We note that there such repetitions also occur in the other S-values. □

Corollary 1 The relation $SNDO_1O_2$ is not in the Fourth Normal Form (4NF).

Proof: A non-trivial MVD S $\rightarrow\!\!\rightarrow$ NO_1 exists, where S is not a superkey. □

The front-end and back-end indexes of the n-gram/2L index are identical to two relations obtained by decomposing the relation $SNDO_1O_2$ so as to obey 4NF. We prove this proposition in Theorem 2. Thus, it can be proved that the redundancy caused by a non-trivial MVD does not exist in the n-gram/2L index [12].

Lemma 3 The decomposition (SNO_1, SDO_2) is in 4NF.

Proof: See Appendix B □

Theorem 2 The 4NF decomposition (SNO_1, SDO_2) of the relation $SNDO_1O_2$ is identical to the front-end and back-end indexes of the n-gram/2L index.

Proof: The relation SNO_1 is represented as the front-end index by regarding N, S, and O_1 as a term, an m-subsequence identifier, and an offset, respectively. Similarly, the relation SDO_2 is represented as the back-end index by regarding S, D, and O_2 as a term, a document identifier, and an offset, respectively. Therefore, the 4NF decomposition (SNO_1, SDO_2) of the relation $SNDO_1O_2$ is identical to the front-end and back-end indexes of the n-gram/2L index. □

Example 6 Figure 12 shows that the relation $SNDO_1O_2$ in Figure 11 is decomposed into the two relations SNO_1 and SDO_2. In the attribute S of the relation SDO_2, the values in parentheses indicate m-subsequence identifiers. The tuples of the relation SDO_2 are sorted by the m-subsequence identifier. The shaded tuples of the relation $SNDO_1O_2$ in Figure 11 are decomposed into the shaded ones of the relations SNO_1 and SDO_2 in Figure 12. We note that the redundancy, i.e., the Cartesian product between NO_1 and DO_2 in Figure 11 has been eliminated in Figure 12. We also note that the relations SNO_1 and SDO_2, when represented in the form of the inverted index, become identical to the front-end and back-end indexes in Figure 6, respectively. □

5.2 Analysis of the Index Size

The parameters affecting the size of the n-gram/2L index are the length n of n-grams and length m of m-subsequences. In general, n is the value determined by applications. On the other hand, m is the value that can be freely tuned when creating the n-gram/2L index. In this section, we analyze the size of the n-gram/2L index and present the model for determining the optimal length of m that minimizes the index size. We denote the optimal length of m by m_o.

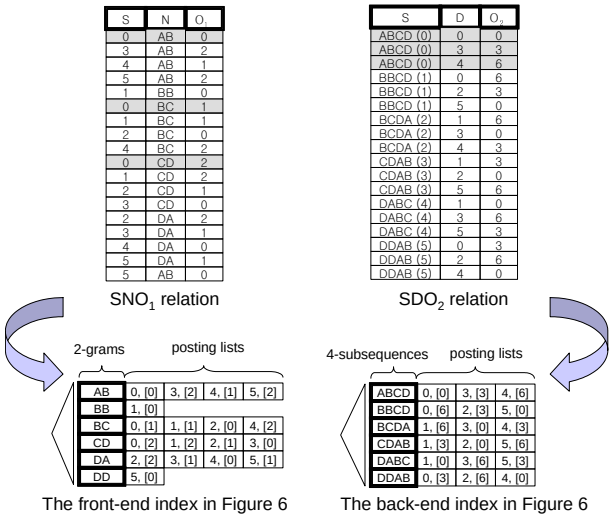


Figure 12: The result of decomposing the relation $SNDO_1O_2$ in Figure 11 into two relations.

In Table 1, we summarize some basic notations to be used for analyzing the size of the n-gram/2L index. Here, we simply regard the index size as the number of the offsets stored because the former is approximately proportional to the latter, the latter representing the occurrences of terms in documents [8].

Table 1: The notations to be used for analyzing the size of the n-gram/2L index.

Symbols	Definitions
$size_{ngram}$	the size of the n-gram index
$size_{front}$	the size of the front-end index
$size_{back}$	the size of the back-end index
$\mathbb{S}$	the set of unique m-subsequences extracted from the document collection
$k_{ngram}(s)$	the number of the n-grams extracted from an m-subsequence s
$k_{doc}(s)$	the frequency of an m-subsequence s appearing in the document collection
$avg_{ngram}(\mathbb{S})$	the average value of $k_{ngram}(s)$ where $s \in \mathbb{S}$ (= $(\sum_{s \in \mathbb{S}} k_{ngram}(s))$ / $\lvert\mathbb{S}\rvert$)
$avg_{doc}(\mathbb{S})$	the average value of $k_{doc}(s)$ where $s \in \mathbb{S}$ (= $(\sum_{s \in \mathbb{S}} k_{doc}(s))$ / $\lvert\mathbb{S}\rvert$)

Now, in order to determine the value of m_o, we define the *decomposition efficiency* in Definition 4.

Definition 4 The *decomposition efficiency* is the ratio of the size of the n-gram index to that of the n-gram/2L index. Thus,

$$\text{decomposition efficiency} = \frac{size_{ngram}}{size_{front} + size_{back}} \quad (1)$$

□

The decomposition efficiency in Definition 4 is computed through Formulas (1)∼(5). We count the number of offsets in the index using the number of tuples in the relation $SNDO_1O_2$. The number of tuples in the relation

$SNDO_1O_2$ is equal to that of offsets of the n-gram index since the relation $SNDO_1O_2$ is created by normalizing the n-gram index into 1NF. As mentioned in Lemma 2, for any value of the attribute S in the relation $SNDO_1O_2$, the values of attributes NO_1 and those of attributes DO_2 form a Cartesian product. Thus, in the relation $SNDO_1O_2$, the number of tuples having s as the value of S becomes $k_{ngram}(s) \times k_{doc}(s)$. Accordingly, the size of the n-gram index can be calculated as in Formula (2), i.e., the summation of $k_{ngram}(s) \times k_{doc}(s)$ for all unique m-subsequences. We obtain the sizes of the front-end index and the back-end index similary. In the relation SNO_1 corresponding to the front-end index, the number of tuples having s as the value of S becomes $k_{ngram}(s)$. Hence, the size of the front-end index is as in Formula (3), i.e., the summation of $k_{ngram}(s)$ for all unique m-subsequences. In the relation SDO_2 corresponding to the back-end index, the number of tuples having s as the value of S becomes $k_{doc}(s)$. Hence, the size of the back-end index is as in Formula (4), i.e., the summation of $k_{doc}(s)$ for all unique m-subsequences. Consequently, we obtain Formula (5) for the decomposition efficiency from Formulas (1)∼(4).

$$size_{ngram} = \sum_{s \in \mathbb{S}} (k_{ngram}(s) \times k_{doc}(s)) \quad (2)$$

$$size_{front} = \sum_{s \in \mathbb{S}} k_{ngram}(s) \quad (3)$$

$$size_{back} = \sum_{s \in \mathbb{S}} k_{doc}(s) \quad (4)$$

$$\begin{aligned} \underset{efficiency}{decomposition} &= \frac{\sum_{s \in \mathbb{S}} (k_{ngram}(s) \times k_{doc}(s))}{\sum_{s \in \mathbb{S}} (k_{ngram}(s) + k_{doc}(s))} \\ &\approx \frac{|\mathbb{S}|(avg_{ngram}(\mathbb{S}) \times avg_{doc}(\mathbb{S}))}{|\mathbb{S}|(avg_{ngram}(\mathbb{S}) + avg_{doc}(\mathbb{S}))} \end{aligned} \quad (5)$$

The decomposition efficiency is computed by using $\mathbb{S}$, $k_{ngram}(s)$, and $k_{doc}(s)$, which can be obtained by preprocessing the document collection. This can be done by sequentially scanning the document collection only once (i.e., $O(data\ size)$). To determine m_o, we first compute decomposition efficiencies for several candidate values of m, and then, select the one that provides the maximum decomposition efficiency. Experimental results show that m_o is determined approximately in the range $(n+1) \sim (n+3)$, i.e., longer than n by $1 \sim 3$, in the document collection of 10 MBytes∼1 GBytes.

Formula (5) shows that the space complexity of the n-gram index is $O(|\mathbb{S}|(avg_{ngram} \times avg_{doc}))$, while that of the n-gram/2L index is $O(|\mathbb{S}|(avg_{ngram} + avg_{doc}))$. As indicated by Formula (5), the decomposition efficiency is maximized when avg_{ngram} is equal to avg_{doc}. Here, avg_{doc} increases as the database size gets larger, and avg_{ngram} does as m gets longer. avg_{ngram} also becomes larger in a larger database since we choose a longer m_o to obtain the maximum decomposition efficiency. That is, both avg_{ngram} and avg_{doc} become larger as the database size increases. Since $(avg_{ngram} \times avg_{doc})$ increases more rapidly than $(avg_{ngram} + avg_{doc})$ does, the decomposition efficiency increases as the database size does. Therefore, the n-gram/2L index has the characteristic of reducing the index size more for a larger database.

5.3 Analysis of the Query Performance

The parameters affecting the query performance of the n-gram/2L index are m, n, and the length $Len(Q)$ of the query string Q. In this section, we conduct a ballpark analysis of the query performance to investigate the trend depending on these parameters.

For simplicity of our analysis, we first make the following two assumptions: (1) the query processing time is proportional to the number of offsets accessed and the number of posting lists accessed. The latter has a nontrivial effect on performance since accessing a posting list incurs seek time for moving the disk head to locate the posting list; (2) the size of the document collection is so large that all possible combinations of n-grams(=$|\Sigma|^n$) or m-subsequences(=$|\Sigma|^m$), where Σ denotes the alphabet, are indexed in the inverted index (for example, when $|\Sigma| = 26$ and $m = 5$, $|\Sigma|^m = 11,881,376$). Since the performance of query processing is important especially in a large database, the second assumption is indeed reasonable.

The ratio of the query performance of the n-gram index to that of the n-gram/2L index is computed by using Formulas (6)∼(9). Let k_{offset} be the average number of offsets in a posting list, and k_{plist} be the number of posting lists accessed during query processing. Then, the number of offsets accessed during query processing is $k_{offset} \times k_{plist}$. In the n-gram index, since k_{offset} is $\frac{size_{ngram}}{|\Sigma|^n}$ and k_{plist} is $(Len(Q) - n + 1)$, the query processing time is as in Formula (6). In the front-end index of the n-gram/2L index, since k_{offset} is $\frac{size_{front}}{|\Sigma|^n}$ and k_{plist} is $(Len(Q) - n + 1)$, the query processing time is as in Formula (7). In the back-end index of the n-gram/2L index, k_{offset} is $\frac{size_{back}}{|\Sigma|^m}$. Here, k_{plist} is the number of m-subsequences covering Q and is calculated differently depending on whether $Len(Q) < m$ or $Len(Q) \geq m$. If $Len(Q) \geq m$, the number of m-subsequences $S_{i+1}, ..., S_{j-1}$ in Figure 8(a) is $(Len(Q) - m + 1)$, and that of m-subsequences S_i or S_j is $\sum_{i=0}^{m-n-1} |\Sigma|^{m-n-i}$. If $Len(Q) < m$, the number of a m-subsequence S_k in Figure 8(b) is $\left((m - Len(Q) + 1) \times |\Sigma|^{m-Len(Q)}\right)$, and that of m-subsequences S_p or S_q is $\sum_{i=0}^{Len(Q)-n-1} |\Sigma|^{m-n-i}$. Hence, the query processing time in the back-end index is as in Formula (8). Finally, Formula (9) shows the ratio of the query processing times.

$$time_{ngram} = \frac{size_{ngram}}{|\Sigma|^n} \times (Len(Q) - n + 1) \quad (6)$$

$$time_{front} = \frac{size_{front}}{|\Sigma|^n} \times (Len(Q) - n + 1) \quad (7)$$

$$time_{back} = \begin{cases} \frac{size_{back}}{|\Sigma|^m} \times (Len(Q) - m + 1 \\ \quad + 2 \sum_{i=0}^{m-n-1} |\Sigma|^{m-n-i}), \\ \qquad \text{if } Len(Q) \geq m \\ \frac{size_{back}}{|\Sigma|^m} \times ((m - Len(Q) + 1) \times |\Sigma|^{m-Len(Q)} \\ \quad + 2 \sum_{i=0}^{Len(Q)-n-1} |\Sigma|^{m-n-i}), \\ \qquad \text{if } Len(Q) < m \end{cases} \quad (8)$$

$$\frac{time_{ngram}}{time_{front}+time_{back}} =$$

$$\begin{cases} \frac{size_{ngram}\times(Len(Q)-n+1)}{(size_{front}\times(Len(Q)-n+1))+\left(size_{back}\times(\frac{Len(Q)-m+1}{|\Sigma|^{m-n}}+c)\right)}, \\ \qquad \text{if } Len(Q)\geq m \\ \frac{size_{ngram}\times(Len(Q)-n+1)}{(size_{front}\times(Len(Q)-n+1))+\left(size_{back}\times(\frac{m-Len(Q)+1}{|\Sigma|^{Len(Q)-n}}+d)\right)}, \\ \qquad \text{if } Len(Q)<m \end{cases}$$

where $c=2\sum_{i=0}^{m-n-1}(\frac{1}{|\Sigma|})^i,\ d=2\sum_{i=0}^{Len(Q)-n-1}(\frac{1}{|\Sigma|})^i$ (9)

By substituting $size_{ngram}$ with $|\mathbb{S}|(avg_{ngram}\times avg_{doc})$, Formula (9) shows that the time complexity of the n-gram index is $O(|\mathbb{S}|(avg_{ngram}\times avg_{doc}))$ while that of the n-gram/2L index is $O(|\mathbb{S}|(avg_{ngram}+avg_{doc}))$. That is, the time complexities of those indexes are identical to their space complexities. The time complexity indicates that the n-gram/2L index has a good characteristic that the query performance improves compared with the n-gram index, and further, the improvement gets larger as the database size gets larger.

From Formulas (6)∼(9), we note that the query processing time of the n-gram index increases proportionally to $Len(Q)$. In contrast, the query processing time of the n-gram/2L index increases only slightly. In the front-end index, the query processing time increases proportionally to $Len(Q)$, but it contributes a very small proportion of the total query processing time because the index size is very small. The size of the front-end index is much smaller than that of the n-gram index because the total size of m-subsequences is much smaller than the total size of documents (for example, when the size of the document collection is 1 GBytes and m=5, the size of the set of m-subsequences is $13\sim 27$ MBytes). Furthermore, in the back-end index, $Len(Q)$ little affects the query processing time since $|\Sigma|^{m-n}$ is dominant (for example, when $|\Sigma|=26, m=6$, and $n=3$, $|\Sigma|^{m-n}=17,576$). This is also an excellent property since it has been pointed out that the query performance of the n-gram index for long queries tends to be bad[6].

To analyze the query processing time more precisely, we should take the time to locate posting lists into account. Suppose that α is the seek time required for locating a posting list. Then, the total time for locating posting lists is $k_{plist}\times\alpha$. Hence, by using k_{plist} computed in Formulas (6)∼(8), we derive the time for locating posting lists as shown in Formulas (10)∼(12).

$$plist\_time_{ngram}=\alpha\times(Len(Q)-n+1) \quad (10)$$

$$plist\_time_{front}=\alpha\times(Len(Q)-n+1) \quad (11)$$

$$plist\_time_{back}=\begin{cases} \alpha\times(Len(Q)-m+1 \\ \quad +2\sum_{i=0}^{m-n-1}|\Sigma|^{m-n-i}), \\ \qquad \text{if } Len(Q)\geq m \\ \alpha\times((m-Len(Q)+1)\times|\Sigma|^{m-Len(Q)} \\ \quad +2\sum_{i=0}^{Len(Q)-n-1}|\Sigma|^{m-n-i}), \\ \qquad \text{if } Len(Q)<m \end{cases} \quad (12)$$

Formulas (10) and (12) indicate $plist\_time_{ngram}=plist\_time_{front}$. Thus, the time for locating posting lists of the n-gram/2L index is larger than that of the n-gram index by $plist\_time_{back}$. In Formula (12), the dominant factor of k_{plist} is $\left(2\sum_{i=0}^{m-n-1}|\Sigma|^{m-n-i}\right)$ if $Len(Q)\geq m$ and $\left(2\sum_{i=0}^{Len(Q)-n-1}|\Sigma|^{m-n-i}\right)$ if $Len(Q)<m$, where these values increase exponentially as m gets larger. Hence, if we select (m_o-1) instead of m_o for the length of m-subsequences, we can significantly improve the query performance while sacrificing a small increment of the index size. Consequently, we use (m_o-1) for performance evaluation in Section 6.2.

6 Performance Evaluation

6.1 Experimental Data and Environment

The purpose of our experiments is to show that the size and query performance of the n-gram/2L index are superior to those of the n-gram index. We use the *index size ratio* defined in Formula (13) as the measure for the index size and the number of page accesses and the wall clock time for the query performance.

$$\textit{index size ratio} = \frac{\textit{the number of pages allocated for the n-gram index}}{\textit{the number of pages allocated for the n-gram/2L index}} \quad (13)$$

We have performed experiments using two real data sets. The first one is the set of English text databases – WSJ, AP, and FR in the TREC databases – used in information retrieval. We use three data sets of 10 MBytes, 100 MBytes, and 1 GBytes, where tags, spaces, special characters, and numbers are removed. We call each data set TREC-10M, TREC-100M, and TREC-1G, respectively. The second one is the set of protein sequence databases – nr, env_nr, month.aa, and pataa in the NCBI BLAST web site[1] – used in bioinformatics. We use three data sets of 10 MBytes, 100 MBytes, and 1 GBytes. We call each data set PROTEIN-10M, PROTEIN-100M, PROTEIN-1G, respectively. We remove tags, spaces, special characters, and numbers in the TREC databases making the formats of the TREC data and the PROTEIN data similar to exclude the influence of the format to the results of the experiments.

To compare the index size, we measure the estimated and real index size ratios in the PROTEIN and TREC databases while varying m. We use the decomposition efficiency (Formula (5)) presented in Section 4.2 to estimate the index size ratio. Then, we show that the estimation of m_o is correct. When creating the n-gram index and the front-end index, we set n to be 3, which is the most practically used one in n-gram applications [13, 7]. Besides, when creating the back-end index, we vary m from 4, the minimum of m (i.e., $n+1$), to (m_o+1).

To compare the query performance, we measure the number of page accesses and the wall clock time while varying the database size and the query length. To test the effect of the database size, we build three databases of 10 MBytes, 100 MBytes, and 1 GBytes using the PROTEIN data and TREC data, respectively. Here, we repeat the test

[1] http://www.ncbi.nlm.nih.gov/BLAST

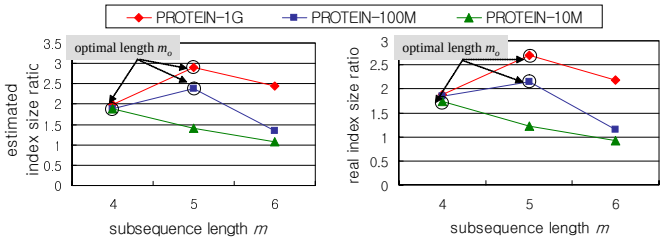
100 times with randomly selected queries whose length is 3∼18 and present the average result. To test the effect of the query length, we vary the query length as follows: 3, 6, 9, 12, 15, and 18. We note that the minimum query length is 3, which is the same as $n = 3$. Using PROTEIN-1G and TREC-1G, we repeat the test 50 times with randomly selected queries and present the average result.

We conduct all the experiments on a Pentinum 2.6 MHz Linux PC with 1 GBytes of main memory and 400 GBytes Segate E-IDE disks. To avoid the buffering effect of the LINUX file system and to guarantee actual disk I/Os, we use raw disks for index files. We use the inverted index implemented in the Odysseus ORDBMS [14] for all the experiments. The page size for data and indexes is set to be 4,096 bytes.

6.2 Results of the Experiments

6.2.1 Index Size

Figure 13 shows the estimated and real index size ratios as the database size and length of m-subsequences are varied in the PROTEIN databases. These results indicate that the size of the n-gram/2L index is significantly reduced compared with that of the n-gram index. Figure 13(b) shows that the size of the n-gram/2L index, when the length of m-subsequences is set to be m_o, is reduced by up to 1.7 times in PROTEIN-10M, by up to 2.2 times in PROTEIN-100M, and by up to 2.7 times in PROTEIN-1G compared with that of the n-gram index.



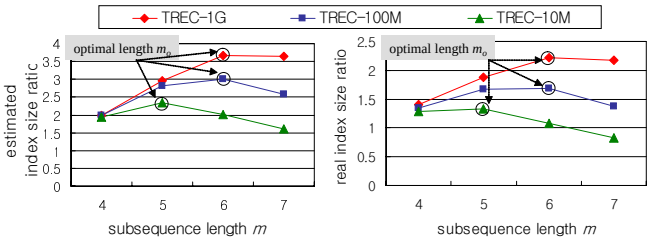
(a) The estimated index size ratio as *m* is varied.

(b) The real index size ratio as *m* is varied.

Figure 13: The estimated and real index size ratio in the PROTEIN databases.

We note that the estimated m_o in Figure 13(a) is identical to the real m_o in Figure 13(b). For PROTEIN-10M, PROTEIN-100M, and PROTEIN-1G, both estimated m_o and real m_o are 4, 5, and 5, respectively. Besides, the real index size ratio is as close as 86% ∼ 98% of the estimated one, showing a small amount of errors (2% ∼ 14%) in estimation. These results indicate that the analysis in Section 5.2 is indeed correct.

Figure 14 shows the estimated and real index size ratio in the TREC databases. We note that again the estimated m_o is identical to the real m_o. However, the real index size ratio is shown to be 52% ∼ 70% of the estimated one, showing a larger amount of errors compared with Figure 13. It is because the inverted index, in real implementation, stores document identifiers or m-subsequence identifiers in addition to offsets while we count only offsets in estimation. The identifier of a document (or an m-subsequence) and offsets at which a term occurs are maintained as a unit in a *posting* (See Figure 4). Here, the space required for identifiers is relatively larger in the n-gram/2L index because the average number of offsets in a posting is smaller than in the n-gram index, thus making the estimation to deviate from the real one. This difference of the average number of offsets is due to duplication of offsets in the n-gram index and its elimination in the n-gram/2L index. This tendency is more marked in Figure 14 than in Figure 13 because in TREC data, a collection of magazines or papers, the words or expressions are more frequently repeated.



(a) The estimated index size ratio as *m* is varied.

(b) The real index size ratio as *m* is varied.

Figure 14: The estimated and real index size ratio in the TREC databases.

Table 2 shows that the index size ratio increases as the database size does. Here, the maximum index size ratio is obtained by using m_o as the length of m-subsequences. (We also present the cases where we use $(m_o - 1)$ to optimize the query performance) This result confirms our analysis in Section 5.2. In Table 2, as the database size is varied by ten fold from 10 MBytes to 1 GBytes, the index size ratio increases by 25% in the PROTEIN databases, and by 29% in the TREC databases on the average when $m = m_o$. Similarly, it increases by 2% in the PROTEIN databases, and by 21% in the TREC databases on the average when $m = m_o - 1$. Here, since m must be longer than n, there is no result for $m = m_o - 1 = 3$ in PROTEIN-10M.

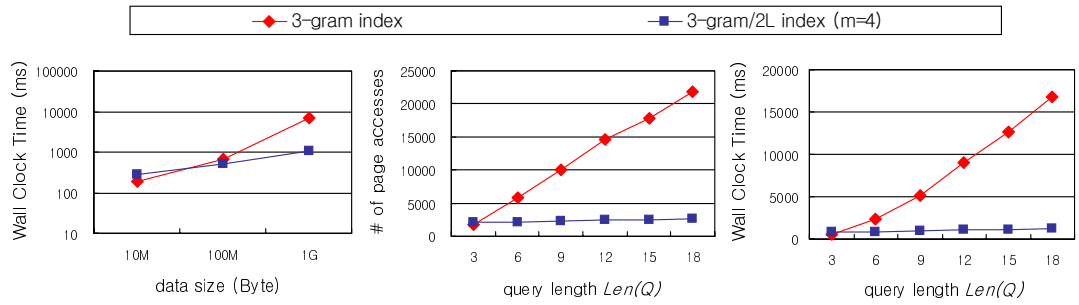
Table 2: The index size ratio as the database size is varied.

data set	10 MBytes	100 MBytes	1 GBytes
PROTEIN	1.734 (m_o=4)	2.153 (m_o=5)	2.705 (m_o=5)
	N/A ($m_o - 1$=3)	1.847 ($m_o - 1$=4)	1.877 ($m_o - 1$=4)
TREC	1.337 (m_o=5)	1.678 (m_o=6)	2.219 (m_o=6)
	1.281 ($m_o - 1$=4)	1.677 ($m_o - 1$=5)	1.878 ($m_o - 1$=5)

6.2.2 Query Performance

Figure 15(a) shows the query processing time of the n-gram index and n-gram/2L index as the database size is varied for the PROTEIN database. Here, we set the length of m-subsequences to $(m_o - 1)$. As indicated by the time complexity in Section 5.3, the n-gram/2L index significantly improves the query performance compared with the n-gram index. Further, we obtain a larger improvement as the database size gets larger. Figure 15(a) shows that the improvement in the query performance is 1.37 times in PROTEIN-100MB and 6.65 times in PROTEIN-1GB.

Figures 15(b) and (c) show the number of page accesses and query processing time as the query length is varied for PROTEIN-1G. We note that they increase only very slightly in the n-gram/2L index as $Len(Q)$ gets longer while increasing rapidly in the n-gram index. In Fig-

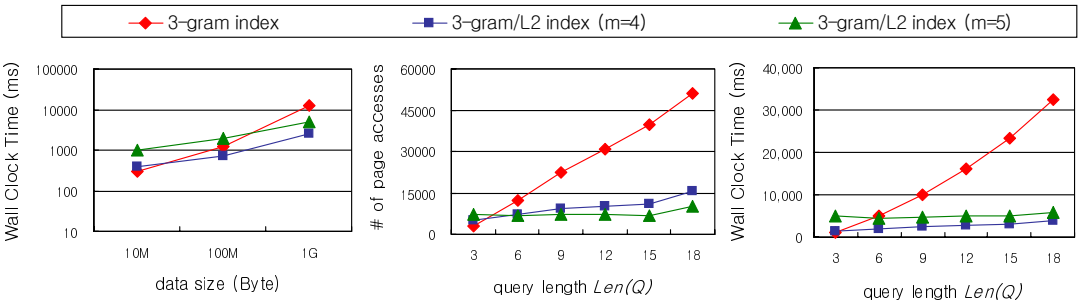


(a) The query processing time as the database size is varied. (*Len(Q)*: 3~18)

(b) The number of page accesses as *Len(Q)* is varied. (data set: PROTEIN-1G)

(c) The query processing time as *Len(Q)* is varied. (data set: PROTEIN-1G)

Figure 15: The query performance for the PROTEIN databases.



(a) The query processing time as the database size is varied. (*Len(Q)*: 3~18)

(b) The number of page accesses as *Len(Q)* is varied. (data set: TREC-1G)

(c) The query processing time as *Len(Q)* is varied. (data set: TREC-1G)

Figure 16: The query performance for the TREC databases.

ure 15(b) and Figure 15(c), as $Len(Q)$ is varied from 3 to 18, the number of page accesses for the n-gram/2L index increases only by 27% and the wall clock time only by 53% on the average, while those for the n-gram index increase by 12.0 times and by 32.9 times, respectively. In effect, the wall clock time — when considering queries shorter than six times of n — is improved by up to 13.1 times compared with those of the n-gram index.

Figure 16 shows the query performance in the TREC databases, showing a tendency similar to that in the PROTEIN databases.

7 Conclusions

In this paper, we have proposed the n-gram/2L index that significantly reduces the size and improves the query performance compared with the n-gram index. The novelty of our approach lies in finding the redundancy of the position information that exists in the n-gram index and eliminating that redundancy. To eliminate the redundancy, we construct the inverted index in two steps: 1) extracting $n-1$ overlapping subsequences of length m from documents and building the back-end index; and 2) extracting n-grams from those subsequences and building the front-end index.

We have theoretically analyzed the properties of the n-gram/2L index. First, we have formally proven in Lemma 2 that the redundancy of the position information that exists in the n-gram index is due to a non-trivial MVD. Then, we have proven in Lemma 3 and Theorem 2 that our index is derived by the relational normalization process that decomposes the n-gram index into 4NF. Second, we have analyzed the space complexity and proposed the model for determining the optimal length of m (i.e., m_o) minimizing the index size. Since the space complexity of our index is $O(|\mathbb{S}|(avg_{ngram} + avg_{doc}))$ and that of the n-gram index is $O(|\mathbb{S}|(avg_{ngram} \times avg_{doc}))$, the reduction of the index size becomes more marked as the database size gets larger. Third, we have analyzed the time complexity. Since the time complexity is shown to be the same as the space complexity, the improvement of the query performance becomes more marked as the database size gets larger. Besides, we have found out that we can speed up query processing by small sacrifice in the index size (i.e., by using $(m_o - 1)$ as the length of m-subsequences.) Fourth, we have shown that the query processing time increases only very slightly as the query length gets longer by using Formula (9).

We have performed extensive experiments for the size and query performance of the n-gram/2L index varying the data set, length of m-subsequences, database size, and query length. We have used $(m_o - 1)$ as the length of m-subsequences to speed up query processing. Experimental results using real text and protein databases of 1 GBytes show that the size of the n-gram/2L index is reduced by up to 1.9 (PROTEIN-1G, $m = 4$) $\sim$ 2.7 (PROTEIN-1G, $m = 5$) times and, at the same time, the query performance — when considering queries shorter than six times of n — is improved by up to 13.1 (PROTEIN-1G, $m = 4$) times compared with those of the n-gram index.

Overall, these results indicate that the n-gram/2L index is a new structure that can replace the n-gram index. We

expect that our index is also capable of efficiently handling approximate matching (e.g., for DNA or protein sequences), which attracts much attention recently. We investigate this method in a future paper.

Acknowledgement

This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc). We would like to thank the anonymous reviewers for their valuable comments that helped enhance the quality of this paper.

References

[1] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan, *Database Systems Concepts*, McGraw-Hill, 4th ed., 2001.

[2] Alistair Moffat and Justin Zobel, "Self-indexing inverted files for fast text retrieval," *ACM Trans. on Information Systems*, Vol. 14, No. 4, pp. 349–379, Oct. 1996.

[3] Ethan Miller, Dan Shen, Junli Liu, and Charles Nicholas, "Performance and Scalability of a Large-Scale N-gram Based Information Retrieval System," *Journal of Digital Information*, Vol. 1, No. 5, pp. 1–25, Jan. 2000.

[4] Falk Scholer, Hugh E. Williams, John Yiannis and Justin Zobel, "Compression of Inverted Indexes for Fast Query Evaluation," In *Proc. Int'l Conf. on Information Retrieval*, ACM SIGIR, Tampere, Finland, pp. 222–229, Aug. 2002.

[5] Gonzalo Navarro, "A Guided Tour to Approximate String Matching," *ACM Computing Surveys*, Vol. 33, No. 1, pp. 31–88, Mar. 2001.

[6] Hugh E. Williams, "Genomic Information Retrieval," In *Proc. the 14th Australasian Database Conferences*, 2003.

[7] Hugh E. Williams and Justin Zobel, "Indexing and Retrieval for Genomic Databases," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 14, No. 1, pp. 63–78, Jan./Feb. 2002.

[8] I. Witten, A. Moffat, and T. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, Morgan Kaufmann Publishers, Los Altos, California, 2nd ed., 1999.

[9] James Mayfield and Paul McNamee, "Single N-gram Stemming," In *Proc. Int'l Conf. on Information Retrieval*, ACM SIGIR, Toronto, Canada, pp. 415–416, July/Aug. 2003.

[10] Jonathan D. Cohen, "Recursive Hashing Functions for n-Grams," *ACM Trans. on Information Systems*, Vol. 15, No. 3, pp. 291–320, July 1997.

[11] Joon Ho Lee and Jeong Soo Ahn, "Using n-Grams for Korean Text Retrieval," In *Proc. Int'l Conf. on Information Retrieval*, ACM SIGIR, Zurich, Switzerland, pp. 216–224, 1996.

[12] Jeffery D. Ullman, *Principles of Database and Knowledge-Base Systems Vol. I*, Computer Science Press, USA, 1988.

[13] Karen Kukich, "Techniques for Automatically Correcting Words in Text," *ACM Computing Surveys*, Vol. 24, No. 4, pp. 377–439, Dec. 1992.

[14] Kyu-Young Whang, Min-Jae Lee, Jae-Gil Lee, Min-soo Kim, and Wook-Shin Han, "Odysseus: a High-Performance ORDBMS Tightly-Coupled with IR Features," In *Proc. the 21th IEEE Int'l Conf. on Data Engineering (ICDE)*, Tokyo, Japan, Apr. 2005.

[15] Ogawa Yasushi and Matsuda Toru, "Optimizing query evaluation in n-gram indexing," In *Proc. Int'l Conf. on Information Retrieval*, ACM SIGIR, Melbourne, Australia, pp. 367–368, 1998.

[16] Raghu Ramakrishnan, *Database Management Systems*, McGraw-Hill, 1998.

[17] Ramez Elmasri and Shamkant B. Navathe, *Fundamentals of Database Systems*, Addison Wesley, 4th ed., 2003.

[18] Ricardo Baeza-Yates and Berthier Ribeiro-Neto, *Modern Information Retrieval*, ACM Press, 1999.

Appendix A. Proof of Theorem 1

Let us consider a document d as a sequence of characters $c_0, c_1, ..., c_{N-1}$. Then, the number of the n-grams extracted from d is $N-n+1$, and the first character of each n-gram is c_i,$(0 \leq i < N-n+1)$. Besides, let us consider m as the subsequence length. Then, the number of the subsequences extracted from d is $\lfloor \frac{N-n+1}{m-n+1} \rfloor$, and the first character of each subsequence is c_j $(0 \leq j < \lfloor \frac{N-n+1}{m-n+1} \rfloor)$. For the convenience of explanation, let $\mathbf{N}_i$ be the n-gram with starting character c_i, and $\mathbf{S}_j$ be the subsequence with starting character c_j. If S_p and S_q are any adjacent two subsequences extracted from d, the n-grams extracted from S_p are $N_p, ..., N_{p+m-n}$, and the n-grams extracted from S_q are $N_q, ..., N_{q+m-n}$. Suppose S_p and S_q overlap with each other by the length l. Then we have the following three cases: (1) $l = n-1$, (2) $l < n-1$, and (3) $l > n-1$. We prove the Theorem by checking for each case whether there are n-grams missed or duplicated in the n-grams extracted from S_p and S_q.

Case $l = n-1$**:** Since S_p and S_q overlap with each other by $n-1$, $q = p+m-n+1$. Thus, the n-grams extracted from S_q are $N_{p+m-n+1}, ..., N_{p+2m-2n+1}$; those from S_p are $N_p, ..., N_{p+m-n}$. Thus, from S_p and S_q, the n-grams $N_p, ..., N_{p+m-n}, N_{p+m-n+1}, ..., N_{p+2m-2n+1}$ are extracted only once without being missed or duplicated.

Case $l < n-1$**:** Let us assume $l = n-2$ without loss of generality. Since S_p and S_q overlap with each other by $n-2$, $q = p+m-n+2$. Thus, the n-grams extracted from S_q are $N_{p+m-n+2}, ..., N_{p+2m-2n+2}$; those from S_p are $N_p, ..., N_{p+m-n}$. The n-gram $N_{p+m-n+1}$ is not extracted from either S_p or S_q, and therefore, is missed.

Case $l > n-1$**:** Let us assume $l = n$ without loss of generality. Since S_p and S_q overlap with each other by n, $q = p+m-n$. Thus, the n-grams extracted from S_q are $N_{p+m-n}, ..., N_{p+2m-2n}$; those from S_p are $N_p, ..., N_{p+m-n}$. The n-gram N_{p+m-n} is extracted once from each S_p and S_q, and therefore, is duplicated.

In summary, if m-subsequences are extracted such that they overlap with each other by $n-1$, no n-gram is missed or duplicated. □

Appendix B. Proof of Lemma 3

MVD's in SNO_1 are $SO_1 \rightarrow\rightarrow N$, $SNO_1 \rightarrow\rightarrow S|N|O_1$. Those in SDO_2 are $DO_2 \rightarrow\rightarrow S$, $SDO_2 \rightarrow\rightarrow S|D|O_2$. All of these MVD's are trivial ones and do not violate 4NF. □