Comput Syst Sci & Eng (2007) 6: 26–40
© 2007 CRL Publishing Ltd

International Journal of
Computer Systems
Science & Engineering

n-Gram/2L-approximation: a two-level n-gram inverted index structure for approximate string matching

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

*Department of Computer Science and Advanced Information Technology Research Center (AITrc) Korea Advanced Institute of Science and Technology (KAIST)
373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Korea
e-mail: {mskim, kywhang, jglee}@mozart.kaist.ac.kr, fax: +82-42-867-3562*

Approximate string matching is to find all the occurrences of a query string in a text database allowing a specified number of errors. Approximate string matching based on the n-gram inverted index (simply, *n-gram Matching*) has been widely used. A major reason is that it is scalable for large databases since it is not a main memory algorithm. Nevertheless, n-gram Matching also has drawbacks: the query performance tends to be bad, and many false positives occur if a large number of errors are allowed. In this paper, we propose an inverted index structure, which we call the *n-gram/2L-Approximation index*, that improves these drawbacks and an approximate string matching algorithm based on it. The n-gram/2L-Approximation is an adaptation of the n-gram/2L index [4], which the authors have proposed earlier for exact matching. Inheriting the advantages of the n-gram/2L index, the n-gram/2L-Approximation index reduces the size of the index and improves the query performance compared with the n-gram inverted index. In addition, the n-gram/2L-Approximation index reduces false positives compared with the n-gram inverted index if a large number of errors are allowed. We perform extensive experiments using the text and protein databases. Experimental results using databases of 1 GBytes show that the n-gram/2L-Approximation index reduces the index size by up to 1.8 times and, at the same time, improves the query performance by up to 4.2 times compared with those of the n-gram inverted index.

Keywords: Approximate String Matching, n-Gram, Inverted Index

1. INTRODUCTION

Approximate string matching is to find all the occurrences of a query string in a text database allowing a specified number of errors [8]. It has a number of applications such as those for searching text documents with typo's errors and for finding DNA or protein sequences with possible mutations. DNA or protein sequences can be regarded as a long text over specific alphabets (e.g., {A,C,G,T} in DNA) [6].

The algorithms for approximate string matching are classified into two categories depending on whether they use indexes: *indexing algorithms* and *online algorithms*. The former uses indexes, while the latter does not. There have been a number of efforts on online algorithms, but relatively few on indexing algorithms. Since online algorithms typically use a sequential scan over the entire database, they are not easily scalable for large databases. Thus, it has been pointed out that developing indexing algorithms is very important [6]. Indexing algorithms are classified depending on the types of the index structures they use. The indexes used include the suffix tree, suffix array, q-sample index, and n-gram inverted index (simply, the *n-gram index*) [8].

Indexing algorithms based on the n-gram index (simply, *n-gram Matching*) has been widely used for approximate string matching due to the following advantage: the n-gram index is easily scalable for large databases because it is not a main memory algorithm, and thus, is not limited by the size of main memory [1, 3, 8]. n-gram Matching retrieves candidate results by finding documents that contain n-grams extracted from a query string, and then, performs refinement by using an online algorithm [7]. Despite these advantages, n-gram Matching also has the following drawbacks. First, the query performance tends to be bad because of large index size [7, 8, 14]. Second, many false positives occur if a large number of errors are allowed [7, 8]. In an extreme case, n-gram Matching is not capable of reducing the candidates despite of using an index.

In order to solve the these drawbacks, we adapt the n-gram/2L index [4], which the authors have earlier proposed for exact matching, to approximate string matching. The n-gram/2L index reduces the size of the index and improves the query performance compared with the n-gram index by eliminating the redundant information that exists in the n-gram index. In this paper, we propose the *n-gram/2L-Approximation index*, which is an adaptation of the n-gram/2L index. We then propose an approximate string matching algorithm based on it. The n-gram/2L-Approximation index is constructed in two levels just in the same way as the n-gram/2L index is: the back-end index and the front-end index.

The n-gram/2L-Approximation index inherits the following excellent properties of the n-gram/2L index. First, the n-gram/2L-Approximation index improves the query performance compared with n-gram Matching, and such improvement becomes more marked in a larger database or for a longer query string. Second, the n-gram/2L-Approximation index reduces the size compared with the n-gram index, and such reduction becomes more marked in a larger database. We investigate the reasons for these desirable properties in Section 5.

The rest of this paper is organized as follows. Section 2 explains approximate string matching. Section 3 presents n-gram Matching. Section 4 proposes the structure and algorithm of the n-gram/2L-Approximation index. Section 5 presents the formal model of the n-gram/2L-Approximation index and analyzes the size of the index and the query performance. Section 6 presents the results of performance evaluation. Section 7 summarizes and concludes the paper.

Table 1 Summary of notation.

Symbols	Definitions
N	the number of documents
D_i	the i-th document ($1 \leq i \leq N$)
d_i	the identifier of the i-th document ($1 \leq i \leq N$)
$D_i[p:q]$	the substring of D_i, consisting of characters from the p-th one through the q-th one ($p \leq q$)
Q	the query string
$Len(s)$	the length of the string s
k	the error tolerance (user-specified maximum acceptable edit distance)
α	the error ratio ($= \frac{k}{Len(Q)}$)

2. PROBLEM DEFINITION

In this paper, we deal with approximate string matching. Approximate string matching is to find the set of pairs of the document and offset where a query string matches within a given error tolerance [6]. A number of distance measures have been proposed for measuring errors between two strings, and the edit distance [6] is the most widely used one. We also use the edit distance as the distance measure.

The *edit distance* between two strings x and y is defined as the minimal number of edit operations needed to convert x into y (or y into x). Here, an *edit operation* means insertion, deletion, or substitution of a character. $edit(x, y)$ denotes the edit distance between x and y, and x matches y within k errors if and only if $edit(x, y) \leq k$.

We first summarize in Table 1 the notation to be used throughout the paper. In Table 1, the *error ratio* α means the ratio of the error tolerance to the length of query string, i.e., $\frac{k}{Len(Q)}$.

Now, we formally define approximate string matching in Definition 1 by using the notations in Table 1.

Definition 1 Suppose that a query string Q, an error tolerance k, and a set of documents $\{D_1, D_2, ..., D_N\}$ are given. *Approximate string matching* is to find the set $\{(d_i, p)\}$ of pairs of the identifier d_i of D_i and offset p that satisfy $edit(Q, D_i[p:q]) \leq k$ for some offset q. □

3. RELATED WORK

In this section, we explain n-gram Matching. Then, we discuss its advantages and disadvantages. n-gram Matching is performed in the following two steps [8]: (1) finding candidate documents that satisfy a necessary condition by searching the n-gram index with n-grams extracted from a query string; (2) doing refinement in order to find final results by using online algorithms. The necessary condition used in the first step is proposed by Navarro et al. [8] and Gravano et al. [3] as in Lemma 1.

Lemma 1 [The necessary condition for n-gram Matching] [3, 8]: Suppose that a query string Q, a document D, and an error tolerance k satisfy $edit(Q, D[p:q]) \leq k$ for some offsets p and q. The query string Q is divided into disjoint n-grams $\{G_i\}(1 \leq i \leq \lfloor\frac{Len(Q)}{n}\rfloor)$. Then, the following two conditions are satisfied: (1) among the set of n-grams $\{G_i\}$, at least $(\lfloor\frac{Len(Q)}{n}\rfloor - k)$ n-grams $\{g_j\}(1 \leq j \leq \lfloor\frac{Len(Q)}{n}\rfloor - k)$ appear in the document D; (2) for any n-gram g_j, the offset o_{qj} of g_j within Q and the offset o_{dj} of g_j within D satisfy $|(o_{dj} - p) - o_{qj}| \leq k$.

PROOF: We prove the Lemma for each condition.

Condition (1): $edit(Q, D[p:q]) \leq k$ means that at most k edit operations are required to convert Q into $D[p:q]$. Each edit operation is able to modify at most one of the $\lfloor\frac{Len(Q)}{n}\rfloor$ n-grams. Thus, among disjoint $\lfloor\frac{Len(Q)}{n}\rfloor$ n-grams $\{G_i\}$, at least $(\lfloor\frac{Len(Q)}{n}\rfloor - k)$ n-grams $\{g_j\}$ must remain unchanged in $D[p:q]$. Those n-grams also must appear in the document D, which contains $D[p:q]$ as a substring.

Condition (2): k edit operations change the offset of g_j by at most k. Thus, the offset o_{qj} of g_j within Q and the offset

o_{pj} of g_j within $D[p:q]$ satisfy $|o_{pj} - o_{qj}| \leq k$. Since $o_{pj} = (o_{dj} - p)$, we have $|(o_{dj} - p) - o_{qj}| \leq k$.

□

n-gram Matching has the advantage of being scalable since it is not bounded by the size of available main memory. In contrast, the suffix tree and suffix array are not easily scalable for large databases because the index should reside in main memory and is limited by its size [1, 8]. Despite this advantage, the query performance of n-gram Matching tends to be bad [7, 8, 14] mainly for the following reasons. First, finding candidate documents is time-consuming because of the large size of the n-gram index. The large size of the index makes n-gram Matching read in many postings from the index during query processing. This large size is caused by the method of extracting terms: the n-gram index extracts n-grams at each character offset in a document, so that a very large number of n-grams are extracted from the document. Second, if a large error tolerance is given, the refinement process is also time-consuming because of a larger number of candidate documents. The number of candidate documents gets larger as the error tolerance k does since a larger k makes n-gram Matching retrieve documents containing fewer n-grams as candidates according to Lemma 1. Furthermore, n-gram Matching can not have the benefit of using the index if the error ratio exceeds a certain threshold [7]. If $\alpha \geq \frac{1}{n}$ (i.e., $k \geq \lfloor \frac{Len(Q)}{n} \rfloor$), n-gram Matching finds documents containing zero or more n-grams by Lemma 1, that is, it selects all documents as candidates. We define the *maximum error ratio* as $\frac{1}{n}$. It has been pointed out as a drawback that n-gram Matching has a low maximum error ratio compared with other indexing algorithms [8].

One can argue that we could decrease the time for performing refinement (simply, *refinement time*) and increase the maximum error ratio by reducing the length n of n-grams [3, 8, 10]. When we use a smaller value of n, since n-gram Matching checks the necessary condition of Lemma 1 with more n-grams, the number of candidate documents is reduced, and, at the same time, the refinement time is reduced. A smaller n also increases the maximum error ratio $\frac{1}{n}$. Nevertheless, a smaller n drastically increases the time for finding candidates (simply, *filtration time*) since both the length and the number of posting lists accessed during query processing are increased. Thus, we can not easily improve the performance by decreasing n, and the problem of a low maximum error ratio of n-gram Matching can not be easily solved.

4. N-GRAM/2L-APPROXIMATION INDEX

4.1 Index Structure

Figure 1 shows the structure of the n-gram/2L-Approximation index. Just like the n-gram/2L index [4], this index consists of the back-end index and the front-end index. The *back-end index* uses an m-subsequence as a term and stores the offsets of the m-subsequence within documents in the posting list of the m-subsequence. The *m-subsequence* is defined as the subsequence of length m. The *front-end index* uses an n-gram as a term and stores the offsets of the n-gram within m-subsequences in the posting list of the n-gram. We note that n denotes the length of the n-gram, and m the length of the m-subsequence.

4.2 Index Building Algorithm

The n-gram/2L-Approximation index is built through the following four steps: (1) extracting m-subsequences, (2) building the back-end index, (3) extracting n-grams, and (4) building the front-end index. Figure 2 shows the algorithm for building the n-gram/2L-Approximation index. We call this algorithm *n-gram/2L-Approximation Index Building*. In Step 1, the algorithm extracts m-subsequences from a set of documents. The building algorithm of the n-gram/2L index extracts m-subsequences such that they overlap with each other by $n-1$ [4]. In contrast, this algorithms extracts m-subsequences such that they are disjoint and do not overlap with each other. This intends to improve the query performance by reducing the number of m-subsequences accessed in the back-end index. We explain the method of extracting m-subsequences in more detail in Section 4.4. In Step 2, the algorithm builds the back-end index using the m-subsequences obtained in Step 1. In Step 3, the algorithm extracts n-grams from the set of m-subsequences obtained in Step 1. Here, the algorithm extracts n-grams by sliding a window of length n by one character in the m-subsequence and recording a sequence of characters in the window at each time. We call it the *1-sliding technique*. In Step 4, the algorithm builds the front-end index using the n-grams obtained in Step 3.

Example 1 Figure 3 shows an example of building the n-gram/2L-Approximation index. Suppose that $n=2$ and $m=4$. Figure 3(a) shows the set of documents. Figure 3(b) shows the set of the 4-subsequences extracted from the documents. Since 4-subsequences are extracted such that they are disjoint, those extracted from the document 0 are “ABCC”, “CDAB”, and “DABC”. Figure 3(c) shows the back-end index built from these 4-subsequences. Since the 4-subsequence “ABCC” occurs at the offsets 0, 8, and 0 in the documents 0, 2, and 3, respectively, the postings $<0,[0]>$, $<2,[8]>$, and $<3,[0]>$ are appended to the posting list of the 4-subsequence “ABCC”. Figure 3(d) shows the set of the 4-subsequences and their identifiers. Figure 3(e) shows the set of the 2-grams extracted from the 4-subsequences in Figure 3(d). Since 2-grams are extracted by the 1-sliding technique, those extracted from the 4-subsequence 0 are “AB”, “BC”, and “CC”. Figure 3(f) shows the front-end index built from these 2-grams. Since the 2-gram “AB” occurs at the offsets 0, 2, and 1 in the 4-subsequences 0, 2, and 3, respectively, the postings $<0,[0]>$, $<2,[2]>$, and $<3,[1]>$ are appended to the posting list of the 2-gram “AB”. □

A small value of n in n-gram index decreases the refinement time and improve the maximum error ratio, but it significantly increases the filtration time as explained in Section 3. In contrast, the increment of the filtration time is not significant in the n-gram/2L-Approximation index since the size of the front-end index is very small compared with that of the n-gram index. Thus, we can use a smaller n than that of the n-gram index in order to decrease the refinement time and improve the maximum error ratio. This is one of the major advantages of the n-gram/2L-Approximation index compared with the n-gram index.

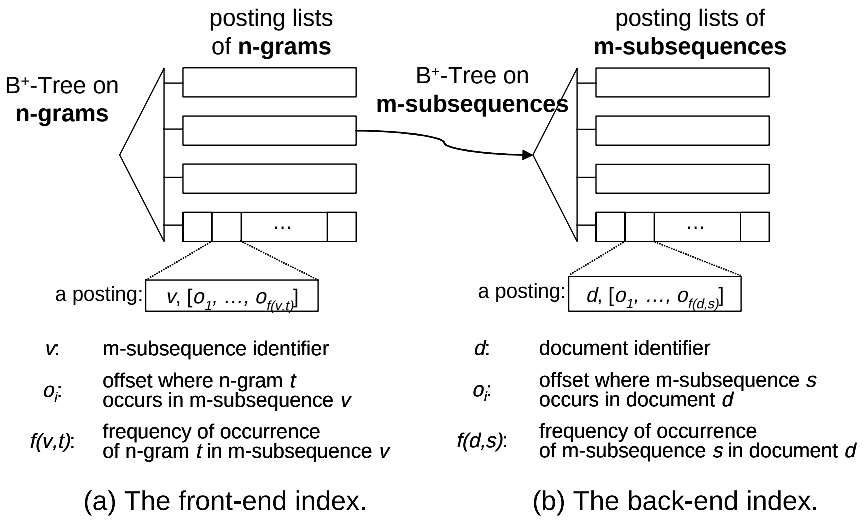

posting lists of n-grams
B+-Tree on n-grams
B+-Tree on m-subsequences
posting lists of m-subsequences
a posting: v, [o1, ..., of(v,t)]
a posting: d, [o1, ..., of(d,s)]
v: m-subsequence identifier
oi: offset where n-gram t occurs in m-subsequence v
f(v,t): frequency of occurrence of n-gram t in m-subsequence v
d: document identifier
oi: offset where m-subsequence s occurs in document d
f(d,s): frequency of occurrence of m-subsequence s in document d
(a) The front-end index.
(b) The back-end index.

Figure 1 The structure of the n-gram/2L-Approximation index.

Algorithm *n-Gram/2L-Approximation Index Building*:

Input: (1) The document collection D, (2) The length m of m-subsequences , (3) The length n of n-grams

Output: The n-gram/2L-Approximation 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 $c_{i*m}$ ($0 \le i < \lfloor N / m \rfloor$) 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 $c_i$ ($0 \le i < L-n+1$) 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 2 The algorithm of building the n-gram/2L-Approximation index.

4.3 Query Processing Algorithm

4.3.1 Overview

The query processing of the n-gram/2L-Approximation index is performed in two steps: (1) searching the front-end index, (2) searching the back-end index. Figure 4 shows an overview of the query processing algorithm that uses the n-gram/2L-Approximation index. In the first step, we find the m-subsequences that approximately match with a query string by searching the front-end index with the n-grams extracted from the query string. In the second step, we find the documents that approximately match with the query string by searching the back-end index with the m-subsequences retrieved in the first step. In each step, we obtain the candidate results by performing filtration, and then, find the final results by performing refinement.

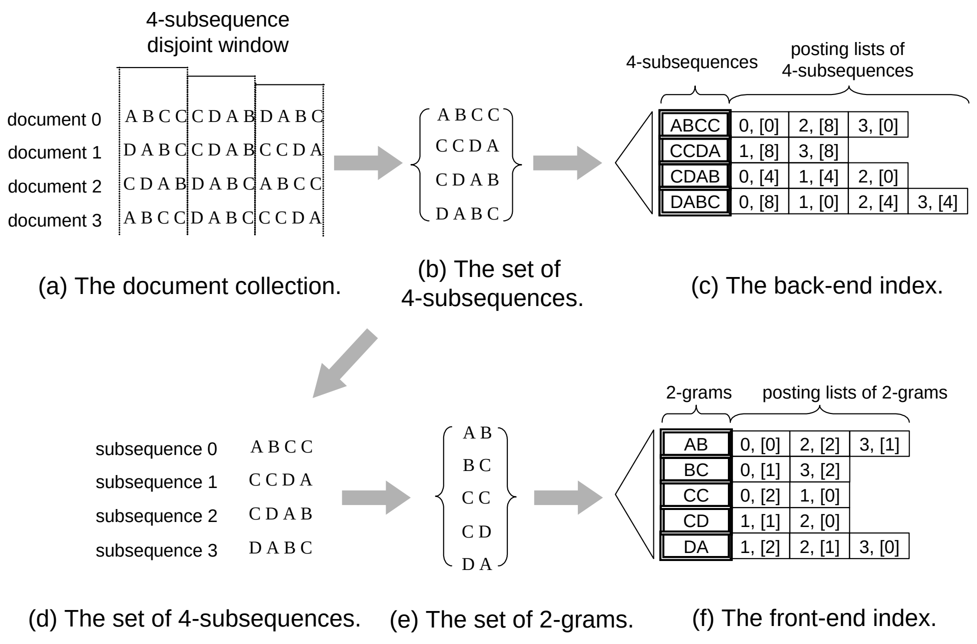


Figure 3 An example of building the n-gram/2L-Approximation index.

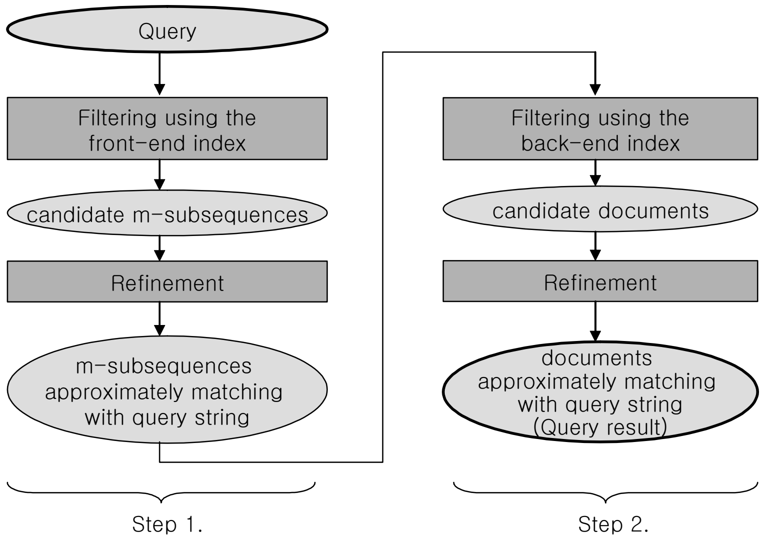

Query
Filtering using the front−end index
candidate m−subsequences
Refinement
m−subsequences approximately matching with query string
Filtering using the back−end index
candidate documents
Refinement
documents approximately matching with query string (Query result)
Step 1.
Step 2.

Figure 4 An overview of the query processing algorithm that uses the n-gram/2L-Approximation index.

4.3.2 The Conditions for Filtration

The filtration operations in Steps 1 and 2 are performed by checking whether m-subsequences or documents satisfy necessary conditions. We are able to reduce the number of candidates by filtering out m-subsequences or documents that do not satisfy the necessary conditions. Since we filter out only the m-subsequences or documents that do not satisfy the necessary conditions, no false drop occurs. Hereafter, we call the filtration operation in Step 1 as the *front-end filtration* and that in Step 2 as the *back-end filtration*. In this section, we explain the necessary conditions for the front-end filtration and back-end filtration.

We first define *ε-match* in Definition 2. When $\varepsilon = 0$ in Definition 2, x exactly matches with $y[p : q]$. This is a special case of ε-match and is denoted as 0-match.

Definition 2 A string x *ε-matches* with a string y if two strings x and y satisfy $edit(x, y[p : q]) \le k$ at some offsets p and q. Here, *ε-offset* denotes the offset p, and *ε-substring(y)* denotes the substring $y[p : q]$. □

We now explain the necessary condition for the front-end filtration in Theorem 1 and that for the back-end filtration in Theorem 2. Theorems 1 and 2 are adapted from Lemma 1. Theorem 1 is different from Lemma 1 in that it assumes n-grams extracted by using the *1-sliding technique* instead of *disjoint* n-grams. Theorem 2 is different from Lemma 1 in that it handles not only *exact* matching but also *approximate* matching with the query string.

Theorem 1 [The necessary condition for the front-end filtration]: Suppose that an m-subsequence S ε-matches with a query string Q at an ε-offset p, and a set of n-grams $\{G_i\}(1 \le$

$i \le m - n + 1$) are extracted from the m-subsequence S by using the 1-sliding technique. Then, the following two conditions are satisfied: (1) among the set of n-grams $\{G_i\}$, at least $r = (m - n + 1) - (\varepsilon \times n)$ n-grams $\{g_j\}(1 \le j \le r)$ 0-matches with the query string Q; (2) for any n-gram g_j, the offset o_{sj} of g_j within S and the 0-offset o_{qj} of g_j within Q satisfy $|(o_{qj} - p) - o_{sj}| \le \varepsilon$.

PROOF: See Appendix A. □

Theorem 2 [The necessary condition for the back-end filtration]: Suppose that a query string Q ε-matches with a document D at an ε-offset p, and ε-substring(D) is divided into disjoint $t = \lfloor \frac{Len(Q)+1}{m} \rfloor - 1$ m-subsequences $\{S_i\}$ $(1 \le i \le t)$. Then, the following two conditions are satisfied: (1) among the set of m-subsequences $\{S_i\}$, at least $r = t - \lfloor \frac{\varepsilon}{\lfloor \frac{\varepsilon}{t} \rfloor + 1} \rfloor$ m-subsequence $\{s_j\}(1 \le j \le r)$ $\lfloor \frac{\varepsilon}{t} \rfloor$-matches with the query string Q; (2) for any m-subsequence s_j, the offset o_{dj} of s_j within D and the $\lfloor \frac{\varepsilon}{t} \rfloor$-offset o_{qj} of s_j within Q satisfy $|(o_{dj} - p) - o_{qj}| \le \varepsilon$.

PROOF: See Appendix B. □

4.3.3 Algorithm

Figure 5 shows the algorithm of approximate string matching that uses the n-gram/2L-Approximation index. We call this algorithm *n-gram/2L-Approximation Matching*. When a query string Q, a error tolerance k, and a set of documents $\{D_i\}$ are given as input parameters, the algorithm outputs the set of documents that k-matches with the query string Q. By Theorem 2, we need to find m-subsequences that $\lfloor \frac{k}{t} \rfloor$-matches with Q in order to find documents that k-matches with Q. Thus, the algorithm first finds m-subsequences that $\lfloor \frac{k}{t} \rfloor$-matches with Q by substituting ε with $\lfloor \frac{k}{t} \rfloor$ in Theorem 1. We now explain each step of the algorithm in more detail.

Step 1: The algorithm extracts n-grams from the query string Q by the 1-sliding technique and searches the posting lists of those n-grams in the front-end index. Then, the algorithm performs merge outer join among those posting lists using the m-subsequence identifier as the join attribute and finds the set $\{S_i\}$ of candidate m-subsequences that satisfy the necessary condition in Theorem 1. Since an candidate m-subsequence S_i typically does not have all the n-grams extracted from Q, the algorithm performs merge outer join in Step 1.2. Next, the algorithm checks whether S_i indeed $\lfloor \frac{k}{t} \rfloor$-matches with Q by performing refinement. If S_i $\lfloor \frac{k}{t} \rfloor$-matches with Q, it adds the identifier s_i of S_i into the set S_{match}.

Step 2: The algorithm performs merge outer join among the posting lists of the m-subsequences in S_{match} using the document identifier as the join attribute and obtains the set $\{D_i\}$ of candidate documents that satisfy the necessary condition in Theorem 2. Since a candidate document D_i typically does not have all the m-subsequences in S_{match}, the algorithm performs merge outer join in Step 2.1. Next, the algorithm checks whether Q indeed k-matches with D_i by performing refinement. If Q k-matches with D_i, it returns the pair of the identifier d_i of D_i and k-offset p, i.e., (d_i, p) as the query result.

4.4 The Method of Extracting m-Subsequences

The method of extracting m-subsequences in the n-gram/2L-Approximation index is different from that in the n-gram/2L index. In the n-gram/2L-Approximation index, we extract m-subsequences such that they are disjoint as mentioned in Section 4.2. This difference intends to decrease the overhead of searching the back-end index by reducing the size of S_{match} to be searched in the back-end index.

Suppose that m-subsequences are extracted such that they overlap with each other by $n - 1$ in Theorem 2. Then, $t' = \lfloor \frac{Len(D)-n+2}{m-n+1} - 1 \rfloor$ m-subsequences are extracted from ε-substring(D). One edit operation is able to give at most two errors to the t' m-subsequences by applying it to the part where two m-subsequences overlap with each other. Thus, k edit operations is able to give at most $2k$ errors to the t' m-subsequences. No matter how we apply $2k$ edit operations to t' m-subsequences, at least one m-subsequence with at most $\lfloor \frac{2k}{t'} \rfloor$ errors appears in a query string Q. That is, at least one m-subsequence $\lfloor \frac{2k}{t'} \rfloor$-matches with Q. Therefore, under this assumption, Theorem 2 is modified so as to search for m-subsequences $\lfloor \frac{2k}{t'} \rfloor$-matching with Q.

Since $\lfloor \frac{2k}{t'} \rfloor > \lfloor \frac{k}{t} \rfloor$ for typical values of n and m (for example, n=2, m=4 or 5), the number of m-subsequences $\lfloor \frac{2k}{t'} \rfloor$-matching with Q becomes larger than that of m-subsequences $\lfloor \frac{k}{t} \rfloor$-matching with Q. We note that the number of m-subsequences $\lfloor \frac{2k}{t'} \rfloor$-matching (or $\lfloor \frac{k}{t} \rfloor$-matching) with Q is the size of S_{match}. Thus, it is preferable to extract m-subsequences such that they are disjoint since a smaller size of S_{match} can improve the query performance.

5. FORMAL ANALYSIS OF THE N-GRAM/2L-APPROXIMATION INDEX

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

5.1 Formalization

Kim et al. [4] have observed that the redundancy of the position information existing in the n-gram index is caused by a non-trivial multivalued dependency (MVD) [2, 11] and shown that the n-gram/2L index can be derived by eliminating that redundancy through relational decomposition to the Fourth Normal Form (*4NF*). In this section, we show that the n-gram/2L-Approximation index can be derived in the same way as the n-gram/2L index is.

For the sake of theoretical development, Kim et al. have first considered the relation that is converted from the n-gram index so as to obey the First Normal Form (1NF). This relation is called the *NDO relation*. It has three attributes N, D, and O. Here, N indicates n-grams, D document identifiers, and O offsets. Further,

```
Algorithm n-Gram/2L-Approximation Matching:
Input: (1) The n-gram/2L-Approximation index
       (2) A query string $Q$
       (3) A error tolerance $k$
       (4) A document collection $\{D_i\}$
Output: the set $\{(d_i, p)\}$ of pairs of the identifier $d_i$ of $D_i$ and offset $p$ that $k$-matches with $Q$
Algorithm:
Step 1.  Searching the front-end inverted index: let $t = \lfloor (Len(Q)+1)/m \rfloor -1$
    1.1  Extract n-grams from $Q$ by the 1-sliding technique 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;
         find a candidate m-subsequence $S_i$ that satisfies the necessary condition in Theorem 1.
         1.2.1 Perform refinement on $S_i$ ;
               If $S_i$ $\lfloor k/t \rfloor$-matches with $Q$, add the identifier $s_i$ of $S_i$ into the set $S_{match}$.
Step 2.  Searching the back-end inverted index:
    2.1 Search the posting lists of m-subsequences in $S_{match}$.
    2.2  Perform merge outer join among those posting lists using the document identifier as the join attribute;
         find a candidate document $D_i$ that satisfies a necessary condition in Theorem 2.
         2.2.1  Perform refinement on $D_i$ ;
                If $Q$ $k$-matches with $D_i$, return the pair $(d_i, p)$ of the identifier $d_i$ of $D_i$ and $k$-offset $p$.
```

Figure 5 The n-gram/2L-Approximation Matching algorithm.

Kim et al. have considered the relation obtained by adding the attribute S and by splitting the attribute O into two attributes O_1 and O_2. This relation is called the *SNDO$_1$O$_2$ relation*. It 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. In the tuple (s, n, d, o_1, o_2) 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.

Kim et al. have proven that non-trivial MVDs hold in the relation SNDO$_1$O$_2$ (i.e., the n-gram index) in Theorem 3, and the n-gram/2L index is derived from the relation SNDO$_1$O$_2$ in Theorem 4.

Theorem 3 [4] The non-trivial MVDs $S \to\to NO_1$ and $S \to\to DO_2$ hold in the relation SNDO$_1$O$_2$. Here, S is not a superkey.

Theorem 4 [4] The 4NF decomposition (SNO$_1$, SDO$_2$) of the relation SNDO$_1$O$_2$ is identical to the front-end and back-end indexes of the n-gram/2L index.

In this paper, we also use the NDO relation and the SNDO$_1$O$_2$ relation for the sake of theoretical development. We note that some n-grams can not be extracted from a document due to our method of extracting m-subsequences. We define those n-grams in Definition 3.

Definition 3 Suppose that m-subsequences are extracted such that they are disjoint and do not overlap with each other. The *missing n-grams* are those that are not extracted since they are located across two consecutive m-subsequences as in Figure 6. □

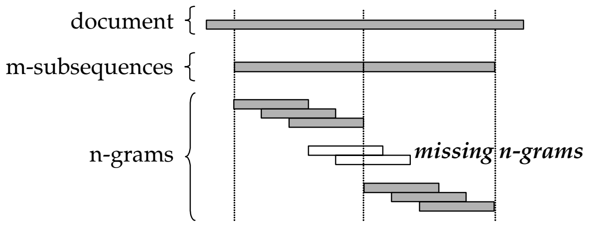


Figure 6 An example of missing n-grams.

Suppose that a missing n-gram n occurs at an offset o in a document d. We call the corresponding tuple (n, d, o) in the NDO relation the *missing tuple*. We call the relation where all the missing tuples are eliminated from the NDO relation as the *reduced NDO relation*. Further, we consider the relation obtained by adding the attribute S to the reduced NDO relation and by splitting the attribute O into two attributes O_1 and O_2. We call this relation the *reduced SNDO$_1$O$_2$ relation*.

Now, we prove that non-trivial MVDs hold in the reduced SNDO$_1$O$_2$ relation (i.e., the n-gram index where all the missing n-grams are eliminated) in Theorem 5.

Theorem 5 The non-trivial MVDs $S \to\to NO_1$ and $S \to\to DO_2$ hold in the reduced SNDO$_1$O$_2$ relation. Here, S is not a superkey.

PROOF: We omit a formal proof since the steps in the proof are similar to those of Lemma 2 by Kim et al. [4]. □

Intuitively, non-trivial MVDs hold in the $SNDO_1O_2$ relation 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. This independency hold not only in the n-gram/2L index but also in the n-gram/2L-Approximation index. Thus, the proof for the n-gram/2L index can be directly applied to that for the n-gram/2L-Approximation index.

Example 2 Figure 7 shows an example showing the existence of the non-trivial MVDs $S \rightarrow\rightarrow NO_1$ and $S \rightarrow\rightarrow DO_2$ in the reduced $SNDO_1O_2$ relation. Suppose that we build the 2-gram index on the documents in Figure 3(a). Figure 7(a) shows the NDO relation converted from that index. Here, the shaded tuples of the NDO relation indicate missing tuples. Suppose that we convert the NDO relation into the reduced NDO relation by eliminating all the missing tuples. Figure 7(b) shows the reduced $SNDO_1O_2$ relation ($m = 4$) derived from that relation. Here, the tuples of the reduced $SNDO_1O_2$ relation are sorted by the values of the attribute S. In the tuples contained in the thick-lined box of the reduced $SNDO_1O_2$ relation in Figure 7(b), there exists the redundancy that the DO_2-values (0, 0), (2, 8), and (3, 0) 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 “ABCC”. We note that such repetitions also occur in the other S-values. □

Kim et al. [4] have shown the process of obtaining the n-gram/2L index by decomposing the $SNDO_1O_2$ relation so as to obey 4NF. Likewise, we obtain the n-gram/2L-Approximation index by decomposing the reduced $SNDO_1O_2$ relation so as to obey 4NF.

5.2 Analysis of the Index Size

The space complexity of the n-gram/2L-Approximation index is the same as that of the n-gram/2L index. This is because, as shown in Section 5.1, the n-gram2/L-Approximation index is obtained through relational decomposition to the 4NF in the same way as the n-gram/2L index is.

Kim et al. [4] have shown that the space complexity of the n-gram index is $O(avg_{ngram} \times avg_{doc})$, while that of the n-gram/2L index is $O(avg_{ngram} + avg_{doc})$. Here, avg_{ngram} is the average number of the n-grams extracted from an m-subsequence, and avg_{doc} is the average number of occurrences of an m-subsequence in the documents. Equation (5.1) shows the ratio of the size of the n-gram index to that of the n-gram/2L index. Both avg_{ngram} and avg_{doc} tend to increase as the database size gets larger. Since $(avg_{ngram} \times avg_{doc})$ increases more rapidly than $(avg_{ngram} + avg_{doc})$ does, the ratio in Equation (5.1) increases as the database size does. Therefore, the n-gram/2L-Approximation index has the characteristic of reducing the index size more for a larger database.

$$\frac{\text{size}_{\text{ngram}}}{\text{size}_{\text{front}} + \text{size}_{\text{back}}} \approx \frac{\text{avg}_{\text{ngram}} \times \text{avg}_{\text{doc}}}{\text{avg}_{\text{ngram}} + \text{avg}_{\text{doc}}} \qquad (5.1)$$

The size of the n-gram/2L-Approximation index is dependent on the length m of m-subsequence. We denote the optimal length of m that minimizes the index size by m_o. To find m_o, we preprocess the document collection before building the index. m_o tends to increase as the database size does. See the work by Kim et al. [4] for the detailed method of finding m_o.

5.3 Analysis of the Query Performance

The parameters affecting the query performance of the n-gram/2L-Approximation index are m, n, the error tolerance k, 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. The query processing time is the sum of the filtration time and refinement time. By using a smaller n, the n-gram/2L-Approximation index reduces the refinement time compared with the n-gram index as explained in Section 4.2. In this section, we focus on the filtration time.

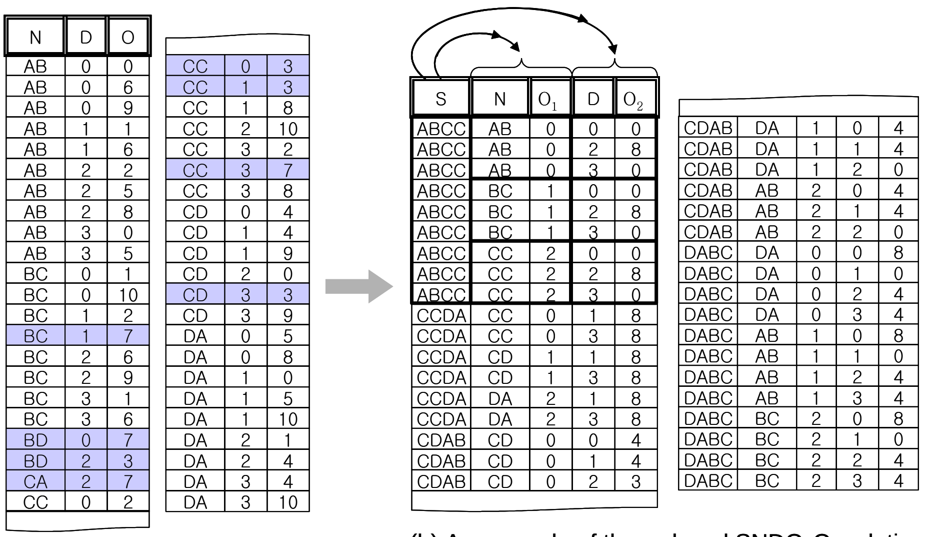
The analysis of the query performance in this paper is done in a way similar to that in Kim et al. [4]. However, The analysis in this paper is for approximate string matching queries, and that in the earlier work is for exact string matching queries. The details are completely different in the number of offsets and the number of posting lists accessed during query processing.

For simplicity of our analysis, we first make the following three assumptions. (1) the query processing time is proportional to the number of offsets 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. (3) the number of disjoint m-subsequences included in a query string Q is $t' = \lfloor \frac{Len(Q)}{m} \rfloor$ rather than $t = \lfloor \frac{Len(Q)+1}{m} \rfloor - 1$. Third assumption is to simplify computation.

We summarize in Table 2 the notation to be used for analyzing the query performance.

5.3.1 Analysis of the Number of Offsets Accessed

The ratio of the query performance of the n-gram index to that of the n-gram/2L-Approximation index is computed through Equations (5.2)~(5.5). 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 $\lfloor \frac{Len(Q)}{n} \rfloor$, the query processing time is as in Equation (5.2). In the front-end index of the n-gram/2L-Approximation 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 Equation (5.3). In the back-end index of the n-gram/2L-Approximation index, K_{offset} is $\frac{size_{back}}{\sigma^m}$. Besides, K_{plist} is the number of m-subsequences that ε-match with Q ($\varepsilon = \lfloor \frac{k}{t'} \rfloor$). K_{plist} is at most $(Len(Q)-m+1) \times C(m, \varepsilon)\sigma^{\varepsilon}$ because the number of m-subsequences extracted from Q is $(Len(Q) - m + 1)$, and there exist at most $C(m, \varepsilon)\sigma^{\varepsilon}$ m-subsequences that ε-match with each m-subsequence. Note that $C(m, \varepsilon)\sigma^{\varepsilon}$ is the maximum



N	D	O
AB	0	0
AB	0	6
AB	0	9
AB	1	1
AB	1	6
AB	2	2
AB	2	5
AB	2	8
AB	3	0
AB	3	5
BC	0	1
BC	0	10
BC	1	2
BC	1	7
BC	2	6
BC	2	9
BC	3	1
BC	3	6
BD	0	7
BD	2	3
CA	2	7
CC	0	2
CC	0	3
CC	1	3
CC	1	8
CC	2	10
CC	3	2
CC	3	7
CC	3	8
CD	0	4
CD	1	4
CD	1	9
CD	2	0
CD	3	3
CD	3	9
DA	0	5
DA	0	8
DA	1	0
DA	1	5
DA	1	10
DA	2	1
DA	2	4
DA	3	4
DA	3	10

(a) An example of the NDO relation.

S	N	O_1	D	O_2
ABCC	AB	0	0	0
ABCC	AB	0	2	8
ABCC	AB	0	3	0
ABCC	BC	1	0	0
ABCC	BC	1	2	8
ABCC	BC	1	3	0
ABCC	CC	2	0	0
ABCC	CC	2	2	8
ABCC	CC	2	3	0
CCDA	CC	0	1	8
CCDA	CC	0	3	8
CCDA	CD	1	1	8
CCDA	CD	1	3	8
CCDA	DA	2	1	8
CCDA	DA	2	3	8
CDAB	CD	0	0	4
CDAB	CD	0	1	4
CDAB	CD	0	2	3
CDAB	DA	1	0	4
CDAB	DA	1	1	4
CDAB	DA	1	2	0
CDAB	AB	2	0	4
CDAB	AB	2	1	4
CDAB	AB	2	2	0
DABC	DA	0	0	8
DABC	DA	0	1	0
DABC	DA	0	2	4
DABC	DA	0	3	4
DABC	AB	1	0	8
DABC	AB	1	1	0
DABC	AB	1	2	4
DABC	AB	1	3	4
DABC	BC	2	0	8
DABC	BC	2	1	0
DABC	BC	2	2	4
DABC	BC	2	3	4

(b) An example of the reduced $SNDO_1O_2$ relation. (sorted by the values of the attribute S)

Figure 7 An example showing the existence of non-trivial MVDs in the reduced $SNDO_1O_2$ relation.

Table 2 Summary of notation used for analyzing the query performance.

Symbols	Definitions
Σ	the alphabet
σ	the size of the alphabet (= $\lvert\Sigma\rvert$)
n	the length of the n-gram in the n-gram index
n'	the length of the n-gram in the n-gram/2L-Approximation index
k	the error tolerance (user-specified maximum acceptable edit distance)
K_{offset}	the average number of offsets in a posting list
K_{plist}	the number of posting lists accessed during query processing
$C(a,b)$	combination: a choose b
t'	the number of disjoint m-subsequences included in a query string Q (= $\lfloor\frac{Len(Q)}{m}\rfloor$)
ε	the error tolerance used in the front-end index (= $\lfloor\frac{k}{t'}\rfloor$)
λ	the time to randomly access a posting list

number of distinct m-subsequences that can be generated by ε edit operations. Hence, the query processing time in the back-end index is as in Equation (5.4). Finally, Equation (5.5) shows the ratio of the query processing times.

$$offset\_time_{ngram} = \frac{size_{ngram}}{\sigma^n} \times \lfloor\frac{Len(Q)}{n}\rfloor \quad (5.2)$$

$$offset\_time_{front} = \frac{size_{front}}{\sigma^{n'}} \times (Len(Q) - n' + 1) \quad (5.3)$$

$$offset\_time_{back} = \frac{size_{back}}{\sigma^m} \times (Len(Q) - m + 1) \times C(m,\varepsilon)\sigma^{\varepsilon} \quad (5.4)$$

$$\frac{offset\_time_{ngram}}{offset\_time_{ngram/2L-Approximation}} =$$

From Equation (5.5), we know that the time complexities of those indexes are identical to their space complexities. By substituting $size_{ngram}$ with $O(avg_{ngram} \times avg_{doc})$, Equation (5.5) shows that the time complexity of the n-gram index is $O(avg_{ngram} \times avg_{doc})$, while that of the n-gram/2L-Approximation index is $O(avg_{ngram} + avg_{doc})$. The time complexity indicates that the n-gram/2L-Approximation 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 Equation (5.5), we note that the query processing time increases at a lower rate in the n-gram/2L-Approximation index than in the n-gram index as $Len(Q)$ gets longer. 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

$$\frac{\frac{1}{\sigma^{n-n'}}\left(size_{ngram} \times \lfloor\frac{Len(Q)}{n}\rfloor\right)}{\left(size_{front} \times (Len(Q) - n' + 1)\right) + \left(size_{back} \times (Len(Q) - m + 1) \times \frac{C(m,\varepsilon)\sigma^{\varepsilon}}{\sigma^{m-n'}}\right)} \quad (5.5)$$

than the total size of documents. Furthermore, in the back-end index, $Len(Q)$ little affects the query processing time since $\frac{C(m,\varepsilon)\sigma^{\varepsilon}}{\sigma^{m-n'}}$ is small[1]. 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 degrade significantly[14].

5.3.2 Analysis of the Number of Posting Lists Accessed

To analyze the query processing time more precisely, we should take the time to locate posting lists into account. The time for locating posting lists is $k_{plist} \times \alpha$. Hence, by using k_{plist} computed in Equations (5.2)~(5.4), we derive the time for locating posting lists as shown in Equations (5.6)~(5.8).

$$plist\_time_{ngram} = \lambda \times \lfloor \frac{Len(Q)}{n} \rfloor \tag{5.6}$$

$$plist\_time_{front} = \lambda \times (Len(Q) - n' + 1) \tag{5.7}$$

$$plist\_time_{back} = \lambda \times (Len(Q) - m + 1) \times C(m,\varepsilon)\sigma^{\varepsilon} \tag{5.8}$$

From Equations (5.6)~(5.8), we note that the time for locating posting lists in the n-gram index is not affected by k, but that, in the n-gram/2L-Approximation index, it increases as k gets larger (we note that $\varepsilon = \lfloor \frac{k}{t'} \rfloor$). The reason for this increment is as follows: the number of m-subsequences ε-matching with Q increases as k gets larger by Equation (5.8), and thus, the number of posting lists accessed in the back-end index increases. Here, the number of m-subsequences ε-matching with Q shows a staircase-like behavior as k gets larger due to the floor function in $\varepsilon = \lfloor \frac{k}{t'} \rfloor$.

From Equation (5.8), we note that the time for locating posting lists in the n-gram/2L-Approximation index is also affected by m. $plist\_time_{back}$ increases 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.

6. PERFORMANCE EVALUATION

6.1 Experimental Data and Environment

The purpose of our experiments is to compare the size and query performance of the n-gram/2L-Approximation index with those of the n-gram index. We use the *index size ratio* defined in Equation (6.1) as the measure for the index size and the wall clock time as the measure for the query performance.

We could use the n-gram/2L index for approximate string matching. However, for approximate string matching, the filtration performance of the n-gram/2L index is worse than the n-gram/2L-Approximation index as explained in Section 4.4. Thus, we do not include experiments for the n-gram/2L index here.

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[2] – used in information retrieval. We use three data sets of 10 MBytes, 100 MBytes, and 1 GBytes. 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[3] – 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 index size ratio in the PROTEIN database and TREC database while varying the database size. When creating the n-gram index, we set the length n of the n-gram to be 3, which is the most practically used one in n-gram applications [5, 15]. Besides, when creating the front-end index of the n-gram/2L-Approximation index, we set the length n' of the n-gram index to be 2, which is shorter than n (i.e., 3) as explained in Section 4.2. When creating the back-end index of the n-gram/2L-Approximation index, we use $(m_o - 1)$ as the length m of the m-subsequence as explained in Section 5.3.1. For the trade-off between the filtration time and the refinement time, $n = 3$, $n' = 2$, and $m = (m_o - 1)$ are the best for our experimental data (10 MBytes $\sim$ 1 GBytes).

To compare the query performance, we perform three kinds of experiments while varying the following parameters: (1) the database size; (2) the length of a query string; (3) the error tolerance. We adapt the experiments done in work by Navarro [6], which is the well-known work in the area of approximate string matching. We summarize in Table 3 the kinds of the experiments and parameters for comparing the query performance.

Experiment 1: We measure the wall clock time while varying the database size. We set $Len(Q)$ to be 50, which is the half of the maximum query length (i.e., 100) used by Navarro [6]. We set α to be $\frac{1}{6}$, which is the half of the maximum error ratio of 3-gram index (i.e., $\frac{1}{3}$). Thus, k is set to be 8 ($k = Len(Q) \times \alpha = 50 \times \frac{1}{6}$).

Experiment 2: We measure the wall clock time while varying the query length for a small k and a large k. We set a small α to be $\frac{1}{9}$, which is a third of the maximum error ratio of 3-gram index (i.e., $\frac{1}{3}$). Thus, k is $Len(Q) \times \frac{1}{9}$. We set a

$$index\ size\ ratio = \frac{the\ number\ of\ pages\ allocated\ for\ the\ n\text{-}gram\ index}{the\ number\ of\ pages\ allocated\ for\ the\ n\text{-}gram/2L\text{-}Approximation\ index} \tag{6.1}$$

[1]For a typical value of k, ε is smaller than $(m - n')$. If k is large enough such that ε is larger than $(m - n')$, we can not have the benefit of using the front-end index. Thus,we do not consider this case.

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

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

large α to be $\frac{2}{9}$, which is two thirds of the maximum error ratio of 3-gram index (i.e., $\frac{1}{3}$). Thus, k is $Len(Q) \times \frac{2}{9}$.

Experiment 3: We measure the wall clock time while varying the error tolerance k for a short $Len(Q)$ and a long $Len(Q)$. We set a short $Len(Q)$ to be 33, which is a third of the maximum query length (i.e., 100) used by Navarro [6]. We set a long $Len(Q)$ to be 66, which is two thirds of the maximum query length used by Navarro.

A query is composed of a query string Q and an error tolerance k. Here, we repeat the experiment 50 times using randomly selected queries from the database and present the average result. We use an online algorithm proposed by Ukkonen et al.[12] for refinement in the same way as Navarro[6] does.

We conduct all the experiments on a Pentium 2.6 GHz Linux PC with 1 GBytes of main memory and 400 GBytes Seagate E-IDE disks. To avoid the buffering effect of the LINUX file system and to guarantee actual disk I/O's, we use raw disks for storing data and indexes. We use the inverted index implemented in the Odysseus ORDBMS [13] for all the experiments. The page size for data and indexes is set to be 4,096 bytes.

6.2 Experimental Results for the Index Size

Figure 8 shows the index size ratio as the database size is varied for the PROTEIN database and TREC database. These results indicate that the size of the n-gram/2L-Approximation index is reduced compared with that of the n-gram index. The index size ratio increases as the database size does as analyzed in Section 5.2. Figure 8(a) shows that the size of the 2-gram/2L-Approximation index is reduced by 1.5∼1.8 times compared with that of 3-gram index in the PROTEIN database. Figure 8(b) shows that the size of the 2-gram/2L-Approximation index is reduced by 1.3∼1.8 times in the TREC database.

6.3 Experimental Results for the Query Performance

6.3.1 Effects of Varying the Database Size

Figure 9 shows the query processing time of the n-gram index and n-gram/2L-Approximation index as the database size is varied for the PROTEIN database and TREC database. These results indicate that we obtain a larger improvement of the query performance as the database size gets larger as analyzed in Section 5.3.1. Figure 9(a) shows that the improvement in the query performance is 0.8 times in PROTEIN-100M, but 3.9 times in PROTEIN-1G. Figure 9(b) shows tendencies similar to Figure 9(a).

In Figure 9, we also show the query processing time of the q-sample index proposed by Navarro et al. [9] for the competitor. A q-sample is a subsequence of length q appearing at fixed intervals h of a document. (Let us consider a document D as a sequence of characters $c_0, c_1, ..., c_{L-1}$. The ith q-sample of D is the sequence $c_{h(i-1)}, ..., c_{h(i-1)+q-1}$.) Since the q-sample index consists of more sparse subsequences than the n-gram index, its size is smaller than that of the n-gram index. However, since the q-sample index checks the necessary condition with less subsequences, the number of candidates increases and, at the same time, the refinement time increases. Figure 9 shows that the query processing time of the q-sample index is much larger than that of the n-gram/2L-Approximation index and even larger than that of the n-gram index. Here, we set the length q and h to be 6, which is the optimal value for the q-sample index used by Navarro et al.

6.3.2 Effects of Varying the Query Length

Figures 10 and 11 show the query processing time of the n-gram index and n-gram/2L-Approximation index as the query length is varied for PROTEIN-1G and TREC-1G. These results indicate that the n-gram/2L-Approximation index improves the query performance compared with the n-gram index with this improvement becoming more marked as $Len(Q)$ gets larger as analyzed in Section 5.3.1. Especially, the query performance is improved by up to 4.2 times compared with that of the n-gram index for a small k (i.e., $k = Len(Q) \times \frac{1}{9}$) and by up to 2.2 times for a large k (i.e., $k = Len(Q) \times \frac{2}{9}$). Besides, the difference in the query performance between two indexes is smaller for a large k in Figure 11 than for a small k in Figure 10. This is because the time for locating posting lists in the n-gram index is not affected by k, but that in the n-gram/2L-Approximation index increases as k gets larger as analyzed in Section 5.3.2.

6.3.3 Effects of Varying the Error Tolerance

Figures 12 and 13 show the query processing time of the n-gram index and n-gram/2L-Approximation index as the error tolerance is varied for the PROTEIN-1G and TREC-1G. These results indicate that the n-gram/2L-Approximation index improves the query performance compared with the n-gram index in most cases. Especially, the query performance is improved by up to 1.9 times compared with that of the n-gram index for a short $Len(Q)$ (i.e., $Len(Q) = 33$) and by up to 3.5 times for a long $Len(Q)$ (i.e., $Len(Q) = 66$).

It is worthwhile to note that the query processing time of the n-gram/2L-Approximation index increases rapidly at $k = 7$ in Figure 12(a). This is because $\varepsilon = \lfloor \frac{k}{t'} \rfloor$ in Equation (5.8) increases by one at $k = 7$. In fact, $\varepsilon = 0$ when $k \leq 6$, and $\varepsilon = 1$ when $k \geq 7$. Figures 12(b), 13(a), and 13(b) show tendencies similar to Figure 12(a).

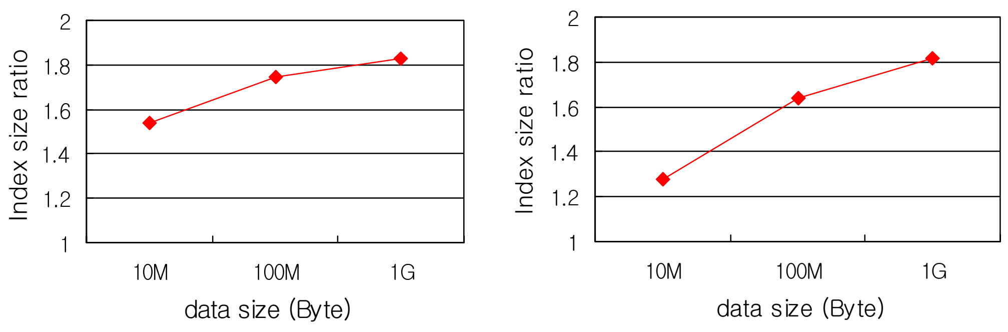
We can verify that the n-gram/2L-Approximation index improves the maximum error ratio compared with the n-gram index in Figures 12 and 13. The maximum error ratio of the 2-gram/2L-Approximation index is $\frac{1}{2}$, while that of the 3-gram index is $\frac{1}{3}$. In Figure 12(a), the 2-gram/2L-Approximation index is able to process queries in a reasonable time at $k = 11$ and $k = 13$ due to a larger maximum error ratio, while the 3-gram index is not. Figures 12(b), 13(a), and 13(b) show tendencies similar to Figure 12(a).

7. CONCLUSIONS

In this paper, we have proposed the n-gram/2L-Approximation index for approximate string matching. The n-gram/2L-Approximation index reduces the index size and improves the

Table 3 The kinds of the experiments and parameters for comparing the query performance.

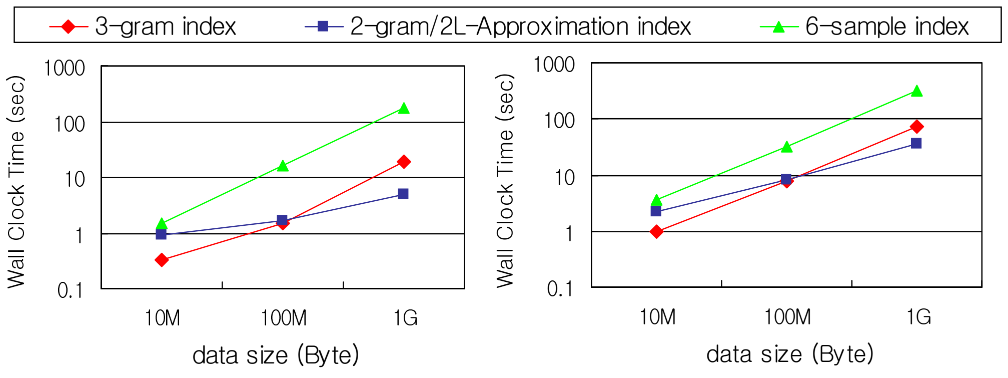
Experiments		Parameters	
Experiment 1	Comparison of the query performance while varying the database size	Data Set	PROTEIN database, TREC database
		Data Size	10 MByte, 100 MByte, 1 GByte
		$Len(Q)$	50
		k	8
Experiment 2	Comparison of the query performance while varying $Len(Q)$	Data Set	PROTEIN-1G, TREC-1G
		$Len(Q)$	20, 40, 60, 80, 100
		k	$Len(Q) \times \frac{1}{9}$, $Len(Q) \times \frac{2}{9}$
Experiment 3	Comparison of the query performance while varying k	Data Set	PROTEIN-1G, TREC-1G
		k	$0 \sim \frac{Len(Q)}{3}$
		$Len(Q)$	33, 66



(a) The index size ratio as the database size is varied. (data set: PROTEIN, $m = m_o$-1)

(b) The index size ratio as the database size is varied. (data set: TREC, $m = m_o$-1)

Figure 8 The index size ratio while varying the database size.



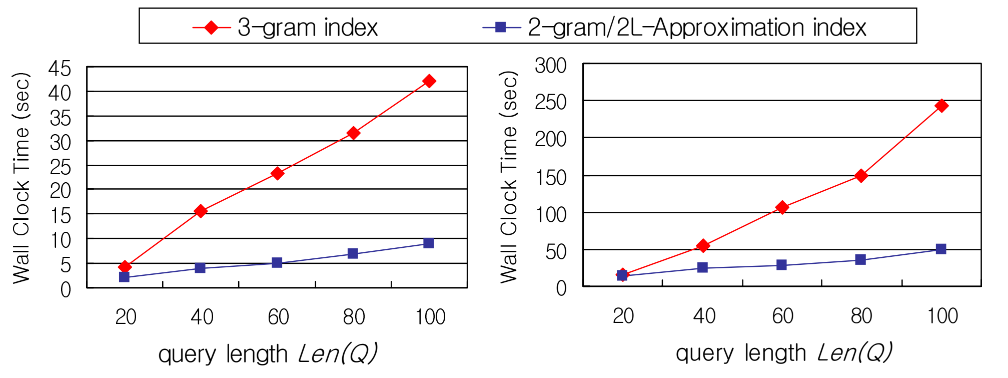
(a) The query processing time as the database size is varied. (data set: PROTEIN, Len(Q)=50, k=8)

(b) The query processing time as the database size is varied. (data set: TREC, Len(Q)=50, k=8)

Figure 9 The query processing time while varying the database size.

query performance compared with the n-gram index. We have modified the n-gram/2L index [4], which the authors have earlier proposed for exact matching, to improve the performance of approximate string matching. The modifications are related to the methods of extracting n-grams and m-subsequences. Due to the modifications, we reduce the number of false positives and improve the maximum error ratio.

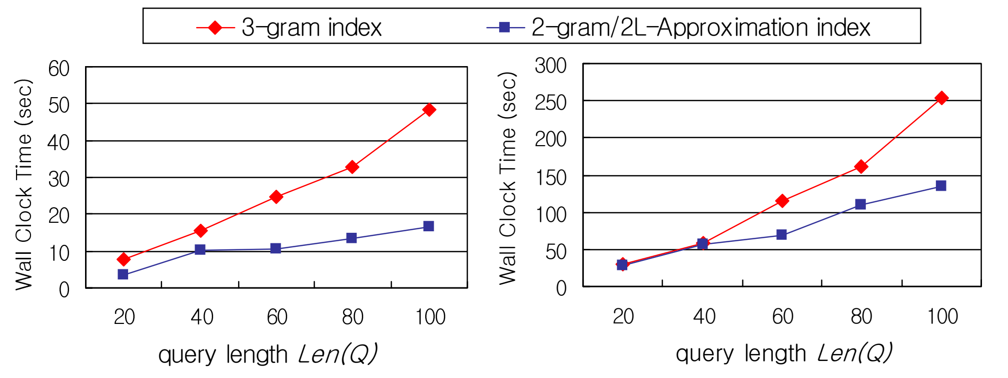
We have theoretically analyzed the properties of the n-gram/2L-Approximation index. First, we have proven in Section 5.1 that our index is derived by the relational normalization process that decomposes the n-gram index into 4NF. Second, we have analyzed the space complexity. Since the space complexity of our index is $O(avg_{ngram} + avg_{doc})$ and that of the n-gram index is $O(avg_{ngram} \times avg_{doc})$, the reduction of the index size



(a) The query processing time as *Len(Q)* is varied. (data set: PROTEIN-1G, *k* = *Len(Q)**1/9, *m*=4)

(b) The query processing time as *Len(Q)* is varied. (data set: TREC-1G, *k* = *Len(Q)**1/9, *m*=5)

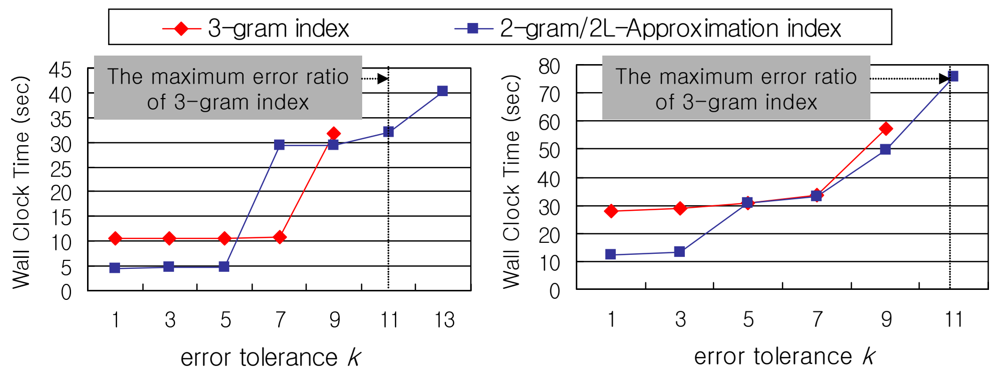
Figure 10 The query processing time while varying $Len(Q)$ for a small k (i.e., $k = Len(Q) \times \frac{1}{9}$).



(a) The query processing time as *Len(Q)* is varied. (data set: PROTEIN-1G, *k* = *Len(Q)**2/9, *m*=4)

(b) The query processing time as *Len(Q)* is varied. (data set: TREC-1G, *k* = *Len(Q)**2/9, *m*=5)

Figure 11 The query processing time while varying $Len(Q)$ for a large k (i.e., $k = Len(Q) \times \frac{2}{9}$).



(a) The query processing time as *k* is varied. (data set: PROTEIN-1G, Len(Q)=33, *m*=4)

(b) The query processing time as *k* is varied. (data set: TREC-1G, Len(Q)=33, *m*=5)

Figure 12 The query processing time while varying k for a short $Len(Q)$ (i.e., $Len(Q) = 33$).

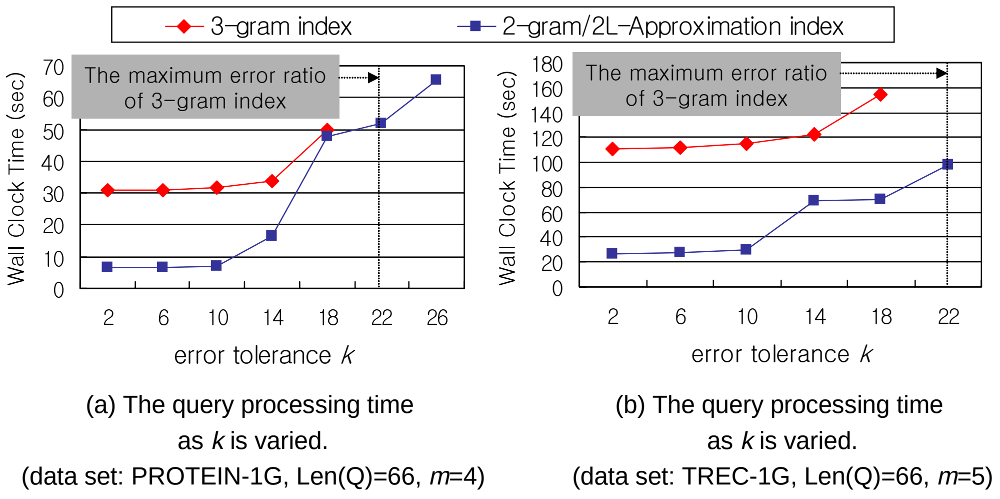


Figure 13 The query processing time while varying k for a long $Len(Q)$ (i.e., $Len(Q) = 66$).

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 shown that the query processing time increases at a lower rate in the n-gram/2L-Approximation index than in the n-gram index as the query length gets longer.

We have performed extensive experiments for the index size and query performance of the n-gram/2L-Approximation index varying the data set, database size, query length, and error tolerance. Experimental results using real text and protein databases of 1 GBytes show that the size of the n-gram/2L-Approximation index is reduced by up to 1.8 (PROTEIN-1G, $m = 4$) times and, at the same time, the query performance is improved by up to 4.2 (PROTEIN-1G, $m = 4$, $\alpha = \frac{1}{9}$) times compared with those of the n-gram index.

Overall, we believe that our index is capable of efficiently handling various applications for approximate string matching, for example, searching text documents with typographical errors and finding DNA or protein sequences with possible mutations.

Acknowledgement

This work was supported by the Ministry of Science and Technology (MOST) / Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc). This work was supported by Brain Korea 21 Project, the school of information technology, KAIST in 2006.

REFERENCES

1. Baeza-Yates, R. and Ribeiro-Neto, B., *Modern Information Retrieval*, ACM Press, 1999.
2. Elmasri, R. and Navathe, S. B., *Fundamentals of Database Systems*, 4th ed., Addison Wesley, 2003.
3. Gravano, L., Ipeirotis, P. G., Jagadish, H. V., Koudas, N., Muthukrishnan, S., and Srivastava, D., "Approximate String Joins in a Database (Almost) for Free," In *Proc. 27th Int'l Conf. on Very Large Data Bases (VLDB)*, Rome, Italy, pp. 491–500, Sept. 2001.
4. Kim, M., Whang, K., Lee, J., and Lee, M., "n-Gram/2L: A Space and Time Efficient Two-Level n-Gram Inverted Index Structure," In *Proc. 31st Int'l Conf. on Very Large Data Bases (VLDB)*, Trondheim, Norway, pp. 325–336, Aug./Sept. 2005.
5. Kukich, K., "Techniques for Automatically Correcting Words in Text," *ACM Computing Surveys*, Vol. 24, No. 4, pp. 377–439, Dec. 1992.
6. Navarro, G., "A Guided Tour to Approximate String Matching," *ACM Computing Surveys*, Vol. 33, No. 1, pp. 31–88, Mar. 2001.
7. Navarro, G. and Baeza-Yates, R., "A Hybrid Indexing Method for Approximate String Matching," *Journal of Discrete Algorithms*, Vol. 1, No. 1, pp. 205–239, 2000.
8. Navarro, G., Baeza-Yates, R., Sutinen, E., and Tarhio, J., "Indexing Methods for Approximate String Matching," *IEEE Data Engineering Bulletin*, Vol. 24, No. 4, pp. 19–27, Dec. 2001.
9. Navarro, G., Sutinen, E., and Tarhio, J., "Indexing Text with Approximate q-grams," *Journal of Discrete Algorithms (JDA)*, Vol. 3, No.2, pp.157-175, 2005.
10. Shi, F., "Fast Approximate String Matching with q-Blocks Sequences," In *Proc. 3rd South American Workshop on String Processing (WSP'96)*, pp. 257–271, Carleton University Press, 1996.
11. Silberschatz, A., Korth, H. F., and Sudarshan, S., *Database Systems Concepts*, 4th ed., McGraw-Hill, 2001.
12. Ukkonen, E., "Finding Approximate Patterns in Strings," *Journal of Algorithms*, Vol. 6, pp. 132–137, 1985.
13. Whang, K., Lee, M., Lee, J., Kim, M., and Han, W., "Odysseus:a High-Performance ORDBMS Tightly-Coupled with IR Features," In *Proc. 21st IEEE Int'l Conf. on Data Engineering (ICDE)*, Tokyo, Japan, pp. 1104–1105, Apr. 2005. (This paper received the Best Demonstration Award.)
14. Williams, H. E., "Genomic Information Retrieval," In *Proc. 14th Australasian Database Conferences*, Adelaide, Australia, pp. 27–35, 2003.
15. Williams, H. E. and Zobel, J., "Indexing and Retrieval for Genomic Databases," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 14, No. 1, pp. 63–78, Jan./Feb. 2002.

Appendix A. Proof of Theorem 1

We prove Theorem 1 for each condition.

Condition (1): One edit operation is able to modify at most n of $(m-n+1)$ n-grams. ε edit operations are able to modify at most $(\varepsilon \times n)$ of $(m-n+1)$ n-grams. Thus, among the set of n-grams $\{G_i\}$, at least $r = (m-n+1) - (\varepsilon \times n)$ n-grams $\{g_j\}$ must appear in the m-subsequence S and the query string Q.

Condition (2): ε edit operations change the offset of g_j by at most ε. Thus, the offset o_{sj} of g_j within S and the 0-offset o_{wj} of g_j within ε-substring(Q) satisfy $|o_{wj} - o_{sj}| \leq \varepsilon$. Since $o_{wj} = (o_{qj} - p)$, we have $|(o_{qj} - p) - o_{sj}| \leq \varepsilon$.

□

Appendix B. Proof of Theorem 2

We prove Theorem 2 for each condition.

Condition (1): No matter how we apply ε edit operations to t m-subsequences, at least one m-subsequence with at most $\lfloor \frac{\varepsilon}{t} \rfloor$ errors appears in a query string Q. That is, at least one m-subsequence $\lfloor \frac{\varepsilon}{t} \rfloor$-matches with Q. Now, we compute the maximum number of m-subsequences $\lfloor \frac{\varepsilon}{t} \rfloor$-matching with Q. We try to minimize the number of m-subsequences $\lfloor \frac{\varepsilon}{t} \rfloor$-matching with Q when ε edit operations are applied. This is achieved by distributing $(\lfloor \frac{\varepsilon}{t} \rfloor + 1)$ edit operations among m-subsequences to the extent possible. The maximum number of m-subsequences which $(\lfloor \frac{\varepsilon}{t} \rfloor + 1)$ edit operations are applied to is $\lfloor \frac{\varepsilon}{\lfloor \frac{\varepsilon}{t} \rfloor + 1} \rfloor$. Thus, at least $r = t - \lfloor \frac{\varepsilon}{\lfloor \frac{\varepsilon}{t} \rfloor + 1} \rfloor$ m-subsequences of t m-subsequences must appear in the query string Q with at most $\lfloor \frac{\varepsilon}{t} \rfloor$ edit operations (i.e., $\lfloor \frac{\varepsilon}{t} \rfloor$-matching with Q).

Condition (2): ε edit operations change the offset of s_j by at most ε. Thus, the offset o_{wj} of s_j within ε-substring(D) and the $\lfloor \frac{\varepsilon}{t} \rfloor$-offset o_{qj} of s_j within the query string Q satisfy $|o_{wj} - o_{qj}| \leq \varepsilon$. Since $o_{wj} = (o_{dj} - p)$, we have $|(o_{dj} - p) - o_{qj}| \leq \varepsilon$.

□