Nordic Journal of Computing 11(2004), 321–343.

APPROXIMATE STRING MATCHING WITH ORDERED q-GRAMS*

ERKKI SUTINEN
Department of Computer Science, University of Joensuu
P.O. Box 111, FIN-80101 Joensuu, Finland
sutinen@cs.joensuu.fi

JORMA TARHIO
Department of Computer Science and Engineering
Helsinki University of Technology
P.O. Box 5400, FIN-02015 HUT, Finland
tarhio@acm.org

Abstract. Approximate string matching with k differences is considered. Filtration of the text is a widely adopted technique to reduce the text area processed by dynamic programming. We present sublinear filtration algorithms based on the locations of q-grams in the pattern. Samples of q-grams are drawn from the text at fixed periods, and only if consecutive samples appear in the pattern approximately in the same configuration, the text area is examined with dynamic programming. The algorithm LEQ searches for exact occurrences of the pattern q-grams, whereas the algorithm LAQ searches for approximate occurrences of them. In addition, a static variation of LEQ is presented. The focus of the paper is on combinatorial properties of the sampling approach.

ACM CCS Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems

Key words: string matching, filtration, q-grams, indexing

1. Introduction

Given a *text* $T = T[1 \ldots n]$ and a *pattern* $P = P[1 \ldots m]$ over an alphabet Σ of size c and an integer k, called the *error level*, the *approximate pattern matching problem* is the following: find all the positions j in T such that an approximate match of P ends at j. The found occurrences are called k-approximate matches of P.

The proximity between two strings α and β can be measured for example by the *edit distance* d. In a *unit cost model*, $d(\alpha, \beta)$ is the minimum number of *edit operations* (insertions, deletions, or changes) needed to convert α to β. Using the unit cost model to evaluate edit distance results in a particular instance of the approximate pattern matching problem, called the k *differences problem*. Furthermore, if only change operations are allowed, the corresponding problem is called the k *mismatches problem*. In this paper, we will consider the k differences problem.

*The work was supported by the Academy of Finland. Most of the work was done at the University of Helsinki.

Received June 26, 2002; revised July 13, 2004; accepted December 20, 2004.

There are several solutions to the k differences problem, see e.g. Navarro's survey [9]. Our main interest is not, however, to compare our solutions with the earlier ones, but derive a different approach, based on the combinatorial characteristics of the problem. We believe that our approach may have other applications than the k differences problem.

The approximate pattern matching problem obviously has applications in diverse areas, such as computational biology, text databases, data security, and information retrieval. In many of these applications, the texts are large. This makes the $O(kn)$ solutions [15, 6, 16], which apply dynamic programming to the whole text, too slow. In order to speed up the search, dynamic programming should be restricted to text regions where approximate matches are supposed to be found. This technique is called *filtration*.

Filtration. Each filtration method has basically two phases. The first one, called the *filtration phase*, uncovers the promising text regions with *potential* approximate matches. These regions are identified by a necessary condition for an approximate match, called the *filtration condition*. The slower but accurate *checking phase* applies dynamic programming to the filtered text regions, verifies or falsifies the potential matches, and returns the locations of *actual* matches. A filtration method can be applied to both the *dynamic* (on-line) and *static* (off-line) search: in the former, the text is not preprocessed for an index, whereas the latter can speed up the search by identifying interesting portions of the text by the index.

A useful filtration scheme is often a compromise between several conflicting requirements. First, the filtration condition should be as *selective* as possible; in an ideal situation, all potential matches should also be actual ones, a situation with no *false*, i.e., potential but not actual, matches. Second, the filtration phase should skip as much text as possible, i.e., process the text in *sublinear*[1] time. Third, checking the filtration condition should be *fast*, with as little overhead as possible. Fourth, the filtration condition should *cover* as diverse problem instances as possible, for example, work for all possible error levels or alphabets. Often, the second and third requirement make too selective filtration impossible. For example, *multilevel filtration* which applies several filtration conditions one after another, might be very powerful in terms of false matches, but not in terms of text skipped or time consumed.

Filtration with q-grams. The use of q-grams, i.e., strings of length q, in filtration is based on the fact that each approximate match must resemble the pattern. This resemblance can be observed in the filtration phase as preserved q-grams of the original pattern. However, the q-gram filtration does not meet the fourth requirement above: a sufficiently large error level k breaks all the q-grams of the original pattern so that they cannot serve as a criterion for a potential match.

The use of the q-gram approach in approximate pattern matching results in various filtration methods. For example, one can utilize the fact that the preserved q-grams retain their original order. The variants of filter algorithms differ from

[1] In this study, we use term "sublinear" in the same sense as Boyer and Moore [2]: an algorithm, searching for a string of length m in a text of length n, is sublinear if it inspects less than n characters in the search task. See also [8].

each other, for example, in the frequency of false matches they yield. When applied to static search, a q-gram method uses an index of q-grams of the text; it is a challenge to design space efficient indexes for these algorithms [7].

We present two dynamic algorithms called LEQ (for Locations of Exact q-grams) and LAQ (Locations of Approximate q-grams), which improve the filtration with the following techniques:

- for better selectivity, require that the preserved q-grams are in the same order as in the original pattern;
- for sublinearity, observe only *q-samples*, i.e., every hth q-gram of the text; and
- for fast evaluation of the filtration condition, apply the *shift-add* method [3] to efficiently use the information gained from preceding evaluation.

Basically, these ideas result in algorithms which observe a sequence of r adjacent q-samples at a time, locate these q-samples with the corresponding, adjacent *pattern blocks*, and use the matching information for checking the following sequence of r q-samples. In LEQ, r equals $k + s$; here $s \geq 1$. It turns out (Theorem 5) that in an approximate match, at least s of the $k + s$ q-samples must exist in the corresponding pattern blocks. In LAQ, the filtration is based on the cumulative error made when comparing each of the r q-samples to the corresponding pattern block. The third algorithm to be presented is SLEQ, a static variation of LEQ. The focus of the paper is on combinatorial properties of the sampling approach.

A preliminary version appeared in two conference papers [11] and [12]. Some of the missing proofs can be found in [10].

Related methods. Takaoka [13] applied the sampling approach in his filtration scheme, with $r = k+1$. This means that the method cannot use the relative order of the preserved pattern q-grams. Wu and Manber [17] also utilize a similar relationship between the found q-grams and the pattern, with $q = \lfloor \frac{m}{k+1} \rfloor$. They scan every q-gram of the text. A potential approximate match occurs if the scanned q-gram is one of the $k + 1$ subsequent, non-overlapping q-grams of the pattern.

In addition to the preserved q-grams, the filtration can be based on the slightly distorted q-grams of the pattern, like in the LAQ algorithm. This idea was also utilized in the BLAST tool [1]. BLAST applies a statistically derived measure to determine the text areas with q-grams not too distant from the q-grams of a given pattern.

Chang and Lawler [4] used a similar idea in their sublinear search algorithm, and their approach was developed further by Chang and Marr [5]. The LAQ algorithm joins the ideas of Chang and Lawler, and Chang and Marr with the sampling approach and the prerequisite that even the distorted q-grams of the pattern shall remain in their mutual order in an approximate match.

General terms and notations. The *length* of a string P is denoted by $|P|$. The *empty string* ε is a string of length 0. In the following, an empty substring of P may also be denoted by $P[i_1 \ldots i_2]$, where $i_1 > i_2$. The set $Q_q(P)$ refers to the q-grams occurring in the string P, i.e., $Q_q(P) = \{P[i \ldots i + q - 1] \mid 1 \leq i \leq m - q + 1\}$.

Finally, let cond be a logical condition. The notation δ_{cond} is 0 if the condition cond is true, otherwise 1.

2. Invariance properties of the edit distance

What is preserved in an approximate occurrence of a pattern? We study this question in order to derive filtration conditions which identify a potential occurrence in terms of preserved or slightly modified substrings of the pattern.

D 1. *Let P and P' be strings over an alphabet Σ. The* edit distance *between the strings P and P', denoted by $d(P, P')$, is defined as the minimum number of edit operations (deletions, insertions, or changes) needed to transform P into P', or equivalently, vice versa.*

E 1. Let us consider strings $P = P[1 \ldots m]$ and $P' = P'[1 \ldots m']$, where m' is at most m. The following bounds hold for the edit distance $d(P, P')$:

$$||P| - |P'|| \leq d(P, P') \leq \max(|P|, |P'|).$$

The edit distance $d(P, P')$ between non-empty strings $P = P[1 \ldots m]$ and $P = P'[1 \ldots m']$ $(m \geq 1, m' \geq 1)$ can be stated as a recursive formula

$$d(P, P') = \min \begin{cases} d(P[1 \ldots m-1], P'[1 \ldots m'-1]) + \delta_{P[m]=P'[m']} \\ d(P[1 \ldots m], P'[1 \ldots m'-1]) + 1 \\ d(P[1 \ldots m-1], P'[1 \ldots m']) + 1 \end{cases} \quad (1)$$

and

$$d(\alpha, \varepsilon) = d(\varepsilon, \alpha) = |\alpha| \quad (2)$$

for every string α over alphabet Σ.

Hence, we can evaluate the edit distance as follows. We need an $(m+1)\times(m'+1)$–matrix E, called the *edit distance matrix*. Each entry $E[i, j]$ gives the edit distance between $P[1 \ldots i]$ and $P'[1 \ldots j]$; in particular, $d(P, P')$ is equal to $E[m, m']$. Since the matrix E is evaluated by dynamic programming, it needs to have initial values $E[i, 0] = i$ in the first column and $E[0, j] = j$ in the first row, corresponding to (2). Other values are calculated using the formula

$$E[i, j] = \min \begin{cases} E[i-1, j-1] + \delta_{P[i]=P'[j]} \\ E[i, j-1] + 1 \\ E[i-1, j] + 1. \end{cases} \quad (3)$$

Note that the entry $E[i, j]$ corresponds to editing the prefix $P[1 \ldots i]$ to the prefix $P'[1 \ldots j]$. The basic method to calculate the values of matrix E is to proceed row-by-row or column-by-column. In case of large m or m', one may benefit from advanced methods [14].

2.1 Tracing the edit operations

In principle, filtration can be based on either a set of subsequences, usually substrings, like q-grams, which cannot exist in an approximate match (*exclusive filtration*), or a set of subsequences which must occur in an approximate match (*inclusive filtration*). To derive any set useful for filtration, one must understand what actually happens when a string is edited under the given set of edit operations, like those associated to the standard edit distance. The *editing process* can be traced from the edit distance matrix, and presented as a corresponding *trace sequence*. The concept of trace sequence turns out to be powerful for stating inclusive filtration conditions which are based on those substrings which were either preserved or only slightly modified under the editing process.

Since the edit distance matrix E gives the edit distances of the prefixes of P and P', it can also be used to trace the $E[m, m'] = d(P, P')$ edit operations needed to convert P to P'. Starting from the entry $E[m, m']$, one selects the entry (either $E[m, m'-1]$, $E[m-1, m'-1]$, or $E[m-1, m']$) that has been used to evaluate $E[m, m']$, according to (3). In general, the edit distance matrix E reveals the edit operation, leading to the edit distance $E[i, j] = d(P[1 \ldots i], P'[1 \ldots j])$, in the following way (note that the choice is not unique):

if $E[i, j] = E[i-1, j-1] + \delta_{P[i]=P'[j]}$, then
    if $P[i] = P'[j]$, then no edit operation;
    else change $P[i] \to P'[j]$;
if $E[i, j] = E[i, j-1] + 1$, then insert $\varepsilon \to P'[j]$;
if $E[i, j] = E[i-1, j] + 1$, then delete $P[i] \to \varepsilon$.

The edit operations are traced until the entry $E[0, 0]$ is encountered (this always happens because of (2) and (3)). The trace defines a *trace sequence* $\Gamma = (E[0, 0], \ldots, E[m, m'])$. Let us consider two successive members $\gamma = E[i, j]$ and $\gamma' = E[i + p_i, j + p_j]$ of the sequence Γ. The pair (γ, γ') corresponds to editing $P'[1 \ldots j]P[i+1 \ldots m]$ to $P'[1 \ldots j + p_j]P[i + p_i + 1 \ldots m]$.

The smaller the edit distance $d(P, P')$ is, the more any trace sequence Γ includes successive members of type $E[i, j]$, $E[i+1, j+1]$, where the corresponding character $P[i+1]$ remains preserved, i.e., $P[i+1] = P'[j+1]$. If $d(P, P')$ is sufficiently small, that is, relatively close to $||P| - |P'||$ (see Example 1), there must be sequences of adjacent preserved characters. Now, the smaller the edit distance, the more there are preserved substrings. If the gram length q is sufficiently small, these preserved sequences also contain preserved q-grams. Like the preserved characters, the preserved q-grams retain their relative order. Let us summarize the idea of the preserved q-grams as a definition:

D 2. *Let $P = P[1 \ldots m]$ and $P' = P'[1 \ldots m']$ be strings, and let E be the corresponding edit distance matrix. Let $\Gamma = (E[0, 0], \ldots, E[m, m'])$ be a corresponding trace sequence. A substring $P'[j-q+1 \ldots j]$ of length q is called a* preserved q-gram *of P if Γ includes successive entries $E[i-q, j-q], E[i-q+1, j-q+1], \ldots, E[i, j]$ and the substring $P[i-q+1 \ldots i]$ is the same as the substring $P'[j-q+1 \ldots j]$. A preserved q-gram is called a* preserved character *of P if $q = 1$.*

Let us consider two strings, P_1 = APPLETREE and P_2 = TRIPLEAPE. Although the character sets of P_1 and P_2 differ only in one character, it is not natural to define the edit distance in a way which would return an edit distance of 1 for P_1 and P_2. While the preserved order of the non-edited characters is a relatively simple property of the edit distance, the algorithms to be introduced in this study are based on it.

2.2 Edit distance between substrings

In this section, we will show how the edit operations that change a string P into a string P' in $d(P, P')$ edit operations are reflected in the edit distance between the substrings of P and P'. These results are crucial for the filtration conditions, to be presented in Section 3.

The properties of a trace sequence make it useful for understanding the edit distance. For example, the elements of any trace sequence Γ compose a sequence of non-decreasing integers. This fact has been used to prove the following theorem.

T 1. *Let P and P' be strings such that $d(P, P') \leq k$. Then for each prefix α of P, and for each suffix β of P, there exists at least one prefix α' of P' such that $d(\alpha, \alpha') \leq k$, and at least one suffix β' of P' such that $d(\beta, \beta') \leq k$.*

A trace sequence Γ of an edit distance matrix E is also useful for partitioning the editing of string P to another string P'. In the following, we will derive a few results on the partitioning technique. In a case where a string P has been edited to a string P' with at most k edit operations, the theorems will be used to correlate a set of substrings of the edited string P' to a corresponding set of substrings of the original pattern P.

D 3. *Let P and P' be strings, and let Γ be a trace sequence. An arbitrary division $\mathcal{P}$ of the string P into its substrings $\alpha_i = P[s_i \ldots e_i], i = 1, \ldots, l$, i.e., $P = \alpha_1 \alpha_2 \cdots \alpha_l$, is called a* partitioning *of the string P. For each partitioning $\mathcal{P}$, the trace sequence Γ has at least one subsequence $\Gamma_{\mathcal{P}} = (E[e_1, e'_1], \ldots, E[e_{l-1}, e'_{l-1}])$ such that $\Gamma_{\mathcal{P}}$ gives the* corresponding partitioning *to the edited string P':*

$$P' = \alpha'_1 \alpha'_2 \cdots \alpha'_l,$$

where $\alpha'_i = P'[s'_i \ldots e'_i]$ for each $i = 1, \ldots, l$. Above, $s_1 = s'_1 = 1$, $e_l = m$ and $e'_l = m'$. In addition, the equations $s_{i+1} = e_i + 1$ and $s'_{i+1} = e'_i + 1$ hold for each $i = 1, \ldots, l - 1$.

Note that for some indices i, the substring α_i (or α'_i) can be the empty string, corresponding to the setting $s_i > e_i$ ($s'_i > e'_i$). For each partitioning of the string P there may be more than one corresponding partitioning of the string P'. Note also that one can construct a partitioning for the string P starting from a partitioning of the string P'. The following theorem states how the edit distance between two strings can be presented as a sum of the edit distances between their substrings.

T 2. *Let P and P' be strings. Let $P = \alpha_1\alpha_2\cdots\alpha_l$ be a partitioning of P, and let $P' = \alpha'_1\alpha'_2\cdots\alpha'_l$ be a corresponding partitioning of P'. Then*

$$d(P, P') = \sum_{i=1}^{l} d(\alpha_i, \alpha'_i).$$

As noted earlier, the partitioning technique is symmetric for the string P and its edited counterpart P'. Thus, for each partitioning $\mathcal{P}'$ of the string P', we can find a corresponding partitioning $\mathcal{P}$ of the string P. This makes it natural to define an *original counterpart* of a substring of the edited string.

D 4. *Let P and P' be strings. Let α' be a substring of P'. We call a substring α of P an* original counterpart *of α' if there exists a partitioning $P = \alpha_1\alpha_2\cdots\alpha_l$ and a corresponding partitioning $P' = \alpha'_1\alpha'_2\cdots\alpha'_l$ such that for one i, $1 \le i \le l$, $\alpha = \alpha_i$ and $\alpha' = \alpha'_i$.*

Note that a substring of the edited string may have multiple original counterparts, as a consequence of multiple trace sequences, each producing at least one partitioning subsequence.

Let us now consider an arbitrary substring α' of the string P'. We can locate the original counterparts of α' in the string P by the following theorem:

T 3. *Let P and P' be strings with an edit distance $d(P, P') = c_I + c_D + c_C$, where c_I stands for the number of insertions to P, c_D gives the number of deletions from P, and c_C is the number of changes in P. Let $P'[i' \ldots j']$ be a non-empty substring of P'. Let $P[i \ldots j]$ be an arbitrary original counterpart of $P'[i' \ldots j']$. Then the following inequalities hold:*

$$j' - c_I \le j \le j' + c_D.$$

In addition, if $i \le j$, then also

$$i' - c_I \le i \le i' + c_D$$

holds.

P . Let $P' = \alpha'_1\alpha'_2\cdots\alpha'_l$ be a partition of P' such that $P'[i' \ldots j'] = \alpha'_\nu$ for some ν, $1 \le \nu \le l$. Let $P = \alpha_1\alpha_2\cdots\alpha_l$ be a corresponding partition of P such that $\alpha_\nu = P[i \ldots j]$ is an original counterpart of $P'[i' \ldots j']$. Let us first consider the location of the index j.

Since $E[j, j']$ gives the number of edit operations needed to transform $P[1 \ldots j]$ to $P'[1 \ldots j']$, and among these operations there are at most c_I insertions and c_D deletions, we get $j' \le j + c_I$ and $j' \ge j - c_D$, which prove the theorem for j.

Let us now assume that the original counterpart $P[i \ldots j]$ is not the empty string, i.e., the condition $i \le j$ is satisfied. If ν equals 1, the inequality holds trivially, because $i = i' = 1$. Now, let $\nu > 1$. Since the substring $P'[i' \ldots j']$ is also non-empty,

the end positions of the substrings $\alpha_{\nu-1}$ and $\alpha'_{\nu-1}$ are $i-1$ and $i'-1$, respectively. Using the same technique as for j and j' above, we get

$$i' - 1 - c_I \leq i - 1 \leq i' - 1 + c_D,$$

which gives

$$i' - c_I \leq i \leq i' + c_D.$$

□

3. Filtration conditions

D 5. *By the* sampling step, *we denote the distance between two adjacent* q-samples, *i.e., q-grams observed in the text. The* sample size *is the number of q-samples observed at a time.*

Conventionally, we refer to the sampling step as h and the sample size as r. The q-samples of a text T are thus $d_j = T[jh - q + 1 \ldots jh]$, where $j = 1, \ldots, \lfloor \frac{n}{h} \rfloor$. In the following, the sampling step h is at most $H(m, k, q, r)$, where

$$H(m, k, q, r) = \lfloor \frac{m - k - q + 1}{r} \rfloor. \tag{4}$$

The upper bound $H(m, k, q, r)$ for the sampling step h ensures that each text substring of length at least $m - k$ contains at least r q-samples (Theorem 4). Since we require that the q-samples do not overlap each other, the value of h must be at least q. If this condition does not hold for an h derived from (4), a smaller value of q has to be used.

D 6. *By a* pattern block $Q_i, 1 \leq i \leq r$, *we refer to the substring* $Q_i = P[(i-1)h + 1 \ldots ih + q - 1 + k]$.

The definition above does not yield index overflows, since even for Q_r, the upper index is at most m:

$$rh + q - 1 + k \leq m - k - q + 1 + q - 1 + k = m.$$

Note that pattern suffix $P[rh + q + k \ldots m]$ does not belong to any block.

Algorithmically, the filtration methods to be presented consist of the following stages:

- preprocessing the pattern into the r pattern blocks;
- scanning every hth q-sample of the text;
- bookkeeping a score $\sigma[i]$ for counting the number of preserved q-grams within each successive sequence $D_i = d_i, \ldots, d_{i+r-1}$ of r q-samples;
- identifying a potential match in the text region around D_i based on the value of the score $\sigma[i]$.

In the following subsections, we will derive the filtration conditions for the algorithms, in terms of the sampling step h, the sample size r, and the value of the score $\sigma[i]$ required for a potential match.

3.1 Sampling based on preserved q-grams

The filtration scheme of the LEQ algorithm is based on identifying the potential approximate matches by finding the sequences of $r = k + s$ consecutive q-samples $d_{b+1}, d_{b+2}, \ldots, d_{b+k+s}$ for which $d_{b+i} \in Q_q(Q_i)$ holds for at least s of the q-samples. Theorems 4 and 5 present necessary conditions for an approximate match in a text, in terms of preserved q-grams, and Theorem 6 defines the text area that needs to be checked by dynamic programming in case of a potential match. Theorem 5 is the main result of this subsection. It defines a potential match in terms of the q-sample location information.

Our goal is a filtration condition which is based on $k + s$ adjacent q-samples. The choice for the sampling step h guarantees that any approximate match of P in T includes at least $k + s$ q-samples of which at least s occur in P. For technical reasons, we state our theorem as follows. The usefulness of the formulation will be clear in the proof of Theorem 5.

T 4. *Let m and r be positive integers, P a pattern, P' a substring of the text T, and k an integer such that $0 \le k \le m$. If $h \le H(m, k, q, r)$ is the sampling step, $h \ge q$ and $|P'| \ge m - k$, then P' contains at least r q-samples. If, in addition, $d(P, P') \le k$ and $s = r - k$ is positive, then among any sequence of $r = k + s$ q-samples in P', at least s occur in P.*

Theorem 4 states that at least s of the $k + s$ q-samples of P' *occur* in P. The theorem does not guarantee that these s q-samples are *preserved* q-grams of P (Def. 2). In fact, the theorem even allows the case where all of the s q-samples are the same, corresponding to a single q-gram in P. We therefore analyze the locations of the pattern q-grams, corresponding to the q-samples of P'. We will show that P' contains at least s preserved q-samples.

Since each edit operation has transformed at most one q-sample and only $d(P, P') \le k$ edit operations are allowed, at least s q-samples have not been edited, i.e., they are preserved q-grams. See Fig. 1.

We have proven the following corollary:

C 1. *Let m and s be positive integers, P a pattern and P' a substring of a text T, such that $d(P, P') \le k$, where $0 \le k \le m$. If $h \le H(m, k, q, k + s)$ is the sampling step, $h \ge q$ and $|P'| \ge m - k$, then P' contains at least $k + s$ q-samples. In addition, among any sequence of $k + s$ q-samples in P', at least s correspond to the preserved q-grams of P.*

The previous corollary is independent of the length of pattern P. However, it is straightforward to apply the theorem to the approximate pattern matching problem, with $|P| = m$:

C 2. *Let P be a pattern of length m and P' a substring of a text T such that $d(P, P') \le k$. If $h \le H(m, k, q, k + s)$ is the sampling step and $h \ge q$, then at least s of the q-samples in P' correspond to the preserved q-grams of P.*

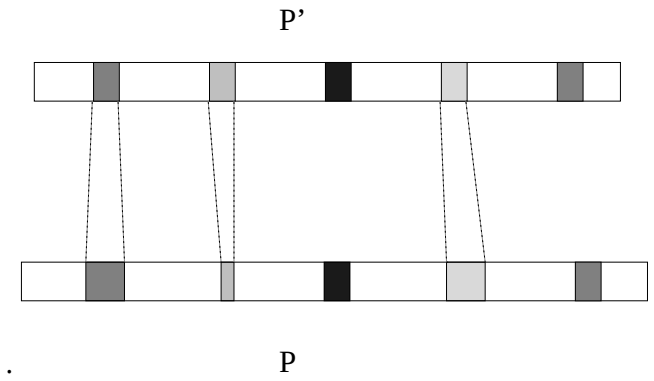


Fig. 1: The q-samples of P' with the corresponding substrings of P for $k = 3$ and $s = 2$. The $k = 3$ edit operations have distorted the first, second, and fourth q-sample of P', while the third one and the fifth one are preserved.

P . Since $|P| = m$ and $d(P, P') \leq k$, the length of P' is at least $m-k$. According to Corollary 1, P' includes at least one sequence of $k + s$ q-samples; moreover, at least s of the q-samples in this sequence correspond to preserved q-grams of P. □

We will now study how to state a necessary condition for an approximate match of P in T in terms of pattern blocks. Theorem 5 will be based on the block information, and it gives a stronger condition than Corollary 2.

D 7. *We call a sequence of $k + s$ subsequent q-samples $d_i, \ldots, d_{i+k+s-1}$ a* promising $(k + s)$-sequence *if at least s of the samples satisfy the condition $d_l \in Q_q(Q_{l-i+1})$.*

E 2. (P) It would be natural to assume that any sequence of $k + s$ subsequent q-samples in an approximate match P' of P is a promising one. This is, however, not the case. As a counterexample, let us consider a string $P =$ ABCDEFGHIJKLMNOPQRSTUV, with $m = 22$, $s = 2$, $q = 2$, and $k = 4$. Using $h = H(m, k, q, s)$, we get $h = 2$. Thus, P contains the following six blocks: ABCDEFG, CDEFGHI, EFGHIJK, GHIJKLM, IJKLMNO, and KLMNOPQ. Let us assume that T includes a substring $P' =$ ABCXDXEXFXGHIJKLMNOPQRSTUV, which is a 4-approximate match of P. Provided that sampling in T starts at the 2-sample AB, the first $(k+s)$- or 6-sequence of 2-samples has only two preserved 2-samples, namely the first, AB, and the sixth, GH. According to our first assumption, AB should belong to the first block and GH to the sixth one. This is not the case. We call this phenomenon the *phase problem*. Basically, it results from inserted characters occurring in an approximate match. However, in this setting there are other sequences of 6 subsequent 2-samples which are promising.

The next theorem states that it is always possible to locate at least *one* promising $(k + s)$-sequence in an approximate match of the pattern. The idea behind the proof

is to locate a substring P'' of the modified pattern P', starting with a promising $(k + s)$-sequence. Intuitively, it seems to be reasonable to search for a promising $(k + s)$-sequence in the middle of P': in this area, the lengthened right ends of the blocks balance the dislocations caused by the inserted characters. The right choice for P'' turns out to be a suffix of P'. To apply Corollary 1, it is essential that the length of P'' is at least $m - k$.

T 5. *Let $P' = T[\alpha \ldots \beta]$ be an approximate match of P with $d(P, P') \leq k$. Then there are $k + s$ consecutive q-samples $d_{b+1}, \ldots, d_{b+k+s}$, included in P', such that $d_{b+i} \in Q_q(Q_i)$ holds for at least s of the samples.*

P . As an approximate match of P, P' consists of c_I insertions, c_D deletions, and c_C changes to the original pattern P. Since $d(P, P') = c_I + c_D + c_C$, we get the inequality

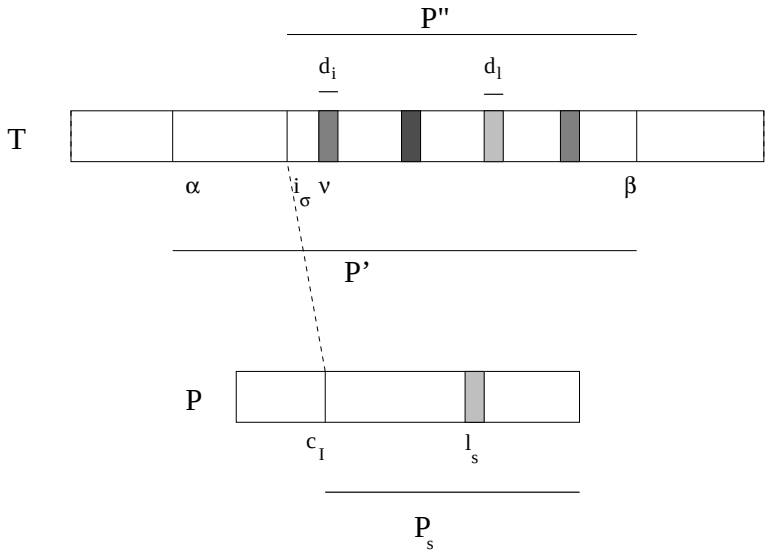
$$c_I + c_D + c_C \leq k. \tag{5}$$

Let us consider a suffix $P_s = P[c_I + 1 \ldots m]$ of P (see Fig. 2). Since $d(P, P') \leq k$, the same applies to their suffixes: according to Theorem 1, a suffix $P'' = T[i_\sigma \ldots \beta]$ of P' exists, such that $d(P_s, P'') \leq k$.

Let us now derive a lower bound for the length of P''. If c_{D_s} denotes the deletions applied to P_s, then $c_D \geq c_{D_s}$ and, using (5), we get

$$|P''| \geq m - c_I - c_{D_s} \geq m - c_I - c_D \geq m + c_C - k \geq m - k,$$

because $|P_s| = m - c_I$.



.

Fig. 2: Locating a promising $(k + s)$-sequence in Theorem 5.

Let now $d_i = T[\nu \ldots \nu + q - 1]$ be the leftmost q-sample in P''. Hence $\nu - i_\sigma \leq h - 1$. According to Corollary 1, suffix P'' includes subsequent q-samples $d_i, d_{i+1}, \ldots, d_{i+k+s-1}$, of which at least s are preserved q-grams of P_s. Note that

$$d_l = T[\nu + (l - i)h \ldots \nu + (l - i)h + q - 1]$$

holds for $i \leq l \leq i + k + s - 1$. Let us denote the set of preserved q-samples by $X = \{d_l \mid i \leq l \leq i + k + s - 1 \text{ and } d_l \text{ preserved}\}$, $|X| \geq s$.

Let us now consider an arbitrary q-sample $d_l \in X$. The q-gram, corresponding to d_l, can be located as $P[l_s \ldots l_s + q - 1]$, where $c_I + 1 \leq l_s \leq m - q + 1$. To prove the theorem, it remains to verify that $d_l \in Q_q(Q_{l-i+1})$, which holds if

$$(l - i)h + 1 \leq l_s \leq (l - i + 1)h + k \tag{6}$$

is satisfied. To prove the first inequality, we first denote by c_{I_s} the number of insertions applied to P_s. Since the distance of d_l from the beginning of P'' is at least $(l - i)h$, we get the following lower bound for the corresponding distance of d_l from the beginning of P_s:

$$l_s - (c_I + 1) \geq (l - i)h - c_{I_s}.$$

Because $c_I \geq c_{I_s}$ holds, this gives a lower bound for l_s:

$$l_s \geq (l - i)h + c_I + 1 - c_{I_s} \geq (l - i)h + 1.$$

To prove the second inequality, we first note that the distance of d_l from the beginning of P'' is at most $h - 1 + (l - i)h = (l - i + 1)h - 1$. We get the following upper bound for the corresponding distance of d_l from the beginning of P_s:

$$l_s - (c_I + 1) \leq (l - i + 1)h - 1 + c_{D_s},$$

which yields

$$l_s \leq (l - i + 1)h + c_I + c_{D_s} \leq (l - i + 1)h + k.$$

Since (6) holds for any $d_l \in X$, we have proven the theorem. □

We still need to define the bounds for the location of an approximate match in case the algorithm has found enough matching q-samples. The next theorem determines the text area that has to be checked by dynamic programming:

T 6. *Let us assume that s of $k + s$ consecutive q-samples $d_{b+1}, \ldots, d_{b+k+s}$ satisfy $d_{b+i} \in Q_q(Q_i)$ where q-sample d_{b+1} ends at text position j. Then an approximate occurrence of the pattern $P[1 \ldots m]$ may be located in the text area*

$$T[j - h - 2k - q + 2 \ldots j + m + k - q].$$

The width of the text area is $m + 3k + h - 1$.

3.2 Sampling based on approximate q-grams

A filtration method can also use of slightly modified q-grams of the pattern P. This observation results in a more sensitive filtration condition compared to the one satisfied with s preserved q-grams among $r = k + s$ observed ones.

The LAQ algorithm utilizes so called *ASM* distances, employed also by Chang and Marr [5]. For any q-gram u and string V, $ASM(u, V)$ is equal to the minimum edit distance (best-match distance) between u and any substring of the string V. The filtration condition of the LAQ algorithm requires that the sum of the *ASM* distances between the consecutive q-samples and the corresponding blocks is at most k, i.e., the inequality

$$\sum_{i=1}^{r} ASM(d_{b+i}, Q_i) \leq k$$

holds for a given parameter b, defining the first q-sample in the inspected test sequence.

Since the *ASM* distance between a string of q characters and any other string is always at most q, the sum $\sum_{i=1}^{r} ASM(d_{b+i}, Q_i)$ cannot exceed rq. Therefore it is required that rq is greater than k, since otherwise the sum is always at most k and the filtration condition triggers for any sequence of r consecutive q-samples. We get the following bounds for the sample size r:

$$\frac{k}{q} < r \leq \lfloor \frac{m - k - q + 1}{q} \rfloor,$$

where the maximum choice for r is given by (4).

We will now present how the *ASM* distance can be utilized to decide whether a given test sequence includes a potential match. As in the case of the LEQ algorithm, we provide two theorems. The first introduces a necessary condition for an approximate match, in terms of the *ASM* distances between the q-samples of a test sequence and the found pattern. The second defines the text area to be checked for a potential match.

T 7. *Let P' be an approximate match of $P = P[1 \ldots m]$ with $d(P, P') \leq k$. Then there are r consecutive q-samples $d_{b+1}, \ldots, d_{b+r}$, included in P', such that the following inequality is true:*

$$\sum_{i=1}^{r} ASM(d_{b+i}, Q_i) \leq k.$$

P . Let us denote $P' = P'[1 \ldots \mu]$. If c_I denotes the number of insertions to the string P and c_D gives the number of deletions from P, the length μ of P' equals $m + c_I - c_D$. Hence, the length μ can be estimated by the limits

$$m - k \leq \mu \leq m + k.$$

Let us now consider a suffix $P'' = P'[c_I + 1 \ldots \mu]$ of the string P'. Since the number of insertions and deletions $c_I + c_D$ is at most k, we get the bound

$$|P''| = \mu - c_I - 1 + 1 = m + c_I - c_D - c_I = m - c_D \geq m - k,$$

because c_D is obviously at most k.

Because (4) defines the sample size r so that any string of length at least $m - k$ includes at least r consecutive q-samples, this is also true for the string P''. Let us denote the leftmost q-sample of P'' by d_{b+1}. This means that an index σ exists for which $d_{b+1} = P'[\sigma \ldots \sigma + q - 1]$. Since d_{b+1} is the leftmost q-sample of P'', we get the bounds

$$c_I + 1 \leq \sigma \leq c_I + h.$$

Let us now consider the sequence $d_{b+1}, \ldots, d_{b+r}$. Let $P' = \alpha'_1 \alpha'_2 \cdots \alpha'_\nu$ be a partitioning of P where for each $i, 1 \leq i \leq r$, an index j_i exists such that $d_{b+i} = \alpha'_{j_i}$. We can choose the indices j_i so that they form an increasing sequence of integers. Let $P = \alpha_1 \cdots \alpha_\nu$ be a corresponding partitioning of the string P. Let us denote the original counterpart α_{j_i} of the q-sample d_{b+i} by $o(d_{b+i})$. According to Theorem 2,

$$d(P, P') = \sum_{l=1}^{\nu} d(\alpha_l, \alpha'_l),$$

which means that it is sufficient to prove that for each $i, 1 \leq i \leq r$, the substring $o(d_{b+i})$ is a substring of Q_i, since then $ASM(d_{b+i}, Q_i) \leq d(d_{b+i}, o(d_{b+i}))$ and therefore

$$\sum_{i=1}^{r} ASM(d_{b+i}, Q_i) \leq \sum_{i=1}^{r} d(d_{b+i}, o(d_{b+i})) \leq \sum_{l=1}^{\nu} d(\alpha_l, \alpha'_l) = d(P, P') \leq k,$$

which proves our theorem.

Now, let us prove that for each $i, 1 \leq i \leq r$, the substring $o(d_{b+i})$ is included in Q_i. Let us consider an arbitrary q-sample $d_{b+i} = P'[\sigma + (i-1)h \ldots \sigma + q - 1 + (i-1)h]$. If $o(d_{b+i})$ is the empty string, it is trivially a substring of Q_i. Let us now assume that $o(d_{b+i})$ is non-empty. According to Theorem 3, the original counterpart $o(d_{b+i})$ is included in $P[\sigma + (i-1)h - c_I \ldots \sigma + q - 1 + (i-1)h + c_D]$. The latter, however, is included in $Q_i = P[(i-1)h + 1 \ldots ih + q - 1 + k]$, since

(i) $\sigma + (i-1)h - c_I \geq c_I + 1 + (i-1)h - c_I$, and

(ii) $\sigma + q - 1 + (i-1)h + c_D \leq c_I + h + q - 1 + (i-1)h + c_D \leq ih + q - 1 + c_I + c_D \leq ih + q - 1 + k$.

This proves our claim and completes the proof. □

The theorem above gives a necessary condition for potential matches. However, we need another result to define the text area which has to be processed in the checking phase. It turns out that this area is slightly larger than the one of the LEQ algorithm, due to the different use of the test samples; we will give an example of this after the proof of the theorem.

T 8. *Let an approximate match P' of the pattern $P[1 \ldots m]$, with $d(P, P') \leq k$, trigger the filtration condition of the LAQ algorithm, i.e., the inequality*

$$\sum_{i=1}^{r} ASM(d_{b+i}, Q_i) \leq k$$

holds. If the q-sample d_{b+1} ends at a text position j, then P' is located in the text area

$$T[j - h - 2k - q + 2 \ldots j + m + k - 1].$$

The width of the text area to be inspected is $m + 3k + h + q - 2$.

P . Let us assume that $P' = T[\alpha \ldots \beta]$, c_I is the number of insertions to P, and c_D is the number of deletions from P. In the same way as in the proof of Theorem 7 we conclude that for a set of original counterparts $o(d_{b+1}), \ldots, o(d_{b+r})$ the following holds: $o(d_{b+i})$ is a substring of Q_i.

Now, at least one of the original counterparts, say $o(d_{b+i})$, is not the empty string, because otherwise $d(P, P') \geq rq > k$. Let us denote $o(d_{b+i})$ by $P[\lambda_1, \lambda_2]$, where $\lambda_2 \geq \lambda_1$, because $o(d_{b+i})$ is not empty. Since $P[\lambda_1, \lambda_2]$ is included in $Q_i = P[(i-1)h + 1 \ldots ih + q - 1 + k]$, we get the inequalities

$$(i-1)h + 1 \leq \lambda_2 \leq ih + q - 1 + k. \tag{7}$$

In addition, since the q-sample d_{b+1} ends at the text position j, the q-sample d_{b+i} ends at the *text* position $j+(i-1)h$, or equivalently, at the position $j+(i-1)h-\alpha+1$ in the *pattern* $P'[1 \ldots \beta - \alpha + 1] = T[\alpha \ldots \beta]$. Now, applying Theorem 3, we get the inequalities

$$j + (i-1)h - \alpha + 1 - k \leq \lambda_2 \leq j + (i-1)h - \alpha + 1 + c_D, \tag{8}$$

because $-k \leq -c_I$.

Combining the right inequality of (7) with the left inequality of (8), we get the lower bound for α:

$$j - h - 2k - q + 2 \leq \alpha. \tag{9}$$

On the other hand, the left inequality of (7) combined with the right inequality of (8) tells us that α is at most $j + c_D$. By noting that $|P'| = \beta - \alpha + 1 \leq m + c_I$, we get the inequality

$$\beta \leq m + c_I + j + c_D - 1 \leq j + m + d(P, P') - 1 \leq j + m + k - 1. \tag{10}$$

The inequalities (9) and (10) prove the claim. □

E 3. Let us consider the pattern $P =$ `ABCDEFGHIJ` of length $m = 10$. Let the error level $k = 3$ and the gram length $q = 2$. Let the sample size $r = 2$. These choices result in a sampling step $h = 3$. The choices are appropriate, since $h \geq q$ and $rq = 4 > 3 = k$. The pattern has two blocks, namely $Q_1 =$ `ABCDEFG` and $Q_2 =$ `DEFGHIJ`. Let us suppose that a text T has an approximate match

P' = AXXXBCDEFGHIJ = $T[91 \ldots 103]$. Since ASM(FG, Q_1) + ASM(IJ, Q_2) = 0 and FG ends at text position $j = 100$, it is appropriate to start the inspection of the text at text position $\alpha = j - h - 2k - q + 2 = 100 - 3 - 6 - 2 + 2 = 91$.

Since the filtration condition already triggers at $j = 94$, this example shows that the bounds for the area to be inspected may be quite pessimistic.

4. Algorithms

4.1 The LEQ algorithm

Matching a sequence of $k + s$ q-samples in the corresponding pattern blocks can be regarded as an instance of the generalized k mismatches problem. Therefore, the shift-add technique [3] can be used in the LEQ algorithm, to efficiently resolve the matching of a sequence $D_i = d_i \ldots d_{i+m'-1}$, based on the information of its predecessor D_{i-1}.

The triggering of the filtration condition, as stated in Theorem 5, can be interpreted as a reduction to a generalized k mismatches problem. In this problem, each position of the pattern has a set of accepted characters of its own. Let us consider the set of q-grams as the alphabet, q-samples as a text $\tau = d_1 \ldots d_{n'}$, and the q-gram sets of blocks of the original pattern as a pattern $\pi = Q_q(Q_1) \ldots Q_q(Q_{m'})$, where $n' = \lfloor n/h \rfloor$ and $m' = k + s$. We can now restate Theorem 5 as follows: a potential approximate match of π with at most k mismatches ends at j if $\tau[j - m' + i] \in \pi[i]$, or equivalently, $d_{j-m'+i} \in Q_q(Q_i)$, holds for at least $m' - k$ indices i, $1 \leq i \leq m'$.

In order to apply shift-add technique [3], we define a bit matrix B as follows: $B[d, j]$ equals 1 if the q-gram d belongs to $Q_q(Q_j)$, otherwise $B[d, j]$ is zero. For each q-gram d, $B[d, *]$ gives the *block profile* of d.

An array $M[1 \ldots m']$ is used to compute the number of matching q-samples in an alignment of the pattern π with $\tau[i \ldots i + m' - 1]$. An approximate match with at most k mismatches can be found only if at least $m' - k$ matched q-samples are aligned, i.e., $M[m'] \geq m' - k = s$.

In order to find all potential matches, the algorithm has to evaluate M for each substring $D_i = d_i \ldots d_{i+m'-1}$ of the text τ. The crucial question, decisive for the applicability of the LEQ algorithm, is how to efficiently update $M[m']$ for D_i, using the results evaluated for D_{i-1}, i.e., how to evaluate the score $\sigma[i]$ from the score $\sigma[i - 1]$. In order to do this, we maintain in M the information about how each suffix of D_i aligns with the corresponding prefix of π. This information provides a simple way to decide whether continuing any prefix of π results in a potential match.

Let us assume that the algorithm has evaluated the values of the array M for $D_i = d_i \ldots d_{i+m'-1}$. For each $j, 1 \leq j \leq m'$, the entry $M[j]$ contains the number of matched q-samples between the suffix $d_{i+m'-j} \ldots d_{i+m'-1}$ and the prefix $\pi[1 \ldots j]$. Initially, each entry of M equals 0. The array M is updated at each text position as follows:

Procedure Shift_add$(M, B[d, *])$
for $j := m'$ **downto** 2 **do** $M[j] := M[j-1]$; *% shift*
$M[1] := 0$;
for $j := 1$ **to** m' **do** $M[j] := M[j] + B[d, j]$; *% add*

In practice, the next value of M is evaluated using bit parallel operations. Let us consider the case $s = 2$. Because the sufficient number of positive q-samples in a text area is then two, only two bits are needed for an element of M. Therefore, to add a new block profile $B[d, *]$ to M, two bits are also reserved for each entry of the index B.

Algorithm LEQ.
1. construct $B[d, *]$ for each $d \in Q_q(P)$;
2. **for** $i := 1$ **to** m' **do** $M[i] := 0$;
3. **for** $j := h$ **to** n **step** h **do**
4. **begin**
5. **if** $T[j-q+1 \ldots j] \notin Q_q(P)$ **then** $bp := [0, \ldots, 0]$
else $bp := B[T[j-q+1 \ldots j], *]$;
6. Shift_add(M, bp);
7. **if** $M[m'] \geq m' - k$ **then**
8. DP$(P, T[j - m'h - 2k - q + 2 \ldots j + m - (m'-1)h + k - q])$;
9. **end**

The procedure DP searches for approximate matches of P in the text area $T[i_1 \ldots i_2]$ using dynamic programming. For an efficient implementation of DP, see [15].

Let us assume that LEQ has found a potential approximate match ending at text position j. In this case, we have two options: either verify the approximate match right away with dynamic programming, or check the condition once more using different samples[2]. The latter alternative is reasonable in cases where a single application of the filtration condition results in too many false matches.

To do another check, the algorithm screens a slightly shifted sequence of $k + s$ q-samples. It backtracks $(m'-1)h + \lfloor \frac{h}{2} \rfloor$ positions in the text and restarts the search with new q-samples. The restart is permitted only if $j - j_p$ is large enough, where j_p is the previous backtracking position, otherwise the checking phase *DP* is called.

The following result is a rough estimate for the probability of a potential match. We apply here (as well as in subsequent theorems) the symmetric Bernoulli model[3].

T 9. *The probability P_c of a potential match in the LEQ algorithm for $s = 2$ is*

$$P_c \leq 1 - (1 - \frac{m + k^2 + k}{kc^q})^{k+2}.$$

[2] This idea can also be applied to some other filtration schemes.

[3] In a random text generated according to the *Bernoulli model*, each character a from the alphabet Σ occurs in the text according to a *fixed* probability $pr(a)$, and independently of other characters, like the preceding ones. The sum of the probabilities $\Sigma_{a \in \Sigma}\, pr(a)$ equals 1. A *symmetric Bernoulli model*, sometimes also called the *i.i.d. model* (for independent and identical distribution), means that each character of the alphabet has an equal probability $\frac{1}{c}$ of occurrence in a random text. In *Markovian models*, an occurrence of a character can depend on its preceding character(s).

The bound shows the clear dependence of the filtration efficiency on the error level k. This is even more important because the algorithm can only apply relatively small values of q because of the condition $h \geq q$. On the other hand, h decreases quickly with an increasing k. This phenomenon can be seen in Fig. 3: the filtration does not work for error levels larger than 11, due to the decreased value of $q = 1$.

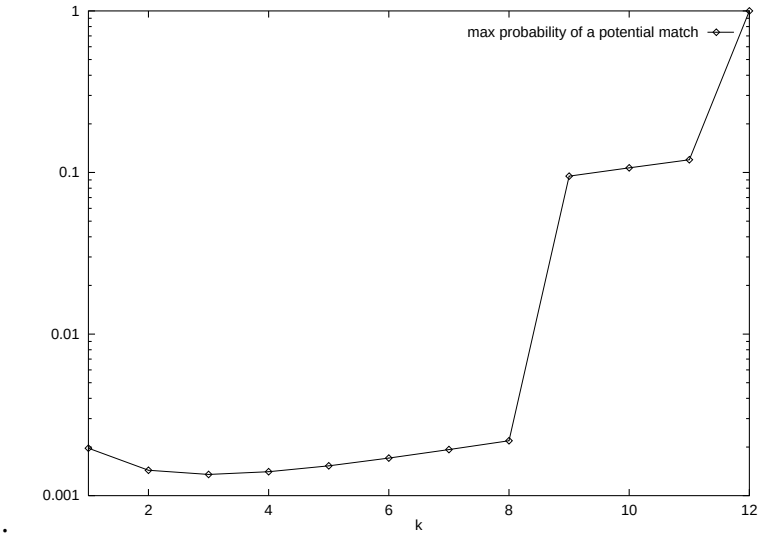


Fig. 3: The upper bound for the probability of a potential match, given by Theorem 9, for the case $c = 40, m = 40$, and $k = 1 \ldots 12$. For error levels k of at most 8, the gram length q equals 3, for $k = 9 \ldots 11$, the gram length is 2, and for $k = 12$ it equals 1.

The time for processing the q-samples dominates over the other phases, especially the dynamic programming phase, as long as the error level k remains sufficiently small. It is straightforward to prove the following result.

T 10. *Let w be the word size in bits. The average time complexity of the LEQ algorithm for $s = 2$ is $O(\frac{n}{m}k(q + \frac{k}{w}))$ in the general case and $O(\frac{n}{m}qk)$ for $k \leq w/2$.*

When $q = \log_c m$ and $k \leq w/2$, the time complexity is $O(\frac{kn}{m} \log_c m)$, which is the same as the average time complexity of the Chang–Lawler algorithm [4] and Takaoka's algorithm [13].

4.2 The LAQ algorithm

The LAQ algorithm is based on sampling and is similar to LEQ.

Algorithm LAQ.
1. preprocess P;
2. **for** $i := 1$ **to** r **do** $M[i] := 0$;
3. **for** $j := h$ **to** n **step** h **do**
4. **begin**
5. $d := T[j - q + 1 \ldots j]$;
6. **for** $l := r$ **downto** 2 **do**
7. $M[l] := M[l - 1] + \mathrm{ASM}(d, Q_l)$;
8. $M[1] := \mathrm{ASM}(d, Q_1)$;
9. **if** $M[r] \leq k$ **then**
10. $\mathrm{DP}(P, T[j - rh - 2k - q + 2 \ldots j + m - (r - 1)h + k - 1])$;
11. **end**

There are certain differences between the two algorithms, though. First, the lines 6–8 correspond to the `Shift_add` operation of the LEQ algorithm. However, whereas LEQ counts the number of correctly located pattern q-grams, occurring as q-samples in the text, the LAQ algorithm counts the differences of the text samples, compared to the corresponding pattern blocks. From the filtration efficiency point of view, the cumulative error $M[r]$ in the LAQ algorithm should be as large as possible, to decrease the number of false matches, while the corresponding entity of the LEQ algorithm should be as small as possible. These observations must be taken into account when applying the algorithms, especially when determining the values for the parameters r (in LAQ), s (in LEQ), q, and h.

Secondly, because of the common `Shift_add` operation, the LAQ algorithm could also utilize bit parallel operations. Contrary to the LEQ approach, the LAQ algorithm benefits less from this technique, since each entry of the M array requires at least $O(\log_2 k)$ bits, compared to 2 bits in LEQ (for $s = 2$).

Thirdly, the preprocessing routine of the LAQ algorithm consumes more time and space than that of the LEQ algorithm. This is because the values $\mathrm{ASM}(d, Q_l)$, $l = 1, \ldots, r$, must be tabulated for each q-gram d of the set Σ^q, whereas the LEQ algorithm records only block profiles for q-grams occurring in the pattern. This means that the LAQ algorithm is appropriate only for long texts where the time used for preprocessing the pattern is marginal compared to the time needed for the whole filtration. For short texts, it is also possible to evaluate the ASM distances on-the-fly: for each read q-sample d, the algorithm evaluates the ASM distances $\mathrm{ASM}(d, Q_l)$ for each $l, l = 1, \ldots, r$, and stores them. However, if these distances have already been computed which means that the read q-sample is not the first occurrence of d, the algorithm utilizes the stored values. Furthermore, it is possible to compromise between the two alternatives and pre-evaluate the ASM distances for common q-grams in the given text.

Fourthly, the overall time consumption for the LAQ algorithm is close to that of the LEQ algorithm, with differences due to the variations in the `Shift_add` operation. However, the probability of a potential match is different.

4.3 SLEQ — a static variation of LEQ

In the area of static filtration, the LEQ approach diminishes one of the main problems: the space needed for the text index. It is possible to reduce the space requirement by modifying the sampling scheme of the LEQ algorithm, to be suitable for storing the locations of only every hth text q-gram. This approach is complementary to other methods of decreasing space consumption, like compressing the index itself [7].

While the LEQ indexing scheme already reduces the space consumption of the q-gram index and location lists by a factor of h, we can save even more: instead of indicating a q-gram position in the text, we just refer to its location among all the q-samples. This reduces the index space from $\{1, \ldots, n\}$ to $\{1, \ldots, \lfloor \frac{n}{h} \rfloor\}$.

The standard implementation of a q-gram index stores the locations of all the q-grams of the text. Since the number of q-grams in a text of length n is $n-q+1$ and storing a position takes $\log_2 n$ bits without compression, the overall space consumption is $n \log_2 n$, provided that q is small compared to n. Let us define a *space saving factor* v_r as the space requirement ratio between our method and the standard approach, i.e.,

$$v_r = \frac{\frac{n}{h} \log_2 \frac{n}{h}}{n \log_2 n} \approx \frac{1}{h} \text{ (for large } n\text{)}.$$

In cases where one can apply the most space-efficient h for each search task, the new approach gives promising results. For example, in the case of $m = 40$, $k = 0, \ldots, 4$, $s = 2$ and $n = 500,000$, the SLEQ approach consumes only 4–17.5% of the space used by a standard q-gram index.

In the SLEQ algorithm, the sampling step h depends on the values of m, k, q, and s in a way similar to the LEQ algorithm. However, while the dynamic LEQ algorithm can choose a different h for each particular search, the static SLEQ algorithm has to apply a preprocessed index, produced according to a specific value of h. This can be done by adjusting s according to h, to be able to utilize the LEQ filtration scheme. The following result is a straightforward consequence of Theorem 5.

T 11. *Let P be a pattern, T a text, and let P' be a substring of T such that $d(P, P') \leq k$. Let h be the sampling step. Then there are $k+s$ consecutive q-samples $d_{b+1}, \ldots, d_{b+k+s}$ in P' such that $d_{b+i} \in Q_q(Q_i)$ holds for at least s of the samples, where*

$$s \leq \lfloor \frac{m-k-q+1}{h} - k \rfloor. \tag{11}$$

The formula (11) for s means that we do not need to create a different index for each pair (m, k). That is, we do not fix s before defining h, but adjust s according to the values of m, k, q, and h. Actually, since s has to be at least 1 for reasonable application of Theorem 11, we get from (11) that $\frac{m-k-q+1}{h} - k$ has to be at least 1, giving an upper bound for the error level k:

$$k \leq \frac{m-q-h+1}{h+1}. \tag{12}$$

For example, assume that a 2-gram index has been built with $h = 5$. If the pattern is of length 20, the index is applicable for error levels $0 \ldots 2$, according to (12). On the other hand, the same index can be used for error levels $0 \ldots 5$ in the case of $m = 40$. The values of s are given by (11). Note that $s = \lfloor \frac{m-k-q+1}{h} - k \rfloor$ is the largest integer for which the index can be utilized.

In the index of the q-samples, each q-sample $d_i = T[ih-q+1 \ldots ih], i = 1 \ldots \lfloor \frac{n}{h} \rfloor$, is hashed into an element of a hash table H. Each entry $H[e]$ points to the beginning of the list $L_H(e)$ consisting of those q-grams which hash to e. In addition to a q-gram u, each entry of the list $L_H(e)$ also contains pointers to the beginning and to the end of the list $L(u)$, which stores the end points of the q-samples equal to u in the text T. Optionally, we could also use a trie structure to store the locations of the text q-samples. This approach is, however, slower in practice than efficient hashing.

The score $\sigma[i]$ gives the number of preserved q-grams within the q-sample sequence $d_i, \ldots, d_{i+k+s-1}$. Let a q-gram u_P of P among the q-samples of T end at a position $j = ih$ for some i. For each block Q_b of P such that $Q_q(Q_b)$ contains u_P, we then increment the score $\sigma[i-b+1]$. For each found pattern q-gram occurrence, the algorithm updates the respective scores, by utilizing the block profiles of the pattern q-grams.

Algorithm SLEQ.

```
1.  s := ⌊(m−k−q+1)/h − k⌋;
2.  construct B[u, ∗] for each u ∈ Q_q(P);
3.  for each u ∈ Q_q(P)
4.  begin
5.      QLIST := L(u);
6.      while QLIST ≠ NULL
7.      begin
8.          i := QLIST.location;
9.          for b := 1 to k + s
10.             if B[u, b] = 1 then σ[i − b + 1] := σ[i − b + 1] + 1;
11.         QLIST := QLIST.next;
12.     end
13. end
14. for i := 1 to ⌊n/h⌋
15.     if σ[i] ≥ s then
16.     begin
17.         j := ih;
18.         DP(P, T[j − h − 2k − q + 2 . . . j + m + k − q]);
19.     end
20. end
```

It is straightforward to show the following result.

T 12. *Algorithm SLEQ has the following characteristics:*

(i) The text preprocessing phase takes $O(q\lfloor\frac{n}{h}\rfloor)$ *time on average.*

(ii) The size of the index is $O(\frac{n}{h}\log_2\frac{n}{h})$.

(iii) The expected length of a q-gram list $L(u)$ *is* $O(\lfloor\frac{n}{h}\rfloor\frac{1}{c^q})$.

(iv) The average time complexity of the text searching phase is $O(\frac{n}{c^q}km)$.

The time complexity of the checking phase is similar to that of the dynamic version, with one exception: SLEQ has to scan all the scores $\sigma[i]$, i.e., $\lfloor n/h\rfloor$ entries. Especially for small error levels k, only few of the elements are non-zero. A better way is to have a score window which slides only through the text areas with pattern q-gram occurrences. Thus, only a fraction of the elements of σ will be processed.

One possibility to maintain the score window is the following. We merge the pattern q-gram lists, the elements of which store the q-sample locations in the increasing order. Using this list, we evaluate the scores only in the text regions containing pattern q-grams.

An efficient method to implement the score window is based on the heap technique. We build a heap K, containing $card(Q_q(P))$ nodes. We start by inserting the first element of each pattern q-gram list $L(u_1),\ldots,L(u_{card(Q_q(P))})$ into K. The invariant of the heap is that the minimum value is always in the root.

After this initialization, we execute the next loop until the heap is empty: extract the root element, indicating a location of q-gram u, into the merged list L_M, and insert the next element of the list $L(u)$ into K.

The time complexity of merging the q-gram lists is given by the following theorem. The result suggests that the method is useful especially for finding a pattern with few common q-grams with the text. To test this condition, the text index should contain the number of occurrences for each different text q-sample.

T 13. *The average time complexity of merging the q-gram lists using a heap is* $O(m\frac{n}{h}\frac{1}{c^q}\log_2 m)$.

5. Concluding remarks

We have presented filtration algorithms based on q-grams for the k-differences problem. Our approach differs from previous q-gram filtration methods in that we take the order of q-grams into account: the found pattern q-grams or their approximate versions must be in the same order as they appear in the original pattern.

We have tested the efficiency of the presented methods in practice. Our experiments [11, 12] confirm that the order requirement improves efficiency for the q-gram based algorithms. In general, compared to other filtration methods, the location-sensitive approach works the better the smaller the allowed error level is. As with every q-gram based approach, the efficiency depends on the alphabet size: the bigger the alphabet size the better the filtration.

The filtration conditions presented were designed for the k differences problem. In a simpler form, they also work for the k mismatches problem. Besides the formulation of these filtration conditions, it is an open question whether they can be evaluated fast enough in practice.

References

[1] A , S. F., G , W., M , W., M , E. W., L , D. J. 1990. Basic Local Alignment Search Tool. *Journal of Molecular Biology 215*, 3, 403–410.

[2] B , R. S. M , J. S. 1977. A Fast String Searching Algorithm. *Communications of the ACM 20*, 10 (Oct.), 762–772.

[3] B -Y , R. G , G. 1992. A New Approach to Text Searching. *Communications of the ACM 35*, 10, 74–82.

[4] C , W. L , E. 1994. Sublinear approximate string matching and biological applications. *Algorithmica 12*, 327–344.

[5] C , W. M , T. 1994. Approximate String Matching and Local Similarity. In *Proc. 5th Annual Symposium on Combinatorial Pattern Matching*, Crochemore, M. and Gusfield, D., Editors, Volume 807 of *Lecture Notes in Computer Science*. Springer-Verlag, 259–273.

[6] G , Z. P , K. 1990. An Improved Algorithm for Approximate String Matching. *SIAM Journal on Computing 19*, 6, 989–999.

[7] K¨ ¨ , J. S , E. 1998. Lempel–Ziv Index for *q*-Grams. *Algorithmica 21*, 1, 137–154.

[8] M , E. W. 1994. A Sublinear Algorithm for Approximate Keyword Searching. *Algorithmica 12*, 4-5, 345–374.

[9] N , G. 2001. A Guided Tour to Approximate String Matching. *ACM Computing Surveys 33*, 1, 31–88.

[10] S , E. 1998. Approximate Pattern Matching with the *q*-Gram Family. PhD Thesis. Report A-1998-3, University of Helsinki, Department of Computer Science, Helsinki, Finland.

[11] S , E. T , J. 1995. On Using *q*-Gram Locations in Approximate String Matching. In *Proc. 3rd Annual European Symposium on Algorithms ESA '95*, Spirakis, P., Editor, Volume 979 of *Lecture Notes in Computer Science*. Springer-Verlag, 327–340.

[12] S , E. T , J. 1996. Filtration with *q*-Samples in Approximate String Matching. In *Proc. 7th Symposium on Combinatorial Pattern Matching CPM '96*, Hirschberg, D. and Myers, G., Editors, Volume 1075 of *Lecture Notes in Computer Science*. Springer-Verlag, 50–63.

[13] T , T. 1994. Approximate Pattern Matching with Samples. In *Proc. 5th Annual International Symposium on Algorithms and Computation ISAAC '94*, Du, D. and Zhang, X., Editors, Volume 834 of *Lecture Notes in Computer Science*. Springer-Verlag, 234–242.

[14] U , E. 1985 Algorithms for Approximate String Matching. *Information and Control 64*, 100–118.

[15] U , E. 1985. Finding approximate patterns in strings. *Journal of Algorithms 6*, 1, 132–137.

[16] U , E. W , D. 1993. Approximate String Matching with Suffix Automata. *Algorithmica 10*, 5, 353–364.

[17] W , S. M , U. 1992. Fast Text Searching Allowing Errors. *Communications of the ACM 35*, 10 (Oct.), 83–91.