Filtration with q-Samples in Approximate String Matching*

Erkki Sutinen and Jorma Tarhio

Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland
E-mail: {sutinen,tarhio}@cs.Helsinki.FI

Abstract. Two filtration schemes are presented for approximate string matching with k differences. In our approach q-samples, which are non-overlapping q-grams, are drawn from the text, and a text area is checked with dynamic programming, if there are enough exact or slightly distorted q-grams of the pattern in the right order in a short sequence of the q-samples. The filtration schemes are applied to searching both in the text itself and in a q-gram index of the text. The results of preliminary experiments support the applicability of the new methods.

1 Introduction

Efficient solutions for approximate string matching are necessary in many application areas, such as molecular biology, text databases, and data communications. The current CD-ROM and World-Wide Web technologies need also smart algorithms with applicable indexes.

The searched pattern often occurs in the data in a slightly transformed or even corrupted form. Therefore, the search task should be defined in terms of *approximate string matching*. Suppose that *text* $T = T[1 \dots n]$ and *pattern* $P = P[1 \dots m]$ are in alphabet Σ of size c. Given integer k, the maximum tolerated *number of errors*, the task of the k *differences problem* is to find (the end points of) the approximate occurrences of pattern P in text T. We call P' an *approximate occurrence* of P, if *edit distance* $d(P, P')$ is at most k, that is, P can be transformed to P' with at most k edit operations which are insertions, deletions, and changes. The reduced form, where only changes are allowed as edit operations, is called the k *mismatches problem*.

A natural solution for the approximate string matching problem is a modification of dynamic programming. However, since the time complexity of this solution is $O(kn)$ [11, 19] and n is typically very large, more efficient approaches are desirable. There are two directions to proceed. One can construct an index of the text [6, 10, 12, 17]. In this form of the problem, called the *static* one, it is assumed that the text does not change between consecutive searches. The index is used to locate the text areas containing potential approximate matches, and

* The work was supported by the Academy of Finland.

thus it is not necessary to process the whole text with time consuming dynamic programming.

The technique of locating potential occurrences is called *filtration*. Besides the static problem, one can apply filtration also to the *dynamic* problem, where the text is not preprocessed. Most filtration methods [4, 5, 13, 16, 18] presented in the literature are designed for the dynamic case. Every filtration method (dynamic or static) is based on finding a necessary condition [8, 15, 18, 22] for an approximate occurrence of the pattern in the text. A crucial property, determining the applicability of a given filtration method A, is its *filtration efficiency* $f_A = \frac{n-n_p}{n}$, where n_p denotes the total width of the text area processed by dynamic programming.

By strengthening filtration conditions, it is possible to increase the filtration efficiency. The point lies, however, elsewhere. One must find an efficient filtration method with as little overhead as possible. An essential question with the static methods is the size of the index. It can be validated only experimentally whether a filtration approach is a practically useful compromise between the opposite requirements. Furthermore, a certain filtration method is usually applicable only for limited ranges of values of problem parameters k, m, c, and n [9].

Takaoka [15] introduces a sublinear[1] filtration technique, based on sampling, for the dynamic problem. In his method every hth q-gram of the text is drawn as a sample (a q-gram is a continuous substring of length q). If a sample appears in the pattern, a neighborhood of the sample is verified with dynamic programming. We present in [14] a more advanced technique, where it is demanded that at least s of the q-grams of the pattern must appear as samples in a potential occurrence and these q-grams preserve their mutual order in the text. In a way, this is a reduction to the k mismatches problem.

Chang and Marr [5] present an optimal sublinear expected time algorithm for the dynamic problem. They divide the text into regions of size $(m-k)/2$ and consider non-overlapping subsequent q-grams of each region. When the cumulative sum of the errors between each q-gram and its best approximate match in P exceeds k, one can skip to the next region.

We will present two new algorithms: a static and a dynamic one. Our motivation is to develop practical methods with a competitive filtration efficiency. The static algorithm given in Section 2 is a modification of our earlier dynamic algorithm [14]. The dynamic algorithm introduced in Section 3 is a cross of the Chang-Marr algorithm [5] and our algorithm [14].

Since general indexing methods [20] are based on words, they can hardly handle even simple errors. Therefore, a q-gram based index is a natural choice for approximate string matching. Jokinen and Ukkonen [10] were the first to present an indexing scheme based on q-grams for approximate searches. Holsti

[1] A string matching algorithm is commonly called sublinear when it does not inspect all characters of the text on the average, even if its average time complexity were linear. There are two types of sublinearity of this kind: the constant of the time complexity depends either only on the problem parameters (e.g. the Boyer-Moore approach in exact string matching [3]) or also on some parameters of the algorithm.

and Sutinen [7] make use of the locations of the q-grams in a different way from our approach and slightly improve the searching time of the Jokinen-Ukkonen algorithm. Both of these approaches [7, 10] use the same indexing scheme. The problem of this scheme is the large size of the index. Our method produces considerably smaller indexes than the two preceding approaches without sacrificing the speed of search.

The results of preliminary experiments in Section 4 support the applicability of the new methods. For example, our static approach consumes only 4–17.5% of the space used by the earlier q-gram based methods, when $c = 40$, $m = 40$, $k = 0, \ldots, 4$, and $n = 500,000$.

We also performed experiments on dynamic algorithms. An essential factor for the applicability of a filtration method is the largest relative error level for which the method still filtrates efficiently. Preliminary experiments reviewed show that for an alphabet of size 20, the introduced dynamic algorithm is efficient for error levels up to over 30%, which is better than with related methods.

2 Indexing with Exact q-Grams

We present a new static algorithm based on the locations of the q-grams of the pattern. The algorithm has been developed from our earlier dynamic algorithm [14] which is called the LEQ algorithm in the following. The abbreviation LEQ stands for the locations of exact q-grams. We start with an overview of the LEQ algorithm.

2.1 The LEQ Algorithm

In the LEQ algorithm the k differences problem is reduced to the k mismatches problem, which can be solved efficiently applying the shift-add technique [2].

In the text, every hth q-gram is examined as a sample, $h \geq q$. These samples are called *q-samples.* Distance

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

between the endpoints of two subsequent q-samples is the *sampling step.* Let $d_1, \ldots, d_{\lfloor n/h \rfloor}$ be the q-samples of the text. It is required that at least s of $k+s$ consecutive q-samples occur in the original pattern; furthermore, they must occur approximately at correct locations in an approximate occurrence of the pattern. A sequence of $k+s$ consecutive q-samples is called a *test sequence* and $r = k+s$ is called the *sample size.*

An approximate location of a q-gram in the pattern is determined by selecting r fixed blocks $Q_1, \ldots, Q_r$ from the pattern using sampling step h, yielding $Q_i = P[(i-1)h + 1 \ldots ih + k + q - 1]$.

The filtration condition is formulated as follows. Let $P' = T[i \ldots j]$ be an approximate occurrence of P, i.e. $d(P, P') \leq k$, and let d_{b+1} be the leftmost q-sample of P'. Then there is integer t, $0 \leq t \leq k+1$, and test sequence

$d_{b+1+t}, \ldots, d_{b+k+s+t}$ included by P' such that $d_{b+i+t} \in Q_i$ holds for at least s of the samples.

The shift-add technique [2] is applied in the searching phase to maintain the number of matching q-samples. The pattern is preprocessed to construct the block profile of each q-gram needed by the shift-add routine. The average time complexity of the LEQ algorithm is $O(\frac{k^2qn}{mw})$, where w is the word size.

2.2 Static Variation

The preprocessing phase creates an index (either using hashing or as a trie) containing all the q-samples of text T. For each q-gram u of the index, ordered list $L(u)$ contains all the end points of the occurrences of u as q-samples in text T.

The filtration phase searches for the potential approximate matches of pattern P. First, a block profile for each q-gram of P is computed in a similar way as in the LEQ algorithm. To find out the potential approximate matches, the algorithm searches for the occurrences of each q-gram of the pattern among the q-samples. Using the location information, the potential occurrences in text T can be identified as follows:

Counter M is maintained for each q-sample. Let q-gram u_P of P be among the q-samples of T and let u_P end at position $j = ih$ for some i. Counter $M[i - b + 1]$ is incremented for each block Q_b of P such that Q_b contains u_P. After the filtration phase, the algorithm checks with dynamic programming all the text areas satisfying $M[i] \geq s$, where s has the maximal value, allowed by formula (1). See an example in Fig. 1.

There are two major distinctions between the LEQ algorithm and its static counterpart. First, while the dynamic method can freely choose the value of s and evaluate h according to it, the static algorithm must adapt itself to an index, based on a fixed value of h. Second, only the dynamic method utilizes the shift-add technique.

In the case of very few potential occurrences, the basic approach wastes time while checking all the counters (because most of them are zero). To speed up the scanning of the counters, one could follow list $L(u_P)$ of each q-gram u_P in a "parallel" way, i.e. one would always get the relevant q-gram occurrences among the q-samples. This leads to a variation which stores only the non-zero counters.

The decision, whether to apply the described heuristic (with the overhead of the bookkeeping of q-gram lists) or not, can be based on the number of occurrences of the q-grams of P among the q-samples.

2.3 Filtration Conditions

The filtration scheme of our method is based on Theorem 2 in [14], stating a necessary condition for an approximate occurrence of pattern P in text T.

Let us assume that text T has been indexed according to sampling step h. It turns out that the filtration phase can utilize this index by adjusting s as long as $s \geq 1$:

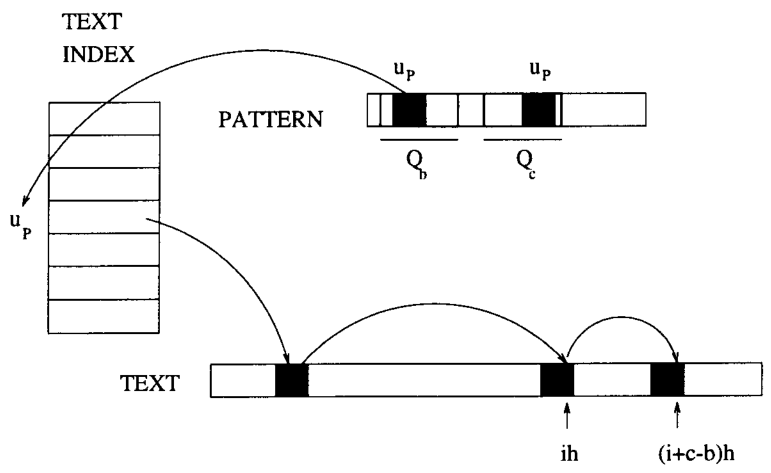

Fig. 1. *Applying the LEQ approach to static texts. Let us assume that q-gram u_P occurs in two blocks Q_b and Q_c. The occurrences of u_P at text positions ih and $(i + c - b)h$ result in $M[i - b + 1] = 2$, indicating a potential occurrence for $s = 2$.*

Theorem 1. *Let P' be a substring of T such that $d(P, P') \leq k$. Let h be the sampling step. Then at least s of the q-samples in P' occur in P, where*

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

The formula for s means that it is not necessary to create a different index for each (m, k) pair. That is, the algorithm adjusts s according to the values of h, m, q, and k to make filtration as efficient as possible.

Since the static algorithm is based on the same filtration scheme as its dynamic counterpart, also the filtration efficiency remains the same. Therefore, the following holds for filtration efficiency f of static algorithm for $s \geq 2$ (Theorem 4 in [14]):

$$f \geq (1 - \frac{m + 2k^2 + 3k}{kc^q})^{k+2}.$$

2.4 Algorithm

In the preprocessing phase a hash table is created for the q-samples of text T. Optionally, one could also use a trie, but this approach is slower than hashing in practice.

The search algorithm, like many algorithms for the dynamic problem, consists of two phases: filtration and checking. The filtration phase filters potential occurrences, which are verified using dynamic programming in the checking phase.

The filtration phase goes through all the q-grams of the pattern and counts the block matches at each q-sample.

The filtration phase is presented below as Algorithm SLEQ (static LEQ). The algorithm scans the occurrences of q-grams of the pattern among the q-samples. For each found q-gram occurrence, the algorithm updates the respective counters, by utilizing the q-gram location information, produced in the pattern preprocessing phase.

Algorithm SLEQ.

```
1.  $s := \lfloor \frac{m-k-q+1}{h} \rfloor - k$;
2.  preprocess pattern $P$;
3.  for each different pattern $q$-gram $u_P$
4.      QLIST := $L(u_P)$;
5.      while QLIST $\neq$ NULL
6.          $i$ := QLIST.pos;
7.          for $b := 1$ to $k + s$
8.              if $u_P \in Q_b$ then $M[i - b + 1] := M[i - b + 1] + 1$;
9.          QLIST := QLIST.next
10.     end
11. end
```

The dynamic programming phase checks each potential occurrence of the pattern, indicated by $M[i] \geq s$ (see Theorem 3 in [14]):

```
1. for $i := 1$ to $\lfloor n/h \rfloor$
2.     if $M[i] \geq s$ then
3.         $j := ih$;
4.         $DP(P, T[j - h - 2k - q + 2 \ldots j + m + k - q])$;
5.     end
```

Procedure DP searches for approximate matches in text region $T[j_1 \ldots j_2]$. This procedure evaluates edit distance matrix $d[0 \ldots m, 0 \ldots (j_2 - j_1 + 1)]$ using dynamic programming, with initial values $d[i, j] = 0$ for $i = 0$ and $d[i, j] = i$ for $j = 0$ and with the recurrence

$$d[i,j] = \min \begin{cases} d[i-1, j-1] + (\text{if } P[i] = T[j_1 + j - 1] \text{ then } 0 \text{ else } 1) \\ d[i-1, j] + 1 \\ d[i, j-1] + 1. \end{cases}$$

There is a match ending at position j if and only if $d(m, j) \leq k$.

A drawback of the approach described above is that it scans through all the elements of array M. Especially for small error levels, only few of the elements are non-zero. A more efficient solution is to use a heap for merging the lists of all the q-grams of the pattern. With this combined list, it is easy to evaluate the counters only in the text regions containing q-grams of the pattern.

2.5 Analysis

Let us consider the time complexities of both the preprocessing and filtration phase in the i.i.d. model, in which the characters in both the text and the patterns are independently and identically distributed. The proofs are skipped.

Theorem 2. *Algorithm SLEQ has the following characteristics:*
a) The preprocessing phase works in time $O(\frac{qn}{h})$ on the average.
b) The size of the index is $O(\frac{n}{h}\log\frac{n}{h})$.
c) The expected length of a q-gram list $L(u)$ is $O(\frac{n}{hc^q})$.
d) The average time complexity of the filtration phase is $O(\frac{nkm}{c^q})$.

The time complexity of the dynamic programming phase follows from the same formula evaluated for the LEQ algorithm in [14], with one exception: it has to scan all the entries of array M, i.e., $\lfloor n/h \rfloor$ entries.

Finally, we give the time complexity of merging the q-gram lists. 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 index must contain the number of occurrences for each q-qram in the text.

Theorem 3. *The average time complexity of merging the q-gram lists using heap is $O(\frac{nm}{hc^q}\log m)$.*

3 Dynamic Filtration with Approximate q-Grams

In this section, we combine the LEQ algorithm with the filtration scheme presented by Chang and Marr [5]. We start with introducing the Chang-Marr algorithm (CM for short).

3.1 The CM Algorithm

The asm distance $asm(u, B)$, introduced by Chang and Marr, denotes the edit distance between q-gram u and its best match in string B. Chang and Marr use the asm distance in their optimal sublinear expected time algorithm to find the approximate matches of pattern P in text T. Text T is divided into consecutive regions of size $(m-k)/2$ in the CM algorithm just like in the earlier sublinear algorithm by Chang and Lawler [4]. Since any approximate occurrence of P covers completely at least one region, Chang and Marr scan the non-overlapping, consecutive q-grams from the beginning of each region until the cumulative asm distance exceeds k or the region ends. If the text area covered by the corresponding q-grams reaches the following region, the algorithm checks the corresponding area by dynamic programming, and otherwise the algorithm skips over to the beginning of the next region.

More formally, let us assume that $T[j \ldots j']$ is the text region to be inspected and $d_i, d_{i+1}, \ldots$ are the consecutive non-overlapping q-grams starting from $T[j]$. Let ρ be the smallest index such that $\sum_{j=i}^{i+\rho-1} asm(d_j, P) > k$. The algorithm checks the corresponding text area by dynamic programming only if q-gram $d_{i+\rho-1}$ ends at text position l, $l > j'$.

The average time complexity of the filtration phase of the CM algorithm is $\Theta((n/m)(k + \log_c m))$.

3.2 Applying the asm Distance with Locations of q-Grams

Intuitively, the filtration condition of the CM algorithm means that any approximate occurrence of P must contain subsequent q-grams with the total error of k or less. This is the point where it is possible to enhance the method by introducing an additional condition: the subsequent q-grams of an approximate occurrence must be in the same order as their error-free counterparts in the pattern. Instead of computing the asm distance between a given q-gram and the pattern, we compute the asm distance between the q-gram and each block of the pattern (see Fig. 2). We call this approach LAQ for the locations of approximate q-grams.

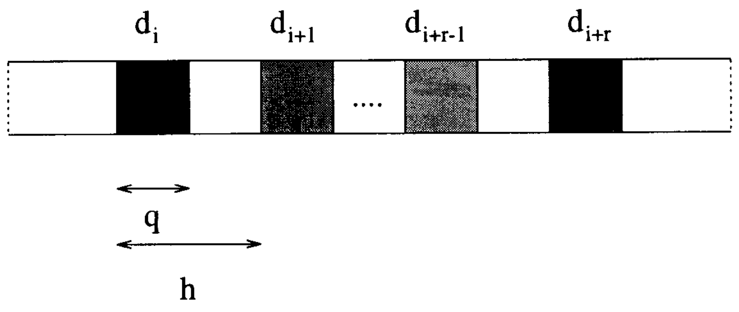

Fig. 2. *Idea of the LAQ Algorithm. In order to decide whether test sequence $d_i, d_{i+1}, \ldots, d_{i+r-1}$ is a part of a potential approximate occurrence, sum $\sum_{j=1}^{r} asm(d_{i+j-1}, Q_j)$ is evaluated. If the sum is at most k, the potential match is verified by dynamic programming.*

In the LAQ approach it is not required that the q-samples are consecutive, but the algorithm inspects every hth q-gram, where $h \geq q$. Moreover, since LAQ is more sampling-oriented than region-oriented, the sum of errors of a fixed number of q-samples is considered instead of cumulative error in a text region.

Let P' be an approximate occurrence of P. Since we base our filtration on the q-grams of P occurring in the approximate match, it is required that there is a test sequence of r q-samples within P' such that the sum of errors of that test sequence is at most k. Since an approximate occurrence can be as narrow as $m - k$ including $m - k - q + 1$ q-grams, we will get the following lower and upper bound for the *sampling step h* (see also Theorem 1 of [14]):

$$q \leq h \leq h_{max} = \lfloor \frac{m - k - q + 1}{r} \rfloor .$$

By choosing the minimum value for h, i.e., $h = q$, we get the following bound for sample size r:

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

because of the condition $rh \leq m-k-q+1$. Therefore, our method does not reduce to the CM algorithm with $h = q$, since Chang and Marr do not use a *fixed* value for r. However, the following bound holds for the number of q-grams r in the CM algorithm:

$$r \leq \lceil \frac{(m-k)/2}{q} \rceil.$$

Note that an efficient application of the presented filtration scheme requires choosing appropriate values for the parameters. The shift-add technique [2] is applied to maintain the cumulative sums of errors. The maximal value of each cumulative sum is $k + q$. As a result, a word of size w bits contains at most $w/(\log(k+q)+1)$ sums.

There are two possibilities to compute asm distances. One way is to preprocess the distances for every q-gram and for every block of the pattern. Another way is to compute the distances on demand and to store the computed values.

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.     $d := T[j-q+1 \dots j]$;
6.     Shift_add($M, asm[d,*]$);
7.     **if** $M[r] \leq k$ **then**
8.         $DP(P, T[j - rh - 2k - q + 2 \dots j + m - (r-1)h + k - q])$;
9. **end**
```

For the range of the dynamic programming area, see Theorem 3 in [14].

It is straight-forward to estimate the average behavior of the algorithm:

Theorem 4. *The average time complexity of the filtration phase of Algorithm LAQ is* $O(n\frac{m\log(k+q)}{q^2 w})$, *where* w *denotes the word size.*

Proof. Since $h \leq (m-k-q+1)/r$, we can bound sample size r:

$$r \leq \frac{m-k-q+1}{h} \leq \frac{m}{h} \leq \frac{m}{q},$$

because $h \geq q$.

The time used for the filtration (lines 3–6 of Algorithm LAQ) is proportional to the product of the number of q-samples and the time for one shift-add operation[2]. We get the following bound for the average time complexity of the filtration phase:

$$\frac{n}{h} \cdot \frac{\log(k+q)r}{w} \leq n\frac{\log(k+q)m}{q^2 w}.$$

□

Remark. The algorithm runs in sublinear time if $g(q) = q^2 w - m\log(k+q) \geq 0$. To get a rough impression of the algorithm's behavior, let us consider two cases:

Let us assume that $q \leq k$. Now $\log(k+q) \leq \log 2k$ holds and the condition is reduced to $\sqrt{m(1+\log k)/w} \leq q \leq k$. Otherwise, $q > k$ holds. Because $1 + \log q \leq q$ is satisfied for $q \geq 1$, the value of q should be at least $\max(k+1, m/w)$.

Because of the requirement $q \leq h$, the value of q cannot be arbitrarily large. Note that function g is only an estimate, and thus our sublinearity condition is pessimistic for a large range of problem parameters.

Variation. Instead of the sampling scheme adopted by the LAQ algorithm, we could apply the region approach of the CM algorithm and compute asm distances based on the blocks of the pattern. See Fig. 3.

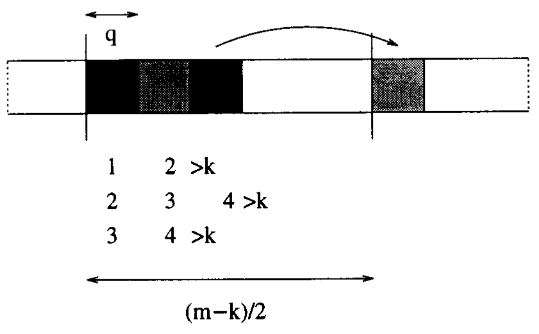

Fig. 3. *Idea of an alternative approach. The q-grams of a text region are first compared to pattern blocks 1 and 2, resulting in cumulative sum of errors which is greater than k. Similarly, the other comparisons produce cumulative sums exceeding k before reaching the next text region. The rest of the region is skipped.*

3.3 Related Methods

Besides the CM algorithm, the LAQ approach is related with the Four Russians' technique (4R for short) which is a divide-and-conquer method originally designed for bit matrix multiplication [1]. In approximate string matching, the

[2] For simplicity, we omit term 1 from the sum $\log(k+q)+1$.

4R technique has been successfully applied by Wu [21], speeding up the $O(3^m)$ pattern preprocessing phase of Ukkonen's dynamic algorithm [19] (with an eventual slow-down in the scanning phase) and giving a somewhat more balanced compromise between the time consumption of the pattern preprocessing phase and the scanning phase. As 4R, the LAQ approach tabulates the subsolutions as the asm-distances and use these values by table lookup, applying the encoded q-samples. Unlike in 4R, we do not use all the substrings but have holes in between. In fact, our approach (as well as the CM method) can be seen as an approximation of dynamic programming, or a relaxed counterpart of the 4R technique.

The LAQ appraoch is also related to the static method of Myers [12]. In a way, Myers' approach resembles that of 4R, by reducing the computation into smaller subcases. Therefore, Myers can solve the problem by generating q-neighborhoods and the preprocessed index contains the occurrences of these q-grams. Since the complexity of generating the q-neighborhoods increases exponentially with q, the reduction of finding k-approximate matches into finding q-approximate matches, where $q < k$, is substantial for his method.

4 Experimental Results

Algorithm SLEQ. Our experiments show that the filtration efficiency of Algorithm SLEQ (Section 2) is similar to that of earlier q-gram based solutions [7, 10]. Let us consider an example of searching for a pattern of size $m = 40$. The text and the patterns were generated according to the i.i.d. model in an alphabet of 40 characters. If the relative error k/m remains less than 20%, practically no extra columns are evaluated, i.e., the dynamic programming checks only the text area with a real match.

Measured by the space consumption, our method shows clear benefit when compared with the earlier methods. Considering the same example as above, with text T of size 500,000, the difference is clear for small relative errors, which give relatively large values for h and q.

Let us consider the space consumption, when $k = 0$. This gives $h = 19$, if $q = 3$. Out of the possible $40^3 = 64,000$ 3-grams, the $\frac{500,000}{19} = 26,315$ 3-samples contain 21,572 different ones.

The main benefit is, however, the reduced number of location nodes: When ordinary q-gram methods store the locations of all the 500,000 3-grams, requiring a space of $500,000 \log 500,000 =$ 9,500,000 bits, our approach is satisfied with $26,315 \log 26,315 =$ 390,000 bits. Table 1 lists the space saving factor v_r for different k, where v_r is defined as the space requirement ratio between our method and the standard approach, i.e.

$$v_r = \frac{\frac{n}{h} \log \frac{n}{h}}{n \log n}.$$

The results show that our method saves space for relative error 0–30%.

In the formula for v_r it is assumed that no index compression technique is applied and the space consumption of an index of d entries within n' possible locations is hence $d \log n'$. Using advanced compression schemes [20] one can decrease the factor $\log n'$ substantially with a small time overhead. The formula for v_r gives thus a somewhat optimistic prediction for the space saving of the SLEQ method.

Table 1. *Space saving factor v_r for $c = 40$, $m = 40$, and $n = 500,000$.*

k	h	v_r
0	19	0.041
1	12	0.068
2	9	0.093
3	7	0.122
4	5	0.175
5...6	4	0.224
7...8	3	0.305
9...12	2	0.474
13...	1	1.000

Algorithms LEQ and LAQ. We compared the filtration efficiency of the LAQ algorithm with that of earlier algorithms. The text and the patterns were generated according to the i.i.d. model. Table 2 shows the results for an alphabet of size $c = 20$. Algorithm ABM is approximate Boyer-Moore [16], T is Takaoka's algorithm [15]. The text is 100,000 characters long, pattern length m is 40, and error level k varies from 0 to 14, i.e., from 0% to 35% (where the relative error is defined as k/m). We have counted the number of columns (i.e. the total width of the area) processed in the dynamic programming phase to evaluate the filtration efficiency of the algorithms.

The parameters of the LEQ algorithm are determined in a slightly different way from [14]: we first compute the maximal h and then adjust s according to the value of h like in our static method. The experiments show that this choice improves the efficiency. Takaoka's algorithm loses its filtration power at relative error level 22.5%. Algorithm ABM is placed in the middle of LEQ and T, measured by the maximum tolerated error level. LEQ is efficient up to 27.5% relative error level.

While the LAQ algorithm marks 3% of the columns for relative error level 27.5%, three times more than LEQ, it does not, however, lose its filtration power as sharply as LEQ does. This is its major advantage over LEQ, making LAQ efficient for relative error levels up to over 30%. The benefit of using the location information, provided by the pattern blocks, is reflected by the gain of LAQ over CM.

Table 2. *Percentage of processed columns for $c = 20$, $n = 100{,}000$, and $m = 40$. In the CM algorithm, $q = 5$ gives the best filtration, while LAQ gets the optimal filtration for $r = 3$, which gives $q = 7$ for $k = 0 \ldots 13$ and $q = 6$ for $k = 14$.*

k	LEQ	ABM	T	CM	LAQ
0–5	0	0	0	0	0
6	0	0	0	1	0
7	0	0	7	12	0
8	0	0	9	56	0
9	0	2	83	99	0
10	0	5	94	100	0
11	1	79	95	100	3
12	91	97	96	100	18
13	100	100	100	100	69
14	100	100	100	100	100

5 Concluding Remarks

We have presented two new solutions for approximate string matching. The SLEQ approach for static texts improves the space efficiency of the earlier q-gram based methods. The dynamic LAQ approach offers improved filtration efficiency at high error levels. Because of the cost of the preprocessing phase, the LAQ method is competitive only for long texts.

The LAQ approach could also be applied to the static case. Let us call u an e-approximate q-gram of string B, if $asm(u, B) \leq e$. An index can in principle be used in two ways: either inclusively to mark the essential text areas or exclusively to mark the text areas guaranteed with no matches. To make this possible one has to generate all the e-approximate q-grams of each block Q_i of the pattern. Value e depends on the number of allowable errors k and sample size r. The implementations of the inclusive and exclusive approaches are only slightly different.

References

1. V. Arlazarov, E. Dinic, M. Kronrod, and I. Faradzev: On economical construction of the transitive closure of a directed graph. *Dokl. Akad. Nauk SSSR* **194** (1970), 487–488 (in Russian). English translation in *Soviet Math. Dokl.* **11** 5, 1209–1210.
2. R. Baeza-Yates and G. Gonnet: A new approach to text searching. *Communications of ACM* **35**, 10 (1992), 74–82.
3. R. Baeza-Yates, G. Gonnet, and M. Régnier: Analysis of Boyer-Moore-type string searching algorithms. In: *Proc. First ACM-SIAM Symposium on Discrete Algorithms*, 1990, 328–343.
4. W. Chang and E. Lawler: Sublinear approximate string matching and biological applications. *Algorithmica* **12**, 4–5 (1994), 327–344.

5. W. Chang and T. Marr: Approximate string matching and local similarity. In: *Combinatorial Pattern Matching, Proceedings of 5th Annual Symposium* (ed. M. Crochemore and D. Gusfield), *Lecture Notes in Computer Science* **807**, Springer-Verlag, Berlin, 1994, 259–273.
6. A. Cobbs: Fast approximate matching using suffix trees. In: *Combinatorial Pattern Matching, Proceedings of 5th Annual Symposium* (ed. Z. Galil and E. Ukkonen), *Lecture Notes in Computer Science* **937**, Springer, Berlin, 1995, 41–54.
7. N. Holsti and E. Sutinen: Approximate string matching using q-gram places. *Proc. Seventh Finnish Symposium on Computer Science* (ed. M. Penttonen), University of Joensuu, 1994, 23–32.
8. R. Grossi and F. Luccio: Simple and efficient string matching with k mismatches. *Information Processing Letters* **33** (1989), 113–120.
9. P. Jokinen, J. Tarhio, and E. Ukkonen: A comparison of approximate string matching algorithms. To appear in *Software – Practice and Experience.*
10. P. Jokinen and E. Ukkonen: Two algorithms for approximate string matching in static texts. In: *Proceedings of Mathematical Foundations of Computer Science 1991* (ed. A. Tarlecki), *Lecture Notes in Computer Science 520*, Springer-Verlag, Berlin, 1991, 240–248.
11. G. Landau and U. Vishkin: Fast string matching with k differences. *Journal of Computer and System Sciences* **37** (1988), 63–78.
12. E. Myers: A sublinear algorithm for approximate keyword searching. *Algorithmica* **12**, 4–5 (1994), 345–374.
13. P. Pevzner and M. Waterman: Multiple filtration and approximate pattern matching. *Algorithmica* **13** (1995), 135–154.
14. E. Sutinen and J. Tarhio: On using q-gram locations in approximate string matching. In: *Proc. 3rd Annual European Symposium on Algorithms ESA '95* (ed. P. Spirakis), *Lecture Notes in Computer Science* **979**, Springer, Berlin, 1995, 327–340.
15. T. Takaoka: Approximate pattern matching with samples. *Proceedings of ISAAC '94, Lecture Notes in Computer Science* **834**, Springer-Verlag, Berlin, 1994, 234–242.
16. J. Tarhio and E. Ukkonen: Approximate Boyer-Moore string matching. *SIAM Journal on Computing* **22**, 2 (1993), 243–260.
17. E. Ukkonen: Approximate string-matching over suffix trees. In: *Combinatorial Pattern Matching, Proceedings of 4th Annual Symposium* (ed. A. Apostolico et al.), *Lecture Notes in Computer Science* **684**, Springer-Verlag, Berlin, 1993, 228–243.
18. E. Ukkonen: Approximate string matching with q-grams and maximal matches. *Theoretical Computer Science* **92**, 1 (1992), 191–211.
19. E. Ukkonen: Finding approximate patterns in strings. *Journal of Algorithms* **6** (1985), 132–137.
20. I. Witten, A. Moffat, and T. Bell: *Managing Gigabytes*, Van Nostrand Reinhold, New York, 1994.
21. S. Wu: Approximate pattern matching and its applications. Ph.D. Thesis, Report TR 92-21, Department of Computer Science, University of Arizona, 1992.
22. S. Wu and U. Manber: Fast text searching allowing errors. *Communications of ACM* **35**, 10 (1992), 83–91.

This article was processed using the LaTeX macro package with LLNCS style