Better Filtering with Gapped q-Grams

Stefan Burkhardt[1]* Juha Kärkkäinen[2]**

[1] Center for Bioinformatics, Saarland University
Postfach 151150, 66041 Saarbrücken, Germany; stburk@mpi-sb.mpg.de
[2] Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany; juha@mpi-sb.mpg.de

Abstract. The q-gram filter is a popular filtering method for approximate string matching. It compares substrings of length q (the q-grams) in the pattern and the text to identify the text areas that might contain a match. A generalization of the method is to use *gapped* q-grams, subsets of q characters in some fixed non-contiguous shape, instead of contiguous substrings. Although mentioned a few times in the literature, this generalization has never been studied in any depth. In this paper, we report the first results from a study on gapped q-grams. We show that gapped q-grams can provide orders of magnitude faster and/or more efficient filtering than contiguous q-grams. The performance, however, depends on the shape of the q-grams. The best shapes are rare and often possess no apparent regularity. We show how to recognize good shapes and demonstrate with experiments their advantage over both contiguous and average shapes. We concentrate here on the k mismatches problem, but also outline an approach for extending the results to the more common k differences problem.

1 Introduction

Given a pattern string P of length m, a text string T of length n, and a distance k, the approximate string matching problem is to find all substrings of the text T that are within a distance k of the pattern P. The most commonly used distance measure, leading to the k *differences problem*, is the Levenshtein distance, the minimum number of single character insertions, deletions and replacements needed to change one string into the other. A simpler variation, the k *mismatches problem*, uses the Hamming distance, the minimum number of replacements needed to change one string into the other, i.e., the number of mismatching characters.

The fastest algorithm in practice for the k differences problem is the bitparallel dynamic programming algorithm of Myers [9]. It works in time $O(nm/w)$, where w is the word size of the machine. An extensive survey and comparison of algorithms is given in [10]. The k mismatches problem is a simpler problem but

* Supported by the DFG 'Initiative Bioinformatik' grant BIZ 4/1-1.
** Partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

we do not know of any asymptotically faster algorithms for it. As the pattern is usually short in comparison to the text, much faster string matching is often possible if the text has been preprocessed to build, e.g., the suffix tree of the text. This is called *indexed* or *offline* string matching. For the k differences and the k mismatches problems, dynamic programming over the suffix tree [1, 17, 4] can be fast, but only for short patterns and small distances k.

Filtering is a way to speed up approximate string matching. The idea is to narrow down the search to a small fraction of the text with some filtering method (the *filtering phase*) and search only those areas using a proper approximate string matching algorithm (the *verification phase*). A good filtering method is *fast* and *efficient*, i.e., leaves only a small area to be verified. A good survey of filtering methods is given in [10]. Among the most popular and studied filtering methods is the *q-gram method*.

A *q-gram* is a substring of length q. The basic q-gram method works as follows. First, find all matching q-grams between the pattern and the text. That is, find all pairs (i, j) such that the q-gram at position i in the pattern is identical to the q-gram at position j in the text. We call such a pair a *hit*. Second, identify the text areas that have enough hits. These are the areas passed to the verification phase. There are different ways of defining the text areas and counting the hits in them (see, e.g., [6, 5]). However, all of them have the same *threshold*, the significant number of q-grams. This number is given by the *q-gram lemma*.

Lemma 1 (The q-gram lemma [6]). *Let P and S be strings of length m with (Levenshtein or Hamming) distance k. Then P and S have at least $t = m - q(k+1) + 1$ common q-grams.*

The threshold given by the lemma is tight in the sense that using any lower value might miss an occurrence (see Lemmas 2 and 4). For example, strings `ACAGCTTA` and `ACACCTTA` have Hamming and Levenshtein distance 1 and have $8 - 3(1+1) + 1 = 3$ common q-grams: `ACA`, `CTT` and `TTA`.

The above description of the q-gram method leaves many details open. Different realizations of the method are described in [6, 16, 5, 2]. There are also many variations, e.g., not using all q-grams [15, 14].

The q-gram method is particularly suitable for indexed string matching. An index of all text q-grams is simple to implement using table lookup, hashing or a trie. This makes the q-gram method very fast unless the number of hits is large. Thus, we would like q to be large since the number of hits decreases exponentially as q increases. On the other hand, as q increases the threshold given by the q-gram lemma decreases, which reduces filtering efficiency. The best trade-off depends on the implementation and the application.

A generalization of the q-gram method uses *gapped* q-grams, subsets of q characters of a fixed non-contiguous *shape*. For example, the 3-grams of shape `##.#` in the string `ACAGCT` are `AC.G`, `CA.C` and `AG.T`. Gapped q-grams have been used in [3, 11, 8]. In [3, 11], the motivation is to increase the filtration efficiency by considering multiple shapes. Pevzner and Waterman [11] use q-grams containing every $(k+1)$st character together with contiguous q-grams for the k mismatches problem. The FLASH algorithm of Califano and Rigoutsos [3] uses as many as 40 different random shapes in a probabilistic manner, i.e., without a guarantee of finding all occurrences. Their approach is effective for high k but they need a huge index (18GB for a 100 million nucleotide DNA database). The Grampse system of Lehtinen et al. [8] uses a shape containing every hth character for some h (similar to [11]) for exact matching. Their motivation of using gapped q-grams is to reduce dependencies between the characters of a q-gram.

In this paper, we will show that gapped q-grams have advantages over contiguous q-grams even when using just one shape and not being concerned with dependencies between characters. Gapped q-grams of suitably chosen shape provide much faster and/or more efficient filtering. We have observed improvements of several orders of magnitude in our experiments.

The results in this paper apply only to the k mismatches problem. The k differences problem causes difficulties for gapped q-grams because they are affected by insertions and deletions in the gaps. However, the difficulties can be tackled by using multiple shapes. For example, we can use the shapes `#####.##` and `#####...##` to handle an insertion or a deletion in the gap of the shape `#####..##`. We are currently working on this approach to extend our results to the k differences problem. As a preliminary result in this direction, we show that even q-grams with just one gap are better than contiguous ones. As the above example shows, q-grams with few gaps are of special interest for the k differences problem.

2 Shapes

A *shape* Q is a set of non-negative integers containing 0. The *size* of Q, denoted by $|Q|$, is the cardinality of the set. The *span* of Q is $s(Q) = \max Q + 1$, i.e., the size of the minimum contiguous interval containing Q. A shape Q with size q and span s is called a q-shape or a (q, s)-shape.

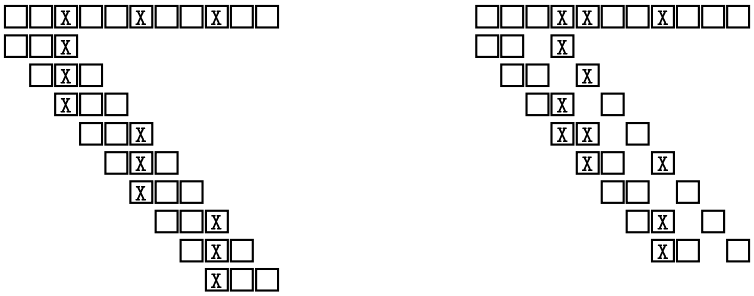
For any integer i and shape Q, the positioned shape Q_i is the set $\{i + j \mid j \in Q\}$. Let $Q_i = \{i_1, i_2, \ldots, i_q\}$, where $i = i_1 < i_2 < \cdots < i_q$, and let $S = s_1 s_2 \ldots s_m$ be a string. For $1 \leq i \leq m - s(Q) + 1$, the *$Q$-gram* at position i in S, denoted by $S[Q_i]$, is the string $s_{i_1} s_{i_2} \ldots s_{i_q}$. Two strings P and S have a *common Q-gram* at position i if $P[Q_i] = S[Q_i]$.

Example 1. Let $Q = \{0, 1, 3, 6\}$ be a shape. Using the notation from the introduction Q is the shape `##.#..#`. Its size $|Q| = 4$ and its span $s(Q) = 7$. The string $S =$ `ACGGATTAC` has three Q-grams: $S[Q_1] = s_1 s_2 s_4 s_7 =$ `ACGT`, $S[Q_2] =$ `CGAA` and $S[Q_3] =$ `GGTC`.

3 Threshold

The q-gram lemma does not apply to gapped q-grams in the form of Lemma 1. A straightforward generalization would give a threshold of $t = m - s(Q) - |Q|k + 1$ (see Lemma 3 below). Pevzner and Waterman [11] give this threshold for shapes of form $\{0, h, 2h, \ldots, (q-1)h\}$.[1] This threshold for gapped shapes is strictly worse than for contiguous shapes of the same size. However, the threshold is not tight for gapped q-grams as shown by the following example.

Example 2. Let $m = 11$ and $k = 3$ and consider the 3-shapes `###` and `##.#`. The above threshold for the two shapes are 0 and -1, respectively. Thus, neither shape would seem to be useful for filtering in this case. However, the real threshold for the shape `##.#` is 1. By full enumeration of all combinations of 3 mismatches it is possible to verify that at least one `##.#`-gram is always unaffected by the mismatches. The following figure gives an example of a worst possible combination of 3 mismatches for both shapes.



We will next define the tight threshold for arbitrary q-grams.

Let $P = p_1, \ldots p_m$ and $S = s_1 \ldots s_m$ be two strings of length m. Let $R(P,S)$ be the set of positions where P and S do not match, i.e., $R(P,S) = \{i \in \{1, \ldots, m\} \mid p_i \neq s_i\}$. Then $|R(P,S)|$ is the Hamming distance of P and S.

To determine the common Q-grams of P and S only the mismatch set $R(P,S)$ is needed: $P[Q_i] = S[Q_i]$ if and only if $Q_i \cap R(P,S) = \emptyset$. The minimum number of common q-grams is the threshold value needed for the q-gram method.

Definition 1. *Let m and k be non-negative integers and Q a shape. The threshold of Q for pattern length m and Hamming distance k is*

$$t(Q, m, k) = \min_{R \subseteq \{1, \ldots, m\}, |R| = k} |\{i \in \{1, \ldots, m - s(Q) + 1\} \mid Q_i \cap R\}|.$$

From the above discussion we get the following tight form of the q-gram lemma for arbitrary shapes.

Lemma 2 (The Q-gram lemma). *Let Q be a shape. For any two strings P and S of length m with Hamming distance k, the number of common Q-grams of P and S, i.e., the size of the set $\{i \in \{1, \ldots, m - s(Q) + 1\} \mid P[Q_i] = S[Q_i]\}$, is at least $t(Q, m, k)$. Furthermore, there exists two strings P and S of length m and Hamming distance k, for which the number of common Q-grams is exactly $t(Q, m, k)$.*

[1] They also give a better threshold for the case when the span of the shape is close to the length of the pattern.

We have not found a closed form for the exact threshold in the general case. The following lower bound we already saw in the beginning of this section.

Lemma 3. $t(Q,m,k) \geq \max\{0, m - s(Q) - |Q|k + 1\}$.

Proof. Let R be the set minimizing the expression in the definition of $t(Q,m,k)$. For each $j \in R$ there is exactly $|Q|$ integers i such that $j \in Q_i$. Therefore, at most $k|Q|$ of the positioned shapes Q_i, $i \in \{1, \dots, m - s(Q) + 1\}$, intersect with R, and at least $m - s(Q) - k|Q| + 1$ do not intersect with R. □

Tighter bounds may be given for special cases. In particular, the old q-gram lemma for contiguous q-grams gives indeed the exact threshold as shown by the following lemma.

Lemma 4. *Let Q be a contiguous shape, i.e., $Q = \{0, \dots, q-1\}$. Then* $t(Q,m,k) = \max\{0, m - s(Q) - |Q|k + 1\} = \max\{0, m - q(k+1) + 1\}$.

Proof. The lower bound is shown by Lemma 3. Let $R = \{q, 2q, \dots, kq\}$. Then Q_i intersects with R if and only if $i \in \{1, \dots, kq\}$, and thus does not intersect with R if $i \in \{kq+1, \dots, m-q+1\}$. This shows the upper bound. □

Using the lower bound of Lemma 3 as the threshold guarantees that all approximate occurrences are found, but it is very inefficient choice for gapped shapes. A difference of just one in the threshold value used makes a big difference in the efficiency of filtering. We have computed the exact thresholds for all shapes for $m = 50$ and $k \in \{4, 5\}$. Tables 1 and 2 give the highest threshold among (q,s)-shapes for all combinations of q and s that have shapes with positive thresholds (except $q = s = 1$).

The tables show that Example 2 was not an isolated case: in many cases, especially for higher values of q, best gapped shapes have much higher thresholds than contiguous shapes of the same or even smaller size. Thus, one can use a higher value of q to get fewer hits, or have a higher threshold and better filtration, or even both.

However, it is not sufficient just to have gaps; the shape has to be chosen carefully. For instance, for the parameters of Example 2, $m = 11$, $k = 3$, $q = 3$, the shape `##.#` and its mirror image `#.##` are the only ones that have a positive threshold. As a more impressive example, for the parameter values $m = 50$, $k = 5$ and $q = 12$, there are only two shapes, `###.#..###.#..###.#` and `#.#.#...#.....#.#.#...#.....#.#.#...#`, (and their mirror images) with a positive threshold. In most cases shown in Tables 1 and 2, only a few shapes achieve the highest threshold. The distribution of threshold values for one typical case is shown in Figure 1.

4 Minimum coverage

The filtering efficiency of a Q-gram clearly depends on the threshold $t(Q,m,k)$. However, the correlation is not direct. The following example shows an additional property of shapes that can affect the filtering efficiency.

Example 3. Let $m = 13$ and $k = 3$. Then both shapes `###` and `##.#` have a threshold of two. If two strings have four consecutive matching characters, they have two common 3-grams of shape `###`. In contrast, to have two common 3-grams of shape `##.#` two strings need to have at least 5 matching characters.

Motivated by the example, we define the following measure.

Definition 2. *Let Q be a shape and t a non-negative integer. The* minimum coverage *of Q for threshold t is*

$$c(Q,t) = \min_{C \subseteq \mathbb{N}, |C|=t} | \cup_{i \in C} Q_i |.$$

The minimum coverage is, in essence, the minimum number of characters that need to match between a pattern and a text substring for there to be t matching Q-grams. This gives a reasonable first order estimator for the probability of random strings having t common Q-grams. We do not analyse this further in this paper, but our experiments support the conjecture that there is a strong correlation between the minimum coverage $c(Q, t(Q, m, k))$ and the filter efficiency (see Figure 2 in Section 5).

We have computed the minimum coverages of all (q, s)-shapes for $m = 50$ and $k \in \{4, 5\}$. Tables 1 and 2 give the highest minimum coverage among (q, s)-shapes for all combinations of q and s that have shapes with positive thresholds (except $q = s = 1$). The tables show that considering the minimum coverage further improves the advantage of the best gapped shapes over the contiguous shapes. There are even cases where the contiguous shape is the q-shape with the highest threshold but some gapped q-shapes have a higher minimum coverage.

As with threshold values, the shapes with the highest minimum coverage are rare. Figure 1 shows the distribution of minimum coverages in one typical case.

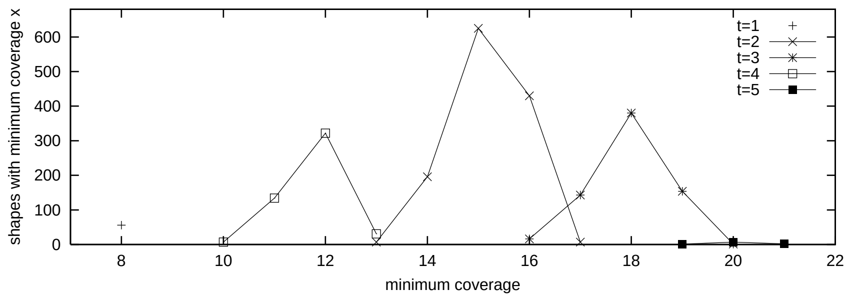


Fig. 1. Distribution of thresholds and minimum coverages of $(8, 17)$-grams for $m = 50$, $k = 5$. Mirror images and one shape with threshold 0 are not included.

Table 1. The best thresholds/minimum coverages for $m = 50$ and $k = 4$.

	q												
s	2	3	4	5	6	7	8	9	10	11	12	13	14
2	41/42												
3	40/41	36/38											
4	39/40	35/38	31/34										
5	38/39	34/38	30/34	26/30									
6	37/38	33/38	29/34	25/30	21/26								
7	36/37	32/38	28/34	24/30	20/26	16/22							
8	35/36	31/38	27/34	23/30	19/26	15/22	11/18						
9	34/35	30/38	26/34	22/30	18/26	14/22	10/18	6/14					
10	33/34	29/38	25/34	21/30	17/26	13/22	9/18	5/14	1/10				
11	32/33	28/36	24/34	20/30	17/27	14/24	10/20	7/17	4/14	0/0			
12	31/32	27/35	23/34	20/31	17/28	13/24	10/21	8/19	5/16	2/13	0/0		
13	30/31	26/34	22/34	19/31	16/28	13/25	10/22	8/20	6/18	3/15	1/12	0/0	
14	29/30	25/33	21/34	18/31	15/28	12/25	10/23	8/21	5/18	4/17	2/15	1/13	0/0
15	28/29	24/32	20/34	17/31	14/28	12/26	9/23	7/21	5/19	3/17	2/16	1/13	0/0
16	27/28	23/31	19/34	16/31	14/29	11/26	9/24	7/22	5/20	3/18	2/16	1/13	0/0
17	26/27	22/30	18/33	16/31[a]	13/29	11/27	9/25	7/23	5/21	4/20	2/17	1/13	0/0
18	25/26	21/28	17/32	15/31	12/29	10/27	8/25	6/23	5/22	3/20	2/18	1/13	0/0
19	24/25	20/27	17/30[a]	14/30	12/29	9/27	7/25	6/24	4/22	3/20	2/18	1/13	1/14
20	23/24	19/26	16/29[a]	13/30	11/29[a]	9/28	7/26	5/24	4/23	3/21	2/18	1/13	0/0
21	22/23	18/25	15/29	12/29	10/28	8/27	6/26	5/24	3/20	2/18	2/17	1/13	0/0
22	21/22	17/24	15/28[a]	12/29[a]	9/29	7/26	6/24	4/22	3/21	2/18	1/12	1/13	0/0
23	20/21	17/24	14/28	12/28[a]	9/28	7/26[a]	5/25	4/22	2/17	2/17	1/12	0/0	0/0
24	19/20	17/24	15/26	12/29[a]	10/27[a]	7/27	5/25	4/20[a]	2/17	1/11	1/12	0/0	0/0
25	20/21	17/24	15/28[a]	12/29[a]	10/29[a]	7/26	5/25	4/23	3/20	2/17	1/12	0/0	0/0
26	21/22	17/24	16/29[a]	12/30	11/31[a]	8/28	6/27	4/24	3/22	2/18	1/12	1/13	0/0
27	20/21	16/23	16/29[a]	12/30	12/30[a]	8/27	7/28[a]	5/26	4/24	3/21	2/18	1/13	0/0
28	19/20	15/21	15/29	11/30	11/30[a]	8/29	7/29[a]	6/25[a]	4/25	3/22	2/19	1/13	1/14
29	18/19	14/20	14/28	10/28	10/30	8/27[a]	7/27[a]	5/27	4/25	3/22	2/19	1/13	1/14
30	17/18	13/19	13/26	9/25	9/30	7/27	6/27	4/25	4/24	3/23	2/19	1/13	1/14
31	16/17	12/18	12/25	9/24	9/28[a]	6/27	6/27[a]	4/25	4/24	2/19	2/19	1/13	1/14
32	15/16	11/17	11/23	9/24[a]	7/25	5/25	5/25	3/22	3/22	2/18	2/18	1/13	1/14
33	14/15	12/17[a]	10/20	8/24	8/24[a]	5/25	4/24	3/22	2/18	2/18	1/12	1/13	0/0
34	13/14	12/18	9/19	8/24	8/24[a]	5/24	4/23	4/21[a]	2/18	2/17	1/12	1/13	0/0
35	12/13	12/18	8/18	8/24	8/25[a]	4/22	4/23	4/21	2/17	2/17	2/16	1/13	1/14
36	11/12	11/17	7/16	7/22	7/24	4/22	4/21	4/21	2/17	2/17	2/16	1/13	1/14
37	10/11	10/15	8/16[a]	6/19	6/21	4/22	4/21	3/18	2/16	2/17	2/16	0/0	0/0
38	9/10	9/14	8/16[a]	5/15	5/20	4/22	4/19[a]	3/18	2/17	2/14	1/12	0/0	0/0
39	8/9	8/13	8/16[a]	5/14[a]	4/18	4/21	4/19[a]	2/16	2/17	2/14	1/12	0/0	0/0
40	7/8	7/12	7/16	5/14	3/15	3/18	3/19	2/15	2/12	1/11	1/12	0/0	0/0
41	6/7	6/10	6/15	6/14[a]	4/14[a]	2/13	2/14	2/14	1/10	1/11	1/12	1/13	0/0
42	5/6	5/9	5/13	5/14	4/14[a]	2/12	2/13	2/14	1/10	0/0	0/0	0/0	0/0
43	4/5	4/8	4/10	4/14	4/14	3/12[a]	2/9	1/9	1/10	0/0	0/0	0/0	0/0
44	3/4	3/6	3/9	3/12	3/14	3/12[a]	2/10	1/9	1/10	0/0	0/0	0/0	0/0
45	2/3	2/5	2/7	2/9	2/11	2/12	2/11	1/9	0/0	0/0	0/0	0/0	0/0
46	1/2	1/3	1/4	1/5	1/6	1/7	1/8	1/9	1/10	0/0	0/0	0/0	0/0

[a] The shapes with the highest threshold and the highest coverage are different.

Table 2. The best thresholds/minimum coverages for $m = 50$ and $k = 5$.

	q										
s	2	3	4	5	6	7	8	9	10	11	12
2	39/40										
3	38/39	33/35									
4	37/38	32/35	27/30								
5	36/37	31/35	26/30	21/25							
6	35/36	30/35	25/30	20/25	15/20						
7	34/35	29/35	24/30	19/25	14/20	9/15					
8	33/34	28/35	23/30	18/25	13/20	8/15	3/10				
9	32/33	27/35	22/30	18/26	14/22	9/17	5/13	0/0			
10	31/32	26/34	21/30	18/27	13/22	10/19	6/15	3/12	0/0		
11	30/31	25/33	20/30	16/26	13/23	10/20	7/17	4/14	2/12	0/0	
12	29/30	24/32	19/30	16/27	12/23	9/20	7/18	4/15	2/13	0/0	0/0
13	28/29	23/31	19/30[a]	15/27	12/24	9/21	6/18	4/16	2/14	1/11	0/0
14	27/28	22/30	17/30	14/27	11/24	8/21	6/19	4/17	2/14	1/11	0/0
15	26/27	21/28	17/29[a]	13/27	10/24	8/22	5/19	3/17	2/15	1/11	0/0
16	25/26	20/27	16/29[a]	13/27	10/24	7/22	5/20	3/17	2/15	1/11	0/0
17	24/25	19/26	15/28	12/27	9/25	7/23	5/21	3/18	2/15	1/11	0/0
18	23/24	18/25	14/27	11/27	8/25	6/23	4/20	3/18	2/16	1/11	0/0
19	22/23	17/24	14/26[a]	11/26[a]	8/25	6/22	4/20	2/15	1/10	1/11	1/12
20	21/22	16/23	13/25[a]	10/24	7/24	5/22	3/18	2/15	1/10	1/11	0/0
21	20/21	15/21	12/25	9/23	7/22	5/21	3/18	2/15	1/10	0/0	0/0
22	19/20	15/21	12/23[a]	9/23	6/21	4/20	2/14	1/9	1/10	0/0	0/0
23	18/19	15/21	12/23[a]	9/23	6/21	4/19	2/14	1/9	0/0	0/0	0/0
24	18/19	15/21	13/24	9/23	7/20[a]	4/19	2/14	1/9	0/0	0/0	0/0
25	19/20	15/21	13/25[a]	9/23[a]	7/22	4/20	3/16	1/9	1/10	0/0	0/0
26	20/21	15/21	14/26[a]	9/24	8/24	5/23	3/19	2/15	1/10	0/0	0/0
27	19/20	14/20	14/26[a]	9/24	9/24[a]	6/23[a]	4/19	2/15	1/10	1/11	0/0
28	18/19	13/19	13/26	8/24	8/24	5/22	4/20	3/17	2/15	1/11	0/0
29	17/18	12/18	12/25	8/23	8/24[a]	5/22	5/22	2/15	2/15	1/11	0/0
30	16/17	11/17	11/23	7/21	6/21	4/20	4/20	2/15	2/15	1/11	0/0
31	15/16	10/15	10/20	7/21	7/20[a]	4/18	4/18[a]	2/14	2/14	1/11	0/0
32	14/15	10/15	9/19	7/19[a]	5/19	3/17	3/16	1/9	1/10	0/0	0/0
33	13/14	10/15[a]	8/18	6/19	6/19[a]	3/17	2/14	1/9	1/10	1/11	0/0
34	12/13	11/17	7/16	6/19	6/20[a]	3/17	3/14	2/12	1/10	1/11	0/0
35	11/12	11/17	6/15	6/19	6/20[a]	3/16	3/14	2/14	1/10	1/11	0/0
36	10/11	10/15	6/13[a]	5/15	5/20	2/13	2/13	2/14	1/10	0/0	0/0
37	9/10	9/14	7/15[a]	4/14	4/18	3/13[a]	2/13	2/13	1/10	1/11	1/12
38	8/9	8/13	7/15[a]	4/12[a]	3/15	2/13	2/12	1/9	1/10	1/11	0/0
39	7/8	7/12	7/15[a]	4/12	3/12	2/12	2/12	1/9	1/10	1/11	0/0
40	6/7	6/10	6/15	4/13	2/10	2/11	1/8	1/9	1/10	0/0	0/0
41	5/6	5/9	5/13	5/13[a]	3/10[a]	1/7	1/8	1/9	0/0	0/0	0/0
42	4/5	4/8	4/10	4/13	3/10	1/7	1/8	1/9	0/0	0/0	0/0
43	3/4	3/6	3/9	3/12	3/12	2/8	1/8	0/0	0/0	0/0	0/0
44	2/3	2/5	2/7	2/9	2/10	2/9	1/8	0/0	0/0	0/0	0/0
45	1/2	1/3	1/4	1/5	1/6	1/7	1/8	0/0	0/0	0/0	0/0

[a] The shapes with the highest threshold and the highest coverage are different.

5 Experiments

To test the q-grams in practice, we performed some experiments on DNA data. We used two different databases of size 50 million, one randomly generated (with even and independent distribution of characters), the other containing the first 50 million basepairs of the GenBank Mouse EST database. The queries we used were random strings of length 500. However, the threshold used in filtering was computed for $m = 50$. The effect is that the filtering is guaranteed to report all positions where there is an approximate occurrence of a substring of length 50 of the query.[2] The distance k varied between 3 and 6. The experimental setting corresponds to the high similarity local alignment problems in shotgun sequencing [18] and EST clustering [7, 2]. No actual matches were found in the databases, i.e., all potential matches reported by the filtering were false positives.

For a filter algorithm the two main properties of interest are speed and filtering efficiency. As a measure of these properties we use the number of *hits* and the number of *matches*, respectively. A hit is a pair (i, j) such that the query q-gram at position i matches the database q-gram at position j. The time to process the hits usually dominates the running time of the filtering phase. A hit (i, j) is counted for the position $j - i$. A match is a position that has at least t hits. The number of matches reflects the amount of work that the verification phase must do.

The expected number of hits is proportional to 4^{-q}. Our conjecture is that there is a similar dependence between the number of matches and the minimum coverage of the shape. We tested a large number of shapes using different values of k and the two databases described above. In figure 2 the relation between the minimum coverage and the number of matches per billion characters are shown. In most cases, the number of matches was computed from an average over 100 queries, although for some of the shapes, 1000 queries was used. It is clear that there is a strong but not stringent correlation of the form we expected.

For a more detailed comparison of shapes, we chose four classes of shapes for different values of q and k:

- *Best.* This is the shape with the highest minimum coverage. To choose between multiple shapes with the same coverage we used the number of distinct covers of the minimum size, the number of distinct covers of the minimum size plus one, and the threshold as secondary keys (in this order).
- *Median.* For each span s, all (q, s)-shapes (without mirror images) were ordered by the minimum coverage, and by the same secondary keys as for the best shape, and the median shape in this order was identified. Of these shapes (one for each s) the best one was used in the experiments. The chance that a randomly chosen shape is better than this shape is at most half.

[2] A better filter efficiency could be achieved by counting the hits separately for each substring of length 50, but we chose the simpler approach. This should not have a significant effect on the relative performance of different shapes. However, looking at the results of the experiments one should keep in mind that we do not achieve maximal filter efficiency.

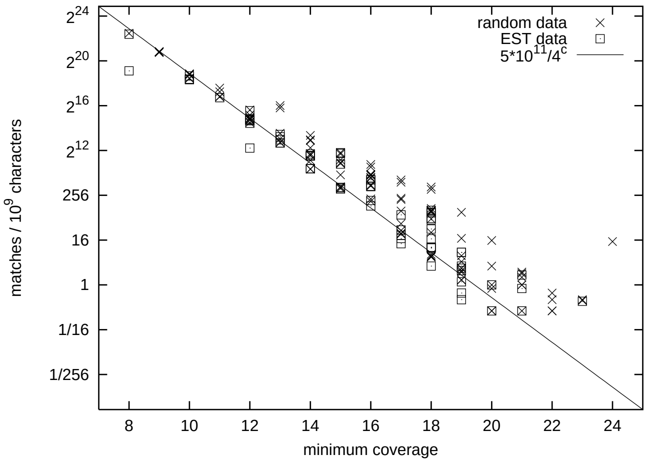


Fig. 2. Correlation of minimum coverage and filter efficiency.

– *1-gap.* The best shape with exactly one gap.
– *Contiguous.*

The chosen shapes except the contiguous ones are shown in Table 3. A missing shape means that the shape in that category had a threshold of 0. Figure 3 compares the chosen shapes both in theory (q vs. minimum coverage) and in practice (hits vs. matches). The experimental results are the averages from 1000 queries against the random database. The missing datapoints in the experimental graph either had a threshold of 0 (bottom of the graph) or had no matches at all (top of the graph).

There are several things of interest in Figure 3. First, there is a high correlation between theoretical and experimental behavior. Second, the performance of the median shapes shows that while a randomly chosen shape is likely to be better than a contiguous one, still much better results can be achieved by a careful choice of the shape. Third, the shapes with one gap, which are of particular interest for the k differences problem, are not as good as the best ones overall but still much better than contiguous ones. Finally, the best shapes have several orders of magnitude better performance than the contiguous shapes.

6 Concluding remarks

We have shown that suitably chosen gapped q-grams can significantly improve the performance of the basic q-gram filtering for the k mismatches problem. While interesting in itself, it also opens the door to the possibilities of gapped q-grams in the numerous other algorithms and applications, where contiguous q-grams and related methods have been found to be useful. In fact, most filtering

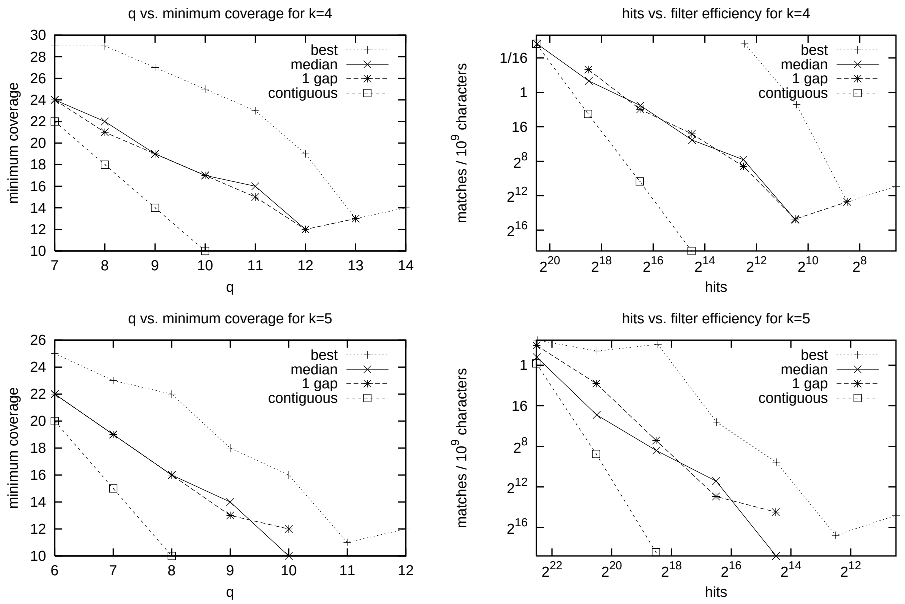


Fig. 3. Comparison of four classes of shapes.

$k=4$			
q	best	median	1 gap
7	##..#........#.........##..#	###.#.....#..#..#	######.....#
8	##..#......#.......#....##.#	#.#..#.#..##...#.#	######....##
9	##..#.#........#......##..#.#	####.#.#......###	#######....##
10	##..#.#.....#......#..##..#.#	###.###..#.....###	#######...###
11	###.#..#......#......#.#.#..##	######.###.#.#	########..###
12	##.#.#...##.........#.##.#...##	##.##.####.###.#	#########..###
13	#########.####		#########.####
14	####.##...####.##		
$k=5$			
q	best	median	1 gap
6	##......#..#..#.#	#.###.....#.#	#####....#
7	#.##......##..#.#	##..#..#..##.#	#####...##
8	##..#.#...............##..#.#	#.#..####...#.#	######...##
9	###..#..#.#...#.##	######..#.#.#	#######...##
10	###..#..#.#..###.#	##.##..#.#.###.#	#######.###
11	#######.##.##		
12	###.#..###.#..###.#		

Table 3. The shapes used in Figure 3.

methods for string matching use short substrings one way or another. We are currently working on extending the use of gapped q-grams to the k differences problem.

There may be applications even beyond string matching. For example, in DNA sequencing by hybridization (SBH) the problem is, in essence, to construct a string given its q-grams. Preparata et al.[12, 13] have recently shown that SBH can be significantly improved by using gapped probes (q-grams) instead of contiguous ones.

Acknowledgements

We thank Kurt Melhorn for useful remarks.

References

1. R. Baeza-Yates and G. Gonnet. All-against-all sequence matching. Technical report, Dept. of Computer Science, University of Chile, 1990.
2. S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivals, and M. Vingron. q-gram based database searching using a suffix array. In S. Istrail, P. Pevzner, and M. Waterman, editors, *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology (RECOMB-99)*, pages 77–83, Lyon, France, 1999. ACM Press.
3. A. Califano and I. Rigoutsos. FLASH: A fast look-up algorithm for string homology. In L. Hunter, D. Searls, and J. Shavlik, editors, *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology*, pages 56–64, Bethesda, MD, 1993. AAAI Press.
4. A. L. Cobbs. Fast approximate matching using suffix trees. In Z. Galil and E. Ukkonen, editors, *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching*, number 937 in Lecture Notes in Computer Science, pages 41–54, Espoo, Finland, 1995. Springer-Verlag, Berlin.
5. N. Holsti and E. Sutinen. Approximate string matching using q-gram places. In *Proceedings of the 7th Finnish Symposium on Computer Science*, pages 23–32, 1994.
6. P. Jokinen and E. Ukkonen. Two algorithms for approximate string matching in static texts. In A. Tarlecki, editor, *Proceedings of the 16th Symposium on Mathematical Foundations of Computer Science*, number 520 in Lecture Notes in Computer Science, pages 240–248, Kazimierz Dolny, Poland, 1991. Springer-Verlag, Berlin.
7. A. Krause and M. Vingron. A set-theoretic approach to database searching and clustering. *Bioinformatics*, 14:430–438, 1998.
8. O. Lehtinen, E. Sutinen, and J. Tarhio. Experiments on block indexing. In R. Baeza-Yates N. Ziviani and K. Guimarães, editors, *Proceedings of the 3rd South American Workshop on String Processing (WSP'96)*, pages 183–193, Recife, Brazil, 1996. Carleton University Press.
9. G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. Assoc. Comput. Mach.*, 46(3):395–415, 1999.
10. G. Navarro. *Approximate Text Searching.* PhD thesis, Dept. of Computer Science, University of Chile, 1998.

11. P. A. Pevzner and M. S. Waterman. Multiple filtration and approximate pattern matching. *Algorithmica*, 13(1/2):135–154, 1995.
12. F. P. Preparata, A. M. Fieze, and E. Upfal. On the power of universal bases in sequencing by hybridization. In *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology (RECOMB-99)*, pages 295–301, Lyon, France, 1999. ACM Press.
13. F. P. Preparata and E. Upfal. Sequencing-by-hybridization at the information-theory bound: An optimal algorithm. In R. Shamir, S. Miyano, S. Istrail, P. Pevzner, and M. Waterman, editors, *Proceedings of the 4th Annual International Conference on Computational Molecular Biology (RECOMB-00)*, pages 245–253, Tokio, 2000. ACM Press.
14. E. Sutinen and J. Tarhio. On using q-gram locations in approximate string matching. In P. G. Spirakis, editor, *Proceedings of the 3rd Annual European Symposium on Algorithms*, number 979 in Lecture Notes in Computer Science, pages 327–340, Corfu, Greece, 1995. Springer-Verlag, Berlin.
15. T. Takaoka. Approximate pattern matching with samples. In Ding-Zhu Du and Xiang sun Zhang, editors, *Proceedings of the 5th International Symposium on Algorithms and Computation*, number 834 in Lecture Notes in Computer Science, pages 236–242, Beijing, P.R. China, 1994. Springer-Verlag, Berlin.
16. E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–212, 1992.
17. E. Ukkonen. Approximate string matching over suffix trees. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*, number 684 in Lecture Notes in Computer Science, pages 228–242, Padova, Italy, 1993. Springer-Verlag, Berlin.
18. J. Weber and H. Myers. Human whole genome shotgun sequencing. *Genome Research*, 7:401–409, 1997.