Alcom-FT Technical Report Series
ALCOMFT-TR-02-43

Computing the Threshold for q-Gram Filters

Juha Kärkkäinen*

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
juha@mpi-sb.mpg.de

Abstract. A popular and much studied class of filters for approximate string matching is based on finding common q-grams, substrings of length q, between the pattern and the text. A variation of the basic idea uses *gapped* q-grams and has been recently shown to provide significant improvements in practice. A major difficulty with gapped q-gram filters is the computation of the so-called *threshold* which defines the filter criterium. We describe the first general method for computing the threshold for q-gram filters. The method is based on a carefully chosen precise statement of the problem which is then transformed into a constrained shortest path problem. In its generic form the method leaves certain parts open but is applicable to a large variety of q-gram filters and may be extensible even to other classes of filters. We also give a full algorithm for a specific subclass. For this subclass, the algorithm has been implemented and used succesfully in an experimental comparison.

1 Introduction

Given a *pattern* string P and a *text* string T, the *approximate string matching problem* is to find all substrings of the text (*matches*) that are within a *distance* k of the pattern P. The most commonly used distance measure is the *Levenshtein distance*, the minimum number of single character insertions, deletions and replacements needed to change one string into the other. A simpler variant is the *Hamming distance*, that does not allow insertions and deletions, i.e., it is the number of nonmatching characters for strings of the same length. The *indexed* version of the problem allows preprocessing the text to build an index while the *online* version does not. Surveys are given in [15, 16, 18].

Filtering is a way to speed up approximate string matching, particularly in the indexed case but also in the online case. A *filter* is an algorithm that quickly discards large parts of the text based on some *filter criterium*, leaving the remaining part to be checked with a proper (online) approximate string matching algorithm. A filter is *lossless* if it never discards an actual occurrence; we consider only lossless filters. The ability of a filter to reduce the text area is called its (*filtration*) *efficiency*.

* Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

Many filters are based on *q-grams*, substrings of length q. The q-gram similarity (defined as a distance in [25]) of two strings is the number of q-grams shared by the strings. The q-gram filter is based on the *q-gram lemma*:

Lemma 1 ([12]). *Let P and S be strings with (Levenshtein or Hamming) distance k. Then the q-gram similarity of P and S is at least $t = |P| - q + 1 - kq$.*

The value t in the lemma is called the *threshold* and gives the minimum number of q-grams that an approximate match must share with the pattern, which is used as the filter criterium. The method is well-suited for indexed matching using an index of text q-grams.

Above we did not define precisely how to count the number of shared q-grams. There are, in fact, many alternatives giving different tradeoffs between filtration efficiency, filter speed and index size. Here are some variations:

- If the same q-gram occurs r_P times in P and r_S times in S, it would be correct to count it as $\min\{r_P, r_S\}$, r_P, r_S, or $r_P r_S$ shared q-grams.
- As noted in [11], a q-gram need to be counted only if it occurs at approximately the same position in P and S.
- Count the shared q-grams between the pattern and large text areas, buckets. Buckets with less than t shared q-grams can be discarded as a whole [12, 4].

For all these variations of counting, the threshold t defining the filter criterium is the one given by Lemma 1; using a higher threshold would make the filter lossy, using a lower threshold would reduce filtration efficiency. This is a reflection of the fact that they are all *upper bound approximations* of the same *core similarity measure*. We define this core similarity measure in Section 2.

There are many ways to generalize the basic method, including the following:

Gapped q-grams The q-grams may contain gaps. For example, the gapped 3-grams of *shape* `##-#` of the string `acgtc` are `ac-t` and `cg-c`. In [5, 6], it is shown that by using q-grams of a carefully chosen shape, the filtration efficiency can be improved significantly. The added complexity makes online filters slower, but on indexed filters the effect is negligible.

Sampling A popular way to reduce time requirement and/or index size at the cost of filtration efficiency is to consider only a sample, say every 5th, of the q-grams of the text or the pattern [8, 24, 14, 22, 23, 21, 17, 19].

Multiple shapes As an opposite to sampling, the number of q-grams can be increased by using gapped q-grams of several different shapes [7, 20]. This improves filtration efficiency but increases time and/or space requirements.

Approximate q-grams Another way to improve filtration efficiency at the cost of slower filtering is to allow errors in q-grams [14, 8, 22, 19].

Of course, various combinations of these methods are possible. For example, sampling and approximate q-grams have often been used together [14, 8, 22, 19]. Gapped q-grams, in particular, offer a lot of possibilities for combination through the use of (possibly multiple) different shapes. The recent results in [5, 6] suggest that the possibilities of gapped q-grams are worth exploring. However, the problem is the difficulty of determining the threshold.

All the filtering methods mentioned above can be formulated using a similarity measure based on counting shared q-grams. A text substring is checked only if the similarity between the pattern and the substring is at least a given *threshold*. Most of the methods mentioned above give a simple equation for the threshold. However, when gapped q-grams are involved, things get more complicated. Even for the simple filter in [5] (Hamming distance, single shape, no sampling or approximate q-grams) no simple equation can be given; the threshold was computed separately for each shape with a dynamic programming algorithm. Pevzner and Waterman [20], too, consider only the Hamming distance and give an equation for a very limited class of regular shapes, and even that is not the optimal value but a lower bound (which guarantees losslessness but at a reduced efficiency). Califano and Rigoutsos [7] use a heuristically chosen threshold supported by probabilistic calculations and experiments; their filter is lossy.

In this paper, we consider the problem of computing the threshold. We give a formal definition of the value of the threshold that captures the essence of the concept and describe an algorithm for computing it. The definition and the algorithm are generic, leaving certain parts open, but being applicable to a large variety of q-gram filters. Filling in the missing pieces for a given class of filters is a non-trivial task, but it is simpler and much more precisely defined than trying to define and compute the threshold from scratch. As a concrete example, we also give the missing pieces for the class of filters considered in [6]. For these filters, the algorithm has been implemented.

The outline of the method (and the paper) is as follows. In section 2, we give a simple, precise statement of the threshold computation problem for *core similarity measures* of a specific form. The threshold problem is then transformed into a *constraint shortest path problem* (defined in Section 4) on a graph that depends on the core similarity (as described in Section 3). When applying the method to a specific filter, two things must be specified. First, a core similarity measure of the specified form must be given (Section 2). The similarity measure used by the filter should be an upper bound approximation of the core similarity. Second, an algorithm for building the above mentioned graph for the chosen core similarity must be given (Section 3). On the other hand, the constrained shortest path algorithm (Section 4) can be used for any filter.

2 Threshold

As in the classic q-gram lemma, we define the threshold of a q-gram filter as a function of the length m of the pattern and the distance limit k. That is, the threshold $t(m, k)$ is the smallest number of matching q-grams between a pattern of length m and a substring of the text that is within distance k of the pattern. The number of matching q-grams is a similarity function for strings. Since we are looking for the minimum similarity, we can assume that there are no “accidentally” matching q-grams, i.e., q-grams match only if they are not affected (too much) by the edit operations. Therefore, the minimum is defined by the worst possible arrangement of the edit operations.

Following [10], we define an *edit transcript* as a string over the alphabet M(atch), R(eplace), I(nsert) and D(elete), describing a sequential character-by-character transformation of one string to another. For two strings P and S, let $\mathcal{T}(P,S)$ denote the set of all transcripts transforming P to S. For example, $\mathcal{T}(\texttt{actg}, \texttt{acct})$ contains MMRR, MMIMD, MIMMD, IRMMD, IDIMDID, etc.. For a transcript $\tau \in \mathcal{T}(P,S)$, the source length $slen(\tau)$ of τ is the length of P, i.e., the number of non-insertions in τ. The Levenshtein cost $c_L(\tau)$ is the number of non-matches. The Hamming cost $c_H(\tau)$ is infinite if τ contains insertions or deletions and the same as Levenshtein cost otherwise. The Levenshtein distance and Hamming distance of P and S are $d_L(P,S) = \min_{\tau \in \mathcal{T}(P,S)} c_L(\tau)$ and $d_H(P,S) = \min_{\tau \in \mathcal{T}(P,S)} c_H(\tau)$, respectively.

Here we defined distance measures for strings using cost functions for edit transcripts. Similarly, we define the q-gram similarity measures for strings using *profit functions* for edit transcripts. Then we can define the threshold as follows.

Definition 1. *The* threshold *for a cost function c and a profit function p is*

$$t_p^c(m,k) = \min_{\tau}\{p(\tau) \mid slen(\tau) = m, c(\tau) \le k\}.$$

The following lemma gives the filter criterium.

Lemma 2. *Let c be a cost function and p a profit function for edit transcripts. Define a distance d of two strings P and S as $d(P,S) = \min_{\tau \in \mathcal{T}(P,S)} c(\tau)$ and a similarity s as $s(P,S) = \max_{\tau \in \mathcal{T}(P,S)} p(\tau)$. Now, if $d(P,S) \le k$, then $s(P,S) \ge t_p^c(|P|,k)$.*

The lemma holds for any choice of cost c and profit p. The cost functions leading to the Hamming and Levenshtein distances were defined above. Below, we give examples of profit functions that define q-gram similarity measures.

Let I be a set of integers. The *span* of I is $span(I) = \max I - \min I + 1$, i.e., the size of the minimum contiguous interval containing I. The *position* of I is $\min I$, and the *shape* of I is the set $\{i - \min I \mid i \in I\}$. An integer set Q with position zero is called a *shape*. For any shape Q and integer i, let Q_i denote the set with shape Q and position i, i.e., $Q_i = \{i + j \mid j \in Q\}$. Let $Q_i = \{i_1, i_2, \dots, i_q\}$, where $i = i_1 < i_2 < \cdots < i_q$, and let $S = s_1 s_2 \dots s_m$ be a string. For $1 \le i \le m - span(Q) + 1$, the *$Q$-gram* at position i in S, denoted by $S[Q_i]$, is the string $s_{i_1} s_{i_2} \dots s_{i_q}$. For example, if $S = \texttt{acagagtct}$ and $Q = \{0,2,3,6\}$, then $S[Q_1] = S[Q_3] = \texttt{aagt}$ and $S[Q_2] = \texttt{cgac}$.

A *match alignment* M_τ of a transcript τ is the set of pairs of positions that are matched to each other. For example, $M_{\texttt{MIMRDMR}} = \{(1,1),(2,3),(5,5)\}$. For a set I of integers, let $M_\tau(I)$ be the set to which M_τ maps I, i.e., $M_\tau(I) = \{j \mid i \in I \text{ and } (i,j) \in M_\tau\}$. A *$Q$-hit* in a transcript τ is a pair (i,j) such that $M_\tau(Q_i) = Q_j$. Note that a Q-hit (i,j) in $\tau \in \mathcal{T}(P,S)$ implies $P[Q_i] = S[Q_j]$.

Now we are ready to define our first profit function. The *Q-profit* $p_Q(\tau)$ of a transcript τ is the number of its Q-hits, i.e., $p_Q(\tau) = |\{(i,j) \mid M_\tau(Q_i) = Q_j\}|$. Using p_Q as the profit function defines the *Q-similarity* of two strings P and S as $s_Q(P,S) = \max_{\tau \in \mathcal{T}(P,S)} p_Q(\tau)$. If Q is the contiguous shape $\{0,1,\dots,q-1\}$,

s_Q is the core similarity measure underlying the classic q-gram filter, and the threshold of Definition 1 agrees with the one in Lemma 1 for both Hamming and Levenshtein distance. For a gapped shape Q, s_Q is the core similarity measure for the Hamming distance filters described in [5].

As a more complicated example, let us define the profit functions for the Levenshtein distance filters in [6]. The filters use a basic shape with only one gap and two other shapes formed from the basic shape by increasing and decreasing the length of the gap by one. For example, with the basic shape `##-#` we would also use the shapes `##--#` and `###`. The filter compares the q-grams of all three shapes in the pattern to the q-grams of the basic shape in the text.

For any $b_1, g, b_2 > 0$, let (b_1, g, b_2) denote the *one-gap shape* $\{0, \ldots, b_1 - 1, b_1 + g, \ldots, b_1 + g + b_2 - 1\}$. For a one-gap shape $Q = (b_1, g, b_2)$, let $Q^{+1} = (b_1, g+1, b_2)$ and $Q^{-1} = (b_1, g-1, b_2)$ (or $Q^{-1} = \{0, \ldots, b_1 + b_2 - 1\}$ if $g = 1$). Then, a $Q \pm 1$-*hit* in a transcript τ is a pair (i, j) of integers such that $Q_j \in \{M_\tau(Q_i^{-1}), M_\tau(Q_i), M_\tau(Q_i^{+1})\}$. If $\tau \in \mathcal{T}(P, S)$, then a $Q \pm 1$-hit (i, j) in τ implies $S[Q_j] \in \{P[Q_i^{-1}], P[Q_i], P[Q_i^{+1}]\}$, which is the criterion for a shared q-gram for the filters in [6]. The $Q \pm 1$-*profit* of τ, denoted by $p_{Q\pm 1}(\tau)$, is the number of $Q \pm 1$-hits in τ, i.e., $p_{Q\pm 1}(\tau) = |\{(i, j) \mid Q_j \in \{M_\tau(Q_i^{-1}), M_\tau(Q_i), M_\tau(Q_i^{+1})\}\}|$. The $Q \pm 1$-*similarity* of two strings P and S is $s_{Q\pm 1}(P, S) = \max_{\tau \in \mathcal{T}(P,S)} p_{Q\pm 1}(\tau)$.

The similarities s_Q and $s_{Q\pm 1}$ are *core similarity measures*. Computing them for given strings is not straightforward. Instead filters use simpler *upper bound approximations* that may give a higher similarity value. Designing good upper bound approximations is nontrivial and beyond the scope of this paper.

3 Profit Automata

In the remainder of the paper, we describe a general technique for computing the threshold according to Definition 1. The technique is based on automata for computing the values $slen(\tau)$, $c(\tau)$, and $p(\tau)$ for any transcript τ. These automata have four transitions out of each state labeled with M, R, I and D. Each transition has an output value, which is the change in the target value caused by the transition. Thus, the target value is the sum of the output labels on the transition path corresponding to τ. Such automata are more generally known as *weighted finite automata* [3] or *string-to-weight transducers* [13]. Fig. 1 shows simple examples.

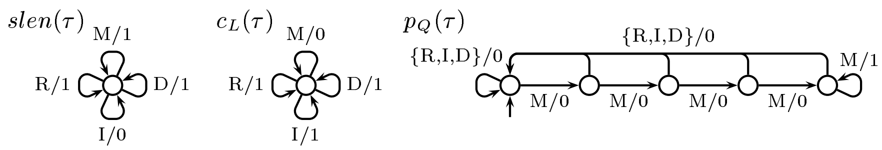

Fig. 1. Automata for computing the source length $slen(\tau)$, Levenshtein cost $c_L(\tau)$ and Q-profit $p_Q(\tau)$, where $Q = \{0, 1, 2, 3, 4\}$, for any transcript τ

For more complicated forms of profits the automata are also more complicated. We show how to build the automaton for the profit $p_{Q\pm 1}$ for any one-gap shape $Q = (b_1, g, b_2)$. Automata for other profits can be build using similar ideas.

We start by describing a dynamic programming algorithm for finding the $Q \pm 1$-hits for a given transcript $\tau = \tau_1 \ldots \tau_n$. Table 1 defines the computation of an entry $P[i, j]$ in a table $P[0..s, 0..n]$, where $s = b_1 + g + b_2$ is the span of Q. A column j in table P stores the state of the computation after reading the prefix $\tau_1 \cdots \tau_j$ of τ. Each entry represents a set of shape Q overlapping or touching the cut point between τ_j and τ_{j+1}. The value infinite means that the set cannot be a part of a $Q \pm 1$-hit. When the cut point is in the gap, a finite value represents the change in the length of the gap (insertion–deletion difference). An example is shown in Fig. 2. In the last row of the table P, each zero signifies a hit. Thus the profit can be computed as $p_{Q\pm 1}(\tau) = \big|\{j \in \{1, \ldots, n\} \mid P[s, j] = 0\}\big|$.

Table 1. Rules for computing the values in the dynamic programming table $P[0..s, 0..n]$

<table>
<tr><th></th><th>$j = 0$</th><th>$\tau_j = \mathrm{M}$</th><th>$\tau_j = \mathrm{R}$</th></tr>
<tr><td>$i = 0$</td><td colspan="3">0</td></tr>
<tr><td>$1 \le i \le b_1$</td><td rowspan="4">∞</td><td rowspan="2">$P[i-1, j-1]$</td><td>∞</td></tr>
<tr><td>$b_1 \le i \le b_1 + g$</td><td>$P[i-1, j-1]$</td></tr>
<tr><td>$i = b_1 + g + 1$</td><td>0 if $-1 \le P[i-1, j-1] \le 1$
∞ otherwise</td><td rowspan="2">∞</td></tr>
<tr><td>$b_1 + g + 1 < i \le s$</td><td>$P[i-1, j-1]$</td></tr>
</table>

<table>
<tr><th></th><th>$\tau_j = \mathrm{I}$</th><th>$\tau_j = \mathrm{D}$</th></tr>
<tr><td>$i = 0$</td><td colspan="2">0</td></tr>
<tr><td>$1 \le i < b_1$</td><td colspan="2">∞</td></tr>
<tr><td>$i = b_1$</td><td rowspan="2">$P[i-1, j-1] + 1$</td><td rowspan="2">$P[i, j-1] - 1$ if $P[i, j-1] + b_1 + g - i \ge 0$
∞ otherwise</td></tr>
<tr><td>$b_1 < i \le b_1 + g$</td></tr>
<tr><td>$b_1 + g < i \le s$</td><td colspan="2">∞</td></tr>
</table>

τ_j	τ_{j+1}			M	M	R	D	M	M	I	R	M	M	M	M	D	D	I	M	M	D	D	D	M
	## - #	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
#	# - #	1	∞	0	0	∞	∞	0	0	∞	∞	0	0	0	0	∞	∞	∞	0	0	∞	∞	∞	0
##	- #	2	∞	∞	0	∞	∞	∞	0	∞	∞	∞	0	0	0	-1	-2	∞	∞	0	-1	-2	∞	∞
## -	#	3	∞	∞	∞	0	-1	∞	∞	$+1$	∞	∞	∞	0	0	-1	∞	-1	∞	∞	∞	∞	∞	∞
## - #		4	∞	∞	∞	∞	∞	0	∞	∞	∞	∞	∞	∞	0	∞	∞	∞	0	∞	∞	∞	∞	∞

Fig. 2. The dynamic programming table P for $Q = (2, 1, 1) = \{0, 1, 3\}$

A column j in table P depends only on the previous column $j-1$ and the symbol τ_j. Thus the computation can be done with an automaton, where each state represents a distinct column. The first and last entry can be omitted from the state description: the first because it is always 0, the last because it does not affect the next column. Instead, the last entry determines the output value of the transition: 1 when last entry is 0, 0 when the last entry is ∞. The result is an automaton for computing the profit $p_{Q\pm1}(\tau)$ similar to the automatons in Fig. 1. An example is given in Fig. 3.

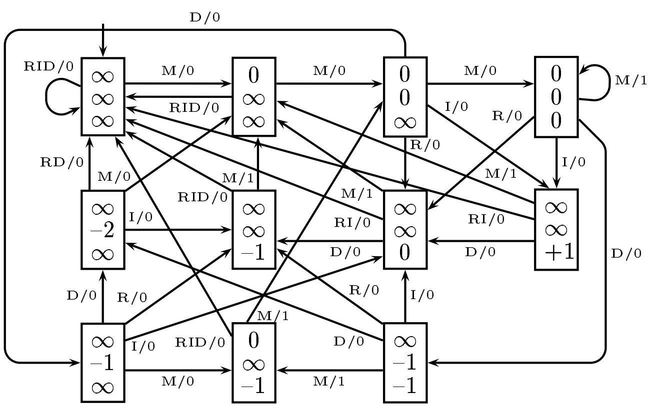

Fig. 3. An automaton for computing the profit $p_{Q\pm1}(\tau)$ for $Q = (2,1,1) = \{0,1,3\}$

The size of the automaton is bounded in the following lemma. The proof is omitted in this extended abstract.

Lemma 3. *The number of states in the profit automaton for the profit $p_{Q\pm1}$, where $Q = (b_1, g, b_2)$, is less than $R = b_1 b_2^2 (g+2)^{3\lceil g/b_1 \rceil}$. The automaton can be constructed in $\mathcal{O}(R(b_1 + g + b_2))$ time and $\mathcal{O}(Rg/b_1)$ space.*

4 Constrained Shortest Path Problem

We are now ready to describe the threshold computation problem as a constraint shortest path problem. The graph is formed by combining the three automata for computing $slen(\tau)$, $c(\tau)$ and $p(\tau)$. In general, the state set of the combined automaton is the Cartesian product of the state sets of the three automata, but since the automata for source length and Leveshtein or Hamming cost have just one state, the graph is essentially the profit automaton with additional labels.

Thus, we have a directed graph $G = (V, E)$, where each edge $e \in E$ is labeled with three non-negative values: $l(e)$, $c(e)$ and $p(e)$ corresponding to the (source) length, cost, and profit, respectively. The values are additive along paths. One node is designated as the source node s (the initial state). Each node has at most four outward edges.

Now, the treshold computation problem can be stated as follows:

Problem 1. Find the minimum profit $p(\pi)$ of a path π in G starting from the source node (and ending anywhere) that satisfies: (1) $c(\pi) \leq k$ and (2) $l(\pi) = m$.

The constraint (2) can be replaced with $l(\pi) \geq m$ without changing the solution.

This problem is very similar to the constrained shortest path (CSP) problem [26], differing in two ways from the standard form. First, no target node for the path is specified. However, by creating an additional node t as the target and adding an all-zero edge from every other node to t, the problem can be stated in the more usual form. Second, in the standard form all constraints are of type (1), i.e., limited from above. Limited-from-below constraints change the problem in a nontrivial way, e.g., the shortest path may contain cycles. However, many of the basic techniques used in CSP algorithms are still applicable.

The CSP problem is NP-hard, but there are pseudopolynomial algorithms, i.e., algorithms that work in polynomial time when the edge labels are polynomially bounded integers. We give a pseudopolynomial algorithm that belongs to a well-known class of shortest path algorithms called label-setting algorithms, which can be seen as generalizations of Dikstra's algorithm [1, Chapter 4]. For the CSP problem, label-setting algorithms have been given in [2, 9]. Our algorithm uses one nonstandard trick to deal with the limited-from-below constraint.

The basic idea of the algorithm is to maintain a collection of paths (initially only the empty path) in a priority queue PQ. The paths are selected from PQ in increasing order of their profit, extended along the four outward egdes, and the extensions are added to the PQ. Paths with cost more than k are removed. Since an extension cannot decrease the profit, the first path of required length selected from PQ is an optimal path.

To prune the set of paths, we use the following concept of domination:

Definition 2. *Let π_1 and π_2 be two paths from s to v. We say that π_1* dominates *π_2 if $p(\pi_1) \leq p(\pi_2)$, $c(\pi_1) \leq c(\pi_2)$ and $l(\pi_1) \geq l(\pi_2)$. If equality holds in each case, the paths are called* equivalent. *Otherwise, the domination is* strict.

Clearly, if π_1 dominates π_2, any extension of π_1 dominates the corresponding extension of π_2. Thus, no extensions of π_2 need to be considered. The algorithm extends a path only if it is not dominated by an already processed path and is not strictly dominated by any path to be processed later. To check for domination by already processed paths, information about the dominant paths is maintained at nodes. To avoid extending a path that is strictly dominated by a later path, the priority queue order is refined. The profit remains the primary key but secondary keys are used to break ties. The details are left to the full paper. The complexity of the algorithm is given by the following theorem.

Theorem 1. *Given the graph $G = (V, E)$, the algorithm computes the threshold t in $\mathcal{O}(k|V|\min\{m, t\} + kd_{max}t)$ time and $\mathcal{O}(k|V|p_{max})$ space, where m and k are the limits of length and cost, respectively, p_{max} is the largest profit value of an edge, and d_{max} is the length of the longest zero-cost, zero-profit path in G.*

By combining Theorem 1 with Lemma 3 we get the following result.

Theorem 2. *Given a one-gap shape $Q = (b_1, g, b_2)$ and positive integers m and k, the threshold $t^{c_L}_{p_{Q\pm 1}}(m,k)$ can be computed in $\mathcal{O}(kmb_1b_2^2g^{3\lceil g/b_1\rceil})$ time and $\mathcal{O}((k+g/b_1)b_1b_2^2g^{3\lceil g/b_1\rceil})$ space.*

5 Concluding Remarks

We have described a method for computing the threshold of a q-gram filter that is applicable to a large variety of filters. The practicality of the method is demonstrated by the implementation of the algorithm for the filters based on the $s_{Q\pm 1}$ similarity. It has been used in an experimental comparison of q-gram filters, the results of which are described in [6]. The shortest paths part of the implementation can be reused for other classes of filters without modification.

The method is an important step towards designing good q-gram filters. First, the threshold (or at least a good lower bound) is needed by any filter. A general method gives us a large family of filters to choose from. Second, the value of the threshold is an important criterium in comparing filters (particularly in choosing the shapes of q-grams [5, 6]). Third, the framework developed here may be helpful in designing filters. In particular, the separation of core similarity measures and their upper bound approximations seems a useful concept.

Acknowledgements

Discussions with Stefan Burkhardt, Mark Ziegelmann and Kurt Melhorn have contributed to this paper. The implementation is partly due to Stefan Burkhardt.

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows : Theory, Algorithms, and Applications.* Prentice Hall, 1993.
2. Y. P. Aneja, V. Aggarwal, and K. P. K. Nair. Shortest chain subject to side conditions. *Networks*, 13:295–302, 1983.
3. A. I. Buchsbaum, R. Giancarlo, and J. R. Westbrook. On the determinization of weighted finite automata. *SIAM J. Comput.*, 30(5):1502–1531, 2000.
4. S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivals, and M. Vingron. q-gram based database searching using a suffix array (QUASAR). In *Proc. 3rd Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 77–83. ACM Press, 1999.
5. S. Burkhardt and J. Kärkkäinen. Better filtering with gapped q-grams. In *Proc. 12th Annual Symposium on Combinatorial Pattern Matching*, volume 2089 of *LNCS*, pages 73–85. Springer, 2001.
6. S. Burkhardt and J. Kärkkäinen. One-gapped q-gram filters for Levenshtein distance. In *Proc. 13th Annual Symposium on Combinatorial Pattern Matching*, LNCS. Springer, 2002. To appear.

7. A. Califano and I. Rigoutsos. FLASH: A fast look-up algorithm for string homology. In *Proc. 1st International Conference on Intelligent Systems for Molecular Biology*, pages 56–64. AAAI Press, 1993.
8. W. I. Chang and T. G. Marr. Approximate string matching and local similarity. In *Proc. 5th Annual Symposium on Combinatorial Pattern Matching*, volume 807 of *LNCS*, pages 259–273. Springer, 1994.
9. L. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M. O. Ball et al., editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 2, pages 35–139. North-Holland, 1995.
10. D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
11. N. Holsti and E. Sutinen. Approximate string matching using q-gram places. In *Proc. 7th Finnish Symposium on Computer Science*, pages 23–32, 1994.
12. P. Jokinen and E. Ukkonen. Two algorithms for approximate string matching in static texts. In *Proc. 16th Symposium on Mathematical Foundations of Computer Science*, volume 520 of *LNCS*, pages 240–248. Springer, 1991.
13. M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:269–311, 1997.
14. E. W. Myers. A sublinear algorithm for approximate keyword searching. *Algorithmica*, 12(4/5):345–374, 1994.
15. G. Navarro. *Approximate Text Searching*. PhD thesis, Dept. of Computer Science, University of Chile, 1998.
16. G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.
17. G. Navarro and R. Baeza-Yates. A practical q-gram index for text retrieval allowing errors. *CLEI Electronic Journal*, 1(2), 1998. `http://www.clei.cl`.
18. G. Navarro, R. Baeza-Yates, E. Sutinen, and J. Tarhio. Indexing methods for approximate string matching. *IEEE Data Engineering Bulletin*, 24(4):19–27, 2001. Special issue on Managing Text Natively and in DBMSs.
19. G. Navarro, E. Sutinen, J. Tanninen, and J. Tarhio. Indexing text with approximate q-grams. In *Proc. 11th Annual Symposium on Combinatorial Pattern Matching*, volume 1848 of *LNCS*, pages 350–363. Springer, 2000.
20. P. A. Pevzner and M. S. Waterman. Multiple filtration and approximate pattern matching. *Algorithmica*, 13(1/2):135–154, 1995.
21. F. Shi. Fast approximate string matching with q-blocks sequences. In *Proc. 3rd South American Workshop on String Processing*, pages 257–271. Carleton University Press, 1996.
22. E. Sutinen and J. Tarhio. On using q-gram locations in approximate string matching. In *Proc. 3rd Annual European Symposium on Algorithms*, volume 979 of *LNCS*, pages 327–340. Springer, 1995.
23. E. Sutinen and J. Tarhio. Filtration with q-samples in approximate string matching. In *Proc. 7th Annual Symposium on Combinatorial Pattern Matching*, volume 1075 of *LNCS*, pages 50–63. Springer, 1996.
24. T. Takaoka. Approximate pattern matching with samples. In *Proc. 5th International Symposium on Algorithms and Computation (ISAAC)*, volume 834 of *LNCS*, pages 236–242. Springer, 1994.
25. E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–212, 1992.
26. M. Ziegelmann. *Constrained Shortest Paths and Related Problems*. PhD thesis, Universität des Saarlandes, Germany, 2001.