# Efficient $q$-Gram Filters for Finding All $\epsilon$-Matches Over a Given Length

Kim R. Rasmussen[1], Jens Stoye[2], and Eugene W. Myers[3]

[1] International NRW Graduate School in Bioinformatics and Genome Research,
Center of Biotechnology, Universität Bielefeld, 33594 Bielefeld, Germany
kim.rasmussen@cebitec.uni-bielefeld.de

[2] Technische Fakultät, Universität Bielefeld, 33594 Bielefeld, Germany
stoye@techfak.uni-bielefeld.de

[3] Div. of Computer Science, UC Berkeley, Berkeley, CA 94720-1776, USA
gene@eecs.berkeley.edu

**Abstract.** Fast and exact comparison of large genomic sequences remains a challenging task in biosequence analysis. We consider the problem of finding all $\epsilon$-matches between two sequences, i.e. all local alignments over a given length with an error rate of at most $\epsilon$. We study this problem theoretically, giving an efficient $q$-gram filter for solving it. Two applications of the filter are also discussed, in particular genomic sequence assembly and BLAST-like sequence comparison. Our results show that the method is 25 times faster than BLAST, while not being heuristic.

## 1 Introduction

Searching a biological sequence database for sequences similar to a given query sequence is a problem of fundamental importance in bioinformatics. Of particular interest is the case where sequences are considered similar if they have a local alignment that scores above a given threshold. The first algorithm to solve this problem is due to Smith and Waterman [19], whose names have become synonymous with the search. A Smith-Waterman search is guaranteed to find all local alignments scoring above a given threshold, and is therefore commonly referred to as a *full-sensitivity* search. Unfortunately, the algorithm has quadratic time complexity and spends most of its running time verifying that there are in fact *no* alignments of interest.

This inefficiency led to the development of *heuristics* such as the very popular FASTA [17] and BLAST [1,2] programs. The latter is based on the assumption that biologically interesting alignments must contain at least one pair of highly similar substrings, called a *seed*. Aided by a preprocessed query sequence, the BLAST algorithm efficiently locates and extends each seed to a local alignment containing the seed. However, it is important to note that the regions of the search space thusly disregarded, can actually contain a match.

In contrast, a *filter* is an algorithm that rapidly and stringently eliminates a large part of a search space from consideration. That is, unlike a heuristic, it guarantees not to eliminate a region containing a match. Full sensitivity can therefore be obtained by applying a full sensitivity algorithm on the unfiltered regions. Effective filtration under scoring measures typical for protein searches is very difficult [15]. It is somewhat easier for applications on DNA, when matches of high identity ($\geq 90\%$) are sought. The first filtration algorithms for this problem appeared in the early 90's and include the works of Ukkonen [20], Chang and Lawler [6] and Myers [13].

Typical for these algorithms is the *preprocessing* of the query sequence or the sequence database in order to accelerate searches. In particular, after building an *index*, such as a suffix tree or array (see e.g. [8]) or a simple list of locations of $q$-grams (i.e. strings of length $q$), it can be used for any number of searches. This is important for many applications, where the typically large sequence database remains unchanged over a high number of usually shorter queries. Myers [13] was the first to deliver an asymptotic improvement using such preprocessing of the sequence database. This setting – a filter using an index of the sequence database, aimed for applications on DNA – is the focus of this paper.

**Related Work.** The program QUASAR [3] is the closest precursor to the work presented in this paper and is itself a refinement of the earlier $q$-gram algorithm of Ukkonen [20]. It uses a suffix array to retrieve the positions of any given $q$-gram in the target sequence. At query time the target sequence is logically split into blocks. As a sliding window proceeds over the query sequence, the number of $q$-grams co-occurring in the window and each block is determined. Blocks with less than a certain threshold of $q$-grams in common with the query sequence are eliminated, and BLAST is run on the query sequence and the remaining blocks to find the reported alignments.

The heuristics SSAHA [16] and BLAT [10] use a $q$-gram index of the target sequence, containing however only the positions of the non-overlapping $q$-grams. This reduces the index size and the expected number of $q$-gram seeds by a factor of $q$, but at the cost of a loss in sensitivity. At query time the set of matching $q$-gram positions between the query and target sequences is collected and sorted by diagonal. A linear scan locates stretches of hits on identical diagonals. These stretches are then sorted by target sequence position. Another linear scan identifies contiguous stretches of matching positions in the target sequence. These stretches are extended in traditional BLAST-style to produce the final alignments. We note that the idea of diagonal sorting actually first appeared in FASTA [17].

The use of sample indexing is taken even further in FLASH [5]. Based on a probabilistic model, randomly chosen discontiguous patterns are indexed in multiple highly redundant indexes. Consequently, the index is very large, requiring approximately 18 Gb for a 100 Mb nucleotide database. However, it was shown that this approach yielded high sensitivity in practice. The use of discontiguous, or *gapped*, seed patterns has more recently been refined in PATTERNHUNTER [12] and [4], giving fast and sensitive heuristic searches. The gapped-seed idea is orthogonal to the filtration method we employ here and could conceivable be layered on, but we do not address it in this paper.

**Contributions.** In this work we consider the problem of detecting local alignments under the unit cost measure or Levenshtein distance. That is, the distance between two strings is the number of insertions, deletions, and substitutions in the alignment between them. An absolute threshold on the number of differences in a local alignment is inappropriate as the lengths of the aligned strings are unconstrained. Normalizing by dividing by the length of the aligned strings, we seek instead local alignments where the *error rate* is at most $\epsilon > 0$. In other words, we seek all $\epsilon$-*matches* for small $\epsilon$.

The main contribution of this paper is an efficient filter for identifying regions of the implied edit matrix that are guaranteed to overlap with possible $\epsilon$-matches. The filter is much more selective than QUASAR and Ukkonen's early work, while operating at comparable or superior speeds depending on parameter settings. Moreover, it finds all matches over a given length, a criterion not considered nor met by the earlier $q$-gram filters. It is thus a very effective, full-sensitivity filter for DNA searches.

The organization of the paper is as follows. In Section 2 we formalize the problem definition and present a filter criterion that identifies regions of the query and target sequences that may contain an $\epsilon$-match. Section 3 gives an efficient algorithm for realizing the filter. Section 4 describes several applications of the basic filter and Sect. 5 presents experimental results for these applications.

## 2 $q$-Gram Filters for $\epsilon$-Matches

### 2.1 Problem Definition

Given a string $A$ over a finite alphabet $\Sigma$, $|A|$ is the length of $A$, $A[i]$ refers to the $i$th character of $A$, and $A[i,j]$ is the substring of $A$ that starts with the $i$th character and ends with the $j$th. A substring of length $q > 0$ of $A$ is a *$q$-gram* of $A$. Let $\varepsilon$ denote the empty string. An *alignment* $L$ of strings $A$ and $B$ is a sequence $(\alpha_1 \to \beta_1, \ldots, \alpha_\ell \to \beta_\ell)$ of *edit operations* (i.e., insertions $\varepsilon \to \beta$, deletions $\alpha \to \varepsilon$, and substitutions $\alpha \to \beta$ of single character substrings) such that $A = \alpha_1 \ldots \alpha_\ell$ and $B = \beta_1 \ldots \beta_\ell$. We denote the number of edit operations $\alpha \to \beta$, $\alpha \neq \beta$ in an alignment $L$ by $\delta(L)$. The (*unit cost*) *edit distance* between $A$ and $B$ is then defined as $dist_\delta(A,B) := \min\{\delta(L) \,|\, L \text{ is an alignment of } A \text{ and } B\}$. It is well known that the edit distance can be calculated in quadratic time using dynamic programming. An $(|A|+1) \times (|B|+1)$ *edit matrix* $E_\delta$ is tabulated such that $E_\delta(i,j) := dist_\delta(A[1,i], B[1,j])$. Then, $E_\delta(|A|,|B|) = dist_\delta(A,B)$.

The problem we consider is that of finding $\epsilon$-matches. The normalized relative distance, or *error rate*, is defined as the edit distance divided by the length of the query substring involved in the local alignment. An $\epsilon$-*match* is then a local alignment with an error rate of at most $\epsilon$. More precisely, our problem is defined as follows. Given a *target* string $A$ and a *query* string $B$, a minimum match length $n_0$ and a maximum error rate $\epsilon > 0$, find all $\epsilon$-matches $(\alpha, \beta)$ where $\alpha$ and $\beta$ are substrings of $A$ and $B$, respectively, such that $|\beta| \geq n_0$ and $dist_\delta(\alpha,\beta) \leq \lfloor \epsilon|\beta| \rfloor$.

### 2.2 Filters

Our goal is to devise an efficient filter for identifying the regions between $A$ and $B$ that may contain an $\epsilon$-match. We derive our idea for the filter from the *q-gram method*, which is based on the observation that the substrings of an approximate match must have a certain number of $q$-grams in common [9]. Define a *q-hit* as a pair $(i, j)$ such that $A[i, i+q-1] = B[j, j+q-1]$. The basic $q$-gram method then works as follows. First, find all $q$-hits between the query and target strings. Second, identify regions between the strings that have 'enough' hits. Such candidate regions are subsequently subject to a closer examination.

We now show that all $q$-hits in an $\epsilon$-match occur in a well-defined region of the edit matrix. We first consider $\epsilon$-matches of length $n_0$ and then extend to $\epsilon$-matches of length $n_0$ or greater.

**Finding $\epsilon$-Matches of Length $n_0$.** Let an $n \times e$ *parallelogram* of the edit matrix be a set of entries on $n+1$ consecutive columns and $e+1$ consecutive diagonals. The *A-projection* $p_A$ of an $n \times e$ parallelogram is the substring of $A$ between the last row of the first column and the first row of the last column, implying $|p_A| = n - e$. Similarly, the *B-projection* $p_B$ of an $n \times e$ parallelogram is the substring of $B$ between the first and the last column of the parallelogram, with $|p_B| = n$. A $q$-hit $(i, j)$ between $A$ and $B$ corresponds to a sequence of $q+1$ consecutive entries along the diagonal $j - i$ of the edit matrix. We say that a $q$-hit is *contained* in a given parallelogram if its entries are a subset of those of the parallelogram. Figure 1 illustrates.

An $\epsilon$-match $(\alpha, \beta)$ of length $n$ between $A$ and $B$ with at most $e = \lfloor \epsilon n \rfloor$ differences relates to an $n \times e$ parallelogram by the following lemma.

**Lemma 1.** *Let $\alpha$ and $\beta$ be substrings of $A$ and $B$, respectively, s.t. $|\beta| = n$ and $dist_\delta(\alpha, \beta) \leq e$. Then, there exists an $n \times e$ parallelogram such that (a) it contains at least $T(n, q, e) := (n+1) - q(e+1)$ q-hits, (b) its B-projection is $\beta$, and (c) its A-projection is contained in $\alpha$.*

*Proof.* The proof is straightforward, c.f. Fig. 1. □

Hence, regions between $A$ and $B$ that can hold an $\epsilon$-match of length $n_0$ can be found as follows. First, count the number of $q$-hits in each $n_0 \times \lfloor \epsilon n_0 \rfloor$ parallelogram. Second, identify parallelograms that contain at least $T(n_0, q, \lfloor \epsilon n_0 \rfloor)$ $q$-hits. Then, for each such parallelogram there can be an $\epsilon$-match $(\alpha, \beta)$ where $\alpha$ is intersected by $p_A$ and $\beta = p_B$.

**Finding $\epsilon$-Matches of Length $n_0$ or Greater.** We now consider how to find all $\epsilon$-matches of length $n_0$ or greater. To solve this problem our idea is to look for the existence of a $w \times e$ parallelogram whose projections intersect $\alpha$ and $\beta$, respectively, and which we can guarantee to contain at least $\tau$ $q$-hits. For a given choice of the parameters $q$, $\epsilon$ and $n_0$, the following lemma guarantees the existence of such a parallelogram. Moreover, it provides its dimensions $w$ and $e$, and the $q$-hit threshold $\tau$.

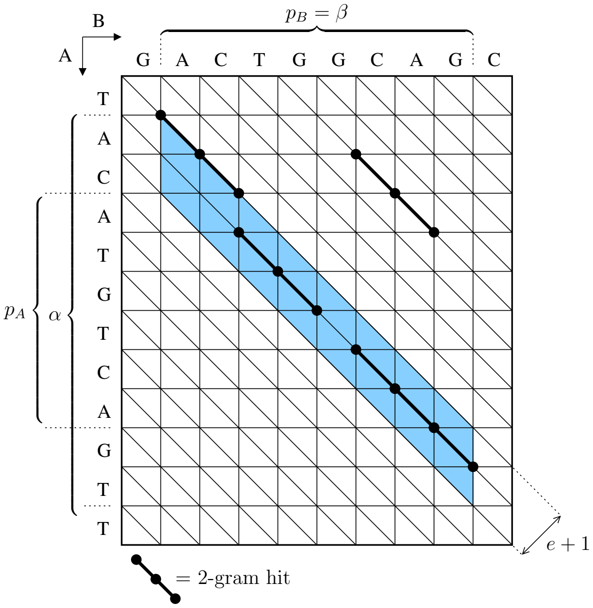

**Fig. 1.** An $8 \times 2$-parallelogram in the edit graph between the sequences $A =$ `TACATGTCAGTT` and $B =$ `GACTGGCAGC`. For $q = 2$, there are four $q$-hits within the parallelogram and $dist_\delta(\alpha, \beta) = 2$.

**Lemma 2.** *Let $\beta$ denote a substring of $B$ of length $n_0$ or greater that has an $\epsilon$-match to a substring $\alpha$ of $A$. Let $U(n, q, \epsilon) := (n+1) - q(\lfloor \epsilon n \rfloor + 1)$ and assume that the $q$-gram size $q$ and the threshold $\tau$ have been chosen such that*

$$q < \lceil 1/\epsilon \rceil \quad \textit{and} \quad \tau \leq min\{U(n_0, q, \epsilon), U(n_1, q, \epsilon)\}, \tag{1}$$

*where $n_1 = \lceil (\lfloor \epsilon n_0 \rfloor + 1)/\epsilon \rceil$. Then, there is guaranteed to exist a $w \times e$ parallelogram containing at least $\tau$ $q$-hits whose projections intersect $\alpha$ and $\beta$, where*

$$w = (\tau - 1) + q(e+1) \quad \textit{and} \quad e = \left\lfloor \frac{2(\tau - 1) + (q-1)}{1/\epsilon - q} \right\rfloor . \tag{2}$$

*Further, if $|\beta| \leq w$ then the $B$-projection of the $w \times e$ parallelogram contains $\beta$, otherwise it is a substring of $\beta$.*

*Proof.* The outline of the proof is as follows. First, we determine the lower bound $\tau$ on the number of $q$-hits contained in the $n \times \lfloor \epsilon n \rfloor$ parallelogram of an $\epsilon$-match of length $n \geq n_0$. Then, we argue that there exists a $w \times e$ parallelogram that contains at least $\tau$ $q$-hits. Finally, we determine the dimensions $w$ and $e$ of such a parallelogram over all values $n \geq n_0$.

Consider $n \geq n_0$ and suppose a substring of $B$ of this length has an $\epsilon$-match to a substring of $A$. Then, there are no more than $\lfloor \epsilon n \rfloor$ differences in the match and so by Lemma 1 there exists an $n \times \lfloor \epsilon n \rfloor$ parallelogram containing at least $T(n, q, \lfloor \epsilon n \rfloor) = U(n, q, \epsilon)$ $q$-hits. For fixed values of $q$ and $\epsilon$, $U(n, q, \epsilon)$ is a saw-toothed function of $n$ for which each 'tooth' has slope 1, dropping to a minimum at each point $\{\lceil i/\epsilon \rceil\}_i$. It is easy to confirm that as long as $q < \lceil 1/\epsilon \rceil$, then $U(\lceil i/\epsilon \rceil, q, \epsilon)$ is a strictly increasing function of $i$, i.e., each successive minimum of the saw-tooth is higher than the previous one. It thus follows that the smallest value of $U(n, q, \epsilon)$ over all values $n \geq n_0$ is either $U(n_0, q, \epsilon)$ or the value $U(n_1, q, \epsilon)$ at the next tooth value $n_1 = \lceil (\lfloor \epsilon n_0 \rfloor + 1)/\epsilon \rceil$, c.f. Fig. 2. Therefore, if one chooses $\tau \leq \min\{U(n_0, q, \epsilon), U(n_1, q, \epsilon)\}$ there will always be at least $\tau$ $q$-hits in an $n \times \lfloor \epsilon n \rfloor$ parallelogram for $n \geq n_0$.

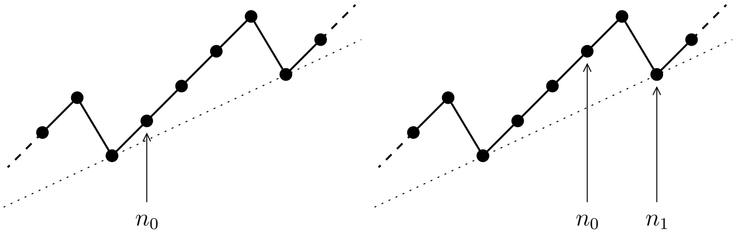

**Fig. 2.** $U(n, q, \epsilon)$ is a sawtoothed function of $n$. Its minimum over all values $n \geq n_0$ is either $U(n_0, q, \epsilon)$ or $U(n_1, q, \epsilon)$, shown left and right respectively, where $n_1$ is the next tooth value.

The $U(n, q, \epsilon) \geq \tau$ hits within an $n \times \lfloor \epsilon n \rfloor$ parallelogram are interspersed between up to $\lfloor \epsilon n \rfloor$ differences. For any $n$, the question is what is the largest $e$ such that for every consecutive sequence of $e$ differences there are less than $\tau$ $q$-hits in the $e$ spaces below them? Some thought reveals that a way to maximize $e$ is to cluster the $q$-hits into groups of $\tau - 1$ and to then spread these as far apart between the $\lfloor \epsilon n \rfloor$ differences, and in this case $e = \left\lfloor \frac{\lfloor \epsilon n \rfloor}{\lceil U(n,q,\epsilon)/(\tau-1) \rceil - 1} \right\rfloor$. Furthermore, it follows that one will have at least $\tau$ hits in a $w \times e$ parallelogram for $w = (\tau - 1) + q(e + 1)$.

It thus remains to find the largest value of $e$ (and its associated $w$) over all values of $n \geq n_0$. First observe that one need only consider the minimum points $\{\lceil i/\epsilon \rceil\}_i$ of the sawtooth as for any given tooth, $U(n, q, \epsilon)$ increases while $\lfloor \epsilon n \rfloor$ stays the same. Therefore we seek

$$e = \max_{i \geq \lceil \epsilon n_0 \rceil} \left\{ \left\lfloor \frac{i}{\lceil U(\lceil i/\epsilon \rceil, q, \epsilon)/(\tau - 1) \rceil - 1} \right\rfloor \right\} \quad . \tag{3}$$

By the definition of the ceiling function and the fact that $\lceil i/\epsilon \rceil$ is the only fractional part of the denominator it follows that this equals

$$e = \max_{i \geq \lceil \epsilon n_0 \rceil} \left\{ \left\lfloor \frac{i}{\lceil U(i/\epsilon, q, \epsilon)/(\tau - 1) \rceil - 1} \right\rfloor \right\} \quad . \tag{4}$$

For all values of $i$ for which $\lceil U(i/\epsilon, q, \epsilon)/(\tau - 1) \rceil$ is the same value, say $m$, the largest value of $i$ will give the largest value of $e$. But this value is

$$\max\{j : U(j/\epsilon, q, \epsilon) \leq m(\tau - 1)\} = \left\lfloor \frac{m(\tau-1) + (q-1)}{1/\epsilon - q} \right\rfloor . \tag{5}$$

Moreover, since $U(n, q, \epsilon) \geq \tau$ for all $n \geq n_0$ we are in effect considering all $m \geq 2$. Thus,

$$e = \max_{m \geq 2} \left\{ \left\lfloor \frac{\left\lfloor \frac{m(\tau-1)+(q-1)}{1/\epsilon - q} \right\rfloor}{m-1} \right\rfloor \right\} = \left\lfloor \max_{m \geq 2} \left\{ \frac{m(\tau-1) + (q-1)}{(m-1)(1/\epsilon - q)} \right\} \right\rfloor , \tag{6}$$

which is clearly decreasing in $m$ and therefore has its maximum at $m = 2$. □

**Feasible Values of $n_0$ Given $\tau$.** For a given choice of the parameters $\epsilon$ and $q$, there is a set of pairs $(n_0, \tau)$ such that for any $\epsilon$-match a $w \times e$ parallelogram exists that contains this match. In Lemma 2, we give the set of feasible $\tau$ for a given $n_0$. We now compute the set of feasible $n_0$ for a given choice of $\tau$.

**Corollary 1.** *If $n_0 \geq q \left\lceil \frac{\tau+q-1}{1/\epsilon-q} \right\rceil + \tau - 1$ then a $w \times e$ parallelogram, as defined by Lemma 2, exists.*

*Proof.* Consider a given choice of $\tau$, $q$, and $\epsilon$. We seek the value $n_0$ for which $U(n, q, \epsilon) \geq \tau$ for all $n \geq n_0$. First we find the smallest tooth point $n_1 = \lceil d_1/\epsilon \rceil$ whose value $U(n_1, q, \epsilon)$ is not less than $\tau$. That is we seek the minimum $d$ such that $\lceil d/\epsilon \rceil + 1 - q(d+1) \geq \tau$. Performing a bit of algebra, we get $d_1 = \min \left\{ d : d \geq \frac{\tau+q-1}{1/\epsilon-q} \right\} = \left\lceil \frac{\tau+q-1}{1/\epsilon-q} \right\rceil$. So, $n_0$ occurs in the previous tooth and satisfies $n_0 + 1 - q((d_1 - 1) + 1) = \tau$. Solving for $n_0$ gives the result. □

Table 1 illustrates the complex relationships between the parameters of the filter, as calculated by Lemma 2 and Corollary 1. Moreover, the values give an indication of the very good selectivity of the filter.

**Table 1.** Filter parameters for $\epsilon = 0.05$, by Lemma 2 and Corollary 1, respectively.

| | $q=7$ | | | $q=9$ | | | $q=11$ | | |
|---|---|---|---|---|---|---|---|---|---|
| $n_0$ | 30 | 50 | 100 | 30 | 50 | 100 | 30 | 50 | 100 |
| $w$ | 44 | 71 | 128 | 48 | 77 | 136 | 40 | 71 | 133 |
| $e$ | 3 | 5 | 9 | 3 | 5 | 9 | 2 | 4 | 8 |
| $\tau$ | 17 | 30 | 59 | 13 | 24 | 47 | 8 | 17 | 35 |

| | $q=11$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\tau$ | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $n_0$ | 28 | 29 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| $w$ | 39 | 40 | 52 | 53 | 54 | 55 | 67 | 68 | 69 |
| $e$ | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 |

## 3 An Efficient Algorithm

We now describe an efficient algorithm for finding all $w \times e$ parallelograms for $\epsilon$-matches of length $n_0$ or greater between the strings $A$ and $B$.

### 3.1 Preprocessing

In the preprocessing step we construct a $q$-gram index for the target sequence $A$. The index consists of two tables. The *occurrence table* is a concatenation of the lists $L(G) := \{i \mid A[i, i+q-1] = G\}$ for all $q$-grams $G \in \Sigma^q$ in $A$, and the *lookup table* is an array indexed by the natural integer encoding of $G$ to base $|\Sigma|$, giving the start of each list in the occurrence table.

### 3.2 Finding $w \times e$ Parallelograms

The $w \times e$ parallelograms that contain at least $\tau$ $q$-hits can be found trivially using a sliding window. The implied edit matrix is split into all overlapping *bins* of $e+1$ adjacent diagonals. At any time, each bin counts the number of $q$-hits contained in the $w \times e$ parallelogram defined by the intersection of the diagonals of the bin and the rows of the sliding window $W_j = B[j, j+w-1]$. As the sliding window proceeds to $W_{j+1}$, the bin counters are updated to reflect the changes caused by the $q$-grams leaving and entering the window. If a bin counter reaches $\tau$, the corresponding parallelogram is reported; overlapping parallelograms are trivially merged on the fly.

**Improving space requirements.** The number of bin counters is reduced by searching for $w \times (e + \Delta)$ parallelograms, where $\Delta > 0$. We associate each bin counter with $e + \Delta + 1$ adjacent diagonals and let successive bins overlap by $e$ diagonals. This is sufficient as the $\tau$ $q$-hits cannot be spread over more than $e+1$ diagonals. In total, only $\left\lceil \frac{|A|-e-\Delta}{\Delta+1} \right\rceil$ bin counters are required. A good choice for $\Delta$ is $2^z$, where $z \in \mathbb{N}$ and $2^z > e$. Bin indices are then calculated with fast bit-operations.

**Improving running time.** We reduce the considering of each $q$-hit from twice to once by use of two observations. First, two $q$-hits that are more than $w - q$ apart (counted as the difference between their starting positions in $B$) cannot both be in the same $w \times e$ parallelogram. Secondly, the $\tau$ $q$-hits in an $\epsilon$-match cannot occur in a string shorter than $q + \tau - 1$. Hence, we relax the search to finding $w' \times e$ parallelograms, where $w' \geq q + \tau - 1$, and update the bins as follows. For each bin we keep track of the minimum and maximum $B$-position of the contained $q$-hits, *min* and *max* respectively. The number of $q$-hits in a bin is counted until a $q$-hit $(i, j)$ is found such that $j - w + q > max$. If the bin counter has reached the threshold $\tau$, we report the matching $(max - min + q) \times e$ parallelogram. We then reset the bin counter and set $min = max = j$ as the current $q$-hit is counted. Algorithm 1 shows the bin updating step in pseudocode.

The improved approach for updating bins has a few subtle points worth mentioning. Unless care is taken, it may return too short parallelograms; in fact as short as $q$. This can happen when a very dense cluster of $q$-hits falls into a bin. In particular, single-character repeat runs are a source of such dense hit clusters. To alleviate the problem, one possibility would be to extend the reporting

**Algorithm 1**: `UpdateBin`$(r, j, d)$

```
Input  : Bin record $r$; $q$-hit position $j$ in sequence $B$; and offset bin diagonal $d$.
Output: Empty or singleton parallelogram set $P$.
$P \leftarrow \emptyset$
if $j - w + q > r.max$ then
    if $r.count \geq \tau$ then
        $p.left \leftarrow |A| - d$
        $p.top \leftarrow r.max + q$
        $p.bottom \leftarrow r.min$
        $P \leftarrow \{ \ p \ \}$
    $r.count \leftarrow 0$
if $r.count = 0$ then
    $r.min \leftarrow j$
if $r.max < j$ then
    $r.max \leftarrow j$
    $r.count \leftarrow r.count + 1$
return $P$
```

criterion such that also the validity of the parallelogram length is checked. A more elegant solution is, however, for each bin to count all $q$-hits with identical $B$ positions as one hit only. Another point is that parallelograms can be generated which are not in accordance with the filter criterion. That is, they do not contain at least $\tau$ $q$-hits within every window of length $w$. In the worst case, this happens when a bin receives one $q$-hit exactly every $w - q + 1$ positions in $B$. Although this is very unlikely to occur in practice, other likewise unfortunate hit distributions can cause the generation of similar strictly invalid filter parallelograms. However, it should be remarked that when searching for local alignments in biological sequences the regions triggering such parallelograms are often of great interest anyway. Algorithm 2 shows the pseudo-code for the main loop of our filtration algorithm.

Summing up, the specificity of our improved approach is slightly lower than that of the simple sliding window approach. This is largely due to the use of larger bins, but also because of the slight risk of producing parallelograms that do not strictly adhere to the filter criterion. On the other hand, our approach improves the time and space requirements considerably.

### 3.3 Complexity

The $q$-gram index is constructed in $\mathcal{O}(|A| + |\Sigma|^q)$ time. Each occurrence list is found in $\mathcal{O}(1)$ time, but the length can be linear in $|A|$. The worst case time for the filter is therefore $\mathcal{O}(|A| \cdot |B|)$. If we assume random strings of uniformly i.i.d. characters, the expected length of each occurrence list is $|A| \cdot |\Sigma|^{-q}$. Hence, under this assumption the filter requires $\mathcal{O}(|B| + |A| \cdot |B| \cdot |\Sigma|^{-q})$ expected time.

**Algorithm 2**: Filter for identifying parallelograms for $\epsilon$-matches

```
Input  : Query $B$; $q$-gram index $I$ for target $A$; parameters $w, e, \tau$; and $\Delta = 2^z$
Output: Set of parallelograms $P$
Allocate and initialize array of bin records $Bins$
$P \leftarrow \emptyset$
for $j \leftarrow 0$ to $|B| - q$ do
    $G \leftarrow B[j, j + q - 1]$
    $L(G) \leftarrow$ lookup occurrence list for $G$ in $I$
    foreach $i \in L(G)$ do
        $d \leftarrow |A| + j - i$
        $b_0 \leftarrow d \gg_{bit} z$
        $b_m \leftarrow b_0$ mod $|Bins|$
        $P \leftarrow P \cup$ UpdateBin($Bins[b_m], j, b_0 \ll_{bit} z$)
        if $(d \;\&_{bit}\; (\Delta - 1)) < e$ then
            $b_m \leftarrow (b_m + |Bins| - 1)$ mod $|Bins|$
            $P \leftarrow P \cup$ UpdateBin($Bins[b_m], j, (b_0 - 1) \ll_{bit} z$)
        if $(j - e)$ mod $(\Delta - 1) = 0$ then
            $b_0 \leftarrow (j - e) \gg_{bit} z$
            $b_m \leftarrow b_0$ mod $|Bins|$
            /* CheckAndResetBin is similar to lines 3–8 of UpdateBin */
            $P \leftarrow P \cup$ CheckAndResetBin($Bins[b_m], j, b_0 \ll_{bit} z$)
$P \leftarrow P \cup$ { remaining parallelograms in $Bins$ }
```

The space complexity is dominated by the $q$-gram index, which requires $|A| + |\Sigma|^q$ integers. Bins and parallelograms are each represented using 3 integers, so with a bin size of $e + 2^z$ we need $3 \cdot 2^{-z}|A|$ integers for all bins, and $3p$ integers to return $p$ parallelograms. In total, the filter requires $(3 \cdot 2^{-z} + 1)|A| + |\Sigma|^q + 3p$ integers. Thus, for 32-bit integers and parameters $|A| = 3 \cdot 10^9$, $q = 11$ and $z = 3$, the filter requires only 5.5 bytes per input character and $12p$ bytes for the result.

## 4 Applications

This section covers two applications of the basic $q$-gram filter. We first consider its use in an overlapper for sequence assembly and then for general purpose BLAST-like alignment searching.

### 4.1 Sequence Assembly

When building a typical DNA assembler, one faces the problem of comparing a collection of $600 - 1000$ bp fragments against each other in search of overlaps, typically say over 50 bp long at 95 % or greater identity. The filter is ideal for this application in that the error rate is low and the requirement is to find all matches under a given percent difference and over a lower length limit. Typically there are thousands or millions of reads $f_1, f_2, \ldots, f_n$, in total $\sum_{i=1}^{n} |f_i|$ bases. The reads are concatenated together to make a single large string $A$ that our filter is run over, with an auxiliary table $map[k] = \sum_{i=1}^{k} |f_i|$ giving the start of each read in $A$. Note that we are comparing $A$ against itself so we carefully modify the filter to ignore hits on or below the diagonal of the implied edit matrix.

In essence after running the filter, all we need to do is identify the pairs of reads that intersect parallelograms and then check each pair for a proper overlap. This is simply a matter of mapping base positions in $A$ to read positions through the inverse of *map*. We used a quick sort of all parallelograms in one dimension and then an insertion sort in the second dimension as the sort buckets are expected to be small. Multiple hits to a given read pair are merged during the insertion sort of a bucket and a parallelogram is required to have more than 5 base pairs in a sequence as often a legitimate parallelogram in one read pair will extend slightly into the next read due to the concatenation. For each read pair, we keep track of the maximum and minimum diagonal of the edit matrix between their sequences that is covered by a parallelogram so that the check for an overlap need only perform dynamic programming within a band consistent with these diagonals and the maximum error rate. The dynamic programming itself is done with a bit-vector acceleration method by Myers [14].

### 4.2 BLAST-like Searching

Another application of the filter is BLAST-like alignment searching. In this setting, we use $\epsilon$-matches as seeds and extend these into longer alignments by dynamic programming. After running the filter, we identify the possible $\epsilon$-matches in each parallelogram by chaining of the contained $q$-hits. Our approach uses a simple variation of sparse dynamic programming [7] and the fact that an $\epsilon$-match must contain at least $\tau$ $q$-hits, which are separated by no more than $e$ differences.

We define the partial ordering relation $\ll$ on a set of $q$-hits as follows. Let $h = (i, j)$ and $h' = (i', j')$ denote $q$-hits and define $diag(h) := j - i$. Let $dist_\infty(h, h')$ refer to the Chebychev distance $\max\{|i' - i - q|, |j' - j - q|\}$ between the ending and starting points of $h$ and $h'$, respectively. Then, $h \ll h'$ if and only if (1) $dist_\infty(h, h') \leq e$, and (2a) $diag(h) \neq diag(h')$ and $i + q \leq i'$ and $j + q \leq j'$, or (2b) $diag(h) = diag(h')$ and $i < i'$. A chain is then a sequence of $q$-hits $\langle h_1, h_2, \ldots, h_l \rangle$ where $h_i \ll h_{i+1}$ for all $1 \leq i < l$. If we assign the score $q$ to each $q$-hit and use $dist_\infty$ for the penalty of connecting successive $q$-hits, the score of a chain $C$ is thus given as $chain(C) := q \cdot l - \sum_{i=1}^{l-1} dist_\infty(h_i, h_{i+1})$. Denoting the maximum score over all chains ending in $q$-hit $h'$ by $chain(h')$, the recurrence relation $chain(h') := \max\{0, \max_{h \ll h'}\{chain(h) - dist_\infty(h, h')\} + q$ immediately gives the basis for the algorithm.

The chaining requires one sweep over each parallelogram. The $q$-hits are found column-wise by lookup in a hash table over the $q$-grams in the sliding window that is defined on $A$ by the first and the last rows of the current column. A balanced search structure $D$ maintains the $q$-hits found on the previous $q + e + 1$ columns, ordered by diagonal number and starting position in $B$. For each $q$-hit $h'$ in the current column, $D$ is searched for $q$-hits in the diagonal range

$[i-j-e, i-j+e]$. In every diagonal in this range, the nearest chain (w.r.t. position in $B$) ending in $q$-hit $h$, such that $h \ll h'$, is candidate for chaining with $h'$. Of all such candidate chains, the maximum scoring is chosen. If multiple candidate chains reach the maximum score, we choose the closest (w.r.t. diagonal). Then, the new chain ending in $h'$ is inserted in $D$.

Chains shorter than $\tau$ cannot be part of an $\epsilon$-match and they are therefore immediately disposed. Otherwise, the chain is rescored by finding the optimal gap position between successive $q$-hits. To avoid reporting multiple only slightly differing chains, we partition the rescored chains into equivalence classes by their first $q$-hit. That is, two chains belong to the same equivalence class if and only if they begin with the same $q$-hit. Only the maximum scoring chain for each equivalence class is extended by gapped $X$-drop extension [21]. We initiate the extension procedure at the end and beginning of the first and last $q$-hits of the chain, respectively.

## 5 Experimental Results

In this section we describe some experimental results for the two applications.

### 5.1 Sequence Assembly

With a gigabyte of memory we can reasonably solve 60 Mbp by 60 Mbp comparisons for 50 bp overlaps at less than 5 % difference in roughly 90 seconds on an Apple PowerBook G4 laptop. Larger problems are solved by partitioning the data set into 60 Mbp segments and solving either serially or in parallel all the necessary pairwise comparisons of segments. On the same laptop, the 1.8 Gbp data set for *D. melanogaster* can be compared in a total of 18 CPU hours. With more memory, larger $q$-grams can be used and larger segments can be accomodated in a single run. For example, on an Intel Itanium II with 16 Gb memory we can compute the same overlaps for *D. melanogaster* in under two hours.

### 5.2 EST Clustering

We compare the performance of our BLAST-like alignment searching application with that of BLAST and the Smith-Waterman algorithm. The Smith-Waterman implementation that we use is SSEARCH [18], which is part of the FASTA package [24]. The BLASTN version is 2.2.9 from NCBI [22]. Our BLAST-like alignment searching application is implemented in SWIFT, available at [25].

The setup resembles that of [11]. Briefly, we select EST sequences from two species and perform a full-sensitivity, all-against-all comparison. All sequence pairs where the best Smith-Waterman local alignment scores above a given threshold are recorded. The all-against-all comparison is then repeated using BLAST and SWIFT. For each, the sensitivity, or recall ratio, is determined as follows. Suppose the full-sensitivity search finds $p$ pairs with a best local alignment score of $s$. If $p'$ of the $p$ pairs are found with score at least $\frac{s}{2}$ by BLAST or SWIFT, the ratio $\frac{p'}{p}$ is the sensitivity for alignment score $s$. Neither BLAST or SWIFT attempt to compute the optimal alignment for found homologies and we therefore consider a sequence pair recalled if its local alignment score is not lower than $\frac{s}{2}$. We note that any other threshold ratio of $s$ can equally well be used.

The EST sequences are obtained from NCBI [23]. We randomly select 40.000 sequences from *H. sapiens* (25 Mbp) and 5.600 sequences from *M. musculus* (2 Mbp). The poly-X tails ($X = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$) typically found in EST sequences due to sequencing errors are trivially masked to Ns. The $q$-gram length is set to 11 in both BLAST and SWIFT. All programs use match/mismatch scores $\pm 1$, and gap open and extension penalties are set to 5 and 1, respectively. The local alignment score threshold is 16. Searches with SSEARCH are conducted on a cluster with 50 UltraSparcIIe/500 MHz nodes, whereas the searches with BLAST and SWIFT run on a 2 GHz AMD Athlon-XP Linux PC. Table 2 lists the running times for the different programs and Fig. 3 compares the sensitivity of BLAST and SWIFT.

Using parameters for typical EST clustering criteria, $\epsilon = 0.05$ and $n_0 = 50$, the sensitivity is a bit lower for SWIFT than for BLAST. However, this is expected as SWIFT requires and guarantees the presence of an $\epsilon$-match before an alignment is recorded. In other words, the extra alignments found by BLAST do not conform to the query criteria. Additionally, note that by using more agressive $(\epsilon, n_0)$ parameters, SWIFT can in general attain sensitivity levels comparable to or better than BLAST, while still being more than 25 times faster.

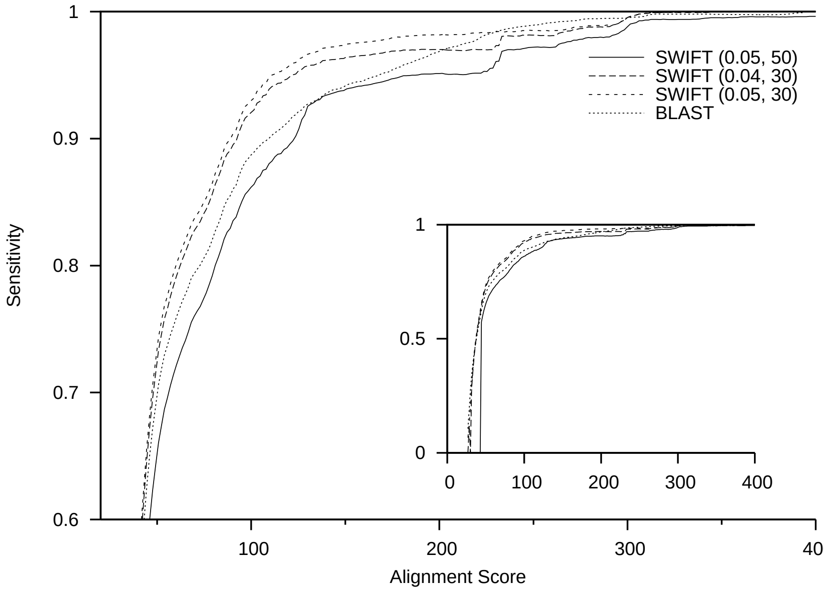

**Fig. 3.** Sensitivity of BLAST and SWIFT. The horizontal line in 1.0 corresponds to the sensitivity of SSEARCH. For SWIFT, the $(\epsilon, n_0)$ parameters are shown in parentheses. The inset shows the complete sensitivity range.

**Table 2.** Running times for EST all-against-all comparison. The time for the database formatting and preprocessing in BLAST (3 s) and SWIFT (12 s) is not included.

| | SWIFT | | | BLAST | SSEARCH |
|---|---|---|---|---|---|
| $(\epsilon, n_0)$ | $(0.05, 50)$ | $(0.04, 30)$ | $(0.05, 30)$ | — | — |
| Running time | 18 s | 29 s | 35 s | 773 s | 8 h |

The filtration efficiency is the primary gauge of the expected running time for the application-specific post-processing of the filter output. We therefore also measure the filtration ratio, which we define as the total area of the unfiltered regions, i.e. the reported parallelograms, divided by the total size of the implied edit matrix. Table 3 shows the resulting filtration ratios and times for SWIFT. For comparison, we include the times and ratios for the closest precursor to our filter, QUASAR. We repeated each run with the QUASAR block size giving the best possible filtration ratio (128 in first; all others 64) and with the smallest block size giving filtration time equal or superior to that of SWIFT (1024 for all runs).

**Table 3.** Filtration ratios and times for EST all-against-all comparison.

| $(\epsilon, n_0)$ | SWIFT Filtration | | QUASAR Filtration, best ratio | | QUASAR Filtration, best time | |
|---|---|---|---|---|---|---|
| | Ratio | Time (s) | Ratio | Time (s) | Ratio | Time (s) |
| $(0.05, 50)$ | $6.5 \cdot 10^{-6}$ | 6.0 | $4.5 \cdot 10^{-4}$ | 36.1 | $2.1 \cdot 10^{-3}$ | 4.2 |
| $(0.04, 30)$ | $4.5 \cdot 10^{-6}$ | 5.0 | $4.0 \cdot 10^{-4}$ | 69.0 | $3.1 \cdot 10^{-3}$ | 4.4 |
| $(0.05, 30)$ | $5.4 \cdot 10^{-6}$ | 6.1 | $4.3 \cdot 10^{-4}$ | 68.5 | $3.5 \cdot 10^{-3}$ | 4.4 |

As stressed by Table 3, our filter is very efficient. Compared to QUASAR, our filter is almost two orders of magnitude more specific while being approximately one order of magnitude faster.

## 6 Conclusion

The problem of finding $\epsilon$-matches is a recurring theme in many DNA searches. We described the theoretical framework and a workable solution for an efficient filter, that identifies regions of the implied edit matrix guaranteed to overlap with possible $\epsilon$-matches. This result is of great practical importance to numerous applications, including e.g. whole-genome alignment.

An interesting direction for further developments is to increase the interval of practically usable values of $\epsilon$ by allowing mismatches in the $q$-grams.

## References

1. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215(3):403–410, 1990.

2. S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, 25(17):3389–3402, 1997.
3. S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivals, and M. Vingron. $q$-gram based database searching using a suffix array. In *Proc. of the 3rd Annu. Int. Conf. on Computational Molecular Biology (RECOMB'99)*, pages 77–83, 1999.
4. S. Burkhardt and J. Kärkkäinen. Better filtering with gapped $q$-grams. In *Proc. of CPM'01*, volume 2089 of *LNCS*, pages 73–85, 2001.
5. A. Califano and I. Rigoutsos. FLASH: a fast look-up algorithm for string homology. In *Proc. of the 1st Int. Conf. on Intelligent Systems for Molecular Biology (ISMB'93)*, pages 56–64, 1993.
6. W. I. Chang and E. L. Lawler. Sublinear expected time approximate string matching and biological applications. *Algorithmica*, 12(4/5):327–344, 1994.
7. D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano. Sparse dynamic programming I: linear cost functions. *J. ACM*, 39(3):519–545, 1992.
8. D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.
9. P. Jokinen and E. Ukkonen. Two algorithms for approximate string matching in static texts. In *Proc. of MFCS'91*, volume 520 of *LNCS*, pages 240–248, 1991.
10. W. J. Kent. BLAT – the BLAST-like alignment tool. *Genome Res.*, 12(4):656–664, 2002.
11. M. Li, B. Ma, D. Kisman, and J. Tromp. PatternHunter II: Highly Sensitive and Fast Homology Search. In *Proc. of the 14th Annu. Int. Conf. on Genome Informatics (GIW'03)*, pages 164–175, 2003.
12. B. Ma, J. Tromp, and M. Li. PatternHunter – faster and more sensitive homology search. *Bioinformatics*, 18:440–445, 2002.
13. E. Myers. A sublinear algorithm for approximate keyword searching. *Algorithmica*, 12(4/5):345–374, 1994.
14. E. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM*, 46(3):539–553, 1999.
15. E. Myers and R. Durbin. A table-driven, full-sensitivity similarity search algorithm. *J. Comp. Bio.*, 10(2):103–118, 2003.
16. Z. Ning, A. J. Cox, and J. C. Mullikin. SSAHA : A fast search method for large DNA databases. *Genome Res.*, 11(10):1725–1729, 2001.
17. W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA*, 85:2444–2448, 1988.
18. W. R. Pearson. Searching protein sequence libraries: Comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. *Genomics*, 11:635–650, 1991.
19. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147(1):195–197, 1981.
20. E. Ukkonen. Approximate string-matching with $q$-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–211, 1992.
21. Z. Zhang, P. Berman, and W. Miller. Alignments without low-scoring regions. In *Proc. of the 2nd Annu. Int. Conf. on Computational Molecular Biology (RECOMB'98)*, pages 294–301, 1998.
22. `ftp://ftp.ncbi.nih.gov/toolbox/ncbi_tools/ncbi.tar.gz`.
23. `ftp://ftp.ncbi.nlm.nih.gov/blast/db/FASTA/est_{human,mouse}.gz`.
24. `ftp://ftp.virginia.edu/pub/fasta`.
25. `http://bibiserv.techfak.uni-bielefeld.de/swift`.