A Fast Filtration Algorithm for the Substring Matching Problem *

Pavel A. Pevzner[1] and Michael S. Waterman[2]

[1] Computer Science Department
The Pennsylvania State University
University Park, PA 16802

[2] Departments of Mathematics and of Molecular Biology
University of Southern California
Los Angeles, California 90089-1113

Abstract. Given a text of length n and a query of length q we present an algorithm for finding all locations of m-tuples in the text and in the query that differ by at most k mismatches. This problem is motivated by the dot-matrix constructions for sequence comparison and optimal oligonucleotide probe selection routinely used in molecular biology. In the case $q = m$ the problem coincides with the classical *approximate string matching with k mismatches* problem. We present a new approach to this problem based on multiple filtration which may have advantages over some sophisticated and theoretically efficient methods that have been proposed. This paper describes a two-stage process. The first stage (multiple filtration) uses a new technique to preselect roughly similar m-tuples. The second stage compares these m-tuples using an accurate method. We demonstrate the advantages of multiple filtration in comparison with other techniques for approximate pattern matching.

1 Introduction

Suppose we are given a string of length n, $T[1\ldots n]$, called the *text*, a shorter string of length q, $Q[1\ldots q]$, called the *query*, and integers k and m. The *substring matching problem with k-mismatches* ([CL90]) is to find all *"starting"* locations $1 \leq i \leq q-m+1$ in the query and $1 \leq j \leq n-m+1$ in the text, such that the substring of the query $Q[i, i+1, \ldots, i+m-1]$ matches the substring of the text $T[j, j+1, \ldots, j+m-1]$ with at most k mismatches. In the case $q = m$ the substring matching problem yields the *approximate string matching problem with k-mismatches.*

The approximate string matching problem with k-mismatches has been intensively studied in computer science. The naive brute-force algorithm for approximate string matching runs in $O(nm)$ time. Landau and Vishkin ([LV86])

* The research was supported in part by the National Science Foundation (DMS 90-05833) and the National Institute of Health (GM-36230). This paper was written when P.A.P. was at the Department of Mathematics, University of Southern California.

gave a $O(kn + km \log m)$ approximate pattern matching algorithm. Galil and Giancarlo ([GG86]) improved the Landau-Vishkin algorithm, achieving a time performance $O(kn + m \log m)$. All these algorithms and their improved versions (see [LV89], [TU90]) are based on the preprocessing of the pattern/text

Recently several approaches emphasizing expected running time have appeared in contrast to earlier results ([BG89], [CL90], [GL90], [TU90], [HS91], [WM92a], [WM92b], [BP92]). In particular, Grossi and Luccio ([GL90]) demonstrated that although earlier algorithms yield the best performance in the worst cases, they are far from being the best in practice. In particular, a simple *filtration* algorithm from [GL90] runs approximately 10 times faster than the algorithm from [GG86] for a wide range of k and m.

The idea of filtration algorithms for approximate matching involves a two-stage process. The first stage preselects a set of positions in the text that are *potentially* similar to the pattern. The second stage verifies each potential position using an accurate method rejecting potential matches with more than k mismatches. Denote p the number of potential matches found at the first stage of the algorithm. Preselection is usually done in $O(n + p)$ time where the coefficient of n is much smaller than for the algorithms based on the preprocessing of the pattern/text. If the number of potential matches is small and the accurate method for potential match verification is not too slow, this idea brings a significant speed up in comparison to the algorithms based on the preprocessing of the pattern/text.

The idea of filtration for information retrieval/pattern matching goes back to early 70's [H71]. The idea of filtration for string matching problems first was described by Karp and Rabin [KR87] for the case $k = 0$. Notice that the idea of filtration in computational molecular biology for related alignment problems was stated even earlier (see [DN82], [WL83], [LP85] for *l-tuple filtration*, [B86], for *filtration by composition*).

For $k > 0$ Owolabi and McGregor [OM88] used an idea of *l-tuple filtration* based on a simple observation that if a pattern approximately matches a substring of the text then they share at least one l-tuple for sufficiently large l. Finding all l-tuples shared by pattern and text can be easily done by hashing. If the number of shared l-tuples is relatively small they can be verified and all *real* matches with k mismatches can be rapidly located. The theoretical analysis of the expected running time of this approach has been recently done by Kim and Shawe-Taylor [KS92]. The idea of l-tuple filtration has been significantly developed by Baeza-Yates and Perleberg [BP92] and by Wu and Manber ([WM92a], [WM92b]).

Grossi and Luccio ([GL90]) observed that if a pattern approximately matches a substring of the text then they have similar letter compositions. This observation leads to a simple algorithm running in $O(n \log |A| + pm)$ time, where A is the alphabet of pattern P and $p < n$ is the number of m-substrings of text T with letter composition having at most k differences with the letter composition of the pattern. Computational experiments with such *filtration by composition* show that $pm < nk$ for a wide range of parameters thus making the Grossi-Luccio algorithm important in practice. Recently Ukkonen ([U92]) generalized the Grossi-Luccio algorithm taking an advantage of l-tuple composition (*filtration by l-tuple composition*) instead of letter (1-tuple) composition.

The complexity of filtration methods depends critically on the ratio $\frac{r}{p}$ (*filtration efficiency*) between r, the number of *real* matches with k mismatches and p, the number of *potential* matches found on the first stage of the algorithm. The larger this ratio the smaller the running time of the second stage of filtration algorithm. In the case $\frac{r}{p} = 1$ we would have an *ideal filtration* but none of the mentioned algorithms provides an ideal filtration or even lower bounds for filtration efficiency. Moreover the filtration algorithms described above do not provide a method for increasing filtration efficiency even at the expense of spending more time on the first (filtration) stage of the algorithm. Also filtration by composition does not allow an efficient implementation for the substring matching problem. We give an algorithm that allows exponential reduction of the number of potential matches at the expense of a linear increase of the filtration time. Therefore we drastically reduce the time of the 2nd stage of the algorithm (potential match verification) for the cost of linearly increased time of the first stage (filtration). Taking into account that the 2nd stage is frequently more time-consuming than the first one, the technique provides a trade-off for an optimal choice of filtration parameters.

Methods described in this paper can be applied to optimal oligonucleotide probe selection ([DMDC87]) and efficient algorithms for dot-matrices ([ML81]) in molecular biology applications. (See Landau et al. [LVN88] for a dynamic programming algorithm for substring matching problem and dot-matrix applications). Some of the described techniques have been implemented in the *OligoProbeDesignStation* software package. (Mitsuhashi M., Cooper A., Waterman M., Pevzner P. *OligoProbeDesignStation: a computerized method for designing optimal DNA probes.* Pending application for United States Letters Patent (1992).)

2 Filtration methods for approximate pattern matching

The following simple observation (compare with Theorem 5.1 from [U92]) provides a basis for *l-tuple filtration* and *filtration by l-tuple composition*.

Lemma 1. *A boolean word $v[1,\ldots,m]$ with at most k zeros contains at least $m - (k+1)l + 1$ l-runs of ones.*

Substituting $l = \lfloor \frac{m}{k+1} \rfloor$ in Lemma 1 we derive

Lemma 2. *A boolean word $v[1,\ldots,m]$ with at most k zeros contains at least one l-run of ones with $l = \lfloor \frac{m}{k+1} \rfloor$.*

Notice that every match with at most k mismatches between strings $P[1,\ldots,m]$ and $S[1,\ldots,m]$ corresponds to a boolean word $v[1,\ldots,m]$ by the rule

$$v[i] = \begin{cases} 1, & \text{if } P[i] = S[i] \\ 0, & \text{otherwise} \end{cases}$$

This remark and lemma 2 imply the following observation of Baeza-Yates and Perleberg [BP92] and Wu and Manber [WM92b]

Lemma 3. *Let the strings* $P[1,\ldots,m]$ *and* $S[1,\ldots,m]$ *match with at most* k *mismatches and* $l = \lfloor \frac{m}{k+1} \rfloor$. *Then the strings* P *and* S *share a* l*-tuple, i.e.* $\exists i$: $P[i, i+1, \ldots, i+l-1] = S[i, i+1, \ldots, i+l-1]$.

Lemma 3 motivates a simple two-stage *l-tuple filtration* algorithm for approximate substring matching with k mismatches between a query $Q[1,\ldots,q]$ and a text $T[1,\ldots,n]$:

Algorithm 1. Detection of all m-matches between Q and T with up to k mismatches.

- *Potential match detection.* Find all occurrences of l-tuples in both the pattern and the text.
- *Potential match verification.* Verify each potential match by extending it to the left and to the right until either the first $k+1$ mismatches are found, or the beginning/end of Q or T is found □

Lemma 3 guarantees that Algorithm 1 finds *all* matches of length m with k or fewer mismatches between Q and T if $l \leq \lfloor \frac{m}{k+1} \rfloor$. Stage 1 (potential match detection) of the Algorithm 1 can be implemented by hashing or by building the trie ([K73]). The running time of Algorithm 1 is $O(n + p_1 m)$ where p_1 is the number of potential matches detected at the first stage of the algorithm (see [BP92] and [WM92b] for details of the implementation). For a Bernoulli text with A equiprobable letters the expected number of potential matches equals

$$E(p_1) = \frac{(n-l+1)(q-l+1)}{A^l}$$

yielding a fast algorithm for large A and l.

For Bernoulli texts with equiprobable A letters define the *filtration efficiency* e of a filtration algorithm to be the ratio of the expected number of matches with k mismatches $E(r)$ to the expected number of potential matches $E(p)$. For example for $k = 1$ the efficiency of the l-tuple filtration, (see Wu and Manber, [WM92b]) $e \approx \frac{A-1}{A^{\lfloor \frac{m}{2} \rfloor}}$ is rapidly decreasing with m and A increasing. This observation raises the question of devising a filtration method with a larger filtration efficiency.

3 The idea of multiple filtration

A set of positions $i, i+t, i+2t \ldots, i+jt, \ldots, i+(l-1)t$ is called a *gapped l-tuple* with *gapsize* t and *size* $1+t(l-1)$ (Fig.1). Continuous l-tuples are simply gapped l-tuples with gapsize 1 and size l.

Similarly to lemmas 1 and 2 we derive

Lemma 4. *A boolean word* $v[1,\ldots,m]$ *with at most* k *zeros contains at least* $m-s+1-kl$ *gapped* l*-tuples with gapsize* t *of size* s *containing only ones.*

Lemma 5. *A boolean word* $v[1,\ldots,m]$ *with at most* k *zeros contains at least one gapped* $\lfloor \frac{m}{k+1} \rfloor$*-tuple with gapsize* $k+1$ *containing only ones.*

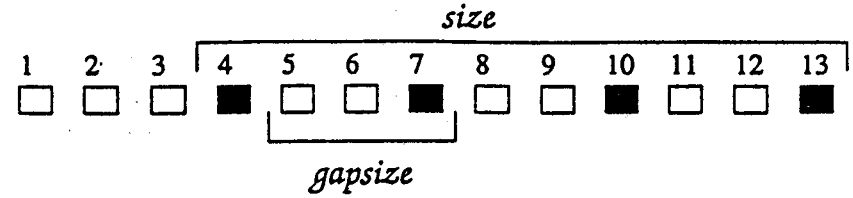

Fig. 1. Gapped 4-tuple with gapsize 3 and size 10 starting at position 4.

If an l-tuple shared by the pattern and the text starts at position i of the pattern and at position j of the query, we call (i,j) the *coordinate* of l-tuple. Define the *distance* $d(v_1, v_2)$ between l-tuples with coordinates (i_1, j_1) and (i_2, j_2) as

$$d(v_1, v_2) = \begin{cases} i_1 - i_2, & \text{if } i_1 - i_2 = j_1 - j_2 \\ \infty, & \text{otherwise} \end{cases}$$

Combining lemma 2 and lemma 5 we derive

Lemma 6. *Let the strings* $P[1,\ldots,m]$ *and* $S[1,\ldots,m]$ *match with at most* k *mismatches and* $l = \lfloor \frac{m}{k+1} \rfloor$. *Then the strings* P *and* S *share both a continuous* l*-tuple and a gapped* l*-tuple with gapsize* $k+1$ *with distance* d *between them satisfying the condition*

$$-k \leq d \leq m - l$$

Lemma 6 is the basis of a two-stage *double filtration* algorithm for approximate string matching with k mismatches between a query $Q[1,\ldots,q]$ and a text $T[1,\ldots,n]$:

Algorithm 2. Detection of all m-matches between Q and T with up to k mismatches.

- *Potential match detection.* Find all such occurrences of continuous l-tuples from the pattern in the text where there exists a gapped l-tuple with gapsize $k+1$ of the distance $-k \leq d \leq m-l$ from the continuous l-tuple.
- *Potential match verification.* Verify each potential match by extending it to the left and to the right until either the first $k+1$ mismatches are found or the beginning/end of Q or T is found. □

Lemma 6 guarantees that Algorithm 2 finds *all* matches between P and T with k mismatches. Stage 1 (potential match detection) of Algorithm 2 can be implemented by hashing. The running time of algorithm 2 is $O(n + p_2 m)$ where

p_2 is the number of potential matches detected at the first stage of the algorithm (the details of an implementation are given in section 5). Define $\delta = \lceil \frac{l}{k+1} \rceil$. For a Bernoulli text with equiprobable A letters the expected number of potential matches can be roughly estimated by

$$E(p_2) \leq \frac{(n-l+1)(m-l+1)}{A^l} \cdot \frac{m}{A^{l-\delta}},$$

thus yielding better filtration than l-tuple filtration when $m < A^{l-\delta}$. The efficiency of double filtration is at least $\frac{A^{l-\delta}}{m}$ better than the efficiency of l-tuple filtration. For typical parameters of oligonucleotide probe selection ($A = 4, m = 25, k = 2$) double filtration is at least 40 times more efficient than l-tuple filtration.

In the next section we estimate the efficiency of double filtration.

4 Efficiency of double filtration

According to lemma 6 every match with k mismatches corresponds to both a continuous l-tuple and a gapped l-tuple located close to each other that contain only ones. In this section we estimate the expected number of such occurrences in a random Bernoulli boolean word.

Fix m and k and let $l = \lfloor \frac{m}{k+1} \rfloor$. We say that position j is in the *vicinity* of position i if $-k \leq i - j \leq m - l$ (see lemma 6).

A position i in a boolean word $v[1, \ldots, n]$ is a *potential match* if

(i) $v[i, \ldots, i+l-1]$ is a run of ones,
and
(ii) there exists a gapped l-tuple with gapsize $k+1$ starting in the vicinity of i that contains only ones.

We denote a continuous l-tuple starting at position i as $c(i)$, and a gapped l-tuple of gapsize $k+1$ starting at position j as $g(j)$. Notice that a continuous l-tuple $c(i)$ and a gapped l-tuple $g(j)$ of gapsize $k+1$ can share at most $\delta = \lceil \frac{l}{k+1} \rceil$ positions. If $g(j)$ contains a position $i+s$ ($0 \leq s \leq k$), then $c(i)$ and $g(j)$ can share at most $\delta(s) = \lceil \frac{l-s}{k+1} \rceil$ positions (Fig.2).

Lemma 7. *Let $v[1, \ldots]$ be a Bernoulli boolean word with the probabilities of letters $p(1) = p$ and $p(0) = 1 - p = q$. Then the probability of a potential match at position $i > m - l$ equals*

$$p^l \cdot \{(1 - \prod_{s=0}^{s=k} (1 - (l - \delta(s)) p^{l-\delta(s)} q - p^{l-\delta(s)})\}$$

Proof. For a l-tuple $c(i)$ starting at i define $G_s(i) = \{g(t)\}$ to be the set of gapped l-tuples with gapsize $k+1$ starting in vicinity of $c(i)$ and fulfilling the condition: $i - t = s \bmod k+1$ (Fig.2). Let $P_s(i)$ be the probability that at least one l-tuple in $G_s(i)$ contains only ones given that $c(i)$ contains only ones. Let

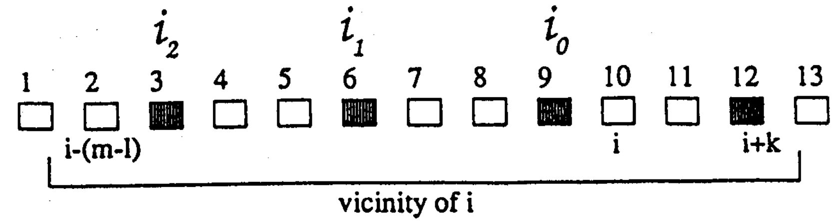

Fig. 2. Vicinity of the position $i = 10$ ($m = 12, k = 2, l = \lfloor \frac{m}{k+1} \rfloor = 4, gapsize = k+1 = 3$. Solid boxes indicate the starting positions of gapped l-tuples from $G_2(i) = \{g(3), g(6), g(9), g(12)\}$. A gapped 4-tuple and a continuous 4-tuple can share at most $\delta = \lceil \frac{l}{k+1} \rceil = \lceil \frac{4}{2+1} \rceil = 2$ positions. Gapped 4-tuples from $G_2(i)$ and continuous 4-tuple $c(i)$ can share at most $\delta(s) = \lceil \frac{l-s}{k+1} \rceil = \lceil \frac{4-2}{2+1} \rceil = 1$ positions ($l' = l - \delta(s) = 3$).

$P(i)$ be the probability to have a potential match at position i given that $c(i)$ contains only ones. As the sets $G_s(i)$ are non-overlapping for $0 \leq s \leq k$,

$1 - P(i) = P\{$there is no gapped l-tuple $g(t)$ in the vicinity of i containing only ones$\} =$

$\prod_{s=0}^{s=k} P\{$there is no gapped l-tuple $g(t) \epsilon G_s(i)$ in the vicinity of i containing only ones$\} =$

$$\prod_{s=0}^{s=k} (1 - P_s(i)).$$

Each l-tuple from $G_s(i)$ shares at most $\delta(s)$ positions with $c(i)$. Denote $l' = l - \delta(s)$. Fix i and consider the following positions of v to the left of i (Fig.2):

$$i_0 = i + s - (k+1), \quad i_1 = i + s - 2(k+1), \ldots, \quad i_{l'-1} = i + s - l'(k+1)$$

and let $left$ be the minimum index such that $v[i_{left}] = 0$ (we assume $left = l'$ if $v[i_0] = v[i_1] = \ldots = v[i_{l'-1}] = 1$). Similarly consider the positions of v to the right of $i + l - 1$

$$j_0 = i+s+(\delta(s))(k+1), j_1 = i+s+(\delta(s)+1)(k+1), \ldots, j_{l'-1} = i+s+(\delta(s)+l'-1)(k+1)$$

and let $right$ be the minimum index such that $v[j_{right}] = 0$ (we assume $right = l'$ if $v[j_0] = v[j_1] = \ldots = v[j_{l'-1}] = 1$).

The positions $i_{l'-1}, \ldots, i_0, j_0, \ldots j_{l'-1}$ represent possible positions of gapped l-tuples from $G_s(i)$. We denote $P^*(i_l, j_r)$ the probability that $left = i_l$ and $right = j_r$ in a random word v. Obviously $G_s(i)$ contains a l-tuple with only ones if and only if $left + \delta(s) + right \geq l$). Therefore

$$1 - P_s(i) = \sum_{0 \leq i_l \leq l'-1, 0 \leq j_r \leq l'-1} P^*(i_l, i_r)$$

where the product is taken over all values i_l and j_r fulfilling the conditions $i_l + j_r < l'$. As $P(left = t) = P(right = t) = qp^t$, the probabilities $P\{left + right = t\}$ constitute the *negative binomial distribution* ([F70]) and

$$1 - P_s(i) = \sum_{i_l + j_r < l'} qp^{i_l} \cdot qp^{j_r} = q^2 \sum_{t=0}^{l'-1} \binom{t+1}{t} p^t = q^2 \cdot \left(\sum_{t=0}^{l'-1} p^{t+1}\right)' = q^2 \cdot \left(\frac{p - p^{l'+1}}{1-p}\right)' =$$

$$1 - l'p^{l'}q - p^{l'}$$

□

Denote $\delta_{min} = \min_s \delta(s) = \lceil \frac{l-k}{k+1} \rceil$. Lemma 7 implies

Lemma 8. *Let $Q[1, \ldots]$ and $T[1, \ldots]$ be random query and text and let p be the probability that arbitrary letters from the query and from the text are equal. Then the probability of potential match between the query and the text at position (i, j) is less than or equal $p^{2l-\delta_{min}}((m - l + k)(1 - p) + k + 1)$*

Proof. Let $P(i, j)$ be the probability of a potential match at (i, j) given the condition that the continuous l-tuples of Q and T starting at positions i and j are equal. Without loss of generality assume that $i - j = \Delta > 0$ and consider a boolean word $v[1, \ldots]$ corresponding to a diagonal Δ:

$$v[t] = \begin{cases} 1, & \text{if } Q[t + \Delta] = T[t] \\ 0, & \text{otherwise} \end{cases}$$

Applying lemma 7 to a word v with $p(1) = p$ and taking into account that $\delta_{min} \leq \delta(s) \leq \delta$ we derive

$$1 - P(i, j) = \prod_{s=0}^{k} (1 - (l - \delta(s))p^{l-\delta(s)}q - p^{l-\delta(s)})$$

$$\geq \prod_{s=0}^{k} (1 - (l - \delta_{min})p^{l-\delta}q - p^{l-\delta}) \geq 1 - (k + 1)(l - \delta_{min})p^{l-\delta}q - (k + 1)p^{l-\delta}$$

Therefore

$$P(i, j) \leq ((k + 1)l - (k + 1)\delta_{min})p^{l-\delta}q + (k + 1)p^{l-\delta} \leq$$

$$((k+1)\lfloor \frac{m}{k+1} \rfloor - (k+1)\lceil \frac{l-k}{k+1} \rceil)p^{l-\delta}q + (k+1)p^{l-\delta} \leq p^{l-\delta}((m-l+k)(1-p)+k+1).$$

□

Lemma 8 demonstrates that the efficiency of double filtration is approximately $\frac{A^{l-\delta}}{m(1-\frac{1}{A})}$ times larger than the efficiency of l-tuple filtration for a wide range of parameters m and k. Fig.3 presents the results of comparison of the efficiency of double filtration with the efficiency of l-tuple filtration for a 4-letter DNA alphabet.

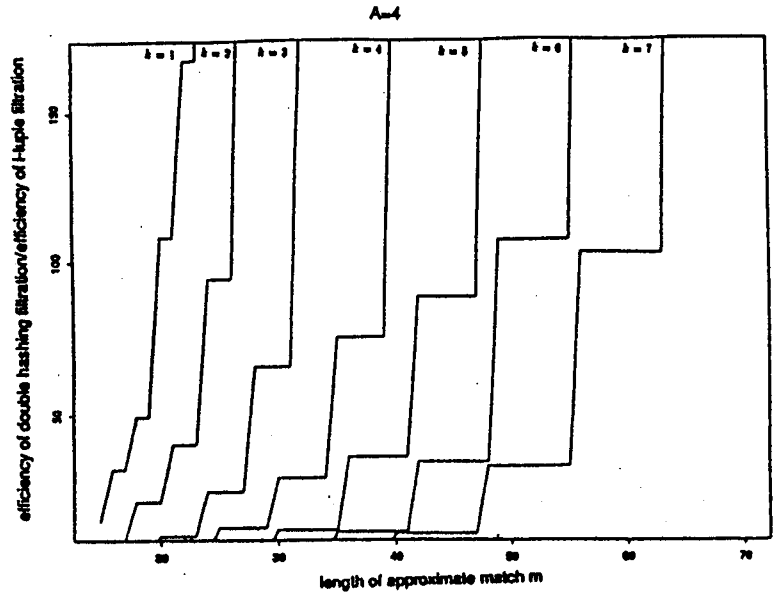

Fig. 3. The comparison of the efficiency of double filtration and l-tuple filtration. The plot shows the ratio of the efficiency of double filtration and l-tuple filtration in 4-letter alphabet for different parameters m and k.

Comment: The definition of filtration efficiency when applied to comparison of l-tuple and double filtration should be taken with caution. The definition does not take into account the number of potential matches relative to the size of the text. When l-tuple filtration is already very efficient there is no reason to apply further filtration. In other words, if the total number of expected potential matches is, say, 1.3 for the whole text, vs. 0.013 for true matches, the ratio is large but is meaningless in practice.

5 Double filtration for approximate substring matching

In this section we present a sketch of the implementation of double filtration for approximate substring matching problem. For simplicity we concentrate on double filtration described by Algorithm 2 and consider the alphabet $\mathcal{A} = \{0, \ldots, A - 1\}$.

Let p be the number of potential matches between the query and the text found at the filtration stage of the Algorithm 2, and p_c (p_g) be the number of continuous (gapped) l-tuples shared by the query and the text. It is not difficult to see that the filtration stage of Algorithm 2 can be implemented in $O(q + n + p_c + p_g)$ time by hashing (compare with [U92]).

Query hashing

We need an encoding of every l-tuple v as an integer. A natural encoding is to interpret each l-tuple as an A-ary integer. For a l-tuple $v[1,\ldots,l]$ a *hash value* of v is

$$\hat{v} = v[1]A^{l-1} + v[2]A^{l-2} + \ldots v[l]A^0 \tag{1}$$

For query $Q[1,\ldots,q]$ define $v_i = Q[i,\ldots,i+l-1]$, $1 \le i \le m-l+1$, be the l-tuple of Q starting at position i. Obviously

$$\hat{v}_{i+1} = (\hat{v}_i - Q[i]\cdot A^{l-1})\cdot A + Q[i+l] \tag{2}$$

By setting v_1 and then applying (2) for $1 \le i \le m-l$, we get the hash values for all l-tuples of Q. Assuming that each application of (2) takes constant time (we consider relatively small A and l) we can build hash table H_1 for continuous l-tuples in $O(q)$ time. Continuous l-tuples from the query with the same hash value h are put in a linked list pointed by $H_1[h]$([K73]).

Similarly we can build a hash table H_2 for gapped l-tuples with gapsize gap in $O(q)$ time. Denote $w_i = Q[i, i+gap, i+2\cdot gap,\ldots,i+j\cdot gap,\ldots,i+(l-1)\cdot gap]$. Using the same hash function (1) for w_i we get

$$\hat{w}_{i+gap} = (\hat{w}_i - Q[i]\cdot A^{l-1})\cdot A + Q[i+l\cdot g] \tag{3}$$

By setting $\hat{w}_1,\ldots,\hat{w}_{gap}$ and then applying 3 we get hash values for all gapped l-tuples with gapsize gap. Gapped l-tuples from the query with the same hash value h are put in a linked list pointed by $H_2[h]$. Note that with such an implementation memory requirements of double filtration are doubled in comparison with l-tuple filtration.

Text scanning with double filtration

Figure 4 presents a sketch of the filtration stage for approximate substring matching with k mismatches by double filtration. We assume $l = \lfloor\frac{m}{k+1}\rfloor$ and $size = (k+1)(l-1)+1$. Given a $q \times n$ matrix we number its $q+n-1$ diagonals assigning number $j-i+q$ to a diagonal containing position (i,j). To implement double filtration we have to test efficiently if there exists a gapped l-tuple in the vicinity of a continuous l-tuple. To provide this test we use an array $diag[1,\ldots,n+q]$ and assign

$$diag[j-i+q] = j$$

every time we find a gapped l-tuple starting at (i,j). Therefore for each of $n+q$ diagonals of the $q \times n$ matrix representing all possible coordinates, $diag[t]$ equals the starting position in T of the *last* gapped l-tuple found at this diagonal. On the preprocessing stage of the algorithm we put a dummy value $diag[t] = -1$ for $1 \le t \le n+q$. Although memory requirements of substring matching problem are not crucial in many applications notice that to reduce memory requirements *diag* can be actually implemented as an array of size q that is scanned in a circular manner (not shown at Fig.4).

Algorithm Text scanning with double filtration

```
for (j = 1; j < n - l + 1; j++) /*n is the length of the text*/
{
   if (j < n - size + 1 - k) /*size is the size of gapped l-tuple with gapsize k + 1*/
   {
      compute the hash value ŵ_{j+k} of the gapped l-tuple starting at j + k
      if (linked list H2[ŵ_{j+k}] is not empty)
      {
         for  all gapped l-tuples from H2[ŵ_{j+k}] find their starting positions
                 i[1],...,i[t1] in the query
         for (t = 1; t <= t1; t++)
            diag[(j + k) - i[t] + q] = j + k;
      }
   }
      compute the hash value v̂_j of the continuous l-tuple starting at j
      if (linked list H1[v̂_j] is not empty)
      {
         for  all continuous l-tuples from H1[v̂_j] find their starting positions
                 i[1],...,i[t2] in the query
         for (t = 1; t <= t2; t++)
            if(j - diag[j - i[t] + q] <= m - l) /*see lemma 8*/
               report potential match (i[t], j)
      }
}
```

Fig. 4. Sketch of the filtration stage of the double filtration approximate substring matching algorithm.

Potential match verification

Brute-force implementation of the verification stage of the algorithm adds $O(m)$ time for verifing of each potential match. For the approximate pattern matching problem it leads to an algorithm with linear expected running time for $k = O(\frac{m}{\log m})$ (see [BP92] for details).

6 Overlapping potential matches and fast dot-matrix drawing

Several optimizations are included that deviate from the simple description of the algorithm given in Fig.4. We observe that m-matches with k-mismatches frequently overlap. To exclude redundant output we use an *extended potential match* data structure.

Let (i,j) be a potential match on the diagonal $j-i+q$ where i (j) is the starting position of the l-tuple representing potential match in the query (text).

Consider the positions on the diagonal $j-i+q$ *behind* (i,j) and define an array $b[1,..]$:

$$b[t]=\begin{cases}1, & \text{if } Q[i-t]=T[j-t]\\ 0, & \text{otherwise}\end{cases}$$

Similarly, consider the positions on the diagonal $j-i+q$ *ahead* (i,j) and define an array $a[1,\ldots]$:

$$a[t]=\begin{cases}1, & \text{if } Q[i+l-1+t]=T[j+l-1+t]\\ 0, & \text{otherwise}\end{cases}$$

(for the sake of simplicity we neglect border effects when, for example $i-t<0$). Let $behind[0,\ldots,k]$ be an array with the positions of the first $(k+1)$ zeros in b. Similarly let $ahead[0,\ldots,k]$ be an array with the positions of the first $(k+1)$ zeros in a. We call the structure

$$(i,j),\quad behind[0,\ldots,k]\quad ahead[0,\ldots,k]$$

an *extended potential match* starting at (i,j).

Let (i,j) be a potential match and $Q[i+l]=T[j+l]$ (it means that $(i+1,j+1)$ is a potential match also). Notice that in this case an extended potential match $(i+1,j+1)$ does not provide any additional information in comparison with (i,j) and we can exclude such overlapping extended potential matches from further consideration.

Arrays $behind[0,\ldots,k]$ and $ahead[0,\ldots,k]$ can be easily derived by simply scanning diagonal $j-i+q$ behind (i,j) and ahead $(i+l-1,j+l-1)$ or by faster methods (see, for example Wu and Manber [WM92b]). We say that an approximate match with k-mismatches starting at (i',j') is *generated* by a potential match (i,j) if it belongs to the same diagonal $j'-i'=j-i$ and

$$i-behind[k]<i' \text{ and } i+l-1+ahead[k]>i'+m-1$$

Lemma 6 guarantees that each approximate match is generated by at least one potential match. On the other hand a potential match (i,j) generates an approximate match with k mismatches if and only if there exists $0\le t\le k$: $ahead[t]+behind[k-t]+l\ge m$. This condition gives an efficient algorithm for potential match verification. Notice that for biological applications extended potential match data structure provides a useful tool for dot-matrices drawing without looking at all approximate matches generated by a given potential match.

7 Computational experiments

We have implemented the double filtration (Algorithm 3) and compared its performance with l-tuple filtration (Algorithm 1). Recent studies ([WM92b]) demonstrate that l-tuple filtration runs much faster than other approximate pattern matching algorithms. Our study indicates that double filtration outperforms l-tuple filtration for approximate substring matching in a wide range of

parameters. We presented the results of the computational experiments with the parameters $l=\lfloor\frac{m}{k+1}\rfloor$ and k as they are more convenient for comparison of running times than usual parameters m and k.

Algorithm 1 (l-tuple filtration) and Algorithm 2 (double filtration) were implemented in 'C' and all tests have been run on a SUN SparcStation 2 running UNIX. Stage 2 (potential match verification) was implemented in the same straightforward way in both Algorithm 1 and Algorithm 3. Our primary interest was to reveal the advantages and disadvantages of the filtration stage; that's why we ignored fast implementations of the verification stage. The numbers given in Figure 5 should be taken with caution. They depend on our program implementation, the architecture, the operating system, and the compilers used. However we tried to avoid optimizations and fancy programming implementations which might give an advantage to the double filtration over l-tuple filtration. The only difference between two programs was the implementation of the filtration stage.

Let $t_{fil}(t_{ver})$ be a running time of the filtration (verification) stage of the l-tuple filtration algorithm. Denote $e=\frac{E(p_1)}{E(p_3)}$ the ratio the filtration efficiency of double filtration to the filtration efficiency of l-tuple filtration. Roughly speaking a running time of double filtration algorithm will be $2\cdot t_{fil}+\frac{t_{ver}}{e}$. In the case

$$t_{fil}+t_{ver}>2\cdot t_{fil}+\frac{t_{ver}}{e}$$

double filtration is faster than l-tuple filtration. It means that in the case $e>\frac{t_{ver}}{t_{ver}-t_{fil}}$ double filtration might be better than l-tuple filtration. Figure 3 indicates that this is the case for various m and k as e is very large for a wide range of parameters. Figure 5 presents the results of comparisons for $q=10000$ and $n=100000$ indicating that double filtration might be better for a range of parameters frequently used for dot-matrices constructions and optimal oligonucleotide probes selection ($m=14,\ldots,30$, $k=1,\ldots,5$). Note that the ratio of the running time of the l-tuple filtration algorithm to the running time of the double filtration algorithm depends on $\frac{n}{q}$ (data are shown only for $\frac{n}{q}=10$)

8 Other filtration techniques

The basic idea of all l-tuple filtration algorithms suggested to date is to reduce a (m,k) approximate pattern matching problem to a $(m',0)$ exact pattern problem and to use a fast exact pattern matching algorithm on the filtration stage. The drawback of such approachs is relatively low filtration efficiencies. In this section we suggest reducing (m,k) approximate pattern matching to (m',k') approximate matching with $m'<m$ and $0<k'<k$ and application of the fast approximate pattern matching technique with small k' on the filtration stage. We demonstrate that this allows an increase of filtration efficiency without significant slowing down the filtration stage. For the sake of simplicity we illustrate this idea on a simple example reducing a (m,k) problem to a $(m',1)$ problem.

Knuth [K73] has suggested a method for approximate pattern matching with 1 mismatch based on the observation that strings differing by a single error must

k \ l	2	3	4	5	6	7	8	9	10
1		0.98	1.32	1.25	1.09	0.75	0.62	0.62	0.72
		89.6	*17.0*	*5.1*	*2.1*	*1.6*	*1.6*	*1.9*	*2.5*
2	0.87	1.27	1.55	1.58	1.23	0.75	0.62	0.62	0.72
	493.3	*84.8*	*17.6*	*4.8*	*2.1*	*1.6*	*1.6*	*1.9*	*2.5*
3	0.89	1.30	1.90	1.87	1.38	0.81	0.62	0.62	0.69
	574.5	*100.1*	*17.1*	*4.7*	*2.1*	*1.6*	*1.6*	*1.9*	*2.6*
4	0.90	1.31	1.92	2.27	1.52	0.87	0.62	0.62	0.69
	658.4	*114.5*	*19.4*	*4.4*	*2.1*	*1.6*	*1.6*	*1.9*	*2.6*
5	0.90	1.31	2.04	2.47	1.66	0.93	0.68	0.62	0.69
	746.0	*128.6*	*20.8*	*4.6*	*2.1*	*1.6*	*1.6*	*1.9*	*2.6*
6	0.91	1.31	2.14	2.62	1.85	1.00	0.68	0.62	0.65
	834.6	*144.2*	*22.2*	*4.8*	*2.1*	*1.6*	*1.6*	*1.9*	*2.7*
7	0.91	1.31	2.12	2.78	2.00	1.06	0.68	0.62	0.65
	921.4	*159.8*	*24.8*	*5.0*	*2.1*	*1.6*	*1.6*	*1.9*	*2.7*
8	0.91	1.31	2.15	2.82	2.19	1.12	0.75	0.62	0.70
	1011.1	*175.3*	*26.8*	*5.4*	*2.1*	*1.6*	*1.6*	*1.9*	*2.8*
9	0.91	1.29	2.25	3.13	2.28	1.12	0:75	0.62	0.70
	1098.5	*193.2*	*28.3*	*5.3*	*2.1*	*1.6*	*1.6*	*1.9*	*2.8*
10	0.91	1.29	2.20	3.12	2.47	1.18	0.75	0.62	0.70
	1189.5	*210.7*	*30.9*	*5.7*	*2.1*	*1.6*	*1.6*	*1.9*	*2.8*

Fig. 5. The comparison of the running time of the double filtration (Algorithm 3) and l-tuple filtration (Algorithm 1) for random Bernoulli words in 4-letter alphabet with $q = 10000$ and $n = 100000$ for different parameters k (number of mismatches) and $l = \lfloor \frac{m}{k+1} \rfloor$ (size of l-tuple). Lower cell on the intersection of k-th row and l-column represents the running time of the double filtration algorithm (in seconds). Upper cell on the intersection of k-th row and l-column represents the ratio of the running time of the l-tuple filtration algorithm to the running time of the double filtration algorithm. The area shown by a solid line represent the set of parameter (k, l) for which double filtration outperforms l-tuple filtration.

match exactly in either the first or the second half. For example, $(9, 1)$ approximate pattern matching problem can be reduced to $(4, 0)$ exact pattern matching problem. This provides an opportunity for 4-tuple filtration algorithm. In this section we demonstrate how to reduce $(9, 1)$ approximate pattern matching to a 6-tuple filtration algorithm thus increasing the filtration efficiency by a factor of $\frac{4^2}{2}$.

Let $(l_1, g_1, l_2, g_2, \ldots, l_t, g_t, l_{t+1})$-tuple be an tuple having l_1 positions followed by a gap of length $g_1 + 1$, then l_2 positions followed by a gap of length $g_2 + 1$,

Fig. 6. Example of (4,1,2,3,3)-tuple.

..., then l_t positions, followed by a gap of length $g_t + 1$ and finally l_{t+1} positions (Fig.6). Fig.7 demonstrates that every boolean word of length 9 with at most 1 zero contains either a continuous 6-tuple or a $(3, 3, 3)$-tuple containing only ones. Two 6-tuples and one $(3, 3, 3)$-tuple shown in Fig.7 are *packed* into 9-letter word v so that every position in v belongs to exactly two of these tuples. Therefore the only zero in v belongs to two of these tuples leaving the third. In Fig.7 the (3.3,3)-tuple contains only ones.

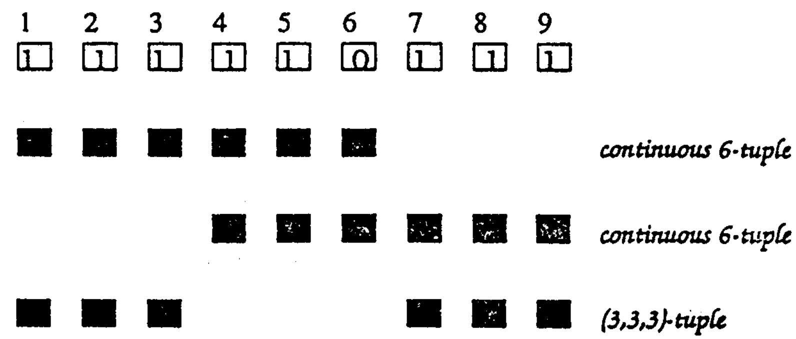

Fig. 7. A boolean word of length 9 with only zero contains either continuous 6-tuple or $(3, 3, 3)$-tuple containing only ones.

The following lemma generalizes the example above and allows one to perform $\lfloor \frac{2}{3} m \rfloor$-tuple filtration instead of $\lfloor \frac{1}{2} m \rfloor$-tuple filtration in Algorithm 1, thus increasing filtration efficiency approximately $\frac{4^{\frac{m}{6}}}{2}$ times.

Lemma 9. *A boolean word $v[1, \ldots, m]$ with at most 1 zero contains either a continuous $\lfloor \frac{2}{3} m \rfloor$-tuple or a $(\lfloor \frac{1}{3} m \rfloor, \lfloor \frac{1}{3} m \rfloor, \lfloor \frac{1}{3} m \rfloor)$-tuple containing only ones.*

The following lemma generalizes lemma 2 and reduces the (m, k) approximate pattern matching problem to the (m', k') problem with $m' < m, k' < k$.

Lemma 10. *A boolean word $v[1, \ldots, m]$ with at most k zeros contains a subword of length m' with at most $k' < k$ zeros for $m' = \lfloor \frac{(k'+1) \cdot m + k'}{k + k' + 1} \rfloor$.*

Proof. Fix $0 < t < m$ and consider all $m - t + 1$ subwords of v of length t. Every position in v belongs to at most t of these t-words. Therefore the total number of zeros in these t-words is $z \leq k \cdot t$.

If all t-subwords of v contain at least $k' + 1$ zeros then the total number of zeros in these t-words is $z \geq (k' + 1) \cdot (m - t + 1)$ and therefore

$$k \cdot t \geq z \geq (k' + 1) \cdot (m - t + 1)$$

If this inequality fails then there exists a t-subword of v containing less than $k' + 1$ zeros. Therefore the maximum t fulfilling the inequality

$$k \cdot t < (k' + 1) \cdot (m - t + 1)$$

provides the upper bound for the length of subword containing at most k' zeros.

$$t < \frac{(k' + 1) \cdot (m + 1)}{k + k' + 1}.$$

□

Substituting $k' = 1$ in the last lemma provides a reduction of (m, k) approximate pattern matching problem to $(\lfloor \frac{2 \cdot m + 1}{k+2} \rfloor, 1)$ approximate pattern matching problem. Lemma 9 allows further implementation of filtration with $\lfloor (\frac{2}{3} \lfloor \frac{2 \cdot m + 1}{k+2} \rfloor) \rfloor$-tuples. For large m lemmas 9 and 10 allows one to implement l-tuple filtration with $l \approx \frac{4}{3} \frac{m}{k}$ which improves the filtration of Algorithms 1 and 3 with $l \approx \frac{m}{k}$.

Finally, there is no approximate pattern/substring matching algorithm that is the best for all possible cases. It is an open problem to find the optimal filtration techniques depending on the parameters and applications. Note that the proposed methods does not support insertions and deletions. This motivates the problem of finding an efficient filtration technique for approximate pattern matching with k differences. To solve this problem Myers, 1990 ([M90]) proposed a related method based on a reduction of the $(m, \epsilon m)$ approximate pattern matching problems with a database of length n to the $(\log n, \epsilon \log n)$ pattern matching problems. The method requires a prebuilt inverted index and so is an off-line algorithm while all the others mentioned here are on-line. This technique provides approximate pattern matching with k differences in sublinear time and gives 50- to 500-fold improvement over dynamic programming algorithms for approximate pattern matching ([U85], [MM86]).

9 Acknowledgements

We are grateful to William Chang, Udi Manber and Gene Myers for useful suggestions.

References

[BG89] Baeza-Yates R.A., Gonnet G.H. A new approach to text searching. in *Proc. of the 12th Annual ACM-SIGIR conference on Information Retrieval*, Cambridge, MA, (1989), 168-175

[BP92] Baeza-Yates R.A., Perleberg C.H. Fast and practical approximate string matching. In A.Apostolico, M.Crochermore, Z.Galil, U.Manber (eds.) Combinatorial Pattern Matching 92, Tucson, Arizona, *Lecture Notes in Computer Science*, 644, Springer-Verlag, (1992) , 185-192

[B86] Blaisdell B.E. A measure of the similarity of sets of sequences not requiring sequence alignment. *Proc. Nat. Acad. Sci. U.S.A.*, 83, (1986), 5155-5159.

[CL90] Chang W.I., Lawler E.L. Approximate string matching in sublinear expected time. *Proceedings of 31st IEEE FOCS*, (1990), 116-124

[DMDC87] Danckaert A., Mugnier C., Dessen P., and Cohen-Solal M. A computer program for the design of optimal synthetic oligonucleotides probes for protein coding genes. *CABIOS*, 3, (1987) 303-307.

[DN82] Dumas, J.P., Ninio, J. Efficient algorithms for folding and comparing nucleic acid sequences. *Nucl. Acids Res.*, 10, (1982), 197-206.

[F70] Feller W. *An introduction to probability theory and its applications.* John Wiley & Sons, New York, (1970)

[GG86] Galil, Z. and Giancarlo, R. Improved string matching with k mismatches. *SIGACT News*, April, (1986), 52-54.

[GL90] Grossi R., Luccio F. Simple and efficient string matching with k mismatches. *Information Processing Letters*, 33, (1990), 113-120

[H71] Harrison M.C. Implementation of the substring test by hashing. *C.ACM*, 14, (1971), 777-779

[HS91] Hume A., Sunday D. Fast string searching. *Software - Practice and Experience*, 21, (1991), 1221-1248

[KR87] Karp R.M., Rabin M.O. Efficient randomized pattern-matching algorithms. *IBM J. Res. Develop.*, 31, (1987), 249-260

[KS92] Kim J.Y. Shawe-Taylor J. An approximate string matching algorithm. *Theoretical Computer Science*, 92, (1992), 107-117

[K73] Knuth D.E. *The art of computer programming, vol.III: sorting and searching.* Addison-Wesley, Reading, Mass., (1973)

[LV86] Landau G.M., Vishkin U. Efficient string matching with k mismatches, *Theoret. Computer Sci.*, 43, (1986), 239-249

[LV89] Landau G.M., Vishkin U. Fast parallel and serial approximate string matching. *J. of Algorithms*, 10, (1989), 157-169

[LVN88] Landau, G.M., Vishkin, U., and Nussinov, R. Locating alignments with k differences for nucleotide and amino acid sequences. *CABIOS*, 4, (1988), 19-24.

[LP85] Lipman, D.J., Pearson, W.R. Rapid and sensitive protein similarity searches. *Science*, 227, (1985), 1435-1441.

[ML81] Maizel, J. V., Jr. and Lenk, R.P. Enhanced graphic matrix analysis of nucleic acid and protein sequences. *Proc. Nat. Acad. Sci. USA*, 78, (1981), 7665-7669.

[MM86] Myers E.W., Mount D. (1986) Computer program for the IBM personal computer that searches for approximate matches of short oligonucleotide sequences in long target DNA sequences. *Nucleic Acids Research*, 14, 501-508

[M90] Myers E.W. (1990) A sublinear algorithm for approximate keyword searching. Technical Report TR-90-25, Department of Computer Science, Thwe University of Arizona, Tucson, Arizona. (to appear in *Algorithmica*)

[OM88] Owolabi O., McGregor D.R. Fast approximate string matching. *Software-Practice and Experience*, 18, (1988), 387-393

[TU90] Tarhio J., Ukkonen E. Boyer-Moore approach to approximate string matching *Lecture Notes in Computer Science, 447*, Springer, Berlin, (1990), 348-359

[U85] Ukkonen U. Finding approximate patterns in strings. *Journal of Algorithms*, 6, (1985), 132-137

[U92] Ukkonen U. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92, (1992), 191-211

[WL83] Wilbur W. J., Lipman D.J.. Rapid similarity searches of nucleic acid and protein data banks. *Proc. Nat. Acad. Sci. USA* , 80, (1983), 726-730.

[WM92a] Wu S., Manber U. Agrep - A Fast Approximate Pattern-Matching Tool. *Usenix Winter 1992 Technical Conference*, San Francisco (January 1992), (1992), 153-162.

[WM92b] Wu S., Manber U. Fast Text Searching Allowing Errors. *Comm. of the ACM*, 35, No.10 (1992),83-90