International Journal of Foundations of Computer Science

BDD-Based Analysis of Gapped q-Gram Filters

Marc Fontaine
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
fontaine@studcs.uni-sb.de

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

and

Juha Kärkkäinen
Department of Computer Science
P.O.Box 68 (Gustaf Hällströmin katu 2 B)
FI-00014 University of Helsinki, Finland
Juha.Karkkainen@cs.helsinki.fi

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

Recently, there has been a surge of interest in gapped q-gram filters for approximate string matching. Important design parameters for filters are for example the value of q, the filter-threshold and in particular the shape (aka seed) of the filter. A good choice of parameters can improve the performance of a q-gram filter by orders of magnitude and optimizing these parameters is a nontrivial combinatorial problem. We describe a new method for analyzing gapped q-gram filters. This method is simple and generic. It applies to a variety of filters, overcomes many restrictions that are present in existing algorithms and can easily be extended to new filter variants. To implement our approach, we use an extended version of BDDs (Binary Decision Diagrams), a data structure that efficiently represents sets of bit-strings. In a second step, we define a new class of multi-shape filters and analyze these filters with the BDD-based approach. Experiments show that multi-shape filters can outperform the best single-shape filters, which are currently in use, in many aspects. The BDD-based algorithm is crucial for the design and analysis of these new and better multi-shape filters. Our results apply to the k-mismatches problem, i.e. approximate string matching with Hamming distance.

Keywords: approximate string matching, filter, gapped q-gram, BDD

1. Introduction

String matching involves searching a given string or textual database T for occurrences of substrings that match a search pattern P. The approximate string matching problem allows the search pattern and the matches to have some difference or distance according to a given distance function.

Many applications depend on efficient solutions of this problem, especially in the field of bio-informatics, where databases may consist of sequences of 10^9 nucleotides of DNA or of long sequences of amino acids.

Filter algorithms are a common approach for approximate string matching. They speedup string matching by quickly generating a set of potential matches and discarding the rest of the database. The true matches can then be found in a second step, the verification phase, by inspecting all potential matches. Designing a good filter usually means optimizing the tradeoff between the complexity of the filtration phase and the efficiency of the filter.

Many efficient filters work with precomputed indexes, in particular, indexes that are based on *gapped q-grams* or *shapes*. For example the three 3-grams of string `ACAGCT` for shape `##-#` are `AC-G`,`CA-C` and `AG-T`. A matching pair of q-grams between a pattern and a substring of T is called a *hit*. The q-gram index stores the positions of all q-grams of the database and allows to find hits efficiently. If the number of hits between the search pattern and a substring of the database exceeds a certain *threshold* t, that substring is called a *potential match*.

As first shown in [6, 7], the performance of the filter depends crucially on the shape. Good shapes are found by analyzing large sets of shapes, because no method for directly generating good shapes has been found yet. Even analyzing a single shape is non-trivial and a lot of effort has gone into developing methods for this purpose. Recently gapped shapes have been the focus of quite a bit of attention [16, 9, 10, 12].

In [6, 7], Burkhardt and Kärkkäinen compute the *optimal threshold*. It is the highest threshold that still allows the filter to return all true matches, i.e., substrings that are within a fixed Hamming distance from the pattern. They also compute a measure called the minimum coverage, which provides a rough estimate of how many false matches get through the filter. The true positive rates and the false positive rates were determined experimentally for selected shapes.

If the threshold or Hamming distance is increased, the filter also discards some true matches, which are then called *false negatives*. In [5] an abstract measure for the false negative probability, the so-called *recognition rate* was defined and analyzed experimentally. The exact computation of both false positive and false negative rates was done by Ma, Tromp and Li [15]. However, their algorithms are restricted to filters that count only non-overlapping hits. This is a significant restriction as the lower correlation of overlapping q-grams is a big advantage of gapped q-grams over un-gapped ones [7].

Brejová, Brown and Vinař [1, 2] developed new variants of the algorithm of Ma, Tromp and Li. In [1], a true match is defined not directly by a Hamming distance but as a probability distribution represented by a hidden Markov model. In [2], they further generalized the approach to approximate hits and multiple shapes. However, the restriction to non-overlapping hits remains in all of their work. Very recently, we became aware of several papers using multiple gapped shapes for approximate string matching in various different approaches [13, 17, 18, 14].

We present a new and flexible method for computing various properties of q-gram filters. This algorithm is based on a simple natural abstraction of the problem and applies to a general class of filters. At the same time, it overcomes many restrictions present in previous algorithms in particular the non-overlapping hit restriction.

Our method consists of two steps. The first step is an algorithm based on sets of bit-strings. These sets can be of exponential size and we have to use a compact and efficient representation of the sets to actually implement our algorithm. A data structure called BDDs (Binary decision diagrams or Binary decomposition diagrams) implements such a representation of sets [3, 4]. Only the use of a data structure like BDDs makes it feasible to run our algorithm.

In the second step the BDDs generated in the first step can be used to efficiently compute interesting properties of the sets they represent. Most properties can be computed in linear time of the size of the BDDs. This clear split of the problem into two steps distinguishes our method from previous algorithms, which were mostly based on dynamic programming.

In the second part of this work, we apply our method to design new and better filters. The basic idea in this part is to filter with a set of shapes simultaneously. Multi-shape filters have been used before [8]. Our new idea is to use a carefully selected set of shapes together with a specifically computed *filtration criterion*. This filtration criterion replaces the optimal threshold of a single shape filter.

The BDD-based algorithm allows us to compute the best filtration criterion for a set of shapes and at the same time determine the important quality measures of the resulting multi-shape filter. To investigate the potential of multi-shape filters based on a specific filtration criterion, we analyze large sets of randomly generated filters. These experiments show that good multi-shape filters are very rare, but the experiments also yield filters that are superior to single-shape filters in several important aspects.

2. Representing Match-Mismatch-Patterns with BDDs

Let A and B be two strings of length l. We call the bit-string $p(A, B) \in \{0, 1\}^l$ the match-mismatch-pattern of the two strings. A "1" in $p(A, B)$ denotes a matching position and "0" a mismatch. The Hamming distance of A and B is then the number of zeros in $p(A, B)$. We represent the number of zeros and ones in a bit-string p by $|p|_0$ and $|p|_1$. The *k-differences* version of approximate string matching allows a pattern string and a match to have a Hamming distance of at most k.

For most filters, the match-mismatch-pattern of an alignment between the pattern string and the database at some position x contains enough information to decide whether x is returned as a potential match or not. Therefore it is, in principle, sufficient to enumerate all possible match-mismatch-patterns to analyze the performance of filters for the *k-differences* problem.

A drawback of this brute-force-approach is, that there are 2^l possible match-mismatch-patterns for strings of length l and realistic filters usually work with pattern length $l \geq 50$.

To overcome this complexity-problem we use a data structure called BDDs. BDDs allow a compact and efficient representation of sets of equal-length bit-strings. They can be seen as an abstract data structure that supports the following operations:

- Creation of a new BDD for the base-cases $\emptyset$ and $\{\epsilon\}$.
- Composition of two BDDs $S = comp(S_0, S_1)$.
- Decomposition of a BDD into two BDDs according to the first position of the bit-strings in the BDD.
- Computing $\cup$ and $\cap$ of two BDDs and the complement $\neg$ of a BDD.

The composition $comp(S_0, S_1)$ represents the set:

$$comp(S_0, S_1) = \{s \mid (s = 0a \wedge a \in S_0) \vee (s = 1b \wedge b \in S_1)\}$$

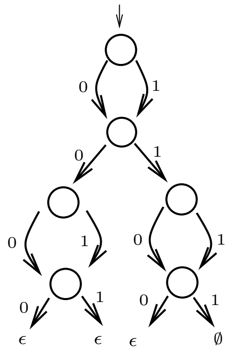
For two sets A and B that are given as decompositions $A = comp(A_0, A_1)$ and $B = comp(B_0, B_1)$, $A \cup B$ and $A \cap B$ can be computed recursively as:

$$A \cup B = comp(A_0 \cup B_0, A_1 \cup B_1)$$

and:

$$A \cap B = comp(A_0 \cap B_0, A_1 \cap B_1)$$

BDDs are implemented with DAGs (Directed Acyclic Graphs) and they are similar to finite automata without loops. In a BDD always the minimal, smallest possible DAG is used to represent a set of bit-strings and equal sets are represented by one canonical node of the DAG. A collection of BDDs can share the structure of a single DAG and BDD-implementations usually make use of hash-tables to maintain the canonical-representation-property. Hash tables are also used to avoid re-computations during the computation of $\cup$ and $\cap$.



0000
0001
0010
0011
0100
0110
1000
1001
1010
1011
1111
1011

A simple BDD and the set it represents

BDDs have many different applications where they often make it possible to handle exponential size sets within non-exponential complexity. An introduction to BDDs can be found in [3, 4], where BDDs are used to represent boolean functions over a finite set of variables. The actual performance of BDDs in an application depends on the structure of the sets they represent. For sets of size 2^l the space complexity of BDDs can range from $O(l)$ to $\Theta(2^l)$.

It is important to note, that we use BBDs only to analyze q-gram filters. This is a one-time computation and the complexity of the BDDs does not interfere with the complexity of the filters under consideration. The theoretical complexity of BDDs is therefore secondary in our application. Our experience is, that BDDs work well to reduce the complexity of filter analysis. They make is possible to analyze all interesting filters within reasonable time and space limits.

In [11] Fontaine described a extension of standard BDDs, which uses $\{0,1\}^*$ as an additional base case for the decomposition (so called $*$-BDDs). This extension allows more compact representation than standard BDDs and it was used for all our experiments. For code listings and runtime measurements of a prototype $*$-BDD implementation see [11]. The prototype implementation only consists of about 10kb of C++ code and computing the filter properties for a typical shape only takes a few seconds.

3. q-Gram Similarity-Based Filters

We use strings from $\{\texttt{\#},\texttt{-}\}^*$ to denote different shapes. `#` stands for a position that must match, whereas `-` is a “don’t care” or wild-card position. $span(s) = |s|$ is the span of a shape s.

Let A and B be two strings of length l and s a shape. A position $0 \leq i \leq l - span(s)$ is a hit of shape s if $\forall\, 0 \leq n < span(s) : s(n) = \texttt{\#} \Rightarrow A(i+n) = B(i+n)$.

The number of different hits of a shape s for two strings A and B is called the q-gram similarity $qgs_s(A,B)$ of the two strings. For strings of length l the q-gram-similarity can be at most $l - span(s) + 1$.

$A=$	`ACTGTACTGCCGTACT`
$B=$	`ACTGTAATGCAGTACT`
$p(A,B)=$	`1111110111011111`
shape $s=$	`###---##--##`
$qgs_s(A,B)=$	2

```
ACTGTACTGCCGTACT
###---?#--?#
 ###---##--##  <- hit
  ###---##--## <- hit
   ###---#?--##
    ##?---?#--##
ACTGTAATGCAGTACT
```

Match-mismatch-pattern and q-gram-similarity

A q-gram filter computes the set of potential matches with the help of a threshold t. A potential match is the position of a substring in the database with a q-gram similarity of at least t with the pattern string. Increasing the threshold of a filter reduces the number of potential matches at the cost of a decreased filter sensitivity, i.e. the filter is more likely to overlook true matches.

The match-mismatch-pattern of two strings contains sufficient information to compute their q-gram-similarity. Therefore a filter can be analyzed by looking at all possible match-mismatch-patterns. We can partition the set of all possible match-mismatch-patterns according to the q-gram-similarity they represent for a given shape.

For any fixed shape s and any $h, l \in \mathbb{N}_0$ we define:

$$P_l^h = \{p \in \{0,1\}^l \mid s \text{ produces exactly } h \text{ hits in } p\}$$

It follows that the set PM of match-mismatch-pattern that represent a *potential match* is:

$$PM = \bigcup_{h \geq t} P_l^h$$

A set P_l^h can easily be computed based on the sets P_{l-1}^{h-1} and P_{l-1}^h. A match-mismatch-pattern $p \in \{0,1\}^l$ is in P_l^h if:
either: its suffix of length $l-1$ is in P_{l-1}^{h-1} and it has an additional hit of shape s at position 0
or: its suffix of length $l-1$ is in P_{l-1}^h and it does not have an additional hit at position 0.

This algorithm can be formulated as a simple equation for sets

$$P_l^h = (\mathit{expand}(P_{l-1}^{h-1}) \cap S_l(s)) \cup (\mathit{expand}(P_{l-1}^h) \cap \bar{S}_l(s))$$

with the following three definitions:

$$S_l(s) = \{p \in \{0,1\}^l \mid s \text{ has a hit in } p \text{ at position } 0\}$$

$$\bar{S}_l(s) = \{p \in \{0,1\}^l \mid s \text{ does not have a hit in } p \text{ at position } 0\}$$

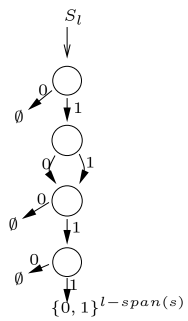
$$\mathit{expand}(M) = \{x \mid x = 0m \vee x = 1m, m \in M\}$$

BDDs directly support $\cup$ and $\cap$, and $\mathit{expand}(M)$ can be implemented as $\mathit{expand}(M) = \mathit{comp}(M, M)$. BDDs also support the creation of $S_l(s)$ and $\bar{S}_l(s)$ for any shape s. $S_l(s)$ can be computed recursively as:

$$S_l(s) = \begin{cases} \{0,1\}^l & \text{if } s = \epsilon \\ \emptyset & \text{if } l < \mathit{span}(s) \\ \mathit{comp}(\emptyset, S_{l-1}(r)) & \text{if } s = \texttt{\#}r \\ \mathit{comp}(S_{l-1}(r), S_{l-1}(r)) & \text{if } s = \texttt{-}r \end{cases}$$

Shape=`#-##`
$P_4^1 = S_4 = \{1011, 1111\}$
$\mathit{expand}(P_4^1) = \{01011, 01111, 11011, 11111\}$

$P_5^1 = \{01011, 01111, 11011, 10110, 11110, 10111\}$
$P_5^2 = \{11111\}$
$P_5^0 = \{00000, 00001, 00010, \ldots\}$



P_l^h and $S_l(s)$ for shape `#-##`

As an alternative to our definition of the q-gram-similarity $qgs(A, B)$ it is possible to require individual hits to be non-overlapping [15, 1]. For such filters the set PM can be computed with an algorithm similar to the one described above. (Compute the sets $P_l^{(h,i)}$, where i is the offset of the first hit.)

4. Filter Analysis with BDDs

The algorithm described in the previous section allows us to generate BDD-representations for the sets P_l^h. These BDD-representations can be used to compute many interesting properties of the sets and thereby the underlying filters. Note that the computation of the various properties is independent of what filter the sets P_l^h represent and how they were computed. This is in contrast to previous approaches using dynamic programming where the filter definition is deeply involved in the property computation.

4.1. Specificity

The specificity of a filter describes its ability to reduce a large database to a small set of potential matches. For a given random model, the filter specificity is equivalent to the probability that a random substring of length l is a potential match of a random search pattern.

Every match-mismatch-pattern p describes one possible event that can occur while aligning a database and a search pattern and we can use several probability models to assign probabilities to these events. We can then simply extend these probabilities from one match-mismatch-pattern to sets of match-mismatch-patterns by summing up the probabilities of the elements of the sets.

For example, to analyze a filter for a DNA database, we might assume that the database and pattern string are independent random strings with an even distribution of the letters $\{A, C, G, T\}$. It follows that every single character has a $\frac{1}{4}$ chance of being a match and the probability of any match-mismatch-pattern p is:

$$prob(p) = (\frac{1}{4})^{|p|_1}(\frac{3}{4})^{|p|_0}$$

With this we can compute the probability of a potential match, i.e the specificity of the filters as:

$$specificity = \sum_{p \in PM} prob(p)$$

Given the binary decomposition $comp(P_0, P_1)$ of a set P the probability $Prob(P)$ of the set is:

$$Prob(P) = (\frac{3}{4}) * Prob(P_0) + (\frac{1}{4}) * Prob(P_1)$$

The base-cases for the binary decomposition are also the base-cases for this recursion:

$$Prob(\emptyset) = 0 \quad Prob(\epsilon) = 1$$

This shows that, if BDDs are used to represent the sets, $Prob(P) = \sum_{p \in P} prob(p)$ can be computed in linear time of the size of the BDDs.

It can be seen that a similar approach allows to compute the probabilities of sets for many different probability models efficiently. In particular it is also possible to use hidden Markov models (HMMs) as probability model. HMMs have been used in [1] to model real DNA sequences of different species.

4.2. Recognition Rate

For approximate string matching with Hamming distance we can define the recognition rate $r(j)$ of a filter as the expected fraction of potential matches among substrings of the database with *exactly* Hamming distance j. The match-mismatch-patterns of length l and Hamming distance j can easily be computed with the single-character shape `#` as $P_l^{l-j}(\texttt{\#})$. It follows that a filter with potential matches PM has the recognition rate:

$$r(j) = \frac{Prob(PM \cap P_l^{l-j}(\texttt{\#}))}{Prob(P_l^{l-j}(\texttt{\#}))}$$

Recognition rates have been defined and determined experimentally in [5].

4.3. Threshold

The set of potential matches of a filter with shape s, and with it the recognition rates of the filter, heavily depends on the threshold t.

A filter is *lossless* for a threshold t and Hamming distance k if $\forall j \leq k : r(j) = 1$, otherwise it is lossy. If one is interested in a fixed maximal Hamming distance k and lossless filtering, then there exists an optimal threshold t_{best}. A dynamic programming algorithm for computing t_{best} is described in [7].

BDD-based threshold computation is also possible. For each set P_l^h we compute:

$$m(P) = \min_{p \in P} |p|_0$$

We use the notation $|p|_0$ for the number of occurrences of "0" in string p. $m(P_l^h)$ is the minimum number of mismatching positions of any match-mismatch-pattern $p \in P_l^h$. This minimum can be found in linear time in the size of the BDD. Any set P_l^h with $m(P_l^h) \leq k$ contains at least one match-mismatch-pattern with Hamming distance at most k. The optimal threshold t_{best} for a lossless filter is the smallest h such that $m(P_l^h) \leq k$.

shape $s =$ `#-#---#-#-#------#`
$span(s) = 18$
pattern length $l = 50$
number of hits $h \in \{0, \ldots, 33\}$

h	0	1	2	3	4	5	6	7	8	9	...	31	32	33
$m(P_l^h)$	8	7	7	6	6	6	5	5	5	4	...	1	1	0
$k = 7$		$t_{best} = 1$												
$k = 6$				$t_{best} = 3$										
$k = 5$							$t_{best} = 6$							
$k = 4$										$t_{best} = 9$				

5. Multi-shape Filters

Shapes can be better than contiguous q-grams because they introduce irregularity in the way the mismatching positions affect the q-grams. For good shapes, only a few worst case configurations of the mismatching characters affect many q-grams. A reasonable approach to further improve the performance of filters is therefore to use two or more somehow *orthogonal* shapes in parallel. The idea is, that those configurations of mismatches, that are particularly bad for one shape, are better covered by a second shape and vice versa.

Designing a good multi-shape filter is a nontrivial combinatorial problem, just like finding good individual shapes. One could assume that the best individual shapes also form the best multi-shape filter, however our experiments suggest that this is often not the case.

Multi-shape filters are the most important application for our BDD-based approach. The extension of our algorithm to multi-shape filters is straight-forward and it leads to a new concept: the generic *filtration criterion* C. The generic filtration criterion C replaces the threshold t of a single-shape filter. It enables a multi-shape filter to make full use of the relations between the single shapes.

A filter with n shapes $s_1 \dots s_n$ can use the q-gram similarities $h_1 = qgs_{s_1}(M,P) \dots h_n = qgs_{s_n}(M,P)$ to decide whether M is a potential match or not. (P is the pattern string and M is any substring of the database.) We call a set $C \subset \mathbb{N}^n$ a *filtration criterion* for the shapes $s_1 \dots s_n$ and define:

$$M \text{ is a potential match } \Leftrightarrow (h_1, \dots, h_n) \in C$$

This generic filtration criterion C can model many different strategies for multi-shape filters. For example it can model filters that require at least one hit of one shape, filters the require one hit of each shape, filters that sum up the hits of the shapes, or filters that use each shape with its individual threshold t_{best}.

In Section 3 we used the notation $P_l^h(s)$ for the set of all match-mismatch-patterns with exactly h hits of a single fixed shape s. To analyze multi-shape filters we extend this notation to sets of shapes $\{s_1, \dots, s_n\}$. We define $P_l^{(h_1,\dots,h_n)}(s_1, \dots, s_n)$ as the set of all match match-mismatch-patterns with exactly h_i hits of shape s_i ($1 \le i \le n$). The sets $P_l^{(h_1,\dots,h_n)}(s_1, \dots, s_n)$ can be computed as:

$$P_l^{(h_1,\dots,h_n)}(s_1, \dots, s_n) = \bigcap_{1 \le i \le n} P_l^{h_i}(s_i)$$

With this, the set of match-mismatch-patterns, that represent a potential match according to a filtration criterion C is:

$$PM = \bigcup_{(h_1,\dots,h_n) \in C} P_l^{(h_1,\dots,h_n)}(s_1, \dots, s_n)$$

Together with the set PM, all statistical performance measures (recognition rate, specificity), which we computed for single-shape filters in Section 3, are now also available for our model of multi-shape filters.

The definition of $P_l^{(h_1,\ldots,h_n)}(s_1,\ldots,s_n)$ also makes it possible to compute an *optimal filtration criterion* C_{best} for a lossless filter with some fixed Hamming distance k. It is:

$$C_{best} = \{(h_1 \ldots h_n) \mid m(P_l^{(h_1,\ldots,h_n)}(s_1,\ldots,s_n)) \leq k\}$$

C_{best} replaces the threshold t_{best} of single shape filters. To reduce the high complexity involved in the computation of C_{best} Fontaine [11] describes a straight forward approximation.

6. Designing Better Filters

The design of a filter is always a compromise between three objectives:

- high sensitivity
- fast filtration phase
- high specificity of the filter, i.e. a fast verification phase

There are several trade-offs between these objectives. For example, a higher sensitivity is usually at the cost of a lower specificity and a faster filtration often yields lower sensitivities and specificities [5, 15, 8].

Using a well chosen shape for the q-grams and the appropriate threshold can greatly improve overall filter performance compared to filtering with un-gapped q-grams [6, 7]. In this section we will show that multi-shape filters with a carefully selected set of shapes and a specifically computed filtration criterion can further boost filter performance for all three objectives compared to single-shape filters.

A good estimate for the runtime of a q-gram filter is the number of hits in the database that have to be processed. It is roughly proportional to $|\Sigma|^{-q}$. (This assumes a database with a random distribution of letters from Σ and it is also a good estimate for example for DNA sequences [7].) High values of q are desirable because they make the filtration fast however they also mean lower sensitivities.

In this section we only consider q-gram filters that work *lossless* for a fixed Hamming distance k and we use k to compare the sensitivities of such filters (a higher value of k means a higher sensitivity). To compare the specificities of different filters, we always use the shapes with the optimal threshold t_{best} (the optimal filtration criterion C_{best} for multi-shape filters) that still guarantees lossless filtering for the fixed k. For all experiments in this section, we use a pattern length $l = 50$ and assume a DNA-like database with $|\Sigma| = 4$.

There is a trade-off between k and the highest value of q that can be used for lossless filtering. For example for pattern length $l = 50$ and $k = 5$ the highest possible q for a lossless single-shape filter is $q = 10$, for $k = 6$ it is $q = 9$. Similar constraints between q and k also exist for multi-shape filters. However we found that they can have higher values of *both q and k* than is possible for single shapes.

Therefore multi-shape filters make it possible to increase q, which makes them faster, or increase k, i.e the sensitivity. In some cases it is even possible to increase q and k at the same time. This is not at the cost of a lower specificity, but instead it is even possible to increase the specificity also.

Pairs of shapes: $q = 10$, $k = 6$

Consider for example the following three lossless two-shape filters for $k = 6$:

Three good two-shapes filters			
$k = 6$, $l = 50$,$\|\Sigma\| = 4$			
	s_1	s_2	$specificity$
a)	`##-##---##-####`	`###-#-###----#--##`	$8.091782 * 10^{-8}$
b)	`#-##-###-####`	`####----###--##-#`	$9.306443 * 10^{-8}$
c)	`#-##-##--#####`	`####--#--#---##--##`	$7.763605 * 10^{-8}$

	$\neg C_{best}$
a) and c)	$\{(0,0),(0,1),(1,0),(1,1),(2,0)\}$ not $(0,2)$!
b)	$\{(0,0),(0,1),(0,2),(1,0),(1,1),(2,0)\}$

The best single shape filter for this problem is `######-#-##` with $q = 9$, $t_{best} = 2$ and a specificity of $3.838350 * 10^{-6}$. Compared to this single shape filter, each of these three multi-shape filters with two ($q = 10$)-shapes improves the specificity by a factor of 50. The runtime of a multi-shape filter is approximately the sum of the run-times computed for its individual shapes. This means that the filters with two ($q = 10$)-shapes for $|\Sigma| = 4$ are also about two times as fast as a $q = 9$ single-shape filter.

The three multi-shape filters of this example were found by scanning 5000 pairs of random ($q = 10$)-shapes. In this sample set, good pairs were extremely rare. 3439 of the pairs, i.e. more than two-thirds, did not yield a lossless filter for $k = 6$ at all. It is interesting that none of the six shapes that comprise the three best pairs, we found, work particularly well as a single-shape filter. This suggests that combining good single-shape filters is not necessarily the best method to construct a good multi-shape filter. Also note, that 5000 pairs of shapes is a relatively small random set. It is very likely that much better two-shape filters can be found with more extensive experiments.

4-tuples of shapes: $q = 10$, $k = 8$

In a second experiment we fixed q to 10 and tried to increase k. We generated $1,500,000$ filters with 4-tuples of random ($q = 10$)-shapes and analyzed each of these 4-tuples with our algorithm. In this sample set, the good 4-tuple filters were again rare. Nevertheless, we found 15 lossless filters for $k = 8$ with specificities of about 10^{-3}. For comparison, the highest possible k for a lossless single-shape filter with $q = 9$ is $k = 6$. (A single-shape filter with $q = 9$, is about as fast as our 4-tuple filters.) The filtration criterion of the 15 4-tuples filters for $k = 8$ is that they require at least one hit of any of the four shapes.

4-tuples of shapes: $q = 10$, $k = 7$

Alternatively, each of the 15 good 4-tuple filters, we found, can also be used for $k = 7$ with a stricter filtration criterion. Although the computation of the exact filtration criterion C_{best} for this problem has a high complexity, it is easy to compute an suitable approximation C_{approx} [11]. The complement $\neg C_{approx}$ of one such approximation consists of 40 elements. This filtration criterion $\neg C_{approx}$ guarantees lossless filtering for $k = 7$ and a specificity of $5.2288 * 10^{-8}$.

The best set of four shapes out of 1,500,000 random	
$l = 50$, $k = 7$, $specificity = 5.228823e - 08$	
s_1=##-#-###---#--#-----##	s_2=##-#--#--#-#-#--###
s_3=###-#-##-#-###	s_4=##-###-----##--#-#--#

$\neg C$ for the best 4-tuple filter and $k = 7$
$\{(0,0,0,0), (0,0,0,1), (0,0,0,2), (0,0,0,3), (0,0,0,4), (0,0,0,5), (0,0,1,0),$ $(0,0,1,1), (0,0,1,2), (0,0,2,0), (0,0,2,1), (0,0,3,0), (0,0,3,1), (0,0,4,0),$ $(0,0,4,1), (0,0,5,0), (0,1,0,0), (0,1,0,1), (0,1,0,2), (0,1,1,0), (0,1,1,1),$ $(0,1,2,0), (0,1,3,0), (0,1,4,0), (0,2,0,0), (0,2,0,1), (0,2,1,0), (0,3,0,0),$ $(1,0,0,0), (1,0,0,1), (1,0,0,2), (1,0,0,4), (1,0,1,0), (1,0,2,0), (1,0,4,0),$ $(1,1,0,0), (1,1,0,1), (1,2,0,0), (2,0,0,0), (5,0,0,0)\}$

The experiments show that multi-shape filters can have significantly better specificities and work for higher values of k than single-shape filters. At the same time they can also speed up the filtration. It remains an open question if there is an algorithm to construct good sets of shapes for multi-shape filters.

7. Conclusion

We described a new method for the analysis of gapped q-gram filters. This method uses bit-strings, we call them match-mismatch-patterns, to describe possible alignments between the database and the search patterns. Sets of match-mismatch-patterns provide a simple abstraction of filter algorithms for the k-differences problem.

The first step of our approach is to generate sets of match-mismatch-patterns, in particular the set of match-mismatch-patterns representing the potential matches. To implement this step efficiently, we use BDDs as a data structure to represent sets of bit-strings. In the second step, we can then use these BDD representations to compute many interesting properties of filters like the recognition rate and specificity for various probability models.

Our approach is simple and general and applies to a variety of filter algorithms. For example, it can model single-shape filters with any threshold and generic multi-shape filters. Previous algorithms for filter-related problems were often based on dynamic programming. Compared to dynamic programming, our approach is more general and more natural and allows many interesting extensions.

The most important application of our approach is the analysis of multi-shape filters, which work with a set of shapes in parallel. For any set of shapes, our approach can compute an optimal filtration criterion C_{best}, which guarantees lossless filtering for the k-differences problem and also the sensitivities and specificities of the resulting multi-shape filter.

We found, that good multi-shape filters with a carefully selected set of shapes and a specifically computed filtration criterion C_{best} are much better than single-shape filters. They allow higher specificities and sensitivities than single shape filters and higher values of k are possible (for lossless filtering). Multi-shape filters can also be faster than single shape filters, because they still work with higher values of q.

The BDD-based approach makes it possible to find good multi-shape filters by scanning a large number of randomly generated candidates. However, only a small fraction of these candidates show the desired properties. Since full enumeration as for single-shape filters [6] is not possible for multi-shape filters, a constructive algorithm to generate good sets of shapes remains an interesting open problem.

Acknowledgments

This work was conducted in part at the MPI for computer science, Saarbrücken with support from the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT) and at the University of Helsinki supported by the Academy of Finland grant 201560.

References

1. B. Brejová, D. G. Brown, and T. Vinař. Optimal spaced seeds for hidden Markow models, with applications to homologous coding regions. In *Proc. 14th Annual Symposium on Combinatorial Pattern Matching*, volume 2676 of *LNCS*, pages 42–54. Springer, 2003.
2. B. Brejová, D. G. Brown, and T. Vinař. Vector seeds: an extension to spaced seeds allows substantial improvements in sensitivity and specificity. In *Proc. 3rd International Workshop on Algorithms and Bioinformatics*, volume 2812 of *Lecture Notes in Bioinformatics*, pages 39–54. Springer, 2003.
3. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986(8).
4. R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24:293–318, 1992(3).
5. S. Burkhardt. *Filter Algorithms for Approximate String Matching*. PhD thesis, Department of Computer Science, Saarland University, 2002. http://www.mpi-sb.mpg.de/~stburk/thesis.ps.
6. 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.
7. S. Burkhardt and J. Kärkkäinen. Better filtering with gapped q-grams. *Fundamenta Informaticae*, 56(1–2):51–70, 2003.
8. 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.
9. K. P. Choi, F. Zeng, and L. Zhang. Good spaced seeds for homology search. *Bioinformatics*, 20(7):1054–1059, 2004.
10. K. P. Choi and L. Zhang. Sensitivity analysis and efficient method for identifying optimal spaced seeds. *Journal of Computer and System Sciences*, 68:22–40, 2004.
11. M. Fontaine. Computing the filtration efficiency of shape-index-filters for approximate string matching. Master's thesis, Dept. of Computer Science, Saarland University, Nov 2003. http://www.mpi-sb.mpg.de/~fontaine/thesis.ps.

12. U. Keich, M. Li, B. Ma, and J. Tromp. On spaced seeds for similarity search. *Discrete Applied Mathematics*, 138(3):253–263, 2004.
13. G. Kucherov, L. Noé, and M. Roytberg. Multi-seed lossless filtration. In *Proc. 15th Annual Symposium on Combinatorial Pattern Matching*, volume 3109 of *LNCS*, pages 297–310. Springer, 2004.
14. M. Li, B. Ma, D. Kisman, and J. Tromp. PatternHunter II: Highly Sensitive and Fast Homology Search. *Journal of Bioinformatics and Computational Biology*, 2004. Early version in GIW 2003.
15. B. Ma, J. Tromp, and M. Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18:440–445, 2002.
16. L. Noè and G. Kucherov. YASS: Similarity search in DNA sequences. Technical report, INRIA Tech report 4852, 2003.
17. Y. Sun and J. Buhler. Designing multiple simultaneous seeds for DNA similarity search. In *Proceedings of the eighth annual international conference on Computational molecular biology*, pages 76–84, 2004.
18. J. Xu, D. Brown, M. Li, and B. Ma. Optimizing multiple spaced seeds for homology search. In *Proc. 15th Annual Symposium on Combinatorial Pattern Matching*, volume 3109 of *LNCS*, pages 47–58. Springer, 2004.