Mismatch sampling

Raphaël Clifford*[1], Klim Efremenko[2], Benny Porat[3], Ely Porat[3], and Amir Rothschild[4]

[1]Dept. Computer Science, Bristol, UK
[2]Dept. Computer Science, Bar-Ilan University and Dept. Computer Science and Applied Mathematics, Weizman institute, Israel
[3]Dept. Computer Science, Bar-Ilan University, Israel
[4]Dept. Computer Science, Tel-Aviv University, Israel

Abstract

We reconsider the well known problem of pattern matching under the Hamming distance. Previous approaches have shown how to count the number of mismatches efficiently, especially when a bound is known for the maximum Hamming distance. Our interest is different in that we wish to collect a random sample of mismatches of fixed size at each position in the text. Given a pattern p of length m and a text t of length n, we show how to sample with high probability up to c mismatches from every alignment of p and t in $O((c + \log n)(n + m \log m) \log m)$ time. Further, we guarantee that the mismatches are sampled uniformly and can therefore be seen as representative of the types of mismatches that occur.

1 Introduction

Approximate pattern matching is one of the most studied problems in computer science. Numerous measures of approximation have been developed over the years with wide ranging applications from computer vision to bioinformatics. The challenge of approximate matching is that with every different measure of distance between strings comes the need to develop new algorithmic techniques to cope with ever larger amounts of data.

Our focus will be on a particularly popular and simple measure of distance between strings known as the the Hamming distance. Given a pattern p of length m and a text t of length n, the task is to return the number of mismatches between p and every substring of t of length m. Much work has gone into fast solutions to this general problem as well as to a restricted version where only distances up to a predefined bound are reported. However, in many situations

*clifford@cs.bris.ac.uk

it is desirable to know not only how many mismatches occur but also to have some idea about the identity of mismatching symbols. It is of course possible to output the full set of mismatches by applying known approximate matching methods. Unfortunately, in the worst case such an approach will require $\Theta(nm)$ time as this is the size of the output.

In order to be able to understand which mismatches occur frequently we develop a fast method that returns a fixed size sample of the mismatches at each alignment, even when there is no prior knowledge of the number of mismatches that will be found. We call the problem we consider *mismatch sampling* and define it as follows. Given an integer c, we sample uniformly at random c distinct mismatches that occur between the pattern and text at each possible alignment. Where the Hamming distance is less than c, all mismatches are to be reported. Such samples will have a variety of interpretations depending on the context but can be seen as representing typical spelling errors when searching text or for example, common DNA mutations in the context of bioinformatics.

In the process of tackling the mismatch sampling problem, we also develop a randomised algorithm for the so-called 1-mismatch problem which may be of independent interest. This returns a single mismatch chosen uniformly at random, where at least one occurs, at every alignment. The algorithm takes $O(n + m \log m)$ time overall and linear time after preprocessing the pattern. This efficiency comes at the cost of a fixed probability of not returning an answer at all. We give a probabilistic bound which we prove is sufficiently strong for the main mismatch sampling result.

This version of the 1-mismatch problem can be applied directly to speed up previous solutions for both the k-mismatch with don't cares problem [CEPR07] and generalised pattern matching [PE08], for example. In the latter case, the new 1-mismatch algorithm gives an immediate log factor speedup and in the former, the same speedup applies when the don't cares in the input occur only in the pattern.

2 Preliminaries

Let Σ be a set of characters which we term the *alphabet* and let $t = t_1 t_2 \dots t_n \in \Sigma^n$ be the text and $p = p_1 p_2 \dots p_m \in \Sigma^m$ the pattern. We first reduce the alphabet to the range $\{1, \dots, m+1\}$ by sorting the pattern and text and relabelling both in $O(n \log n)$ time. Symbols $\{1, \dots, m\}$ are reserved for symbols that occur in both the pattern and text and $m+1$ is reserved for any symbol in the text that does not occur at all in the pattern. This relabelling does not affect the overall time complexity of our algorithms.

The terms *symbol* and *character* are used interchangeably throughout. Similarly, we will sometimes refer to a *location* in a string and synonymously at other times a *position*. We will also refer to an *alignment* of the pattern and text which is to be understood as the location in the text where the pattern starts to be compared.

Definition 1. *Define $HD(i)$ to be the Hamming distance between p and $t[i, \ldots, i+m-1]$.*

Our algorithms make extensive use of the fast Fourier transform (FFT). An important property of the FFT is that in the RAM model, the cross-correlation,

$$(t \otimes p)[i] \stackrel{\text{def}}{=} \sum_{j=1}^{m} p_j t_{i+j-1}, \quad 1 \leq i \leq n-m+1,$$

can be calculated accurately and efficiently in $O(n \log n)$ time both over the integers $\mathbb{Z}$, and a finite field F_q (see e.g. [CLR90], Chapter 32). As we assume the unit-cost RAM model, arithmetic operations on words take constant time. In particular, we can compute ab and a/b in F_q in constant time, as $q \leq cn$ for some constant $c \geq 1$. Division is performed by taking advantage of a precomputed lookup table of size $O(q)$ giving the inverse of every element of F_q.

By a standard trick of splitting the text into overlapping substrings of length $2m$, the running time of the cross-correlation can be further reduced to $O(n \log m)$. We will at times assume for ease of presentation that the text is of length $2m$ and that the reader is familiar with this splitting technique.

In order to fix terminology we give a definition of the term *with high probability* which is also abbreviated to w.h.p. Our definition is at the stricter end of the scale compared to that which is normally found in the literature. One beneficial consequence is that the probabilistic bounds we state remain true even if the pattern matching algorithm is repeated a polynomial number of times.

Definition 2. *We say that an algorithm outputs an answer with high probability or* w.h.p. *in time $\Theta(f(n))$ if for every $\alpha \geq 1$, there exist a value $c_\alpha > 0$ depending on α, such that after $\Theta(f(n))$ time, the algorithm outputs an answer with probability at least $1 - \frac{c_\alpha}{n^\alpha}$. In this definition, the constant in the Θ notation may also depend on α.*

3 Related work and previous results

Much progress has been made in finding fast algorithms for the Hamming distance problem over the last 20 years. $O(n\sqrt{m \log m})$ time solutions based on repeated applications of the FFT were given independently by both Abrahamson and Kosaraju in 1987 [Abr87; Kos87]. The major improvements have concentrated on a bounded version of the problem called k-mismatch. In this problem an integer bound k is given in advance and only Hamming distances less than or equal to k need be reported. In 1985 Landau and Vishkin gave a beautiful $O(nk)$ algorithm that is not FFT based which uses constant time lowest common ancestor (LCA) operations on the suffix tree of p and t [LV86]. This was subsequently improved in [ALP04] to $O(n\sqrt{k \log k})$ time by a method based on filtering and FFTs again. More recently, $\tilde{O}(nk)$ time randomised and deterministic algorithms for the k-mismatch problem with single character wildcard or don't care symbols have been shown [CEPR07; CEPR09]. Efficient solutions to

the problem of approximating the Hamming distance to within a multiplicative factor of $(1+\epsilon)$ have also been developed [Kar93; Ind98].

The existing fast k-mismatch algorithms do also return the mismatches that have been found and so might appear to be helpful for our problem of mismatch sampling. However, in our case k can be as large as m which would give an algorithm whose time complexity is no better than a naive $\Theta(nm)$ solution. Despite this limitation, some of the techniques we will employ are related to those given in the previous work of [CEPR07]. The most important similarity is the idea of sampling single mismatches using masked versions of the pattern. However the solution we present is different and more efficient than those given before and as we will show, a number of further technical obstacles need to be overcome before we are able to provide a full solution for the mismatch sampling problem.

4 Results and new techniques

We summarise the main results and discuss the techniques that were developed.

- The first result in Section 5 is a randomised algorithm that solves the 1-mismatch problem in $O(n + m \log m)$ time. The algorithm gives an answer with constant probability as long as the true Hamming distance $HD(i)$, can be estimated to within a constant factor.

  The main new technique is a way to restrict the cross-correlation calculations to be only performed on a constant number of arrays of length $2m$ rather than the full text of size n. The only computations that have to be performed on an array of length n run in linear time. This saves a log factor in the overall time complexity as well as being practically more efficient due to the constant factor overheads inherent in FFT calculations.

  We also show that by performing all calculations modulo a large prime q, we can ensure that the single mismatches found are chosen uniformly at random from the set of mismatches at each alignment without detriment to the time complexity.

- In Section 6 we present the first solution for the mismatch sampling problem which samples $\min(c, HD(i))$ mismatches w.h.p. at every alignment. The 1-mismatch algorithm is repeatedly run over $O(\log m)$ stages with each stage providing a different estimate of the Hamming distance between pattern and text alignments. This gives an $O(c \log n(n + m \log m) \log m)$ time algorithm. It is important to note that the probabilistic bounds we give are particularly strong and hold for every position in the text simultaneously.

- Finally we show how by using a k-mismatch algorithm as a preprocessing step, we are able to speed up the mismatch sampling algorithm and still output the answer w.h.p. The main idea is quickly to eliminate all positions where the Hamming distance is less than $2c$ and then concentrate

only on those remaining positions. The overall time complexity is therefore reduced to $O((c + \log n)(n + m \log m) \log m)$ time. We also show that the algorithm can easily be made Las Vegas while maintaining the same running time w.h.p.

5 Randomised 1-mismatch

We first present the main algorithmic tool that will be used to sample distinct mismatches. The 1-mismatch problem is to sample a single mismatch, where at least one occurs, between the pattern and every alignment of the text. The overall strategy for mismatch sampling will be repeatedly to sample single mismatches from each alignment of the pattern and text using an algorithm for the 1-mismatch problem.

Our solution to the 1-mismatch problem is randomised and requires $O(n + m \log m)$ time per iteration, returning a single sampled mismatch for each alignment where $HD(i) \geq 1$, with constant probability. The key property we require is that each sampled mismatch will be chosen uniformly at random from the set of possible mismatches at each alignment. In order to find the mismatches we will also require an estimate within a constant factor of the Hamming distance at each alignment of the pattern and text. For the time being we assume that such an estimate is available and in Section 6 we show that only $O(\log m)$ distinct estimates will be required overall.

In order to be able to sample individual mismatches we must find a way to eliminate all other mismatches that could have occurred. We first create masked versions of the pattern, so that an alignment of the masked pattern and the text is likely to only contain one mismatch. To perform this efficiently we create a random array r of length $2m$. A *sampling rate* s is then defined which determines the probability that a given r_j will be set to zero. For a given sampling rate s, the aim is for 1-mismatch to find single mismatches for every alignment i for which $HD(i) \leq s \leq 2HD(i)$. The aim will be to mask out all but one mismatch at each alignment by multiplying the difference $(p_j - t_{i+j-1})$ by the random element r_{i+j-1}.

We define the random array r such that each value $r_j = 0$ with probability $(s-1)/s$ and is chosen independently and uniformly at random from $[1, \ldots, q-1]$ with probability $1/s$. We set q to be a prime which is larger than $\max(\max_{i,j}(|p_j - t_i|), n)$. We then compute the cross-correlation between the pattern and r and an array r' of the same length as r such that $r'_i = ir_i$. This gives two arrays $A = p \otimes r$ and $C = p \otimes r'$. In this way, any values set to zero in r will effectively eliminate the contribution from corresponding values in p. The cross-correlation calculations over arrays of length $2m$ need only to be performed once per iteration of the 1-mismatch algorithm and do not require the input text. In this way it can be seen as a preprocessing step.

For each position i in the text we then calculate $\sum_{j=1}^{m} r_{i+j-1}(i+j-1)(p_j - t_{i+j-1}) / \sum_{j=1}^{m} r_{i+j-1}(p_j - t_{i+j-1})$ with all calculations performed modulo q. This calculation is the main body of the 1-mismatch algorithm and Algorithm 1

describes the main steps assuming the text is of length $2m$. Using the standard method of splitting the text into segments of length $2m$ with overlap m described in Section 2 the algorithm can then be applied to the whole text.

Input: Pattern p, text t, random array r and prime q
Output: $E[i]$ contains a single mismatch location with probability at least $1/(2\sqrt{e})$
Compute A s.t. $A[i] = \sum_{j=1}^{m} r_{i+j-1}p_j$ for each $1 \leq i \leq m$;
Compute B s.t. $B[i] = \sum_{j=1}^{m} r_{i+j-1}t_{i+j-1}$ for each $1 \leq i \leq m$;
Compute C s.t. $C[i] = \sum_{j=1}^{m}(i+j-1)r_{i+j-1}p_j$ for each $1 \leq i \leq m$;
Compute D s.t. $D[i] = \sum_{j=1}^{m}(i+j-1)r_{i+j-1}t_{i+j-1}$ for each $1 \leq i \leq m$;
Compute $E = (C-D)/(A-B)$;

Algorithm 1: Randomised 1-MISMATCH for text of length $2m$

For any i where there is exactly one mismatch between p and $t[i,\ldots,i+m-1]$, $E[i]$ is the location in t of the mismatch and $E[i]-i+1$ is the location in p. As we have the location of the proposed mismatch in both the pattern and text a simple constant time check per alignment will tell us if we have indeed found a mismatch.

Lemma 1. *Algorithm 1 run over a text of length n takes $O(n+m\log m)$ time.*

Proof. Calculating B, D and E for every partition of the text into sections of length $2m$ takes $O(n)$ time in total. The time required to compute A and C is dominated by the running time of the FFT on an array of length $O(m)$ and is therefore $O(m\log m)$. A and C do not need to be recalculated for each segment of the text of length $2m$. The total running time of Algorithm 1 is therefore $\Theta(n+m\log m)$ □

Before we can show that Algorithm 1 returns a uniformly sampled mismatch, we will need a preliminary mathematical lemma that gives us the probability that various sums in our algorithm will be exactly zero modulo q.

Lemma 2. *For prime q, let X_i be uniform random variables over the range $\{1,\ldots,q-1\}$, let a_i be distinct arbitrary but fixed integers in the same range and let $Q = 1/(q-1)$. Define $p_k = P(\sum_{i=1}^{k} a_iX_i = 0)$ and $r_k = P(\sum_{i=1}^{k} a_iX_i = 0 \wedge \sum_{i=1}^{k} X_i = 0)$. The following is then true:*

$$p_k = \frac{Q+(-1)^kQ^k}{1+Q}$$

and

$$r_k = \frac{(k-2)(-Q)^{k+1}-(k-1)(-Q)^k+Q^2}{(1+Q)^2}.$$

Proof. We approach the problem by setting up and solving recurrences for both p_n and r_n. In the first case it follows from the definition that $p_{n+1} = (1 - p_n)Q$ and $p_1 = 0$ which when solved gives us the first result. We then see that

$$\begin{aligned}
r_{n+1} &= P\left(\sum_{i=1}^{n+1} a_i X_i = 0 \wedge \sum_{i=1}^{n+1} X_i = 0\right) \\
&= P\left(\sum_{i=1}^{n} a_i/a_{n+1} X_i = \sum_{i=1}^{n} X_i \neq 0\right) Q \\
&= P\left(\sum_{i=1}^{n} a_i/a_{n+1} X_i = \sum_{i=1}^{n} X_i\right) Q - P\left(\sum_{i=1}^{n} a_i X_i = \sum_{i=1}^{n} X_i = 0\right) Q \\
&= P\left(\sum_{i=1}^{n} (a_i - a_{n+1}) X_i = 0\right) Q - r_n Q \\
&= (p_n - r_n)Q.
\end{aligned}$$

Setting the base case $r_1 = 0$ and using the solution for p_n found before, the second result of the lemma follows. □

We are now able to prove the main result for the randomised 1-mismatch algorithm.

Theorem 3. *For a given alignment i and sampling rate $HD(i) \leq s \leq 2HD(i)$, Algorithm 1 samples a single mismatch uniformly from the set of mismatches at alignment i with probability at least $1/2\sqrt{e}$.*

Proof. Suppose $k = HD(i) \geq 1$, and let $i_1, ..., i_k$ be the locations in the pattern of the mismatches at alignment i. Algorithm 1 will find a single mismatch at location i_1 if $r_{i+i_j-1} = 0$ for all $1 < j \leq k$ and $r_{i+i_1-1} > 0$. Therefore the probability of a single mismatch being found is $k/s(1 - 1/s)^{k-1}$ if $k \geq 2$ or $1/s$ if $k = 1$. To bound the probability let $s = 2k$ which is its largest possible value, recall that $\lim_{k\to\infty}(1 - 1/(2k))^{k-1} = 1/\sqrt{e}$ and that this limit is reached from above. Therefore the probability of finding a mismatch is bounded below by $1/2\sqrt{e}$.

It is also possible that the algorithm will accidentally return a mismatch location even when there are two or more mismatch positions available. However, this can only increase the probability of a mismatch being found. Finally, it is also possible that $A - B = 0$ or $C - D = 0$. In this case we know that we have found more than one mismatch at the relevant alignment and so we can simply discard the result. We now need to show that any mismatch found is selected uniformly at random.

In the case where only one mismatch is aligned with a non-zero element of r, uniformity follows immediately from the observation that the zero elements of r are chosen independently and uniformly at random. Where two or mismatches align with non-zero elements of r, the algorithm may also return the location

of a single mismatch by chance. We must establish that such locations will also be uniformly chosen from the set of mismatches that exist at the alignment.

For a fixed alignment i, We define the random variable $X_j = (r_{i+j-1})(p_j - t_{i+j-1})$. In this way the output of Algorithm 1 for a fixed alignment can be viewed as a random variable $A = \sum_{j=1}^{k} X_j a_j / \sum_{j=1}^{k} X_j$, where k is the number of mismatches at that alignment and each coefficient a_j corresponds to a position in the text where a mismatch occurs. Unfortunately, A is not distributed uniformly as can straightforwardly be seen by setting $k = 2$. In that case A cannot equal either a_1 or a_2 unless either $X_1 = 0$ or $X_2 = 0$, which cannot occur by definition. However, it is the case as we will now show that A is uniformly distributed as long as the outcome of the random variable A is equal to a_j for some j. This is all that is required as the a_j are the locations of the mismatches that can be reported.

We now calculate wlog $P(A = a_k)$ or equivalently $P(\sum_{j=1}^{k-1}(a_j - a_k)X_j = 0 \wedge \sum_{j=1}^{k} X_j \neq 0)$. We will use the terms p_k and r_k as defined in Lemma 2 and let $Q = 1/(q-1)$ as before.

$$
\begin{aligned}
&= P\left(\sum_{j=1}^{k-1}(a_j - a_k)X_j = 0\right) - P\left(\sum_{j=1}^{k-1}(a_j - a_k)\,X_j = 0 \wedge \sum_{j=1}^{k} X_j = 0\right) \\
&= p_{k-1} - P\left(\sum_{j=1}^{k-1}(a_j - a_k)X_j = 0 \wedge \sum_{j=1}^{k-1} X_j \neq 0\right) Q \\
&= p_{k-1} - \left(P\left(\sum_{j=1}^{k-1}(a_j - a_k)X_j = 0\right) - P\left(\sum_{j=1}^{k-1}(a_j - a_k)X_j = 0 \wedge \sum_{j=1}^{k-1} X_j = 0\right)\right) Q \\
&= p_{k-1} - Q(p_{k-1} - r_{k-1}) \\
&= (1-Q)p_{k-1} + Qr_{k-1}
\end{aligned}
$$

The probability that the value returned is one of the mismatch locations is therefore uniform over all possible choices of mismatch locations.

□

Given that the distribution of A is not uniform, but that it is uniform when the outcome is one of the a_j values, it is an interesting question to ask what the difference is between the probability of accidentally finding a specified mismatch location and finding any other location. This value can now be calculated directly and works out to be $(-Q)^{k-1}$ which decreases exponentially as the number of mismatches aligned with non-zero values of the random array r increases.

6 The Mismatch Sampling Algorithm

Input: Pattern p, text t and an integer c
Output: $O[i]$ = sample of up to $\min(HD(i), c)$ distinct mismatches
`/* Iterate over` $O(\log m)$ `sample rates */`
for $\ell = 1$ *to* $\log_2 m$ **do**
  repeat
    Create random array r with sampling rate $s = 2^{\ell-1}$;
    Perform 1-MISMATCH(p, t, r);
    Add new mismatches to output O;
  until $O(c \log n)$ *iterations* ;
end

Algorithm 2: Simple mismatch sampling

We are now able to present the main mismatch sampling algorithm. This will be done in two phases. In the first we will give a simple algorithm based on repeated applications of 1-mismatch which runs in $O((c \log n)(n + m \log m) \log m)$ time and samples c mismatches w.h.p. wherever there are at least c mismatches. We will then show how to speed up the approach using k-mismatch as a preprocessing stage resulting in the final $O((c + \log n)(n + m \log m) \log m)$ mismatch sampling algorithm. We also discuss how this algorithm can be made Las Vegas without increasing the time complexity w.h.p.

To start, recall from Section 5 that the 1-mismatch algorithm requires an estimate of the Hamming distance. In order to apply it to the full problem where the Hamming distance at each alignment is not known, we will require $O(\log m)$ stages overall. At each stage ℓ we set the sampling rate s set to $2^{\ell-1}$. The algorithm which is set out in Algorithm 2 will repeat 1-mismatch a sufficient number of times at each sampling rate so that when the correct sampling rate is found, we will find c mismatches w.h.p., assuming the true Hamming distance is at least c.

The following lemma shows that when the correct sampling rate is found for a particular alignment i, all $\min(c, HD(i))$ mismatches will be found w.h.p.

Lemma 3. *For all alignments i such that $s \leq HD(i) \leq 2s$, at least $\min(c, HD(i))$ distinct mismatches will be found after $O(c \log n)$ iterations of 1-mismatch w.h.p. and they will be chosen uniformly at random from the set of mismatches at alignment i.*

Proof. The lemma is proved by an application of the coupon collector's problem (see e.g. [Fel68]) and the observation that there is a constant probability of getting a new mismatch at each iteration. Although the usual analysis of this problem requires only $O(c \log c)$ iterations to get all the required distinct mismatches, we increase the number of iterations to $O(c \log n)$ in order to ensure that the probabilistic bound holds at all alignments in the text simultaneously. □

We can now give the running time of the first mismatch sampling algorithm.

Theorem 4. *For each $1 \leq i \leq n$, Algorithm 2 samples $\min(c, HD(i))$ mismatches w.h.p. uniformly at random in $O((c \log n)(n + m \log m) \log m)$ time.*

Proof. From Lemma 3 we know that after $O(c \log n)$ iterations we will find $\min(c, HD(i))$ mismatches w.h.p. for $s \leq HD(i) \leq 2s$. Therefore, by repeating this process for each of $O(\log m)$ sampling rates, s, we will find $\min(c, HD(i))$ mismatches w.h.p. at every alignment i. Our algorithm performs $O(c \log n \log m)$ 1-mismatch procedures. Therefore the overall running time is $O(c \log n(n + m \log m) \log m)$. □

Mismatch Sampling in $O((c + \log n)(n + m \log m) \log m)$ time

We now show the final speedup and give the full mismatch sampling algorithm. First, we observe that some positions are easier to sample c mismatches from than others. In particular, the following lemma shows that if there are more than $2c$ mismatches, we can sample c mismatches more quickly than before.

Lemma 4. *For all alignments i such that $s \leq HD(i) \leq 2s$, and $HD(i) \geq 2c$, c distinct mismatches will be found after $O(c + \log n)$ iterations of 1-mismatch w.h.p. and will be chosen uniformly at random from the set of mismatches at alignment i.*

Proof. By Lemma 3 we will find one mismatch at every iteration of 1-mismatch algorithm with constant probability. Because $HD(i) \geq 2c$, if we have found fewer than c mismatches then the probability that a discovered mismatch will be new is at least $1/2$. So at every iteration of the 1-mismatch algorithm we will have found a *new* mismatch with constant probability. So after $O(c + \log n)$ iterations we will have found at least c mismatches w.h.p. As before, this bound holds at every alignment in the text simultaneously. □

The second part of the improvement is to eliminate all the alignments where fewer than $2c$ mismatches occur. This can be done by using the k-mismatch algorithm of Landau and Vishkin [LV86] and setting $k = 2c$. After this preprocessing step we will have found $HD(i)$ mismatches for all alignments where $HD(i) \leq 2c$ in $O(nc)$ time. We can now concentrate only on those alignments where $HD(i) > 2c$.

By doubling the sampling rate at each iteration as before but this time starting at a rate of c, Algorithm 3 sets out the main steps that need to be performed. Theorem 5 now gives the final running time.

Theorem 5. *For each $1 \leq i \leq n$, Algorithm 3 samples $\min(c, HD(i))$ mismatches w.h.p. uniformly at randomly in $O((c + \log n)(n + m \log m) \log m)$ time.*

Proof. The $2c$-mismatch algorithm handles the cases with at most $2c$ mismatches and runs in $O(nc)$ time. Then we choose a random array r and perform the 1-mismatch algorithm $O((\log(m/c))(c + \log n)$ times taking $O((n + m \log m)(\log(m/c))(c + \log n))$ time overall. After performing these two stages, at each alignment i we have found $\min(c, HD(i))$ mismatches w.h.p. Therefore we will also find $\min(c, HD(i))$ mismatches at all alignments w.h.p. □

Input: Pattern p, text t and an integer c
Output: $O[i]$ =sample of up to $\min(HD(i), c)$ distinct mismatches
`/* Eliminate alignments with few mismatches */`
Run $2c$-mismatch(t,p) ;
`/* Many mismatches stage */`
for $\ell = \lfloor \log 2c \rfloor$ *to* $\log m$ **do**
  repeat
    Create random array r with sampling rate $s = 2^{\ell-1}$;
    Perform 1-MISMATCH(p, t, r);
    Add new mismatches to output O;
  until $O(c + \log n)$ *iterations* ;
end

Algorithm 3: Mismatch sampling

By repeating the main loop of Algorithm 3 until $\min(c, HD(i))$ mismatches are found at each alignment, the algorithm can straightforwardly be made Las Vegas and it follows from Theorem 5 that the running time will be $O((c + \log n)(n + m \log m) \log m)$ w.h.p.

7 Extensions to related problems

The motivation for mismatch sampling can be applied to any number of approximate pattern matching problems and it is of interest to know which will allow a fixed size sample to be given efficiently. The most obvious direct application of our method is to mismatch sampling where single character don't cares are permitted in the input. In [CEPR07], where a randomised algorithm is given for the k-mismatch problem with don't cares, a deterministic 1-mismatch algorithm allowing don't cares forms a core part of the solution. If the problem is restricted to only allow don't cares in the pattern, rather than the text, then the new faster randomised 1-mismatch algorithm we have presented in this paper can be used as a direct replacement. Following the same overall strategy of sampling single mismatches over $O(\log m)$ stages will now give a mismatch sampling algorithm allowing don't cares in the pattern that runs in $O((c + \log n)(n \log^2 m))$ time. The problem of error sampling where pattern matching is to be performed over more sophisticated approximation measures, such as the edit distance for example, appears to be considerably more challenging.

8 Acknowledgements

This work was supported in part by the Binational Science Foundation (BSF), Israel Science Foundation (ISF) and the Engineering and Physical Sciences Research Council (EPSRC).

References

[Abr87] K. Abrahamson. "Generalized string matching". In: *SIAM Journal on Computing* 16.6 (1987), pp. 1039–1051.

[ALP04] A. Amir, M. Lewenstein, and E. Porat. "Faster Algorithms for String Matching with k Mismatches". In: *Journal of Algorithms.* 50.2 (2004), pp. 257–275.

[CEPR07] R. Clifford, K. Efremenko, E. Porat, and A. Rothschild. "k-Mismatch with Don't Cares". In: *ESA '07: Proc. 15th Annual European Symp. on Algorithms.* 2007, pp. 151–162.

[CEPR09] R. Clifford, K. Efremenko, E. Porat, and A. Rothschild. "From Coding Theory to Efficient Pattern Matching". In: *SODA '09: Proc. 20th ACM-SIAM Symp. on Discrete Algorithms.* 2009, pp. 778–784.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[Fel68] W. Feller. *An introduction to probability theory and its applications. - Vol. 1.* Wiley, 1968. ISBN: 0-471-25708-7.

[Ind98] P. Indyk. "Faster Algorithms for String Matching Problems: Matching the Convolution Bound." In: *FOCS '98: Proc. 39th Annual Symp. Foundations of Computer Science.* 1998, pp. 166–173.

[Kar93] H. Karloff. "Fast Algorithms for approximately counting mismatches". In: *Information Processing Letters* 48.2 (1993), pp. 53–60.

[Kos87] S. R. Kosaraju. "Efficient string matching". Manuscript. 1987.

[LV86] G. M. Landau and U. Vishkin. "Efficient string matching with k mismatches". In: *Theoretical Computer Science* 43 (1986), pp. 239–249.

[PE08] E. Porat and K. Efremenko. "Approximating general metric distances between a pattern and a text". In: *SODA '08: Proc. 19th ACM-SIAM Symp. on Discrete Algorithms.* 2008, pp. 419–427.