

Mismatch sampling

Raphaël Clifford¹, Klim Efremenko², Benny Porat³ Ely Porat³ and Amir Rothschild⁴

¹ University of Bristol, Dept. of Computer Science, Bristol, BS8 1UB, UK
clifford@cs.bris.ac.uk

² Bar-Ilan University, Dept. of Computer Science, 52900 Ramat-Gan, Israel and
Weizman institute, Dept. of Computer Science and Applied Mathematics, Rehovot,
Israel klimefrem@gmail.com

³ Bar-Ilan University, Dept. of Computer Science, 52900 Ramat-Gan, Israel
bennyporat@gmail.com, porately@cs.biu.ac.il

⁴ Tel-Aviv University, Dept. of computer science, Tel-Aviv, Israel and Bar-Ilan
University, Dept. of Computer Science, 52900 Ramat-Gan, Israel
rotshch@post.tau.ac.il

Abstract. We consider 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 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 c mismatches where possible from every alignment of p and t in time. Further, we guarantee that the mismatches are sampled uniformly and that they 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 way of measure the distance between two strings comes the need to develop often entirely novel techniques to cope with the need for ever larger amounts of data to be processed efficiently.

One of the most common and simplest measures of approximation is 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 a restricted version called k -mismatch where only distances up to k are to be reported. However, in many situations it is desirable to know not only how many mismatches occur but also to have some idea what the mismatches are. It is clear that in the worst case no algorithm that returns all mismatches can run in less than $\Theta(nm)$ time.

In order to be able to understand which mismatches occur it will be necessary to return a sample of the mismatches at each alignment of fixed size and it is this problem that we consider. Given an integer value c , the task is to 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 general case where the maximum Hamming may be as large as m , the authors are not aware of any existing techniques which improve on the naive $\Theta(nm)$ time algorithm in the worst case. We also improve the running time of 1-mismatch algorithm from $O(n \log m)$ to $O(n + m \log m)$ which is used in many other algorithm like [CEPR07, PE08]. For example this reduces running time of algorithm in [PE08] by $\log m$ factor. Our the techniques we use could also support the case when some of symbols of the pattern are wildcards.

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. 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 $t[i, \dots, i+m-1]$.

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,

$$(t \otimes p)[i] \stackrel{\text{def}}{=} \sum_{j=1}^m p_j t_{i+j-1}, \quad 0 \leq i \leq n-m+1,$$

can be calculated accurately and efficiently in $\mathcal{O}(n \log n)$ time both over the integers \mathbb{Z} , and a finite field F_q (see e.g. [CLR90], Chapter 32. By a standard trick of splitting the text into overlapping substrings of length $2m$, the running time can be further reduced to $\mathcal{O}(n \log m)$.

We will often assume that the text is of length $2m$ in the presentation of an algorithm or analysis in this Section 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 those found in the literature when analysing randomised algorithms and has as one consequence that the bounds still hold even if the algorithm is repeated a polynomial number of times.

Definition 2 We say that an algorithm outputs the correct 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 the correct answer with probability at least $1 - \frac{c_\alpha}{n^\alpha}$. Note that 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. Approximations within a multiplicative factor of $(1 + \epsilon)$ to the Hamming distance can also be found in $O(n/\epsilon^2 \log m)$ time [Ind98]. More recently, a randomised algorithm for the k -mismatch problem with *don't cares* was given which runs in $O(n(k + \log m \log k) \log n)$ time and returns all the mismatches found [CEPR07].

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 $O(nm)$ solution. Despite this limitation, some 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 patterns. 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 a problem called 1-mismatch with constant probability in $O(n + m \log m)$ time. The 1-mismatch problem is to find a single mismatch, where at least one occurs, at every alignment. The algorithm gives the correct 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 here is a sampling trick which masks out enough of the pattern so that only one mismatch is likely to remain. This enables us to sample single mismatches even when the true Hamming distance may be considerably larger. In order to perform this sampling efficiently, we show that the expensive cross-correlation calculations using FFTs need only be performed over a constant number of arrays of length $2m$. The only computations that have to be performed on an array of the length of the text run in linear time. This saves a log factor in the overall time complexity as well as being practically more efficient due to the overheads inherent in the 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. The overall time complexity of the algorithms is maintained by performing the FFT calculations and hence the cross-correlations, over the field F_q .

- In Section 6 we present the Mismatch Sampling algorithm which samples $\min(c, HD(i))$ mismatches with high probability at every alignment of the pattern and text in $O((c + \log n)(n + m \log m) \log m)$ time. The 1-mismatch algorithm is repeatedly run over $O(\log m)$ stages with each stage providing a different estimate of the Hamming distances between pattern and text. By carefully accounting for the number of distinct mismatches found in previous stages of the algorithm, we show how the number of iterations of 1-MISMATCH can be minimised.
- 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 find the correct answer w.h.p. The main idea is to eliminate all but a small number of positions where we are still required to find extra mismatches. These positions can then be checked naively in $O(m)$ time each. This gives the final running time for Mismatch Sampling of $O((c + \log m)(n + m \log m) \log m)$.

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 find 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 to repeatedly 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 such that $HD(i) \leq 1$, with constant probability. However, in order to find the mismatches with constant probability we will require a reasonably accurate estimate 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 $s \leq HD(i) \leq 2s$. 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 $r_j = 0$ with probability $((s-1)/s)$ and with probability $1/s$, r_j is chosen independently and uniformly at random from $[1, \dots, q-1]$. We set q to be a prime which is larger than $\max((\max_{i,j} |p_j - t_i|), m)$. 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 over the finite field F_q . This calculation is the main body of the 1-MISMATCH algorithm and Algorithm 1 describes the main steps assuming the text 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 $1/2e$

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 $t[i, \dots, i+m-1]$, $E[i]$ is the location in t of the mismatch and $E[i] - i + 1$ is the location in p . We can check for all the positions where there are no mismatches in linear time using any of the well known exact matching algorithms. 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 $O(n)$ 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 $\Theta(n + m \log m)$

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

Proof. Suppose $k = HD(i) \geq 1$, and let i_1, \dots, i_k be the locations of the mismatches. Algorithm 1 will find a single mismatch i_1 if $r_{i+i_j-1} = 0$ for all $1 < j \leq k$ and $r_{i_1} > 0$. Therefore the probability of a single mismatch being found is: $\frac{k}{s}(1 - \frac{1}{s})^k$ bounded below by $1/2e$. It is also possible that the algorithm will accidentally find a mismatch even when there are two or more mismatch positions available. However this can only increase the probability of a mismatch being found. It is also possible that the denominator $(A - B) = \sum_{j=1}^m r_{i+j-1}(p_j - t_{i+j-1}) = 0$. However, if this occurs then we know that there can't be a single mismatch at the relevant alignment and so we can simply discard the result. Lets see also that the probability that every mismatch is chosen are equal in order to see it if only single mismatch is chosen by r_i then all mismatches are equal. On over hand if more then one mismatches are chosen by r_i then the probability of every mismatch at place say l to found is lets look on some mismatch place not equal to l then exists only one r_j that finds this mismatch l .

$$\Pr\left(\frac{(i+j)r_{i+j}(p[j] - t[i+j]) + c}{r_{i+j}(p[j] - t[i+j]) + d} = k\right) = \Pr\left(r_{i+j} = \frac{kd - c}{(i+j-k)(p[j] - t[i+j])}\right) = \frac{1}{q}$$

So the probability to choose every mismatch is equal.

Non-uniform sampling

In our application we require that the 1-MISMATCH algorithm returns mismatches chosen uniformly at random in order that the full Mismatch Sampling algorithm will return uniformly random subsets of the possible mismatches. However, 1-MISMATCH is also useful as a tool in its own right and there can be situations where it is not required that the mismatches are chosen at random. In this case a simplification to Algorithm 1 is possible which may provide a practical speedup. Instead of choosing r_j from a large range, we can simply make r a binary array with r_j set to 0 with probability $(s-1)/s$ and 1 with probability $1/s$. The calculations, including the FFTs can then be performed over the integers instead of the finite field F_q .

In this case, the mismatches found will no longer be chosen uniformly at random from the possible mismatches at each alignment. Why this is true is best shown with an example. Consider $p = 4, 4, 4$ and $t = 3, 2, 3$ so that $E =$

```

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
    for  $O(c \log n)$  times do
        Create random array  $r$  with sampling rate  $s = 2^{\ell-1}$ ;
        Perform 1-MISMATCH( $p, t, r$ );
        Add new mismatches to output  $O$ ;
    end
end

```

Algorithm 2: Simple Mismatch Sampling

$(r_1 + 4r_2 + 3r_3)/(r_1 + 2r_2 + r_3)$. In this case, if r has two or more non-zero positions then the only mismatch that will be found is at position 2. However, single mismatches will still be found with probability at least $1/2e$ although no longer uniformly at random.

6 The Mismatch Sampling Algorithm

In this Section we will 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 where possible w.h.p. We will then show how to speed up the approach resulting in the final $O((c + \log m)(n + m \log m) \log m)$ Mismatch Sampling algorithm.

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. The proof is an application of the *coupon collector's problem* (see e.g. [Fel68]).

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 alignment i .*

Proof. Lets consider some specific offset i with $HD(i) = k$ such that $s \leq k \leq 2s$. By 1-mismatch algorithm will choose uniformly at random one mismatch

with constant probability. So after $O(c \log n)$ iterations the we will find at least $\min(c, k)$ random uniform distinguish mismatches w.h.p. because if $c < 2k$ then for every mismatch probability that we will not found it is at most $\frac{1}{n^2}$ therefor the probability that we will not found all the mismatches is at most $\frac{1}{n}$. The case when $c > 2k$ will be proved later at 4

We can now give the running time of the first Mismatch Sampling algorithm.

Theorem 3. *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. As we already saw in 3 after $O(c \log n)$ iterations we will find c mismatches w.h.p. for $s \leq HD(i) \leq 2s$ and therefor after enumeration on s we will find for all $i = 1, 2, \dots, n - m$. Our algorithm runs $O(\log(m)c \log n)$ 1-mismatch procedure. Therefor overall running time is: $O(\log m(n + m \log m)c \log n)$

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

Two further improvements are possible to give the final algorithm and running time. First, we observe that it is not necessary to find all c mismatches at only one stage of the algorithm.

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 2 we will find one mismatch at every iteration of 1-mismatch algorithm with constant probability. Because $HD(i) \geq 2c$ if we found less than c mismatches then probability that this mismatch will be new is at least $1/2$. So at every iteration of 1-mismatch algorithm we will found *new* mismatch with constant probability. So after $O(c + \log n)$ iterations we will found more then c mismatches w.h.p.

The second improvement is to eliminate all the alignments were fewer than a constant times c 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 $\min(c, HD(i))$ mismatches for all alignments where $HD(i) \leq 2c$ in $O(nc)$ time. We can now concentrate only on those alignments where $HD(i) > 2c$.

Algorithm 3 sets out the main steps.

Theorem 4. *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 handling the cases with few mismatches, runs in $O(nc)$ time. Then we choose a random array r and perform the 1-mismatch algorithm $O((\log \frac{m}{c})(c + \log n))$ times, therefore taking $O((n + m \log m)(\log \frac{m}{c})(c + \log m))$ time. After performing this two stages, at each alignment i we have found c mismatches with probability at least $1 - n^{-d}$. Therefor we will find c mismatches at all places with probability at least $1 - n^{-(d-1)}$

```

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
    for  $O(c + \log n)$  times do
        Create random array  $r$  with sampling rate  $s = 2^{\ell-1}$ ;
        Perform 1-MISMATCH( $p, t, r$ );
        Add new mismatches to output  $O$ ;
    end
end

```

Algorithm 3: Mismatch sampling

7 Discussion

The problem of Mismatch Sampling can be applied to any number of approximate pattern matching problems and it is of interest to know which will allow a sample to be given efficiently. The most obvious direct extension is to Mismatch Sampling for the Hamming distance with don't cares problem considered in [CEPR07] where a deterministic 1-mismatch algorithm allowing don't cares is developed. The sampling rate s from Section 6 can now be used to choose positions from the patterns with the remaining positions replaced with don't care characters. 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 that runs in $O((c + \log n)(n \log^2 m))$ time. The problem of sampling errors 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.

References

- [Abr87] K. Abrahamson. Generalized string matching. *SIAM journal on Computing*, 16(6):1039–1051, 1987.
- [ALP04] Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004.
- [CEPR07] R. Clifford, K. Efremenko, E. Porat, and A. Rothschild. k -mismatch with don't cares. In *European Symposium on Algorithm (ESA '07)*, pages 151–162, October 2007.
- [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.
- [Ind98] P. Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 166–173, 1998.

- [Kos87] S. R. Kosaraju. Efficient string matching. Manuscript, 1987.
- [LV86] G. M. Landau and U. Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986.
- [PE08] Ely Porat and Klim Efremenko. Approximating general metric distances between a pattern and a text. In Shang-Teng Huang, editor, *SODA*, pages 419–427. SIAM, 2008.