From coding theory to efficient pattern matching

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

Klim Efremenko
Bar-Ilan University, Dept. of Computer Science
52900 Ramat-Gan and
Weizmann Institute
Rehovot 76100, Israel

Ely Porat*
Bar-Ilan University, Dept. of Computer Science
52900 Ramat-Gan, Israel

Amir Rothschild
Bar-Ilan University, Dept. of Computer Science
52900 Ramat-Gan, Israel

Abstract

We consider the classic problem of pattern matching with few mismatches in the presence of promiscuously matching wildcard symbols. Given a text t of length n and a pattern p of length m with optional wildcard symbols and a bound k, our algorithm finds all the alignments for which the pattern matches the text with Hamming distance at most k and also returns the location and identity of each mismatch. The algorithm we present is deterministic and runs in $\tilde{O}(kn)$ time making it optimal to within logarithmic factors. The solutions we develop borrow from the tool set of algebraic coding theory and provide a new framework in which to tackle approximate pattern matching problems.

1 Introduction

Given a pattern represented as a string of characters, the problem of finding all its occurrences in some very large dataset is fundamental in computer science. As well as the theoretical interest that derives from elegant algorithmic solutions, entire industries are based on the ability to perform these searches quickly and accurately. The list of such applications provides us not only with the Internet search facilities now commonly taken for granted but also for example, the hugely successful field of high throughput computational genetics, which is the lifeblood of the pharmaceutical industry. However, despite the relatively long history of efficient algorithms for pattern matching, new problems in urgent need of solution arise continually.

A particular challenge for the design of efficient pattern matching algorithms is how to handle two or more different kinds of approximation simultaneously. In this paper we present a deterministic algorithm

*Research supported in part by the Binational Science Foundation (BSF)

which combines two of the most common and simplest forms of approximation in pattern matching. The first involves the introduction of promiscuously matching single character *wildcard* symbols which are said to match any symbol in the input alphabet. Wildcards are a very common way of describing unknown data and the problem of exact matching with wildcards is well studied [12, 13, 14, 10, 7]. The second form of approximation is pattern matching under the *Hamming distance*, where the number of mismatches is counted as the distance (see e.g. [1, 16, 15, 13, 4]). In the version we consider, the distance is only reported if it is less than or equal to some integer bound k. More formally, given a text t of length n and a pattern p of length m both which may contain optional wildcard symbols and a bound k, our algorithm finds all the locations for which the pattern matches the text with Hamming distance at most k. This combined approximate matching problem was first introduced recently and is known as *k-mismatch with wildcards* [8]. The algorithm we present here is deterministic and runs in $O(nk \log m(\log^2 k + \log\log m))$ time. As the algorithm also outputs the location and identity of all $O(nk)$ mismatches, the running time of our solution is within logarithmic factors of being optimal.

For exact matching with wildcards, the key observation given by [7] but implicit in previous work is that for numeric strings, if there are no wildcard symbols then for each location $0 \leq i \leq n - m$ we can calculate

$$\sum_{j=0}^{m-1}(p_j - t_{i+j})^2 = \sum_{j=0}^{m-1}(p_j^2 - 2p_j t_{i+j} + t_{i+j}^2)$$

in $O(n \log m)$ time using fast Fourier transforms (FFTs). Wherever there is an exact match this sum will be exactly 0. If p and t are not numeric, then an arbitrary one-to-one mapping can be chosen from the alphabet to the set of positive integers $\mathbb{N}$. In the case of matching with wildcards, each wildcard symbol in p or t is replaced by a 0 and the sum is modified to be

$$\sum_{j=0}^{m-1} p'_j t'_{i+j}(p_j - t_{i+j})^2$$

where $p'_j = 0$ ($t'_i = 0$) if p_j (t_i) is a wildcard symbol and 1 otherwise. This sum equals 0 if and only if there is an exact match with wildcards and can also be computed in $O(n \log m)$ time using FFTs.

Although the FFT has also been central to many of the algorithms for pattern matching under the Hamming distance, the fastest solutions for the bounded k-mismatch problem rely crucially on the ability to search a suffix tree of the pattern and text as well. For example, in 1985 Landau and Vishkin gave a beautiful $O(nk)$ algorithm for the k-mismatch problem that is significant as it does not require the FFT at all. Their method uses constant time lowest common ancestor queries on the suffix tree of p and t [18] in a technique that has subsequently come to be known as "kangaroo hopping". Almost 20 years afterwards, the asymptotic running time was finally improved in [4] to $O(n\sqrt{k \log k})$ time by a method based on filtering, the suffix tree (with kangaroo hopping) and FFTs again.

However, the use of suffix trees is problematic in the presence of wildcards. In particular, there is no known equivalent for the process of kangaroo hopping that runs in sublinear time when wildcards are permitted in the input. To compound the difficulties still further, the filtering method used to speed up search in [4] provides another seemingly insurmountable obstacle when wildcards are permitted in both the pattern and text. It would therefore appear that a completely different approach is required when the k-mismatch and wildcards matching problems are combined. Recently such a solution was given in a randomised setting by an algorithm that runs in $O(n(k + \log n \log\log n) \log m)$ time with high probability (w.h.p.) [8]. The same paper gave a deterministic algorithm based on group testing which runs in $O(nk^2 \log^3 m)$ time. Both results, which do not use suffix trees, rely on the ability to

sample single mismatch positions at every location of the pattern. This has left open the question of whether an $\tilde{O}(kn)$ time deterministic algorithm could be found, matching the best known randomised complexity and it is this question that we answer in the positive here.

We develop a new set of tools for approximate pattern matching and show how they can be used to circumvent the difficulties of using suffix trees or filters in the presence of wildcards. We find these tools by delving into the rich field of algebraic coding theory where we find deep connections to approximate pattern matching. The overall approach is to set up a system of equations whose form is similar to those found in error correcting codes, in our case the "key equation" of BCH codes. The solution to these equations will provide the locations of any mismatches. However, we can now use the full set of tools prevoiusly developed for decoding error correcting codes in order to solve the equations. We then show that by considering the input over a finite field of small characteristic this mismatch location step can be performed efficiently without randomisation.

We believe that our approach of exploiting the connections between approximate matching in large data sets and error correcting over noisy channels will be a fruitful one for further study. We hope that this work will be just the first step in this direction.

2 Related work and previous results

Fischer and Paterson [12] presented the first solution for the exact matching with wildcards based on fast Fourier transforms (FFT) in 1974. Their solution takes $O(n \log m \log |\Sigma|)$ time, where Σ is the alphabet from which the input symbols are chosen. Subsequently, the major challenge has been to remove this dependency on the alphabet size. Indyk [13] gave a randomised $O(n \log n)$ time algorithm which was followed by a simpler and slightly faster $O(n \log m)$ time randomised solution by Kalai [14]. In 2002, the first deterministic $O(n \log m)$ time solution was given [10] which was then further simplified in [7]. For a more limited version of the k-mismatch with wildcards problem, where wildcards are only permitted in either the pattern or text, but not both, a filtering algorithm has been developed which runs in $O(nm^{1/3}k^{1/3} \log^{2/3} m)$ time [9]. A variant of the edit-distance problem (see e.g. [17]) called the k-difference problem with wildcards was also considered in [2].

Two solutions for pattern matching without wildcards under the Hamming distance were given independently by both Abrahamson and Kosaraju in 1987 [1, 16] which both run in $O(n\sqrt{m \log m})$ time. These methods can also be easily extended to handle wildcards in both the pattern and text with little extra work because the algorithms count matches and not mismatches. The running time of these algorithms is independent of the bound k as they report the Hamming distance at every position irrespective of its value. The fastest known algorithm for the bounded k-mismatch problem without wildcards runs in $O(n\sqrt{k \log k})$ time [4]. A separate approach has been to find the Hamming distance within a $(1 + \epsilon)$ multiplicative factor. Karloff gave a such randomised algorithm that runs in $O((n/\epsilon^2) \log^2 m)$ [15] which was subsequently improved in [13] to $O(n/\epsilon^3 \log m)$.

The system of equations (4.1) set out in Section 4 was also used previously to solve a related problem known as the k-aligned ones with location problem [3]. However their relation to the encoding of BCH or error correcting codes in general does not appear to have been noticed at that time and so a relatively inefficient $\tilde{O}(k^3)$ time solution was given. Our improved method will therefore also provide a direct speedup for this previously considered problem.

3 Problem definition and preliminaries

Let Σ be a set of characters which we term the *alphabet*, and let ϕ be the wildcard symbol. Let $t = t_0 t_1 \dots t_{n-1} \in \Sigma^n$ be the text and $p = p_0 p_1 \dots p_{m-1} \in \Sigma^m$ the pattern. Both the pattern and text may also include ϕ in their alphabet. 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

the *position*.

- Define $HD(i)$ to be the Hamming distance between p and $t[i, \ldots, i+m-1]$ and define the wildcard symbol to match any symbol in the alphabet.
- We say that at position i in t, p is a k-mismatch if $HD(i) \leq k$.

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

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

can be calculated accurately and efficiently in $O(n \log n)$ time (see e.g. [11], Chapter 32). All cross-correlation calculations in this paper will in fact be performed over the Galois Field of characteristic 2. There are some technical complications with performing FFTs over fields where for example, 2 is not a unit and so we will require slightly more sophisticated methods. These are provided by a variant of the classic integer multiplication algorithm of Schönhage and Strassen which allows us to perform polynomial multiplication of degree n over a field of characteristic 2 in $O(n \log n \log \log n)$ arithmetic operations [20]. As our field size will be at most of size $O(m)$, we will represent the elements in normal basis form and build a logarithm table of size $O(m)$ to allow the multiplications to be performed in constant time. Throughout this paper we assume the RAM model when giving the time complexity of the FFT. This is in order to be consistent with the large body of previous work on pattern matching with FFTs. Specifically, the time complexities we give for the cross-correlation calculations over a finite field are in terms of the number of arithmetic operations required.

By a standard trick of splitting the text into overlapping substrings of length $2m$, the running time of polynomial multiplication can be further reduced to $O(n \log m \log \log m)$. This is also what allows us to consider the size of the field to be only $O(m)$. For ease of notation and presentation, we also often assume that the input length is an exact power of 2 and that logarithms are to be taken base 2.

4 Deterministic k-mismatch with wildcards

We can now present the algorithm for the k-mismatch pattern matching problem with wildcards. The overall structure of the solution is to first set up a system of equations whose solution will provide the position of the mismatches at each location in the text. This will be done by performing a number of cross-correlation calculations over a finite field of characteristic 2. We then use tools previously developed for decoding BCH and Reed-Solomon error correcting codes and an efficient polynomial root finding algorithm to solve these equations efficiently.

Before we start, we will require that the input pattern and text are transformed so that their symbols are chosen from a field of characteristic 2 that is sufficiently large to be able to index all the locations in each section of the text of length $2m$ and also represent all the symbols in the input alphabet. We choose the field $GF(2^{\log 4m})$. A one to one mapping from the input alphabet Σ to $GF(2^{\log 4m})$ can be made simply using a constant number sort operations and one linear traversal, taking $O(n \log n)$ time overall. We also require a mapping from $[2m]$ to $GF(2^{\log 4m})$ as we will use elements of the finite field to index the positions in the text. After this mapping has been made the input can be transformed as necessary. All operations on the symbols in the input will be carried out using operations in the finite field from here on. Here and in the remainder of this paper we will use $\sum$ as an abbreviation for $\sum_{j=0}^{m-1}$.

We can now calculate $2k+1$ arrays such that

$$s_{\ell,i} = \sum (p_j - t_{i+j})(i+j)^{\ell} p'_j t'_{i+j} \tag{4.1}$$

where ℓ is in the range $0 \leq \ell \leq 2k$. By abuse of notation we write p_j, t_{i+j} and $i+j$ when we mean the corresponding elements in $GF(2^{\log 4m})$. We also create binary arrays p' and t' so that $p'_j = 0$ ($t'_j = 0$) if $p_j = \phi(t_i = \phi)$ and $p'_j = 1$ ($t'_j = 1$) otherwise.

It follows that for any given alignment i,

$$s_{\ell,i} = \sum_{j=1}^{HD(i)} r_j x_j^{\ell}, \tag{4.2}$$

where r_j is difference between the pattern and text at the jth mismatch between p and $t[i, \ldots, i+m]$. Therefore, if there is a k'-mismatch at position i then we can rewrite (4.1) to be in terms of the unknown mismatch positions x_j and the differences found at those positions $r_j = p_{j'} - t_{i'+j'}$.

$$\begin{array}{ccccccccc}
r_1 & + & r_2 & + & \ldots & + & r_{k'} & = & s_{0,i} \\
r_1 x_1 & + & r_2 x_2 & + & \ldots & + & r_{k'} x_{k'} & = & s_{1,i} \\
r_1 x_1^2 & + & r_2 x_2^2 & + & \ldots & + & r_{k'} x_{k'}^2 & = & s_{2,i} \\
 & & & \vdots & & & & & \vdots \\
r_1 x_1^{2k} & + & r_2 x_2^{2k} & + & \ldots & + & r_{k'} x_{k'}^{2k} & = & s_{2k,i}
\end{array}$$

A solution to this set of equations will give the positions of the mismatches between p and $t[i, \ldots, i+m]$. We first show that these equations can be set up in $O(nk \log m \log\log m)$ time.

THEOREM 4.1. *The $2k+1$ arrays with elements $s_{\ell,i}$ can be calculated in $O(nk \log m \log\log m)$ time*

Proof. Observe that for any ℓ, $s_{\ell,i}$ can be written as:

$$\sum (p_j - t_{i+j})(i+j)^{\ell} p'_j t'_{i+j} =$$
$$\sum (p_j p'_j)(i+j)^{\ell} t'_{i+j} - \sum (p'_j (i+j)^{\ell} t_{i+j} t'_{i+j})$$

Therefore $s_{\ell,i}$ can be computed for all $i = 0, 1, \ldots n-m$ by two cross-correlation calculations. In order to compute $s_{\ell,i}$ for every i and ℓ we therefore require $O(k)$ cross-correlations in total.

The task now is to solve the set of simultaneous equations efficiently for each i. We consider the set of equations for each alignment independently from each other. As a result we fix some i and make all the remaining computations with respect to this location i. We also write s_ℓ where it is understood that this refers to $s_{\ell,i}$ for some location i and set $k' = HD(i)$. We will also assume for the moment that $k' = HD(i) \leq k$ and show later how we can handle the case where the Hamming distance is large than k at location i. The two main steps taken to solve the system of equations are as follows.

1. Calculate the coefficients of the polynomial

$$P(z) = \prod_{i=1}^{k'} (x_i z - 1)$$

   from the $2k+1$ values s_ℓ.

2. Find the roots of the polynomial $P(z)$. The inverse of these roots will give us the position of the mismatches of the pattern and text.

Given the equations (4.1), step 1 can be computed using the Berlekamp-Massey algorithm [5, 19] applied over a field of characteristic 2. The problem that Berlekamp-Massey solves is as follows. Take

as input a sequence of elements $s_0, s_1, \ldots s_{2k}$ and output the shortest linear recursion on this sequence. That is, the shortest sequence $a_0, a_1, \ldots a_v$ such that $\sum_{i=0}^{v} s_{i+j} a_i = 0$ for every $j = 0, 1, \ldots 2k - v$. Fast solutions to this problem play a key role in decoding BCH codes for example and the current best upper bound gives a time complexity of $O(k \log^2 k)$ [6]. Another equivalent formulation of the problem is given polynomial $F(z)$ find the polynomial with smallest degree $H(z)$ such that $F(z)H(z) \mod z^{degF(z)+1}$ has degree less than the degree of $H(z)$.

THEOREM 4.2. *For a fixed location i and set of $2k+1$ sums s_ℓ, if $k' = HD(i) \leq k$, then the fast Berlekamp-Massey algorithm applied to all s_ℓ for $0 \leq \ell \leq 2k$ will return the coefficients of the polynomial $P(z) = \prod_{i=1}^{k}(zx_i - 1)$.*

Proof. We have assumed that there are $k' < k$ mismatches at the current location and that the degree of $P(z)$ is therefore k'. We define the polynomial

$$F(z) = \sum_{\ell=0}^{2k} s_\ell z^\ell.$$

If we run the fast Berlekamp-Massey algorithm on the polynomial $F(z)$ then it will return some polynomial $\tilde{P}(z)$ with smallest degree such that $F(z)\tilde{P}(z) \mod z^{2k+1}$ has degree less than k. We want to prove that $\tilde{P}(z) = P(z)$. In order to do this we have to prove that

1. $F(z)P(z) \mod z^{2k+1}$ has degree less than k.
2. $P(z)$ is the polynomial with smallest degree for which $F(z)P(z) \mod z^{2k+1}$ has degree less than k.

By Equation (4.2) we have

$$F(z) = \sum_{\ell=0}^{2k} \left(\sum_{i=1}^{k'} r_i x_i^\ell \right) z^\ell$$

A change of order of the summation gives us

$$F(z) = \sum_{i=1}^{k'} r_i \sum_{\ell=0}^{2k} (x_i z)^\ell$$

Therefore

$$F(z)P(z) = \sum_{i=1}^{k'} r_i \left(\sum_{\ell=0}^{2k} (x_i z)^\ell \prod_{j=1}^{k'} (zx_j - 1) \right)$$

Recall that

$$\sum_{\ell=0}^{2k} (x_i z)^\ell (zx_i - 1) = (x_i z)^{2k+1} - 1$$

So we have

$$F(z)P(z) = \sum_{i=1}^{k'} r_i ((zx_i)^{2k+1} - 1) \prod_{j=1, j\neq i}^{k'} (zx_j - 1)$$

And therefore taking both sides $\mod z^{2k+1}$ we get

$$F(z)P(z) \mod z^{2k+1} = -\sum_{i=1}^{k'} r_i \prod_{j=1, j\neq i}^{k} (zx_j - 1)$$

Therefore $F(z)P(z) \mod z^{2k+1}$ is a polynomial of degree $k' - 1 \leq k - 1$ and the first part is proved.

Now let us assume that there is a linear recursion $\{a_i\}_{i=0}^{k'-1}$ of degree $k'-1$ such that $\sum_{i=0}^{k'-1} a_i s_{i+j} = 0$ for every $j = 0 \ldots 2k - k'$. If so we can rewrite it as

$$\sum_{i=0}^{k'-1} a_i s_{i+j} = \sum_{i=0}^{k'-1} a_i \sum_{l=0}^{k'-1} r_l x_l^{i+j} = 0 \tag{4.3}$$

Consider the same equations in matrix form so that $\vec{a}M = \vec{0}$ where M is a k' by k' matrix with $M(i,j) = \sum_{v=1}^{k'} r_v x_v^{i+j}$ and $\vec{a} = (a_0, a_1, \ldots a_{k'-1})$. Now we want to prove that $\det M \neq 0$ as then it will follow that $\vec{a} = \vec{0}$ and therefore a_i is not a linear recursion for s_ℓ.

We now show that $\det M \neq 0$. Consider the Vandermonde matrix $V(i,j) = x_i^j$. Then $M = DV^TV$ where D is a diagonal matrix with r_ℓ on the diagonal. We know that the determinant of a Vandermonde matrix is non zero. Therefore $\det(M) = \det(DV^TV) = \det(D)\det^2(V) \neq 0$.

Now what remains is to find the mismatch positions. These are encoded as the inverse of the roots of the polynomial $P(z) = \prod_{i=1}^{k}(zx_i - 1)$ which will again be computed separately for each location in the text. In general, factorising a polynomial deterministically is a hard problem. However, in our cas e we are only interested in degree one factors and have chosen the field that we are working in to have small characteristic. This allows us to apply an efficient polynomial root finding technique.

THEOREM 4.3. *Given the polynomial $P(z) = \prod_{i=1}^{k}(zx_i - 1)$, its roots can be found deterministically in $O(k \log^2 k \log m)$ time in the field $GF(2^{\log 4m})$.*

Proof. This is an immediate consequence of the factorisation technique of [21] applied over a finite field with small characteristic. The complexity follows from the fact that we only need to find linear factors.

For any location i such that $HD(i) \leq k$, we can now find all the mismatch locations as required. However, if the Hamming distance is in fact larger than k there is no guarantee what the output will be from the above process. We therefore require one further verification stage which will check that the mismatches we find are all the mismatches that occur at a given location. This final check can be performed in a number of ways with the following being a particularly simple example. First, using integers instead of elements of a finite field, we compute the sum

$$D[i] = \sum (p_j - t_{i+j-1})^2 p'_j t'_{i+j-1}.$$

We now go through the list of $O(nk)$ mismatch positions found and "correct" $D[i]$ at each location i where a mismatch is found. Given that we know the mismatch locations we can simply calculate $(p_j - t_{i+j-1})^2$ for those positions and subtract it from $D[i]$. If all mismatches have been found then at the end $D[i] = 0$. Otherwise, it must be that the true Hamming distance was greater than k at that location. The time for this verification stage is $O(n \log m + nk)$ and so does not affect the overall running time.

THEOREM 4.4. *Algorithm 1 solves the k-mismatch with wildcards problem in $O(nk \log m(\log^2 k + \log\log m))$ time.*

Proof. The time to perform the encoding step is $O(nk \log m \log \log m)$ following Theorem 4.1. The time to create the error locater polynomials for every position in the text is $O(nk \log^2 k)$ in total following Theorem 4.2. The time needed to find all the roots of all the polynomials is $O(nk \log^2 k \log m)$ by Theorem 4.3. Finally an extra $O(n \log m + nk)$ time is needed to verify which of the locations found were in fact k-mismatches. The total time taken by Algorithm 1 to solve the k-mismatch with wildcards problem is therefore $O(nk \log m(\log^2 k + \log \log m))$

Input: Pattern p, text t and an integer k
Output: Positions i where $HD(i) \leq k$ and all mismatch locations
/* Encoding
Create syndrome equations for code with up to k errors
/* Decoding
Create error locater polynomial $P(z) = \prod_{i=1}^{k}(z - x_i)$
Find the roots of $P(z)$ over a field of small characteristic
/*Verification
Check that all mismatches are found at each position i

Algorithm 1: Deterministic k-mismatch with wildcards

References

[1] K. Abrahamson. Generalized string matching. *SIAM journal on Computing*, 16(6):1039–1051, 1987.
[2] T. Akutsu. Approximate string matching with don't care characters. *Information Procesing Letters*, 55:235–239, 1995.
[3] A. Amir and M. Farach. Efficient 2-dimensional approximate matching of half-rectangular figures. *Information and Computation*, 118(1):1–11, 1995.
[4] Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004.
[5] E. R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill, New York, 1968.
[6] R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun. Fast solution of toeplitz systems of equations and computation of padé approximants. *Journal of Algorithms*, 1(3):259–295, 1980.
[7] P. Clifford and R. Clifford. Simple deterministic wildcard matching. *Information Processing Letters*, 101(2):53–54, 2007.
[8] 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.
[9] R. Clifford and E. Porat. A filtering algorithm for k-mismatch with don't cares. In *14th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 130–136, October 2007.
[10] R. Cole and R. Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 592–601, 2002.
[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.
[12] M. Fischer and M. Paterson. String matching and other products. In R. Karp, editor, *Proceedings of the 7th SIAM-AMS Complexity of Computation*, pages 113–125, 1974.
[13] 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.
[14] A. Kalai. Efficient pattern-matching with don't cares. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 655–656, 2002.
[15] H. Karloff. Fast algorithms for approximately counting mismatches. *Information Processing Letters*, 48(2):53–60, 1993.
[16] S. R. Kosaraju. Efficient string matching. Manuscript, 1987.

[17] G. M. Landau and U. Vishkin. Efficient string matching in the presence of errors. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 126–136, 1985.

[18] G. M. Landau and U. Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986.

[19] J. L. Massey. Shift-register synthesis and bch decoding. *IEEE Trans. on Information Theory*, 15:122–127, 1969.

[20] A. Schönhage. Schnelle multiplikation von polynomen über körpern der charakteristik 2. *Acta Inform.*, 7:395–398, 1976.

[21] Victor Shoup. A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic. In *ISSAC '91: Proceedings of the 1991 international symposium on Symbolic and algebraic computation*, pages 14–21, New York, NY, USA, 1991. ACM Press.