Faster algorithms for string matching problems: matching the convolution bound

Piotr Indyk*
Department of Computer Science
Stanford University
indyk@cs.stanford.edu

Abstract

In this paper we give a randomized $O(n \log n)$-time algorithm for the *string matching with don't cares* problem. This improves the Fischer-Paterson bound [10] from 1974 and answers the open problem posed (among others) by Weiner [30] and Galil [11]. Using the same technique, we give an $O(n \log n)$-time algorithm for other problems, including *subset matching* and *tree pattern matching* [15, 21, 9, 7, 17] and *(general) approximate threshold matching* [28, 17]. As this bound essentially matches the complexity of computing of the Fast Fourier Transform which is the only known technique for solving problems of this type, it is likely that the algorithms are in fact optimal. Additionally, the technique used for the threshold matching problem can be applied to the *on-line* version of this problem, in which we are allowed to preprocess the text and require to process the pattern in time sublinear in the text length. This result involves an interesting variant of the Karp-Rabin fingerprint method [22] in which hash functions are *locality-sensitive* [25], i.e. the probability of collision of two words depends on the distance between them.

1 Introduction

In 1974 Fischer and Paterson [10] observed that several string matching problems can be viewed as linear products over proper algebraic structures. As a result, they obtained $O(n \log m \log |\Sigma|)$-time[1] algorithm for the problem of string matching with *don't cares*, or $*$-*matching* for short (here and in the whole paper n denotes the length of the text, m denotes the length of the pattern and Σ denotes the alphabet). This approach turned out to be one of the most powerful tools of stringology, resulting in the design of efficient algorithms for many problems, including *pattern matching with mismatches* and *generalized string matching* [5, 20, 18], *two-dimensional approximate matching* [2, 3] and *tree pattern matching* [15, 21, 9, 7, 17]; it also influenced complexity and structural investigations in pattern matching [27, 26]. However, to this date, no asymptotic improvement over the Fischer-Paterson bound has been made, although Muthukrishnan and Palem [26] managed to reduce the constant factor. Determining the complexity of this problem appeared as an open problem in the list by Galil [11] and is one of few problems from this list for which no progress has been made up to date. In fact, Muthukrishnan and Palem [26] showed a lower bound of $\Omega(\log |\Sigma|)$ for the number of boolean products needed to solve this problem. As the only known way to compute boolean products is to use a Fast Fourier Transform (with running time $O(n \log m)$), it seemed likely that the Fischer-Paterson bound was optimal.

*Supported by a Stanford Graduate Fellowship and NSF Award CCR-9357849.
[1] In the RAM model; the times are slightly higher for the bit-cost model, which we do not consider in this paper

In this paper we introduce a new randomized technique for computing boolean products, which yields an $O(n \log n)$-time[2] Monte Carlo algorithm for the $*$-matching problem. In fact, the technique is applicable to a wide class of string matching problems, including tree pattern matching, approximate threshold matching and approximate general threshold matching (see the definitions and background in Section 1.1). In all cases, our technique yields randomized algorithms with $O(n \log n)$ running time; the algorithms are simple and easy to implement. Thus for all of these problems we obtain upper bounds essentially matching the complexity of the Fast Fourier Transform (FFT). As the latter is the only known technique for solving this class of problems, it is likely that these bounds are in fact optimal (see also Conclusions).

The technique which we use for the threshold matching problem turns out to have additional applications to the *on-line* version of this problem, in which we are allowed to preprocess the text and require to process the pattern in time sublinear in the text length. This result involves an interesting variant of the Karp-Rabin fingerprint method [22] in which hash functions are *locality-sensitive* [25], i.e. the probability of collision of two words depends on the distance between them.

1.1 Problem definitions, background and results

In this section we define the problems considered in this paper and present the currently known upper bounds. Throught this paper we denote the alphabet of symbols by Σ and we assume $\Sigma = \{0 \ldots k-1\}$.

Definition 1 *The* string matching with don't cares *(or $*$-matching) is defined as follows: given a (text) string $t[0 \ldots n-1]$ and a (pattern) string $p[0 \ldots m-1]$ such that $t[i] \in \Sigma \cup \{*\}, p[j] \in \Sigma \cup \{*\}$ for $i = 0 \ldots n-1, j = 0 \ldots m-1$, output a binary array $o[0 \ldots n-m]$ such that $o[i] = 1$ iff for all $j = 0 \ldots m-1$ either $p[j] = t[i+j]$ or $p[j] = *$ or $t[i+j] = *$.*

The first (and essentially the only) algorithm for this problem was given by Fischer and Paterson [10]. They showed that this problem can be reduced to $2 \log |\Sigma|$ boolean convolutions of a binary sequence of length m with a sequence of length n. As the latter can be done in $O(n \log m)$ time using the FFT, the bound follows. The constants were improved later by Muthukrishnan and Palem [26], who showed that $\log |\Sigma| + o(\log |\Sigma|)$ convolutions are sufficient (and neccessary in a restricted model).

Definition 2 *The* subset matching problem *is defined as follows: given a (text) string $t[0 \ldots n-1]$ and a (pattern) string $p[0 \ldots m-1]$ such that $t[i] \subset \Sigma, p[j] \subset \Sigma$ for $i = 0 \ldots n-1, j = 0 \ldots m-1$, output a binary array $o[0 \ldots n-m]$ such that $o[i] = 1$ iff $p[j] \subset t[i+j]$ for all $j = 0 \ldots m-1$.*

This problem was defined by Cole and Hariharan [7]; in the same paper they gave an $O(n \log^2 m)$ algorithm for this problem. The importance of subset matching comes from the fact that (as shown in [7]) the *tree pattern matching* problem can be reduced to it with only a constant factor increase in space and time. The latter problem has been heavily investigated: the first $O(nm)$ bound from [15] was subsequently improved to $\tilde{O}(nm^{3/4})$ [20] to $\tilde{O}(nm^{1/2})$ [9] and finally to $O(n \log^2 m)$ [7] (the last bound comes from the above reduction). The subset matching problem turns out to be of interest for other reasons as well. For example, it was shown in [16] that the application of subset matching enables the design of efficient algorithms for several geometric pattern matching problems. It was also shown [17] that subset matching can be used to give a fast solution to general pattern matching [26].

[2] The $\log n$ factor (instead of $\log m$) comes from our goal to obtain an algorithm with probability of failure inversely polynomial in n. If we needed only to ensure that each occurence is reported with probability $1-\delta$, the factor would be $\log 1/\delta$.

Definition 3 *The* ϵ-threshold matching *problem is defined as follows: given an integer* $k > 0$*, a (text) string* $t[0 \ldots n-1]$*, a (pattern) string* $p[0 \ldots m-1]$ *such that* $t[i] \in \Sigma, p[j] \in \Sigma$ *for* $i = 0 \ldots n-1, j = 0 \ldots m-1$*, output a binary array* $o[0 \ldots n-m]$ *such that*

- $o[i] = 0$ *if the cardinality of the set of* mismatches $\{j | p[j] \neq t[i+j]\}$ *is less than* k*,*
- $o[i] = 1$ *if the cardinality of the set of mismatches is greater than* $(1+\epsilon)k$*,*
- $o[i]$ *is either* 0 *or* 1 *otherwise*

The exact version of the ϵ-threshold matching problem was defined by Galil [11]. The papers by Landau and Vishkin [24] and Galil and Giancarlo [13] gave algorithms working in $\tilde{O}(n(k+m))$ time. The more general problem of *counting* the number of mismatches was shown by Kosaraju [20] and Abrahamson [5] to be solvable in $\tilde{O}(n\sqrt{m})$ time. Karloff [18] showed that approximating the number of mismatches up to a factor of $(1+\epsilon)$ can be done in randomized $O(n \log n \log m/\epsilon^2)$ time. In this paper we give a randomized algorithm for ϵ-threshold matching running in $O(n \log n/\epsilon^3)$ time. Note that by applying binary search on k (and thus multiplying the time by $O(\log \log_{1+\epsilon} n)$) we can find the alignment of the pattern in the text which approximately minimizes the number of mismatches. This in turn allows us to speed-up the algorithm for so-called *inverse pattern matching problem* by Amir et al [1], by using our algorithm instead of Karloff's.

Definition 4 *The* general ϵ-threshold matching problem *is defined as follows: given a (text) string* $t[0 \ldots n-1]$*, a (pattern) string* $p[0 \ldots m-1]$ *such that* $t[i] \in \Sigma, p[j] \in \Sigma$ *for* $i = 0 \ldots n-1, j = 0 \ldots m-1$*, a number* $k > 0$ *and a match graph* $M \subset \Sigma \times \Sigma$ *of maximum degree* d*, output an array* $o[0 \ldots n-m]$ *such that* $o[i] = 1$ *if the value of* $\sum_{j=0}^{m-1} M(t[i+j], p[j])$ *is approximately greater than* k *(in the sense of the above definition) and* $o[i] = 0$ *otherwise. We define* $M(x,y)$ *to be equal to* 1 *if* $(x,y) \in M$ *and* 0 *otherwise.*

In [28] a $O(nd \log^2 n)$-time randomized algorithm was given which approximated the number of mismatches up to a multiplicative factor of $\log m$. In [17] an algorithm was given which for any $\epsilon > 0$ finds a $(1+\epsilon)$-approximation in randomized $O(nd \log^2 n/\epsilon^2)$ time. Here we give a bound of $O(nd \log n/\epsilon^3)$ for the decision version of that problem.

Definition 5 *The* on-line ϵ-threshold matching problem *is defined as follows: given an integer* $k > 0$*, a (text) string* $t[0 \ldots n-1]$*, devise a data structure which given any pattern* $p[0 \ldots m-1]$*:*

- *outputs* YES *if there is* i *such that the cardinality of the set of* mismatches $\{j | p[j] \neq t[i+j]\}$ *is less than* k*,*
- *outputs* NO *if the cardinality of the set of mismatches is greater than* $(1+\epsilon)k$*,*
- *outputs* YES *or* NO *otherwise.*

We give an algorithm which after $O(n \log n)$ time preprocessing, $O(n)$ storage and query time $O(m)$. The probability of correctness of the answer is constant; it can be amplified to $1-\delta$ by using $\log 1/\delta$ data structures in parallel (for any $\delta > 0$). The approximation factor is $\epsilon = O(\log n)$. By using the techniques of [25] one can obtain an algorithm with preprocessing/storage $\tilde{O}(n^{1+1/(1+\epsilon)})$ and query time $\tilde{O}(mn^{1/(1+\epsilon)})$.

We are not aware of any previous result of this type (the direct application of the techniques from [25] would yield $\Omega(mn)$ preprocessing time). However, our result implies that the ϵ-threshold matching with *multiple patterns* $p_1 \ldots p_l$ can be solved in time $O(n \log n + ml)$. To our best knowledge, this is the first time when the convolution-based approach gives a multiple pattern matching algorithm with almost linear complexity (see [28], p. 317, Problem 9.5).

2 Preliminaries

All of the algorithms given in this paper assume the Random Access Machine (RAM) model [6]. More specifically, we assume that all the usual operations (boolean, arithmetic etc) on arguments of size $O(\log n)$ can be performed in constant time. We do not assume the availability of any non-standard operations (like multiplication in $GF(2^s)$ etc) as such an assumption is often not satisfied in practice.

Convolutions. Let $a[0 \ldots n-1]$ and $b[0 \ldots m-1]$ be vectors of elements over a semiring $R = (\oplus, \otimes)$. The *convolution* of a and b is defined as a vector $a *_R b$ such that $(a *_R b)[i] = \oplus_{j=0}^{m-1}(a[i-j] \otimes b[j])$ for $0 \leq i \leq n-m$ (out-of-range values are assumed to be 0). The index R will be skipped if it follows from the context. Notice that convolution of a and b corresponds naturally to multiplication of univariate polynomials $\oplus_{j=0}^{n-1} a[j]x^j$ and $\oplus_{j=0}^{m-1} b[j]x^j$. We also define $a \cdot b = \oplus_i a[i] \otimes b[i]$.

In this paper we consider the following cases:

- *boolean convolution*, where $R = B = (\vee, \wedge)$
- *polynomial convolution*, where $R = P = (+, \times)$ is a field (in this paper we consider only the Galois field $\mathrm{GF}(2^s)$)

We also define $a \circ b$ to be an array such that $(a \circ b)[i] = \oplus_{j=0}^{m-1}(a[i+j] \otimes b[j])$ for $0 \leq i \leq n+m-2$. It it easy to see that this operation can be computed within the same bound as convolution and vice versa. Thus in the following we will use either $\circ$ or $*$ depending on which is more convenient.

The following fact is well known:

Fact 1 *If $n | 2^s - 1$, then the polynomial convolution can be performed in $O(n \log n)$ time.*

Other notation. Let F be a field. The ring of univariate polynomials over F is denoted by $F[x]$. The subset of $F[x]$ of polynomials of degree up to d is denoted by $F^d[x]$.

Let H^d denote the Hamming space, i.e. the set $\{0,1\}^d$. The Hamming distance between two vectors u, v is defined as the number of coordinates on which x and y differ. The weight of x is defined as the number of 1's in x. For any binary operation $\bullet$ on $\{0,1\}$ we define its extension to $\{0,1\}^d$ as $(a \bullet b)[i] = a[i] \bullet b[i]$.

We use $[n]$ to denote the set $\{0 \ldots n-1\}$.

3 Randomized boolean convolution

In this section we describe the $O(n)$-time randomized algorithm for boolean convolution, which is the basis for the rest of our results. The algorithm outputs an array $\hat{o}$ which approximates $a * b$ in the sense that for each i we have $\hat{o}[i] = (a * b)[i]$ with large (constant) probability.

In order to obtain this result we first give an $O(n)$-time algorithm for multiplying two n-degree polynomials p and q over $GF(2)$. The algorithm consists of two steps:

1. Reduce multiplication of p and q to a multiplication of two polynomials p' and q' of degree n/t over $GF(2^t)$, such that $\frac{n}{t} = 2^t$, for $t = O(\log n)$,
2. Multiply p' and q' over $GF(2^t)$.

The first step was given by Cantor [8]; it essentially relies on partitioning the coefficients of p and q into intervals of length $t/2$ and treating each interval as a coefficient over $GF(2^t)$. The second step can be done using $O(\frac{n}{t} \log \frac{n}{t}) = O(n)$ operations in $GF(2^t)$, so it is sufficient to show how to implement each such operation in $O(1)$ time. We will give such an implementation by resorting to lookup tables. We note that an $O(1)$-time algorithm for multiplication over $GF(2)$ using lookup tables was independently discovered in [23]. However, their algorithm is more complicated than ours and has larger constants in the running time.

We will represent the elements of $GF(2^t)$ as polynomials over $GF(2)$. The addition can be performed easily, so we focus on multiplying, say $a(x)$ and $b(x)$. To perform multiplication we need to:

- compute the product $d(x)$ of polynomials $a(x)$ and $b(x)$
- compute $d(x)$ mod $u(x)$, where $u(x)$ is an irreducible polynomial of degree t (which can be found in negligible time during preprocessing, see [12])

We will implement both operations using precomputed lookup tables. Note however that we cannot simply store products of all polynomials a and b, as the preprocessing time and the size of the lookup table would be almost quadratic. Therefore, we reduce each operation to $O(1)$ operations operating on inputs of size t/c, for $c > 1$ large enough to precompute all answers in advance in sublinear time. For the multiplication, this reduction is given already by Cantor's method. For the division we proceed as follows. Consider the natural $O(t^2)$-time algorithm (call it SHIFT) which for $i = 2t-1 \dots t$ performs the following steps (assuming $d^{2t-1} = d$):

- check if the ith coefficient of d^i is 1
- if so, assign $d^{i-1} = d^i - u$; otherwise set $d^{i-1} = d^i$

This algorithm can be implemented in $O(t)$ parallel steps, each step involving subtraction of polynomials, which can be easily implemented using $O(1)$ boolean operations [23]. Here we show that we can accelerate this algorithm by reducing the number parallel steps to $O(1)$. The main idea is to shift the polynomial u by t/c (instead of 1) positions during each step, where each step is performed using a lookup table. More specifically, we will split the sequence of coefficients of d and u into intervals $d_1 \dots d_a$ and $u_1 \dots u_b$; each d_i, u_i being of length t/c. Notice that the behavior of the first t/c steps of the algorithm (i.e. the times when u was subtracted) depends only on d_1 and u_1. We can represent this behavior by a binary sequence s of length t/c (containing 1 if u was subtracted and 0 otherwise) and store all such sequences for all pairs (d_1, u_1), so that we can later find s in constant time. We also notice that the value of d_i after first t/c steps depends only on s, d_i and u_j, u_{j-1}, where u_j and u_{j-1} are the only two parts of u overlapping with d_i. More specifically, the value of d after t/c changes by the value of the product of s (viewed as a polynomial) and u shifted by a proper number of steps; the product can be computed by summing up products of s and u_i, each of which can be computed by using the lookup table. Thus we can simulate the first t/c steps of SHIFT in $O(1)$ time; it is easy to see that the remaining steps can be done in the same way as well.

The randomized boolean convolution can be now computed as follows.

Lemma 1 *Let u, v be vectors from H^n. There exists a randomized algorithm which in $O(n)$ time computes an approximation of the boolean convolution of u and v in the sense that it outputs a vector $w \in H^{2n}$ such that for any i :*

1. *If $(u * v)[i] = 0$ then $w[i] = 0$,*
2. *If $(u * v)[i] = 1$ then $\Pr[w[i] = 1] = \frac{1}{2}$*

Proof: We apply *Freivalds' technique* (see for instance [27], p. 162). More specifically, the algorithm computes a vector $w = (r \wedge u) * v$, where r is a random element of H^n and the convolution is taken over the $GF(2)$. The correctness follows from standard arguments. $\Box$

4 Applications

In this section we give applications of randomized boolean convolution to the above defined problems.

String matching with don't cares. Let $d = c \log n$ for a proper constant $c > 0$ specified later. Also, for a given sequence $r_0 \ldots r_{k-1}$ of elements of H^d define functions $f, g : \Sigma \to H^d$ as:

$$f(a) = \begin{cases} r_a & \text{if } a \in \Sigma \\ 0^d & \text{if } a = * \end{cases}$$

and

$$g(a) = \begin{cases} \overline{r_a} & \text{if } a \in \Sigma \\ 0^d & \text{if } a = * \end{cases}$$

The algorithm is as follows:

- choose r_i $(i = 0 \ldots k-1)$ independently and uniformly at random from H^d
- compute binary vectors $p^i = f(p[0])[i] \ldots f(p[m-1])[i]$ and $t^i = g(t[0])[i] \ldots g(t[n-1])[i]$, for $i = 0 \ldots d-1$
- compute the randomized boolean convolution $a^i = t^i \circ p^i$ for $i = 0 \ldots d-1$
- compute $a = \vee_{i=1}^{d} a^i$
- output $\overline{a}$ as the occurrence vector

Lemma 2 *For any $\alpha > 0$ there is a constant $c > 0$ such that the occurrence vector generated by the algorithm is correct with probability $1 - \frac{1}{n^\alpha}$.*

Proof: It is sufficient to show for each $i = 0 \ldots n-m$ that

1. If p occurs in t at position i, then $a^j[i] = 0$ with probability 1 for any $j = 0 \ldots d-1$
2. If p does not occur in t at position i, then for any $j = 0 \ldots d-1$ we have $\Pr[a^j[i] = 0] > \delta$, for some constant $\delta > 0$.

Case 1. If p occurs in t at position i, then all corresponding pairs of symbols $(p[j], t[i+j])$ are either of the form $(*, a)$ or $(a, *)$ or $(*, *)$ or (a, a), for $a \in \Sigma$. It is easy to see that for any such pair (c, d) we have $f(c) \wedge g(d) = 0^d$; thus $a^j[i] = 0$ for all $j = 0 \ldots d-1$.

Case 2. If p does not occur in t at position i, then there exists j such that $p[j] = a \in \Sigma, t[i+j] = b \in \Sigma$ and $a \neq b$. As $\Pr[r_a[k] = \overline{r_b[k]} = 1] = \frac{1}{4}$ for any $k = 0 \ldots d-1$ (assuming $a \neq b$), we know that the ith bit of the boolean convolution of t^k and p^k is non-zero with the same probability. Thus (by Lemma 1) there is a constant probability $\delta > 0$ that $a^k[i] = 1$, which implies that $\Pr[\overline{a[i] = 1}] \leq (1-\delta)^d = \frac{1}{n^\alpha}$. $\Box$

Subset matching. Cole and Hariharan [7] give a very elegant $O(n \log^2 n)$-time randomized algorithm for this problem. Firstly, they observe that the output o remains unchanged if all occurrences of any symbol are "shifted" in both the text and the pattern by the same amount. More formally, they observed the following. Define 0-1 matrices $T[0 \ldots k-1, 0 \ldots n-1]$ and $P[0 \ldots k-1, 0 \ldots n-1]$ such that $T[a,i] = 1$ iff $t[i] = a$ and $P[a,i] = 1$ iff $p[i] = a$. It is easy to see that p matches t at position i iff

$$\forall_{j=0 \ldots m-1} \forall_a (P[a,j] = 1) \Rightarrow T[a, i+j] \tag{1}$$

For any vector X of length n and any $i = 0 \ldots n-1$ define $X+i$ to be a vector of length n such that for any $j = 0 \ldots n-1$ $X[j] = (X+i)[(j+i) \bmod n]$. We say that $X+i$ is a result of *shifting* X by i positions. Cole and Hariharan [7] observed that the condition (1) remain invariant when the rows of both T and P are shifted by the same number of positions. Based on this observation they shift all occurrences of each symbol a by a random amount $l_a \in [cn]$ for some constant $c > 1$, obtaining a (slightly longer) pattern p' and text t'. Since $s \leq m$ and the shifts are done independently for each column, it can be shown that for each $u \in [(c+1)n]$, the probability that $t'[u]$ contains the empty set is lower bounded by a constant $\delta > 0$. Therefore for any position $i \in [n-m]$ at which the pattern does not occur (i.e. there is an offset j and symbol a such that $a \in p[j] - t[i+j]$) there is a constant probability that $t'[i+j'] = \phi \neq \{a\} \subset p'[j']$ for $j' = j + l_a$. However the existence of such a j' can be checked for all i's by using convolution. More specifically, we define a binary vector $t'' \in H^n$ such that $t''[i] = 1$ iff $t'[i] = \phi$; we also define $p'' \in H^m$ such that $p''[i] = 1$ iff $t'[i] \neq \phi$. In order to detect a mismatch at position i it is now sufficient to check if $(t'' \circ_B p'')[i] = 1$. Each such an operation detects a mismatch with probability δ. Therefore repeating the above procedure $O(\log n)$ times one obtains an algorithm which detects all mismatches with high probability.

One can easily see that if we replace the exact computation of $t'' \circ_B p''$ by the probabilistic one, the probability of mismatch detection is still at least $\frac{\delta}{2}$. Thus, by iterating this procedure we obtain an algorithm working in $O(n \log n)$ time with high probability.

Approximate threshold matching. Karloff [18] gave the following randomized algorithm for the ϵ-counting matching. He chose a sequence of s random elements $r_0 \ldots r_{s-1}$ from H^d, where $d = O(\log n / \epsilon^2)$. With high probability (by Chernoff bound) for any pair of elements r_i, r_j, $i \neq j$ we have $d\frac{1-\epsilon/2}{2} \leq d_H(r_i, r_j) \leq d\frac{1+\epsilon/2}{2}$. Thus by encoding each symbol a by r_a he reduced the original problem to counting matching over a binary alphabet, which can be done easily in $O(n \log n)$ time using convolutions. More specifically, if for each $i = 0 \ldots d$ we create $t^i = r_{t[0]}[i] \ldots r_{t[n-1]}[i]$ and $p^i = r_{p[0]}[i] \ldots r_{p[m-1]}[i]$ and then compute $o^i = t^i \circ_P p^i$, then the array $\tilde{o} = \sum_i o_i$ (after appropriate scaling) gives a good approximation of the number of mismatches for each position.

We cannot apply our technique to the above algorithm, as the randomized boolean convolution returns only $O(1)$ bits of information per position, as opposed to $O(\log n)$ bits returned by the deterministic one. However, the convolution can be used to solve the ϵ-threshold matching using a tool which we call *sampling comparator*, which is as follows. Suppose we are given a set $S \subset [n]$ and numbers $1 \leq k \leq n$ and $0 < \epsilon \leq 1$; the goal is to determine if $|S|$ is approximately greater than k. The only allowed operation is checking if $S \cap X$ is nonempty, for any set X specified by the algorithm. We want a (randomized) algorithm A which outputs 0 or 1 with the property that for $P_1 = \Pr[A(S) = 1 | |S| \geq k(1+\epsilon)]$ and $P_2 = \Pr[A(S) = 1 | |S| \leq k]$ we have $P_1 \geq P_2(1+\alpha)$, for some $\alpha = \alpha(\epsilon)$; we call such an algorithm a $(k, \epsilon, P_1, \alpha)$-*comparator*. The bias between P_1 and P_2 can be amplified using standard methods. Specifically, having a (k,ϵ,P_1,α)-comparator C of complexity T we can get a $(k,\epsilon,\frac{2}{3},1)$-comparator C' with complexity $T'=O(\frac{T}{P_1\alpha^2})$. This is done by running the comparator $O(\frac{1}{P_1\alpha^2})$ times and checking if the fraction of times it outputs 1 is greater than $P_2(1+\alpha/2)$; the proof follows from the Chernoff bound.

The above problem appeared in various forms in [19, 4, 29] and others. The common solution proceeds by choosing X of cardinality $\frac{n}{k}$ uniformly at random, returning 1 if $X\cap S\neq\emptyset$ and 0 otherwise. It is known that this yields a $(k,\epsilon,\Theta(1),\Theta(\epsilon))$-comparator. We also note that the bounds essentially do not change if instead of checking the emptiness of the intersection of X and S we compute the size of this intersection mod 2.

For our purpose we need to generalize the above approach to the case when S is not fixed, but rather is chosen from a fixed distribution D. The goal is now to obtain the estimation for the *mean* cardinality of S. In other words, we are interested in an algorithm A such that for $P_2=\Pr[A(S)=1|E_D[|S|]\leq k]$ and $P_1=\Pr[A(S)=1|E_D[|S|]\geq k(1+\epsilon)]$ we have $P_1\geq P_2(1+\alpha)$; we call such an algorithm a (k,ϵ,P_1,α)-*mean comparator.* The only assumption about D is that $E=E_D[|S|]\geq cM$, where $0<c\leq 1$ is a constant and M is the maximum cardinality of S. Note that without this assumption the complexity of a comparator could be unbounded, as we could assign very small but non-zero probability p to a very large value v, in which case the complexity of a comparator could be as high as $1/p$.

Lemma 3 *There is a constant $a,f>0$ such that for any $k\leq\frac{\epsilon fn}{a}$ there exists a $(k,\epsilon,\Theta(\epsilon),\Theta(\epsilon))$-mean comparator with constant complexity.*

Proof: Let $0<d,b<1$ be auxiliary constants specified later and let $f=bcd$. Define also $\delta=\epsilon/a$, where $a>1$ is a constant specified later. The algorithm proceeds by choosing $t=\delta f\frac{n}{k}$ elements from $[n]$ with replacement and forming the set X from these elements (notice that $t\geq 1$ from the assumption). We need to show that the probabilities P_1 and P_2 are separated by a factor of $1+\Theta(\epsilon)$. To this end, we consider two cases, namely when $k\geq fM$ and when $k<fM$. In the first case we show that the probability of $X\cap S\neq\emptyset$ is approximately linear in E, so the differences in the expectation translate into comparable differences in the probability of intersection. In the second case we show that the probability of intersection is high independently of D.

The formal proof is as follows.

Case 1: $k\geq fM$.

Let $p_i=\Pr[|S|=i]$ and $E=\sum_i p_i i$ be the mean cardinality. Let $q_i=\Pr[X\cap S\neq\emptyset||S|=i]$. We know that $q_i=1-(1-\frac{i}{n})^t$. By standard arguments [27] we have

$$1-e^{-\frac{it}{n}}\leq q_i\leq 1-e^{-\frac{it}{n}}(1-\frac{(\frac{i}{n}t)^2}{t})$$

Notice that $\frac{it}{n}=\frac{if\delta n}{kn}\leq\frac{Mf}{k}\delta\leq\delta\leq\frac{1}{2}$. Therefore, we can use an approximation $e^{-x}=1-x+O(x^2)$, which yields

$$\frac{it}{n}-O((\frac{it}{n})^2)\leq q_i=\frac{it}{n}+O((\frac{it}{n})^2)$$

Alternatively, we can write $q_i=\frac{it}{n}(1+\eta)$, where $|\eta|\leq O(\delta)\leq\epsilon/2$ for a proper constant a. Therefore, we can write the probability of $X\cap S\neq\emptyset$ as

$$\sum_i p_iq_i=\sum p_i(\frac{it}{n}(1+\eta))=\frac{t}{n}E(1+\eta)$$

Thus if $E \leq k$ we have $P_2 \leq \frac{tk}{n}(1+\eta)$, while for $E \geq (1+\epsilon)k$ we have $P_1 \geq \frac{tk}{n}(1-\eta)(1+\epsilon)$.

Case 2: $k \leq fM$.

Let $p_1 = \Pr[|S| \leq cdM]$ and $p_2 = \Pr[|S| > cdM]$; moreover define $E_1 = E_D[|S|||S| \leq cdM]$ $E_2 = E_D[|S|||S| > cdM]$. Clearly, $E = p_1E1 + p_2E_2$, which implies

$$p_2 = \frac{E - p_1E_1}{E_2} \geq \frac{cM - p_1Mcd}{M} \geq c(1-d).$$

Conditioned on the event $|S| > cdM$, the probability of $X \cap S \neq \emptyset$ is at least

$$1 - (1 - \frac{cdM}{n})^t \geq 1 - e^{-\frac{-cdMt}{n}} \geq 1 - e^{-cd\epsilon}.$$

Thus $\Pr[X \cap S \neq \emptyset] \geq (1 - e^{-cd\epsilon}) \cdot c(1-d)$, which can be seen to be greater than the probability

$$P_1 \leq \frac{tk}{n}(1+\eta)(1+\epsilon) = \epsilon f(1+\eta)(1+\epsilon) = \epsilon bcd(1+\eta)(1+\epsilon)$$

from Case 1, by taking b sufficiently small. $\square$

We are now ready to present the algorithm. If $k > \frac{\epsilon fn}{a}$, then for each i we sample $l = O(\log n/\epsilon^2)$ positions $j_0 \ldots j_{l-1}$ independently and uniformly at random from $\{i \ldots i+m-1\}$ and compute the number m_i of positions at which a mismatch occurs (i.e. such that $p[j_r] \neq t[i+j_r]$); we output $o[i] = 1$ iff $m_i > k(1+\epsilon/2)$. It is easy to see that this will give the right answer with high probability.

In the case when $k > \frac{\epsilon fn}{a}$, we perform the following steps. Let $d = \log n/\epsilon^3$.

- choose r_i $(i = 0 \ldots k-1)$ independently and uniformly at random from H^d,
- compute binary vectors $p^i = f(p[0])[i] \ldots f(p[m-1])[i]$ and $t^i = g(t[0])[i] \ldots (t[n-1])[i]$, for $i = 0 \ldots d-1$,
- choose u^i $(i = 0 \ldots d-1)$ from H^n by choosing (independently for each i) t positions at random from $[n]$ with replacement and setting them to 1 while setting the others to 0; the value of t is as in the proof of Lemma 3,
- compute the randomized boolean convolution a^i of p^i and $(t^i \wedge u^i)$,
- output o such that $o[j] = 1$ iff $\sum_i a^i[j] \geq t$ where t is chosen as in Lemma 3.

Lemma 4 *For any $\alpha > 0$ there is a constant $c > 0$ such that the occurrence vector generated by the algorithm is correct with probability $1 - \frac{1}{n^\alpha}$.*

Proof: Consider any $i = 0 \ldots d-1$ and any position $j \in [n-m]$. If the number of mismatches between t and p aligned at position j is M, then the expected value $E[t^i \cdot p^i]$ is equal to $\frac{M}{4}$. We can estimate this value using Lemma 3 by using the fact that $(p^i \cdot t^i) \wedge u^i = p^i \cdot (t^i \wedge u^i)$. Thus selecting u^i corresponds to the selection of X in the proof of Lemma 3. $\square$

General ϵ-threshold matching. The algorithm essentially follows by applying the mean-comparator technique to the algorithm from [17]. The details will be given in the full paper.

On-line ϵ-threshold matching. The algorithm uses the sampling comparator idea in the following way. Firstly, we assume that the alphabet set is equal to $\Sigma = \{1 \dots k\}$, i.e. we remove 0. Let $t = c\frac{m}{k}\log n$, for a constant c specified later. We choose a vector $v \in H^m$ with weight t uniformly at random. Then we choose a vector $u \in \{1 \dots cn\}^m$ such that:

- if $v[i] = 0$ then $u[i] = 0$,
- if $v[i] = 1$ then $u[i]$ is assigned a random value from $\{1 \dots cn\}$

For any vector w of length m define $h(w) = w \cdot u$.

Claim 1 *Let $w, w' \in \Sigma^m$ and let M be Hamming distance between them. Then there are constants $c, \delta > 0$ such that:*

- *if $M \le k$ then $\Pr_v[h(w) = h(w')] \ge \delta$*
- *if $M \ge k \log n$ then $\Pr_v[h(w) = h(w')] \le \frac{1}{2n}$*

Proof: Similar to the proof of Lemma 3 □

Define $t_i = t[i \dots i + m - 1]$. During the preprocessing, the algorithm computes all $h(t_i)$ for $i = 0 \dots n - m$ and stores them in a hash table of size $O(n)$. The computation of $h(t_i)$ be done in $O(n \log n)$ time using convolution. In order to process a query pattern p, the algorithm computes $h(p)$ and checks if such a value is already stored in the hash table. The correctness of the algorithm follows from Claim 1.

By allowing $\epsilon < \log n$ we can obtain an algorithm with preprocessing/storage $\tilde{O}(n^{1+1/(1+\epsilon)})$ and query time $\tilde{O}(mn^{1/(1+\epsilon)})$. The details are similar to those in [25] and therefore omitted.

5 Conclusions

In this paper we introduced a new technique for solving several fundamental pattern matching problems in $O(n \log n)$ time. As this bound essentially matches the running time of Fast Fourier Transform which is the only known technique for solving problems of this type, it is likely that the algorithms are in fact optimal. Unfortunately, showing a matching (or even superlinear) lower bound for these problems appears to be very difficult, as it would imply a superlinear lower bound for the FFT in finite fields, which is a major open problem in algebraic complexity.

Acknowledgments. The author would like to thank Amin Shokrollahi and Daniel Reischert for helpful discussion concerning the complexity of polynomial multiplication. He is also indebted to Aris Gionis and Suresh Venkatasubramanian for useful remarks on the preliminary version of this paper.

References

[1] A. Amir, A. Apostolico and M. Lewenstein, Inverse Pattern Matching, *Journal of Algorithms*, 24, pp. 325-339, 1997.

[2] A. Amir and M. Farach, Efficient 2-dimensional approximate matching of half-rectangular figures, *Information and Computation*, 118(1), pp. 1-11, 1995.

[3] A. Amir and G. M. Landau, Fast parallel and serial multidimensional approximate array matching, *Theoretical Computer Science*, 81(1), pp. 97-115, 1991.

[4] N. Alon, Y. Matias, M. Szegedy, The space complexity of approximating the frequency moments, *Proc. STOC'96*, pp. 20-29, 1996.

[5] K. Abrahamson, Generalized String Matching, *SIAM Journal on Computing*, 16(6), pp. 1039-1051, 1987.

[6] A. Aho, J. Hopcroft, J. Ullman, Design and Analysis of Algorithms, Addison-Wesley, 1974.

[7] R. Cole, R. Hariharan, Tree Pattern Matching and Subset Matching in Randomized $O(n \log^3 m)$ Time, *Proc. STOC'97*, 1997.

[8] D. G. Cantor, On Arithmetical Algorithms over Finite Fields, *Journal of Combinatorial Theory*, A 50, pp. 285-300, 1989.

[9] M. Dubiner, Z. Galil, E. Magen, Faster tree pattern matching, *Journal of the Association for Computing Machinery* vol.41, no.2, p. 205-213, 1988.

[10] M.J. Fischer and M.S. Paterson, String matching and other products, *Complexity of Computation*, R.M. Karp (editor), SIAM-AMS Proceedings, 7, pp. 113–125, 1974.

[11] Z. Galil, Open problems in stringology, *Combinatorial problems on words*, NATO ASI Series, Vol. F12, pp. 1-8.

[12] J. Gordon, Very simple method to find the minimal polynomial of an arbitrary non-zero element of a finite field, *Electronic Letters*, 12, pp. 663-664, 1976.

[13] Z. Galil and R. Giancarlo, Parallel string matching with k mismatches, *Theoretical Computer Science*, 51(3), pp. 341-348, 1987.

[14] A. Gibbons and W. Rytter, Efficient Parallel Algorithms, *Cambridge University Press*, 1988.

[15] C. M. Hoffman, M.J. O'Donell, Pattern Matching in Trees, *Journal of the ACM*, pp. 68-95, 1982.

[16] P. Indyk, R. Motwani, S. Venkatasubramanian, Geometric pattern matching, manuscript.

[17] P. Indyk, Deterministic superimposed coding with applications to pattern matching, *Proc. FOCS'97*.

[18] H. Karloff, Fast algorithms for approximately counting mismatches, *Information Processing Letters*, 48(2), pp. 53-60, 1993.

[19] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, Efficient search for approximate nearest neighbor in high dimensional spaces, *Proc. STOC'98*, 1998.

[20] S. R. Kosaraju, Effiicent string matching, manuscript, 1987.

[21] S. R. Kosaraju, Efficient tree pattern matching, *Proc. FOCS'89*, pp. 178-83.

[22] R. M. Karp, M. O. Rabin, Efficient randomized pattern-matching algorithms, *IBM Journal of Research and Development*, vol.31, no.2, pp. 249-260, 1987.

[23] C. K. Koc and T. Acar, Montgomery multiplication in $GF(2^k)$, *Proc. Third Annual Workshop on Selected Areas in Cryptography*, pp. 95-106, 1996.

[24] G. M. Landau, U. Vishkin, Efficient string matching with k mismatches, *Theoretical Computer Science,* vol.43, no.2-3, p. 239-49, 1986.

[25] Piotr Indyk and Rajeev Motwani, Approximate nearest neighbors: Towards removing the curse of dimensionality, *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 1998.

[26] S. Muthukrishnan, K. Palem, Non-standard Stringology: Algorithms and Complexity, in *Proc. STOC'94*, p. 770-779.

[27] S. Muthukrishan, S.M.H. Ramesh, String Matching Under a General Matching Relation, *Information and Computation*, vol. 122, no.1, p. 140-8, 1994.

[28] S. Muthukrishnan, New Results and Open Problems Related to Non-Standard Stringology, *Proc. 6th Annual Symposium on Combinatorial Pattern Matching*, p. 298-317, 1995.

[29] M. Strauss, Counting in small memory, personal communication.

[30] P. Weiner, Linear pattern matching algorithms, *Proc. 14th Annual Symposium on Switching and Automata Theory*, pp. 1-11, 1973.