# TWO AND HIGHER DIMENSIONAL PATTERN MATCHING IN OPTIMAL EXPECTED TIME*

JUHA KÄRKKÄINEN† AND ESKO UKKONEN‡

**Abstract.** Algorithms with optimal expected running time are presented for searching the occurrences of a two-dimensional $m \times m$ pattern $P$ in a two-dimensional $n \times n$ text $T$ over an alphabet of size $c$. The algorithms are based on placing in the text a static grid of test points, determined only by $n$, $m$ and $c$ (not dynamically by earlier test results). Using test strings read from the test points the algorithms eliminate as many potential occurrences of $P$ as possible. The remaining potential occurrences are separately checked for actual occurrences. A suitable choice of the test point set leads to algorithms with expected running time $O(n^2 \log_c m^2/m^2)$ using the uniform Bernoulli model of randomness. This is shown to be optimal by a generalization of a one-dimensional lower bound result by Yao. Experimental results show that the algorithms are efficient in practice, too. The method is also generalized for the $k$ mismatches problem. The resulting algorithm has expected running time $O(kn^2 \log_c m^2/m^2)$, provided that $k \le (m\lfloor m/\lceil \log_c m^2\rceil\rfloor - 1)/2$. All algorithms need preprocessing of $P$ which takes time and space $O(m^2)$. The text processing can be done on-line, using a rather small window. The algorithms easily generalize to $d$-dimensional matching for any $d$.

**Key words.** multidimensional matching, average case analysis, approximate matching

**AMS subject classifications.** 68P05,68P10,68Q20,68Q25,68U10

**1. Introduction.** The classical *pattern matching problem* for strings is to find all (exact or approximate) occurrences of a *pattern* $P$ in a *text* $T$. In one-dimensional case, $P$ and $T$ are strings over some alphabet $\Sigma$ of size $c$. In $d$-dimensional case, $P$ and $T$ are $d$-dimensional arrays over $\Sigma$. The applications are numerous.

The exact one-dimensional pattern matching is well-understood. For two and higher dimensional versions, several algorithms have been proposed, too, mainly aiming at good worst-case running time; see e.g. [7, 6, 16, 26, 11, 2, 13, 22]. Recently, the two-dimensional problem has been solved in linear worst-case time with algorithms that are alphabet-independent and work in small additional space [10].

In the applications of one-dimensional matching, the Boyer-Moore algorithm [8], whose expected running time is "sublinear", is clearly the most successful; see e.g. [15]. Also in higher dimensions, one should look at algorithms with good expected running time, not only the worst case. The first significant step in this direction was the algorithm of Baeza-Yates and Régnier [5] that finds the occurrences of an $m \times m$ pattern in an $n \times n$ text in expected time $O(n^2/m)$.

**Results.** In this paper, we give better algorithms that find exact occurrences of an $m \times m$ pattern in an $n \times n$ text in expected time $O(n^2 \log_c m^2/m^2)$ under the uniform Bernoulli model of random strings over $\Sigma$. Algorithms with the same expected performance have independently appeared in [17, 12, 23].

Our algorithms are not dimensionality specific. For a $d$-dimensional pattern of size $m^d$ and text of size $n^d$, they run in expected time $O(n^d \log_c m^d/m^d)$.

*Work supported by the Academy of Finland. A Preliminary version appeared in the Proc. of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, January 23–25, 1994.

†Juha.Karkkainen@cs.Helsinki.FI, Department of Computer Science, P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki, Finland.

‡ukkonen@cs.Helsinki.FI, Department of Computer Science, P.O. Box 26 (Teollisuuskatu 23), FIN-00014 University of Helsinki, Finland.

In [25], A. Yao confirmed a conjecture of Knuth [18] by proving a lower bound of $O(n \log_c m/m)$ for the expected time complexity of one-dimensional exact pattern matching. Yao's argument generalizes for $d$-dimensional matching and gives a lower bound $O(n^d \log_c m^d/m^d)$. Hence, our algorithms are the first algorithms for multidimensional matching with optimal expected running time. Experiments show that the algorithms are also fast in practice.

We also generalize our algorithm for approximate pattern matching. We consider the two-dimensional $k$ *mismatches problem* (find all occurrences of $P$ in $T$ with $\leq k$ mismatching symbols) and give an algorithm with expected running time $O(kn^2 \log_c m^2/m^2)$. The algorithm requires that $k \leq (m\lfloor m/\lceil \log_c m^2 \rceil \rfloor - 1)/2$. Under this condition the expected running time is at most linear in $|T|$; previously Chang and Lawler [9] (see also [24]) have given an algorithm with similar properties for the $k$ differences problem of one-dimensional approximate matching. The two-dimensional $k$ mismatches problem has also been studied in [21, 19, 4, 3].

**Overview.** We next informally describe the very simple elimination idea behind our algorithms. Note first that the naive algorithm for pattern matching and its improvements such as the Knuth-Morris-Pratt algorithm [18] are based on a view of the problem that could be called *positive*: the algorithm tries to prove at each text location that there is an occurrence of $P$ at that location.

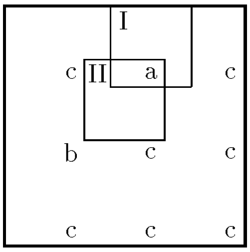
In applications, $T$ often contains relatively few occurrences of $P$. Hence the *negative* view can fit the situation better than the positive one. A negative view algorithm tries to prove for each text location that there can *not* be an occurrence of $P$ at that location. This forms the *elimination phase* of the algorithm. Whenever the elimination proof fails the algorithm has found a potential occurrence of $P$ that needs further consideration. A separate *checking phase* then examines whether or not a potential occurrence really matches $P$. This leads to fast algorithms because the elimination phase needs, on average, to examine only a small fraction of $T$.

The well-known Boyer-Moore algorithm [8] has this flavour. In this paper, we develop algorithms based on the negative view for two and higher dimensional pattern matching.

We explain the idea of our algorithms using two dimensional $P$ and $T$. Let

$$P: \boxed{\begin{array}{ccc} \mathrm{b} & \mathrm{b} & \mathrm{a} \\ \mathrm{b} & \mathrm{b} & \mathrm{b} \\ \mathrm{b} & \mathrm{a} & \mathrm{b} \end{array}} \qquad T: \boxed{\begin{array}{ccccccccc} \mathrm{a} & \mathrm{a} & \mathrm{a} & \mathrm{a} & \mathrm{b} & \mathrm{b} & \mathrm{a} & \mathrm{b} & \mathrm{b} \\ \mathrm{b} & \mathrm{b} & \mathrm{b} & \mathrm{b} & \mathrm{b} & \mathrm{b} & \mathrm{b} & \mathrm{b} & \mathrm{b} \\ \mathrm{a} & \mathrm{b} & \mathrm{c} & \mathrm{c} & \mathrm{b} & \mathrm{a} & \mathrm{b} & \mathrm{a} & \mathrm{c} \\ \mathrm{b} & \mathrm{a} & \mathrm{b} & \mathrm{a} & \mathrm{b} & \mathrm{a} & \mathrm{b} & \mathrm{a} & \mathrm{b} \\ \mathrm{a} & \mathrm{a} & \mathrm{c} & \mathrm{b} & \mathrm{b} & \mathrm{b} & \mathrm{a} & \mathrm{a} & \mathrm{a} \\ \mathrm{b} & \mathrm{c} & \mathrm{b} & \mathrm{b} & \mathrm{b} & \mathrm{c} & \mathrm{b} & \mathrm{b} & \mathrm{c} \\ \mathrm{a} & \mathrm{c} & \mathrm{c} & \mathrm{b} & \mathrm{c} & \mathrm{c} & \mathrm{a} & \mathrm{c} & \mathrm{c} \\ \mathrm{c} & \mathrm{c} & \mathrm{c} & \mathrm{b} & \mathrm{b} & \mathrm{b} & \mathrm{a} & \mathrm{a} & \mathrm{a} \\ \mathrm{c} & \mathrm{c} & \mathrm{c} & \mathrm{c} & \mathrm{a} & \mathrm{c} & \mathrm{b} & \mathrm{b} & \mathrm{c} \end{array}}$$

We have to find all $(I, J)$ such that $T[I+k-1, J+h-1] = P[k, h]$ for $1 \leq k, h \leq 3$. If we are lucky we, can eliminate all possible occurrences of $P$ in $T$ by examining only the entries $T[i, j]$ where $i = 3, 6, 9$, $j = 3, 6, 9$, because any possible occurrence of $P$ in $T$ must overlap at least one of these entries of $T$. The test entries in $T$ are as follows




As the first entry $T[3,3]$ equals $c$, and $c$ does not occur in $P$ at all, no occurrence of $P$ in $T$ can overlap $T[3,3]$. This proves that there can not be an occurrence of $P$ inside subarray $T[1:5, 1:5]$. Test entry $T[3,6] = a$ eliminates all other occurrences of $P$ that overlap $T[3,6]$ except those that have overlapping symbol $a$. The upper left-hand corners of these potential occurrences of $P$ are $T[1,5]$ (marked I in the figure) and $T[3,4]$ (II), because $P[1,3] = P[3,2] = a$. The checking phase then finds that there is an occurrence of $P$ at $T[1,5]$ but not at $T[3,4]$.

The fourth test entry $T[6,3] = b$ is least useful for elimination because $b$ occurs 7 times in $P$. Hence, we are left with 7 potential occurrences of $P$ that overlap $T[6,3]$. However, there is no reason to use only one entry of T in each elimination step. Taking $T[6,2] = c$ as an additional test entry would eliminate all possible occurrences of $P$ that overlap $T[6,2]$ and $T[6,3]$ because substring $cb$ does not occur in $P$.

Increasing the number of test entries used in each elimination step obviously leads to more extensive elimination but possibly takes more time. Therefore, we determine $q$ such that, when each elimination step is based on $q$ entries of $T$, the expected total running time of elimination and (naive) checking is minimized. It turns out that the minimum is given by $q \approx \log_{|\Sigma|} |P|$; this simply has the effect of using a superalphabet of size $\Theta(|P|)$.

The test entries will be distributed over $T$ such that every potential occurrence of $P$ in $T$ covers at least $q$ entries. We use a *static distribution* of test entries; this is a major difference to virtually all other pattern matching algorithms proposed in the literature. The grid of test entries is determined using only the size (and shape) of $P$, $T$ and the underlying alphabet. The elimination phase becomes *oblivious*: the results of earlier elimination tests do not cause dynamic changes on later tests. This property helps in writing fast code for the algorithms, for both sequential and parallel environments.

The idea of using superalphabet of size $\Theta(|P|)$ in two-dimensional matching appears independently in [12, 17, 23]. All achieve the same asymptotic running time as our algorithms. Crochemore and Rytter [12, Sect. 12.7] sketch an algorithm which comes close to one of our algorithms, while the algorithms in [17, 23] differ in that they determine the test entries dynamically.

The above idea is formulated in §2 as a generic pattern matching algorithm. The elimination of potential occurrences of $P$ is based on reading test strings of length $q$ from $T$. The strings are picked up according to a fixed sampling scheme. In §3, we give an example scheme such that each test entry belongs to only one test string. To get minimum total number of test entries, we use test strings whose shape has smallest possible diameter. In §4, we give a bit more efficient scheme in which each test entry belongs to several test strings. Section 5 reformulates the lower bound of Yao [25] for $d$-dimensional pattern matching. Section 6 describes the results of experiments comparing the running times of the algorithms of §3 and §4, the trivial algorithm, and an algorithm by Tarhio [23]. In §7, we generalize the algorithm for the $k$ mismatches problem. The elimination is based on $k+p$ non-overlapping test strings; here $p$ is a parameter used for optimizing the performance of the method.

**2. General Algorithm.** To keep the technical exposition simple we restrict ourselves to the two-dimensional case with square patterns and texts. The generalization of all our results to the $d$-dimensional pattern matching with rectangular patterns and texts is straightforward.

Let a two-dimensional $m \times m$ array $P = (p_{ij})$, $1 \le i, j \le m$, be the *pattern* and a two-dimensional $n \times n$ array $T = (t_{ij})$, $1 \le i, j \le n$, be the *text* over some alphabet $\Sigma$. Let $c$ denote the size of the alphabet $\Sigma$.

We let integer $q$ denote the *sample size*. Selecting a suitable $q$ will minimize the expected running time of our algorithms. A sequence $Q = ((0,0), (i_1, j_1), (i_2, j_2), \ldots, (i_{q-1}, j_{q-1}))$ of $q$ *disjoint* pairs of indexes is called a (two-dimensional) *template* of size $q$. The sequence always begins with the pair $(0,0)$ which is called the *origin* of the template. A template $Q$ and a pair of indexes $(i,j)$ define a *sample* $Q(i,j) = ((i,j), (i+i_1, j+j_1), \ldots, (i+i_{q-1}, j+j_{q-1}))$. Here $Q$ gives the shape of the sample and $(i,j)$ the location of the sample.

Pattern $P$ is said to *contain* sample $Q(i,j)$ if all the pairs listed in $Q(i,j)$ refer to entries of $P$. Such sample $Q(i,j)$ determines a *test string* $s(P, Q(i,j))$ in $P$ as $s(P, Q(i,j)) = p_{i,j} p_{i+i_1, j+j_1} \cdots p_{i+i_{q-1}, j+j_{q-1}}$.

A *sampling scheme* $\mathcal{Q}$ for a template $Q$, a pattern $P$, and a text $T$ is a pair $(\mathcal{Q}_P, \mathcal{Q}_T)$ where *pattern sampling scheme* $\mathcal{Q}_P$ and *text sampling scheme* $\mathcal{Q}_T$ are subsets of all possible $Q$-shaped samples of $P$ and $T$, respectively, i.e.,

$$\begin{aligned} \mathcal{Q}_P &\subseteq \{Q(i,j) \mid P \text{ contains } Q(i,j)\} \text{ and} \\ \mathcal{Q}_T &\subseteq \{Q(i,j) \mid T \text{ contains } Q(i,j)\}. \end{aligned}$$

Let $R$ be a potential occurrence of $P$ in $T$ (i.e., an $m \times m$ subarray of $T$) with upper left-hand corner at $(i,j)$. A *witness* of $R$ is a pair $(Q(i_P, j_P), Q(i_T, j_T)) \in \mathcal{Q}_P \times \mathcal{Q}_T$ such that $Q(i_T, j_T)$ is in $R$ in the same position as $Q(i_P, j_P)$ is in $P$, i.e., $i_T - i + 1 = i_P$ and $j_T - j + 1 = j_P$. Witness $(Q(i_P, j_P), Q(i_T, j_T))$ is positive if $s(P, Q(i_P, j_P)) = s(T, Q(i_T, j_T))$; otherwise the witness is negative. A potential occurrence with a negative witness can not be a real occurrence of $P$ in $T$.

*Example* 1. Let

$$Q: \begin{array}{|cc|} \hline 0 & 2 \\ \hline & 1 \\ \hline \end{array} \qquad P: \begin{array}{|ccc|} \hline b & b & c \\ b & c & a \\ a & a & a \\ \hline \end{array} \qquad T: \begin{array}{|ccccc|} \hline c & b & a & a & b \\ b & b & c & b & c \\ b & c & a & c & a \\ a & a & b & b & a \\ c & a & b & b & b \\ \hline \end{array}$$

The numbers in the template $Q = ((0,0), (1,1), (0,1))$ show the order of the entries. Let $R$ be a potential occurrence of $P$ in $T$ with upper left-hand corner at $(2,1)$ (marked with dashed boundaries above). If now $Q(2,2) \in \mathcal{Q}_P$ and $Q(3,2) \in \mathcal{Q}_T$, then (Q(2,2),Q(3,2)) is a negative witness of $R$ as $s(P, Q(2,2)) = caa \ne cba = s(T, Q(3,2))$. This proves that $R$ is not an occurrence of $P$. All other possible witnesses of $R$ would be positive.

We are now ready to give a general description of our algorithms. The idea is to eliminate potential occurrences with negative witnesses and check the remaining potential occurrences using, e.g., naive checking. We select a sampling scheme $\mathcal{Q}$ such that every potential occurrence has exactly one witness in $\mathcal{Q}$. If that one witness is positive, then the potential occurrence has no negative witnesses and must be checked. In other words, the potential occurrences worth checking are determined by finding all positive witnesses. Uninteresting potential occurrences are implicitly eliminated by ignoring negative witnesses.

ALGORITHM [Generic]

1. [Selection of $q$, $Q$ and $\mathcal{Q}$] Select suitable sample size $q$, template $Q$ and sampling scheme $\mathcal{Q} = (\mathcal{Q}_P, \mathcal{Q}_T)$ for given pattern $P$ and text $T$ over alphabet $\Sigma$ of size $c$. Scheme $\mathcal{Q}$ must provide exactly one witness for each potential occurrence of $P$ in $T$.
2. [Preprocessing of $P$] For each pattern sample $Q(i,j) \in \mathcal{Q}_P$, evaluate the test string $s(P, Q(i,j))$. Construct a suitable data structure to quickly find all pattern samples with a given test string.
3. [Elimination] For each text sample $Q(i_T, j_T) \in \mathcal{Q}_T$, scan $T$ to obtain the string $s(T, Q(i_T, j_T))$ and find, using the data structure constructed in the preprocessing step, all pattern samples $Q(i_P, j_P)$ such that $s(P, Q(i_P, j_P)) = s(T, Q(i_T, j_T))$. For each matching pair $(Q(i_P, j_P), Q(i_T, j_T))$, execute the checking phase for the potential occurrence $R$ with upper left-hand corner at $(i_R, j_R) = (i_T - i_P + 1, j_T - j_P + 1)$.
4. [Checking] Given a potential occurrence $R$ with upper left-hand corner at $(i_R, j_R)$, check whether or not there really is an occurrence, that is, test whether or not $t_{i_R+I-1, j_R+J-1} = p_{I,J}$ for $1 \le I, J \le m$.

Selecting different sampling schemes in the generic algorithm gives different concrete algorithms. The simplest alternative is developed in §3 where we use templates with small diameter to maximize the number of potential occurrences each text sample is contained in. Utilizing overlaps of text samples leads to an algorithm with faster pattern preprocessing but more sophisticated (not necessarily slower) elimination phase. Such methods are described and analyzed in §4.

The sampling schemes of both of these algorithms are designed to minimize the number of text positions inspected during elimination as the text is typically much larger than the pattern. For the same reason our algorithms preprocess the pattern samples. In an application, where multiple patterns are to be searched in the same static text, it might be useful to reverse the handling of text and patterns. However, we will not follow this direction further in this paper.

**3. Square templates.** The requirement of one witness per potential occurrence means that the number of different witnesses in $Q$, $|\mathcal{Q}_P||\mathcal{Q}_T|$, approximately equals the number of potential occurrences of $P$ in $T$.* This implies that one can minimize the number of text samples by maximizing the number of pattern samples. This is the main idea of the algorithm presented in this section. The idea is realized by using what we call *square templates*.

DEFINITION 3.1. *Template* $Q = ((0,0), (i_1, j_1), \ldots, (i_{q-1}, j_{q-1}))$ *of size* $q$ *is a square template iff* $0 \le i_k, j_k \le \lceil \sqrt{q} \rceil - 1$ *for* $1 \le k \le q-1$.

A square sample (sample whose shape is defined by a square template) of size $q$ is contained in a $\lceil \sqrt{q} \rceil \times \lceil \sqrt{q} \rceil$ subarray whose upper left-hand corner is the origin of the sample. It is easy to see that the square shape maximizes the number of possible pattern samples.† Optimal sample size $q$ will be determined in the analysis of the algorithm.

*The non-exactness is caused by witnesses near the text boundaries.

†Actually, for some $q$, we could fit the template in a $\lceil \sqrt{q} \rceil \times \lfloor \sqrt{q} \rfloor$ subarray.

Given a square template $Q$, the pattern sampling scheme $\mathcal{Q}_P$ now contains all possible $Q$-shaped pattern samples, that is

$$\mathcal{Q}_P = \{Q(i,j) \mid 1 \leq i,j \leq m - \lceil\sqrt{q}\rceil + 1\}.$$

To get one witness per potential occurrence the text samples must be placed at the intersections of every $(m - \lceil\sqrt{q}\rceil + 1)$th row and column. This gives

$$\mathcal{Q}_T = \left\{ Q\left(i(m - \lceil\sqrt{q}\rceil + 1), j(m - \lceil\sqrt{q}\rceil + 1)\right) \;\middle|\; 1 \leq i,j \leq \left\lfloor \frac{n - \lceil\sqrt{q}\rceil + 1}{m - \lceil\sqrt{q}\rceil + 1} \right\rfloor \right\}.$$

Now each text sample is a component of the witness of $|\mathcal{Q}_P| = (m - \lceil\sqrt{q}\rceil + 1)^2$ different potential occurrences. For each text sample, we need to find all matching pattern samples. To do this as quickly as possible, we preprocess the pattern to get a data structure which we call *test dictionary*.

Let $w$ be a string of length $q$ over alphabet $\Sigma$. Let $A(w)$ be the set of origins of pattern samples whose test string is $w$. That is,

$$A(w) = \{(i,j) \mid w = s(P, Q(i,j)),\ Q(i,j) \in \mathcal{Q}_P\}.$$

$A(w)$ is called the set of *addresses* of $w$. The test dictionary $D(P, \mathcal{Q}_P)$ for pattern $P$ and sampling scheme $\mathcal{Q}_P$ of $P$ is a data structure supporting queries that for a given string $w \in \Sigma^q$ yield the set of addresses of $w$.

There are two obvious implementations of test dictionary $D(P, \mathcal{Q}_P)$. First, one can use a trie representing the test strings, with the addresses $A(w)$ associated with the leaf that corresponds to string $w$. The test strings and their addresses can be constructed by scanning $P$ for each pattern sample. There are $\leq m^2$ samples, hence the total length of the test strings is $\leq m^2 q$, and the trie representing them can be built in time $O(m^2 qc)$ or $O(m^2 q \log c)$, depending on whether direct indexing or balanced search trees are used for implementing the branches of the trie. The queries for $A(w)$ can be answered in time $O(q)$ or $O(q \log c)$, respectively (plus the size of $A(w)$).

The second possibility to implement $D(P, \mathcal{Q}_P)$ is to convert the test strings into ($c$-ary) integers, and use these integers as indices to an array of size $c^q$ representing the addresses $A(w)$ for each $w$. The optimal value of $q$ turns out to be small enough to keep $c^q$ acceptable. The array initialization takes $O(c^q)$ time and the trivial method of pattern processing requires $O(m^2 q)$. The queries can be answered in time $O(q)$.

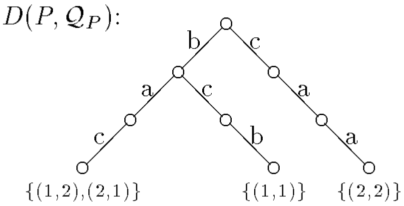
With a suitable ordering of the template entries and a little bookkeeping the pattern processing time can be reduced to $O(m^2)$. The template is ordered like text: row by row from top to bottom with each row ordered left to right. Individual rows of the samples can then be thought as $c$-ary integers of $\lceil\sqrt{q}\rceil$ digits and the whole sample as $c^{\lceil\sqrt{q}\rceil}$-ary integer of $\lceil\sqrt{q}\rceil$ digits (rows). Each sample row is a substring of a pattern row. The sample row values coming from a pattern row can be calculated with a single scan of the row in time $O(m)$ in the obvious way. Similarly, given all the sample row values the sample values can then be calculated by scanning the row values column by column. The operation is further complicated by the fact that, to have the sample size be exactly $q$, one of the sample rows may have to be shorter than $\lceil\sqrt{q}\rceil$. Despite the complexity of this method, it turned out to be faster, in practice, than the simpler $O(m^2 q)$ method.

*Example* 2. Let $Q$, $P$, $T$ and $R$ be as in Example 1. Then $Q$ is a square template and, using the sampling scheme of this section, we have

$$\begin{aligned}\mathcal{Q}_P &= \{Q(1,1), Q(1,2), Q(2,1), Q(2,2)\}\\ \mathcal{Q}_T &= \{Q(2,2), Q(2,4), Q(4,2), Q(4,4)\}.\end{aligned}$$

The witness of $R$ in $\mathcal{Q} = (\mathcal{Q}_P, \mathcal{Q}_T)$ is $(Q(1,2), Q(2,2))$. The addresses of strings $w \in \Sigma^3$ are now $A(bac) = \{(1,2),(2,1)\}$, $A(bcb) = \{(1,1)\}$, $A(caa) = \{(2,2)\}$, and $A(w) = \emptyset$ for all other $w$. The trie implementation of $D(P, \mathcal{Q}_P)$ would be

$D(P, \mathcal{Q}_P)$:




**Analysis.** We determine $q$ such that the expected processing time of $T$ (i.e., elimination + checking) is minimized under the assumption that $T$ is a random text in the uniform Bernoulli model (each symbol of $\Sigma$ can occur in a given position of $T$ with probability $1/c$ independently of the content of other positions). We will next analyze the expected time for the elimination and checking of a fixed potential occurrence $R$ of $P$ in $T$.

Let $Q(i_T, j_T) \in \mathcal{Q}_T$ be the text sample contained in $R$. The elimination phase for $R$ consists of evaluating string $w = s(T, Q(i_T, j_T))$ and finding $A(w)$ from $D(P, \mathcal{Q}_P)$. This takes time $O(q)$, i.e., is proportional to the number $q$ of entries of $T$ that have to be examined to get $w$. As $Q(i_T, j_T)$ is contained in $(m - \lceil\sqrt{q}\rceil + 1)^2$ potential occurrences of $P$, the elimination time amortized for $R$ is proportional to $q/(m - \lceil\sqrt{q}\rceil + 1)^2$.

The analysis of the checking phase is based on the naive checking that compares the corresponding entries of $P$ and $R$ until the first mismatch or an entire occurrence of $P$ is found. The expected number of comparisons is $< \alpha = c/(c-1)$. However, the checking is activated only if the witness of $R$ is positive. The probability of this event is $1/c^q$, and the expected number of comparisons in the checking phase is therefore $< \alpha/c^q$.

Now we choose $q$ to minimize the total expected number of comparisons allocated for $R$,

$$\frac{q}{(m - \lceil\sqrt{q}\rceil + 1)^2} + \frac{\alpha}{c^q}. \tag{3.1}$$

Application of elementary calculus on (3.1) without the ceiling function gives the equation

$$q = \log_c m^2 - \log_c \frac{m^2(m+1)}{(m - \sqrt{q} + 1)^3} + \log_c \frac{c \ln c}{c-1}$$

for the optimal (real) value of $q$. This further implies that

$$\log_c m^2 - 1 < q < \log_c m^2 + \frac{1}{2},$$

when $m \geq 2$ and $c \geq 2$. Based on this we choose $q = \lceil \log_c m^2 \rceil$ (when $m = 1$, define $q = 1$).

Substituting this into (3.1) and noting that there are $\leq n^2$ potential occurrences, gives the following bound for the expected number of symbols examined during the elimination and checking:

$$
\begin{aligned}
& n^2 \left( \frac{\lceil \log_c m^2 \rceil}{\left( m - \left\lceil \sqrt{\lceil \log_c m^2 \rceil} \right\rceil + 1 \right)^2} + \frac{\alpha}{m^2} \right) \leq n^2 \frac{4\lceil \log_c m^2 \rceil + \alpha}{m^2} \\
= \; & O\left( \frac{n^2 \log_c m^2}{m^2} \right) = O\left( \frac{|T| \log_c |P|}{|P|} \right),
\end{aligned}
$$

where we have used the fact that $m - \left\lceil \sqrt{\lceil \log_c m^2 \rceil} \right\rceil + 1 \geq m/2$ if $c \geq 2$ and $m \geq 2$.

We have shown the following result.

THEOREM 3.2. *The occurrences of an $m \times m$ pattern $P$ in an $n \times n$ text $T$ can be found using square templates of size $\lceil \log_c m^2 \rceil$ in expected time $O(n^2 \log_c m^2/m^2)$. The preprocessing of $P$ takes time and space $O(m^2)$.*

When the pattern is large ($|P| = \omega(n\sqrt{\log_c n})$), the pattern preprocessing dominates the matching time. If desirable, the preprocessing time can be reduced by using only a part of the pattern in the preprocessing and elimination phases. Full pattern is used only in the checking phase. When the area of the pattern used in the preprocessing is $\Theta(n\sqrt{\log_c n})$, the expected total time becomes $O(n\sqrt{\log_c n})$.

It should be clear that the algorithm of this section generalizes to $d$-dimensional pattern matching for rectangular patterns $P$ and texts $T$ such that the expected running time is $O\left(|T| \log_c |P|/|P|\right)$ for processing $T$. Preprocessing $P$ needs time $O(|P|)$.

**4. Linear templates.** In this section, we relax the requirement of minimum number of text samples. Instead, we minimize the number of text entries belonging to samples by letting the samples share entries. This together with an efficient method for reading the text samples and comparing them to pattern samples leads to an algorithm with performance similar to the algorithm of previous section.

To make the reading of the text samples simple in the case of overlapping templates, we use *linear templates.* A linear template $Q$ of size $q$ defines the shape of samples to be an evenly spaced subsequence of $q$ entries on a row of a two-dimensional array. Hence $Q$ is of the form $Q = ((0,0), (0,h), (0,2h), \ldots, (0,(q-1)h))$ for some $h \geq 1$. Note that samples $Q(i,j)$ and $Q(i,j+h)$ share $q-1$ entries.

Text sampling scheme $\mathcal{Q}_T$ will consist of linear samples placed on every $m$th row, called a *test row*, such that they start from every $h$th entry. Hence the samples overlap each other as $Q(i,j)$ and $Q(i,j+h)$ above. This gives

$$
\mathcal{Q}_T = \left\{ Q(im, jh) \;\middle|\; 1 \leq i \leq \left\lfloor \frac{n}{m} \right\rfloor, 1 \leq j \leq \left\lfloor \frac{n-m}{h} \right\rfloor + 1 \right\}.
$$

The number of text entries belonging to samples in $\mathcal{Q}_T$ is now $\lfloor n/m \rfloor (\lfloor (n-m)/h \rfloor + q)$. To minimize this, we should maximize $h$. Upper bound for $h$ is given by the fact that every potential occurrence must contain at least one text sample. This gives $h = \lfloor m/q \rfloor$. The pattern sampling scheme is

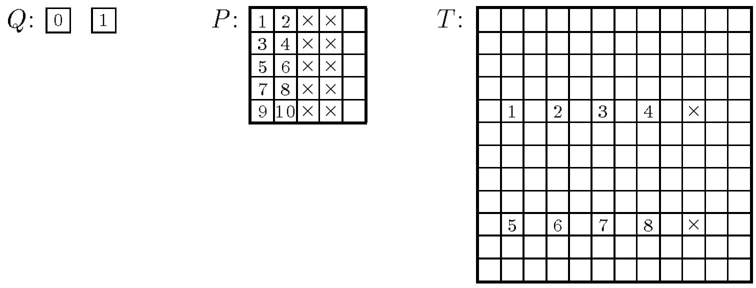
$$
\mathcal{Q}_P = \{ Q(i,j) \mid 1 \leq i \leq m, \; 1 \leq j \leq h \}.
$$

Note that there are no overlaps between pattern samples.

*Example* 3. Let $P$ be a $5 \times 5$ pattern, $T$ a $12 \times 12$ text and $Q = ((0,0),(0,2))$ a linear template of size 2. Using the above method, we get

$$\begin{aligned}\mathcal{Q}_P &= \{Q(1,1), Q(1,2), Q(2,1), Q(2,2), Q(3,1), Q(3,2),\\ &\qquad Q(4,1), Q(4,2), Q(5,1), Q(5,2)\}\\ \mathcal{Q}_T &= \{Q(5,2), Q(5,4), Q(5,6), Q(5,8), Q(10,2), Q(10,4), Q(10,6), Q(10,8)\}.\end{aligned}$$

This is illustrated below:




The numbered entries in the pattern and the text are the origins of the samples. The entries marked with '$\times$' are the other sample entries.

Utilizing the overlaps of text samples makes it possible to read each test string $s(T, Q(i,j))$, $Q(i,j) \in \mathcal{Q}_T$, in constant amortized time. For each test row, we read the sample entries from left to right and maintain a window of size $q$ to hold the current string.

We also need to find the matching pattern samples in constant time. To achieve this we replace the trie of §3 with an Aho-Corasick multipattern matching automaton $AC(P, \mathcal{Q}_P)$ (see [1]). The automaton can be built in the same asymptotic time as the trie; in fact $AC(P, \mathcal{Q}_P)$ is simply the trie augmented with the *failure*-transitions. The addresses of the test strings are attached to the states of $AC(P, \mathcal{Q}_P)$ that correspond to the test strings. As there are $m\lfloor m/q \rfloor$ samples of size $q$ in $\mathcal{Q}_P$, the preprocessing time becomes $O(m^2)$; with $c$ shown explicitly the bound becomes $O(m^2 c)$ or $O(m^2 \log c)$. This is smaller, by a factor of $q$, than the preprocessing time of the previous section as there are now fewer pattern samples.

The method of treating strings as integers can be modified to work with the linear templates, too. During the scanning of $T$, we treat the characters as digits and maintain a window of $q$ digits. The pattern samples can be read one at a time in $O(m^2)$ time. Otherwise, the method works as with square templates.

**Analysis.** As already mentioned, the number of text entries read during elimination is

$$\left\lfloor \frac{n}{m} \right\rfloor \left( \left\lfloor \frac{n-m}{h} \right\rfloor + q \right) = \left\lfloor \frac{n}{m} \right\rfloor \left\lfloor \frac{n-m+qh}{h} \right\rfloor \leq \left\lfloor \frac{n}{m} \right\rfloor \left\lfloor \frac{n}{h} \right\rfloor \leq \frac{n^2}{mh} = \frac{n^2}{m\lfloor m/q \rfloor},$$

which is also the running time of elimination phase as each entry can be processed in constant time. The expected running time of the checking phase is, as before, at most $n^2\alpha/c^q$. Thus, the total expected text processing time is now bounded by

$$n^2 \left( \frac{1}{m \left\lfloor \frac{m}{q} \right\rfloor} + \frac{\alpha}{c^q} \right).$$

Omitting the floor function, we can again apply calculus to find that the optimal (real) value of $q$ is

$$q = \log_c m^2 + \log_c \frac{c \ln c}{c-1},$$

where $0 < \log_c \frac{c \ln c}{c-1} < \frac{1}{2}$ for $c \geq 2$. Thus we again choose $q = \lceil \log_c m^2 \rceil$ * which leads to the bound

$$n^2 \left( \frac{2 \log_c m^2 + \alpha}{m^2} \right) = O\left( \frac{n^2 \log_c m^2}{m^2} \right) = O\left( \frac{|T| \log_c |P|}{|P|} \right)$$

for the total expected number of examined symbols during elimination and checking.

THEOREM 4.1. *The occurrences of an $m \times m$ pattern $P$ in an $n \times n$ text $T$ can be found using overlapping linear templates of size $\lceil \log_c m^2 \rceil$ in expected time $O(n^2 \log_c m^2/m^2)$. The preprocessing of $P$ takes time and space $O(m^2)$.*

Again, the pattern preprocessing time dominates when $|P| = \omega(n\sqrt{\log_c n})$. As with the square template algorithm, the total expected time in such a case can be limited to $O(n\sqrt{\log_c n})$ by using only an area of size $\Theta(n\sqrt{\log_c n})$ of the pattern in the preprocessing.

The algorithm again easily generalizes to $d$-dimensional $P$ and $T$. Note that, for any $d$, the generalized algorithm uses linear (one-dimensional) samples scanned with the Aho-Corasick automaton. In this way, the elimination phase of $d$-dimensional matching reduces directly to one-dimensional techniques.

For larger $d$, it may happen that $q = \lceil \log_c m^d \rceil > m$ in which case the straightforward generalization does not work any more.† An easy correction is to use superrows, consisting of two or more adjacent one-dimensional rows. As $2^{d-1}m \geq \log_c m^d$ for $d \geq 1$, $c \geq 2$, $m \geq 2$, a superrow with diameter 2 is always sufficient. Therefore this does not change the asymptotic behaviour of the algorithm.

The algorithm of this section is quite similar to the (suboptimal) algorithm by Baeza-Yates & Régnier [5], the main difference being that we use test samples with fixed size and location.

**5. Lower bound.** In [25], Yao proved a lower bound of $\Omega(n \log_c m/m)$ for the expected running time of one-dimensional pattern matching. In this section, we generalize Yao's result for $d$-dimensional hypercubic strings.

Let $P$ be a pattern of size $m^d$ and $T$ a text of size $n^d$ over alphabet $\Sigma$ of size $c$. There exists some minimum number of entries of $T$ that must be examined before any algorithm can be sure that all occurrences of $P$ in $T$ have been found. We define $g(P, n)$ to be the minimum of such numbers for pattern $P$ over all texts of size $n^d$. Thus, $g(P, n)$ gives the best case lower bound of pattern matching for pattern $P$.

There exists patterns that are easy to match in the best case (e.g. patterns that do not contain all characters of the alphabet). However, the following theorem shows that such patterns are rare.

---

*Taking the floor function into account, $q' = \lfloor m/\lfloor m/q \rfloor \rfloor$ is always at least as good a choice as $q$, because $q' \geq q$ and $\lfloor m/q' \rfloor = \lfloor m/q \rfloor$.

†For $d = 2$ there is one such case, namely $m = 3$, $c = 2$ with $\log_2 3^2 \approx 3.17$. In this special case, we can simply set $q = 3$.

DEFINITION 5.1. *For $n \geq m > 0$, let*

$$
\begin{aligned}
f_1(m,n) &= \left\lceil \frac{1}{d} \log_c \left( \frac{(n-m)^d}{\ln(m^d+1)} + 2 \right) \right\rceil \quad \text{and} \\
f_2(m,n) &= \left\lfloor \frac{n}{2m} \right\rfloor^d \left\lceil \frac{1}{d} \log_c (m^d+1) \right\rceil .
\end{aligned}
$$

*Define*

$$
f(m,n) = \begin{cases} f_1(m,n) & \text{if } m \leq n \leq 2m, \\ f_2(m,n) & \text{if } n > 2m. \end{cases}
$$

THEOREM 5.2. *There exists a positive number $a$ such that, for any $|\Sigma| = c \geq 2$ and $m > 0$, there exists a set of patterns $L \subseteq \Sigma^{m^d}$ satisfying*

$$
|L| \geq \left(1 - \frac{1}{m^{9d}}\right) c^{m^d},
$$

*and*

$$
\text{for each } P \in L,\ g(P,n) \geq af(m,n) \text{ for all } n \geq m.
$$

*Proof.* The proof is a simple modification of Yao's proof in [25]. As the proof is quite long we will give here only a brief guide for the modification. Almost all that is required is to remember that we are now dealing with $d$-dimensional strings. For example, in Yao's Lemma 5.2 the text has to be divided into hypercubes of size $(2m)^d$ instead of strings of length $2m$. Similarly, we need to replace the occurrences of $m$, $n$, $n-m$ (denoted by $d$ in Yao's paper), $2m$ and $\lfloor n/(2m) \rfloor$ throughout the proof with $m^d$, $n^d$, $(n-m)^d$, $(2m)^d$ and $\lfloor n/(2m) \rfloor^d$, respectively. Additionally, the terms of the form $xq^4 \ldots$ and $xq^{20} \ldots$ must have an extra factor of $2^d$. This is balanced by the factor $1/d$ in the definitions of $f_1$ and $f_2$. □

The above theorem shows that for majority of patterns (i.e., for the patterns in $L$) even the best case complexity of pattern matching is

$$
\Omega\left(\frac{n^d \log_c m^d}{m^d}\right),
$$

for a fixed $d$. This implies that for a fixed $d$ and $T$ and random $P$, the expected running time of pattern matching is $\Omega\left(n^d \log_c m^d / m^d\right)$. Thus our algorithms for fixed-dimensional strings are optimal.

It is possible to have a single version of our algorithms to work for all dimensions $d$. In such a case, the lower bound given by Theorem 5.2, with dependency on $d$ shown explicitly, is

$$
\Omega\left(\frac{1}{d2^d} \frac{n^d \log_c m^d}{m^d}\right),
$$

leaving the upper and lower bounds separated by factor $d2^d$. Our conjecture is that the lower bound can be improved to $\Omega\left(n^d \log_c m^d / m^d\right)$ in this case, too, but that remains an open question.

TABLE 6.1

*Running times of the algorithms in milliseconds for* $n = 1000$.

| | | preprocessing | | | text scanning | | | total | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $c$ | $m$ | ST | LT | TDBM | ST | LT | TDBM | ST | LT | TDBM | TRIV |
| 2 | 2 | 0.16 | **0.10** | 0.13 | 1593 | **1254** | 1539 | 1593 | 1254 | 1539 | **1156** |
| 2 | 4 | 0.27 | **0.14** | 0.15 | 531 | **446** | 565 | 531 | **446** | 565 | 1101 |
| 2 | 8 | 0.64 | 0.42 | **0.24** | 147 | **140** | 188 | 148 | **140** | 189 | 1064 |
| 2 | 16 | 2.30 | **0.58** | 0.67 | 46.1 | **41.7** | 49.6 | 48.4 | **42.3** | 50.3 | 1044 |
| 2 | 32 | 10.1 | **1.64** | 2.24 | 12.0 | 13.0 | **11.8** | 22.1 | 14.6 | **14.1** | 1014 |
| 2 | 64 | 44.9 | **6.34** | 8.69 | 3.01 | 4.59 | **2.90** | 47.9 | **10.9** | 11.6 | 945 |
| 2 | 128 | 202 | **26.7** | 42.0 | 0.77 | 1.19 | **0.74** | 203 | **27.9** | 42.7 | 820 |
| 2 | 256 | 1081 | **107** | 180 | 0.22 | 0.27 | **0.20** | 1082 | **107** | 180 | 594 |
| 256 | 2 | 0.61 | **0.38** | 0.50 | 300 | **270** | 374 | 300 | **270** | 374 | 524 |
| 256 | 4 | 0.78 | **0.46** | 0.53 | 84.6 | **77.7** | 110 | 85.4 | **78.1** | 111 | 522 |
| 256 | 8 | 1.23 | 0.57 | **0.56** | 28.9 | **27.0** | 32.8 | 30.1 | **27.6** | 33.4 | 519 |
| 256 | 16 | 3.19 | 0.94 | **0.80** | 13.1 | 12.6 | **11.2** | 16.3 | 13.6 | **12.0** | 511 |
| 256 | 32 | 96.9 | **54.0** | 83.3 | 2.21 | 4.34 | **2.35** | 99.1 | **58.3** | 85.7 | 496 |
| 256 | 64 | 144 | **59.4** | 88.5 | 0.53 | 0.72 | **0.50** | 144 | **60.1** | 89.0 | 462 |
| 256 | 128 | 331 | **80.9** | 107 | 0.17 | 0.17 | **0.12** | 331 | **81.1** | 107 | 402 |
| 256 | 256 | 1367 | **166** | 182 | 0.10 | 0.06 | **0.04** | 1367 | **166** | 182 | 293 |

**6. Experimental results.** We have conducted some experiments to find out how these algorithms work in practice and how they compare to each other and some other algorithms. The other algorithms in the experiments are the trivial algorithm (check naively each potential occurrence of $P$ in $T$) and an asymptotically optimal algorithm by Tarhio [23]. Tarhio's algorithm represents the dynamic sampling approach as opposed to the static sampling schemes of our algorithms.

Both the square template algorithm (ST) and the linear template algorithm (LT) use the method of converting strings to integers. Also Tarhio's algorithm (TDBM = Two Dimensional Boyer-Moore) uses a similar method, so in this sense these algorithms are very well comparable. The pattern preprocessing of ST is done using the more complicated (but faster) $O(m^2)$ method. All of these algorithms try to choose the parameters in the pattern preprocessing phase so that the running time of the text scanning phase is minimized. The algorithms were written and carefully optimized using C and were run on a Sun 4 workstation.

Table 6.1 shows the results of experiments using independently generated, random texts and patterns. Text size is $1000 \times 1000$ in all of the tests, and pattern and alphabet sizes vary. The times (given in milliseconds) are averages of seven tests with different patterns.

The table gives the pattern preprocessing and text scanning times separately; the total time is the sum of these. The total time for the trivial algorithm (TRIV) is also its scanning time as it doesn't have preprocessing. The times do not include the reading of the text and the pattern from disk.

The trivial algorithm is clearly inferior to other algorithms for all except the smallest patterns. The ST algorithm has poor preprocessing times due to the complicated preprocessing method. Otherwise the algorithms perform similarly.

The scanning times are the most interesting here as that is what the algorithms try to minimize. Figure 6.1 compares the scanning times of the algorithms. The times are shown relative to $\log_c m^2/m^2$ to bring the times with different pattern sizes to the same scale.

As the pattern size increases, the total time decreases at first but starts to increase

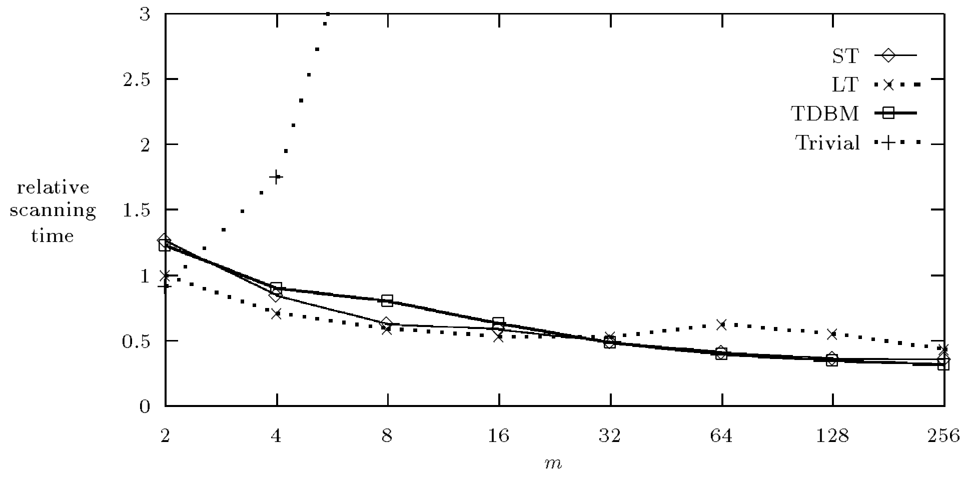



FIG. 6.1. *Scanning times relative to* $\log_c m^2/m^2$ *scaled so that the scanning time of LT is one when* $m = 2$. *The alphabet size c is 2.*

TABLE 6.2

*Running times of LT with total time minimization in milliseconds for* $n = 1000$ *and* $c = 2$.

| $m$ | prepr | scan | total |
|---|---|---|---|
| 2 | 0.10 | 1253 | 1253 |
| 4 | 0.14 | 446 | 447 |
| 8 | 0.42 | 139 | 140 |
| 16 | 0.58 | 41.8 | 42.4 |
| 32 | 1.63 | 13.0 | 14.6 |
| 64 | 2.83 | 5.95 | 8.78 |
| 128 | 2.95 | 5.31 | 8.25 |
| 256 | 3.00 | 3.49 | 6.49 |

when the pattern preprocessing starts to dominate. Both in theory and in practice, the minimum is at $m^2 \approx n\sqrt{\log_c n}$. As suggested earlier, for large patterns the total time can be reduced by using only a part of the pattern in the preprocessing phase. The algorithms LT, ST and TDBM can all be modified in this way. Table 6.2 shows the running times for the modified LT algorithm for alphabet size 2. Figure 6.2 compares the minimized total running time to the preprocessing and scanning times of the basic algorithm that tries to minimize scanning time.

**7. The $k$ mismatches case.** The $k$ *mismatches problem* asks for finding approximate occurrences of $P$ with at most $k$ mismatches, that is, occurrences that match with $P$ except possibly for $\leq k$ symbols. In this section, we generalize the algorithms of §3 and §4 to the $k$ mismatches problem. After the publication of the original version of this paper, Park [20] presented some improvements to the approximate matching algorithms and their analysis. This section includes some small changes to the original inspired by Park's improvements.

One of the requirements of the algorithms in the previous sections was that every potential occurrence has exactly one witness. The $k$ mismatches case, $k \geq 1$, can be handled by increasing the number of witnesses per potential occurrence as shown by the following lemma.

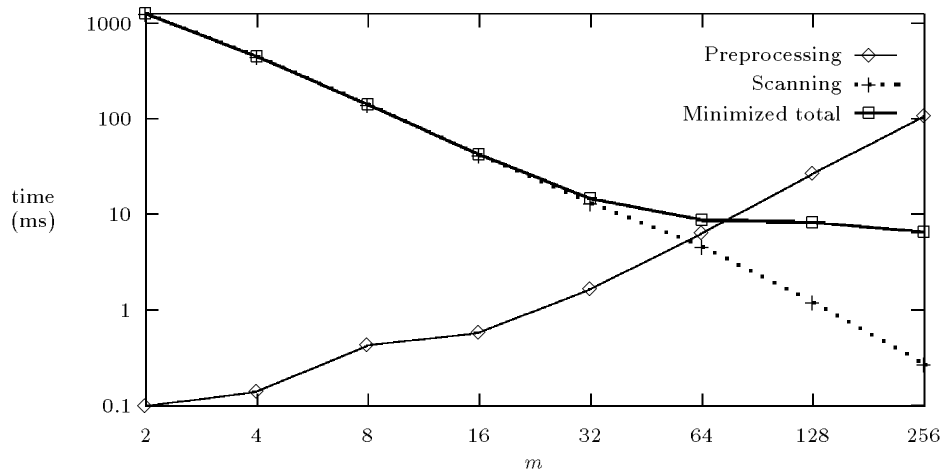



FIG. 6.2. *The minimized total running time of LT compared to the preprocessing and scanning times of basic LT.*

LEMMA 7.1. *Let potential occurrence $R$ have $k+p$ disjoint witnesses. If $< p$ of them are positive then $R$ can not be an occurrence of $P$ with $\leq k$ mismatches.*

Suggested by the lemma, the sampling scheme $\mathcal{Q}$ has to be such that every potential occurrence has $k+p$ non-overlapping witnesses of size $q$. Parameters $q$ and $p$ will be determined in the analysis; note that to get optimal performance we cannot simply choose $p = 1$.

The elimination phase now maintains a counter, initially equal to 0, for every potential occurrence $R$. When a positive witness of $R$ is found the counter of $R$ is increased by one. When the counter achieves value $p$, $R$ is transmitted to the checking phase. The checking phase naively compares $R$ and $P$ until $k+1$ mismatches are found or the whole pattern has been compared.

To avoid initializing all the $\Theta(n^2)$ counters, we do not create a counter until its first incrementation. The counters can be accessed in constant expected time using, e.g., hashing.

We will now show how to modify the sampling schemes of §3 and §4 to have $k+p$ witnesses for each potential occurrence.

**Square templates.** The text sampling scheme described in §3 consists of square samples placed in the form of a regular grid such that every potential occurrence contains exactly one whole sample. If we take $k+p$ copies of such sample grid, each translated such that there are no overlaps, we have a text sampling scheme $\mathcal{Q}_T$ where each potential occurrence contains exactly $k+p$ disjoint text samples. When we define the pattern sampling scheme $\mathcal{Q}_P$ to be the same as in §3, we have the desired sampling scheme $\mathcal{Q} = (\mathcal{Q}_P, \mathcal{Q}_T)$ with $k+p$ witnesses per potential occurrence.

The method works as long as we can keep the text samples disjoint. This means that we must have

$$k + p \leq \left\lfloor \frac{m - \lceil\sqrt{q}\rceil + 1}{\lceil\sqrt{q}\rceil} \right\rfloor^2 . \tag{7.1}$$

**Linear templates.** Using the linear templates of §4, there are two ways to increase the number of witnesses. First, we can increase the number of test rows in $\mathcal{Q}_T$

so that every potential occurrence contains $u$ test rows. If $k+p \le m$, we can select $u = k+p$ and we are done. In the case $k+p > m$, every segment of length $m$ of every test row in $\mathcal{Q}_T$ will contain more than one, say $v$, disjoint samples. This is achieved by selecting a step size $h = \lfloor m/vq \rfloor$. The text sampling scheme can now be defined as

$$\mathcal{Q}_T = \Big\{ Q(im-u+r, jh) \;\Big|\; 1 \le i \le \left\lfloor \frac{n-u+1}{m} \right\rfloor, \quad 1 \le j \le \left\lfloor \frac{n-m}{h} \right\rfloor + 1 + (v-1)q,\ 1 \le r \le \min\{u, n-im+u\} \Big\}.$$

The corresponding pattern sampling scheme is

$$\mathcal{Q}_P = \{ Q(i, j+rqh) \mid 0 \le r \le v-1, 1 \le i \le m,\ 1 \le j \le h \}.$$

This sampling scheme $\mathcal{Q} = (\mathcal{Q}_P, \mathcal{Q}_T)$ gives $uv$ disjoint witnesses per potential occurrence.

For some (large) values of $q$ and $k+p$ it may not be possible to select $u$ and $v$ such that the optimum value $k+p$ of $uv$ is achieved. However, by selecting

$$v = \left\lceil \frac{k+p}{m} \right\rceil \text{ and } u = \left\lceil \frac{k+p}{v} \right\rceil,$$

we can get close as then

$$k+p \;\le\; uv = \left\lceil \frac{k+p}{v} \right\rceil v \le k+p+v-1 = k+p+\left\lceil \frac{k+p}{m} \right\rceil - 1 < (k+p)\left(1+\frac{1}{m}\right).$$

The upper bounds for $u$ and $v$ are given by $u \le m$ and $vq \le m$. Thus, the above selections can be done as long as

$$k+p \le m \left\lfloor \frac{m}{q} \right\rfloor. \tag{7.2}$$

**Analysis.** The running time (without pattern preprocessing) is the sum of three components:

$A =$ the number of entries of $T$ examined during the elimination,
$B =$ the number of updates of the counters for potential occurrences,
$C =$ the number of symbols compared during the checking.

For both of the above sampling schemes we have

$$A \approx \frac{n^2(k+p)q}{m^2}. \tag{7.3}$$

The expected value $\bar{B}$ of $B$ is

$$\bar{B} \le \frac{n^2(k+p)}{c^q} \tag{7.4}$$

because there are at most $n^2$ potential occurrences with $k+p$ witnesses each, and a witness is positive with probability $1/c^q$.

The third component $C$ needs a more careful analysis. First, we denote by $C'$ the number of character comparisons during naive checking of a randomly chosen potential occurrence $R$. As $C'$ is a Pascal distributed random variable with parameters $k+1$ and $\frac{c-1}{c}$, we get

$$\bar{C'} = (k+1)\frac{c}{c-1} \le 2(k+1).$$

However, a candidate reaching the checking phase is not random: it is known to contain $p$ positive witnesses. This increases the number of comparisons by at most $pq$.

Next we analyze the probability, denoted by $H$, that the checking is started for a fixed $R$. As $H$ is the probability that at most $k$ of the $k+p$ witnesses of $R$ are negative, we have

$$H = \sum_{i=0}^{k} \binom{k+p}{i} \left(\frac{1}{c^q}\right)^{k+p-i} \left(1-\frac{1}{c^q}\right)^{i}.$$

Noting that

$$\binom{k+p}{i} = \frac{\binom{k+p}{k}\binom{k}{i}}{\binom{k+p-i}{k-i}},$$

we can rewrite $H$ as follows

$$\begin{aligned} H &= \frac{\binom{k+p}{k}}{c^{qp}} \sum_{i=0}^{k} \frac{\binom{k}{i}}{\binom{k+p-i}{k-i}} \left(\frac{1}{c^q}\right)^{k-i} \left(1-\frac{1}{c^q}\right)^{i} \\ &\le \frac{\binom{k+p}{k}}{c^{qp}} \sum_{i=0}^{k} \binom{k}{i} \left(\frac{1}{c^q}\right)^{k-i} \left(1-\frac{1}{c^q}\right)^{i} \\ &= \frac{\binom{k+p}{k}}{c^{qp}}. \end{aligned}$$

As there are at most $n^2$ potential occurrences, we now have

$$\bar{C} \le n^2 H(\bar{C'} + pq) \le \frac{n^2\binom{k+p}{k}}{c^{qp}}(2k+2+pq). \tag{7.5}$$

Select finally the parameters $q$ and $p$ satisfying

$$q = \lceil \log_c m^2 \rceil \quad \text{and} \tag{7.6}$$

$$p = 1 + \left\lceil \log_{c^q} \binom{k+p}{k} \right\rceil. \tag{7.7}$$

The value of $p$ needs a little more analysis. We first note that

$$1 \le 1 + \left\lceil \log_{c^q} \binom{k+1}{k} \right\rceil \qquad \text{and} \qquad k+1 \ge 1 + \left\lceil \log_{c^q} \binom{2k+1}{k} \right\rceil.$$

The latter inequality holds because

$$\log_{c^q} \binom{2k+1}{k} = \log_{c^q} \left( \frac{2k+1}{k} \frac{2k}{k-1} \cdots \frac{k+1}{1} \right) \le \log_{c^q} (k+1)^k = k \log_{c^q}(k+1) \le k,$$

with the reasonable assumption that $k+1 \le m^2 \le c^q$. Given the above and the fact that the right-hand side of (7.7) is an integer valued non-decreasing function of $p$, we are guaranteed that there exists an integer value of $p$ satisfying (7.7) in the range $[1, k+1]$. If there are many such values, we select the smallest of them. The value can, in practice, be found by trying with $p = 1, 2, \ldots$ until (7.7) is satisfied. Note that if $k = 0$ then $p = 1$ and the algorithm reduces to the exact matching algorithm.

Substituting (7.6), (7.7) and the fact that $p \leq k+1$ into (7.3), (7.4) and (7.5) gives

$$\begin{aligned} A, \bar{C} &= O\left(\frac{n^2(k+1)\log_c m^2}{m^2}\right) \text{ and} \\ \bar{B} &= O\left(\frac{n^2(k+1)}{m^2}\right). \end{aligned}$$

Hence, the expected running time of the algorithm is

$$A + \bar{B} + \bar{C} = O\left(\frac{n^2(k+1)\log_c m^2}{m^2}\right).$$

As $A$ is the most significant component in the running time, the only way to improve the selection of parameters $q$ and $p$ would be to decrease $q$. However, any asymptotically significant decrease in $q$ would increase $\bar{B}$ past $A$. Thus, the above selection of the parameters is asymptotically optimal.

The maximum value of $k$ that the algorithm can handle is limited by (7.1) with square templates and by (7.2) with linear templates. The linear template algorithm is less restrictive in most cases. Substituting $q = \lceil \log_c m^2 \rceil$ and $p \leq k+1$ into (7.2) gives the bound

$$k \leq \frac{m\left\lfloor \frac{m}{\lceil \log_c m^2 \rceil} \right\rfloor - 1}{2}$$

for the maximum value of $k$. For larger patterns, the actual upper bound for $k$ in the linear template algorithm is close to $m^2/\lceil \log_c m^2 \rceil$.

If the counters are destroyed as soon as they can no longer be increased, the space taken by the counters is $O(kn/m)$ on average and $O(nm)$ in the worst case. The space requirement can be reduced with the following technique. Divide the text into slices of $2m-2$ rows that overlap each other by $m-1$ rows, and search each slice separately. All potential occurrences belong to exactly one slice, so no occurrence will be missed or found twice. The space requirement is now reduced to $O(k)$ on average and $O(m^2)$ in the worst case. The number of text samples read at most doubles, so the asymptotic time requirement remains the same. The number of samples can still be reduced by using wider slices, which increases the space requirement, and/or by optimizing the sampling scheme for the slices, which has been done by Park [20].

The next theorem follows.

THEOREM 7.2. *Let $k \leq (m\lfloor m/\lceil \log_c m^2 \rceil \rfloor - 1)/2$. Then the $k$ mismatches problem for two-dimensional $m \times m$ pattern $P$ and $n \times n$ text $T$ can be solved in expected time $O((k+1)n^2 \log_c m^2/m^2)$ and additional space $O(m^2)$. The preprocessing of $P$ takes time and space $O(m^2)$.*

The algorithm requires that $k = O(m^2/\log_c m^2)$. Substituting this into the bound of the above theorem shows that whenever the algorithm works, it is at most linear in $|T|$. Chang and Lawler [9] have obtained similar results for the one-dimensional $k$ differences problem.

**8. Concluding remarks.** There are many possibilities for practical fine-tuning of our algorithms. For example, in the algorithm of §3 it is possible to combine the elimination and checking phases using the spiral-shaped test strings of Gonnet [14]. The text processing in all algorithms can be performed with a window of size $O(m^2)$ into the text.

We ignored worst-case considerations in this paper. It is possible to use some linear worst-case algorithm for the checking phase of our algorithms such that the total time stays $O(n^2)$. However, it is not clear that this would be useful in practice. More important is that the obliviousness of the elimination phase makes parallel implementations of our algorithms simple.

The expected time $O(|T| \log_c |P|/|P|)$ of our exact matching algorithms was shown to be optimal for $d$-dimensional strings when $d$ is considered constant, but not when $d$ is variable. It remains an open problem to either remove the dependence on $d$ from the lower bound or develop an algorithm that is better than ours for variable $d$. Another open problem is whether our approximate matching algorithms are optimal in the expected case.

**Acknowledgements.** We would like to thank Jorma Tarhio for his useful comments and his help in implementing the TDBM-algorithm.

REFERENCES


[1] A. V. AHO AND M. J. CORASICK, *Efficient string matching: an aid to bibliographic search*, Comm. ACM, 18 (1975), pp. 333–340.

[2] A. AMIR, G. BENSON, AND M. FARACH, *An alphabet independent approach to two-dimensional pattern matching*, SIAM J. Comput., 23 (1994), pp. 313–323.

[3] A. AMIR AND M. FARACH, *Efficient 2-dimensional approximate matching of non-rectangular figures*, in Proc. 2nd ACM-SIAM Symposium on Discrete Algorithms (SODA'91), ACM-SIAM, 1991, pp. 212–223.

[4] A. AMIR AND G. M. LANDAU, *Fast parallel and serial multidimensional approximate pattern matching*, Theoretical Comput. Sci., 81 (1991), pp. 97–115.

[5] R. BAEZA-YATES AND M. RÉGNIER, *Fast two-dimensional pattern matching*, Inf. Process. Lett., 45 (1993), pp. 51–57.

[6] T. P. BAKER, *A technique for extending rapid exact-match string matching to arrays of more than one dimension*, SIAM J. Comput., 7 (1978), pp. 533–541.

[7] R. S. BIRD, *Two dimensional pattern matching*, Inf. Process. Lett., 6 (1977), pp. 168–170.

[8] R. S. BOYER AND J. S. MOORE, *A fast string searching algorithm*, Comm. ACM, 20 (1977), pp. 762–772.

[9] W. I. CHANG AND E. L. LAWLER, *Approximate string matching in sublinear expected time*, in Proc. 31st IEEE Symposium on Foundations of Computer Science (FOCS'90), IEEE Computer Society Press, 1990, pp. 116–124.

[10] M. CROCHEMORE, L. GĄSIENIEC, W. PLANDOWSKI, AND W. RYTTER, *Two-dimensional pattern matching in linear time and small space*, in Proc. 12th Symposium on Theoretical Aspects of Computer Science (STACS'95), vol. 900 of LNCS, Springer, 1995, pp. 181–192.

[11] M. CROCHEMORE, L. GĄSIENIEC, AND W. RYTTER, *Two-dimensional pattern matching by sampling*, Inf. Process. Lett., 46 (1993), pp. 159–162.

[12] M. CROCHEMORE AND W. RYTTER, *Text Algorithms*, Oxford University Press, 1994.

[13] Z. GALIL AND K. PARK, *Alphabet-independent two-dimensional witness computation*, SIAM J. Comput., 25 (1996), pp. 907–935.

[14] G. H. GONNET, *Efficient searching of text and pictures (extended abstract)*, Technical Report OED-88-02, Centre for the New OED, University of Waterloo, 1988.

[15] A. HUME AND D. M. SUNDAY, *Fast string searching*, Software—Practice and Experience, 21 (1991), pp. 1221–1248.

[16] R. M. KARP AND M. O. RABIN, *Efficient randomized pattern-matching algorithms*, IBM J. Res. Dev., 31 (1987), pp. 249–260.

[17] J. Y. KIM AND J. SHAWE-TAYLOR, *Fast expected two dimensional pattern matching*, in Proc. 1st South American Workshop on String Processing (WSP'93), 1993, pp. 77–92.

[18] D. E. Knuth, J. H. Morris, and V. R. Pratt, *Fast pattern matching in strings*, SIAM J. Comput., 6 (1977), pp. 323–350.

[19] K. Krithivasan and R. Sitalakshmi, *Efficient two-dimensional pattern matching in the presence of errors*, Information Sciences, 43 (1987), pp. 169–184.

[20] K. Park, *Analysis of two-dimensional approximate pattern matching algorithms*, in Proc. 7th Symposium on Combinatorial Pattern Matching (CPM'96), vol. 1075 of LNCS, Springer, 1996, pp. 335–347.

[21] S. Ranka and T. Heywood, *Two-dimensional pattern matching with $k$ mismatches*, Pattern Recognition, 24 (1991), pp. 31–40.

[22] M. Régnier and L. Rostami, *A simple (but optimal) 2d-witness algorithm*, in Proc. 3rd South American Workshop on String Processing (WSP'96), Carleton University Press, 1996, pp. 243–256.

[23] J. Tarhio, *A sublinear algorithm for two-dimensional string matching*, Pattern Recognition Letters, 17 (1996), pp. 833–838.

[24] E. Ukkonen, *Approximate string-matching with $q$-grams and maximal matches*, Theoretical Comput. Sci., 92 (1992), pp. 191–211.

[25] A. C. Yao, *The complexity of pattern matching for a random string*, SIAM J. Comput., 8 (1979), pp. 368–387.

[26] R. F. Zhu and T. Takaoka, *A technique for two-dimensional pattern matching*, Comm. ACM, 32 (1989), pp. 1110–1120.