INFORMATION AND COMPUTATION 122, 140–148 (1995)

String Matching under a General Matching Relation*

S. MUTHUKRISHNAN† AND H. RAMESH‡

Courant Institute of Mathematical Sciences, New York University, Washington Square, New York, New York 10003

In standard string matching, each symbol matches only itself. In other string matching problems, e.g., the string matching with "don't-cares" problem, a symbol may match several symbols. In general, an arbitrary many-to-many matching relation might hold between symbols. We consider a general string matching problem in which such a matching relation is specified and those positions in a text t, of length n, are sought at which the pattern p, of length m, matches under this relation. Depending upon the existence of a simple and easily recognizable property in the given matching relation, we show that string matching either requires linear (i.e., $O(n+m)$) time or is at least as hard as boolean convolution. As an application, we show that the matching relations of several independently studied string matching problems do indeed fall into the latter (hard) category. We also give a generic string matching algorithm that works for any matching relation and has complexity $o(nm)$ except for very "large" matching relations.

1. INTRODUCTION

In standard string matching, each symbol matches only itself. However, there are some string matching problems in which a symbol may match a number of other symbols. One example is the problem of string matching with "don't cares." Here, the "don't care" symbol matches all other symbols. Another example is the threshold matching problem [DLG91]. In this problem, a distance is defined between each pair of symbols and a symbol matches all those symbols that are within some given threshold. In general, an arbitrary many-to-many matching relation might hold between pattern and text symbols; i.e., each pattern symbol could match a number of text symbols and each text symbol could match a number of pattern symbols.

In this paper, we study the complexity of the string matching problem given an arbitrary matching relation. The matching relation specified all pairs of symbols which match each other. The goal is to find all positions in a text t of length n where the pattern p of length m matches the text. The complexity of this problem clearly depends upon the nature of the matching relation. For simple matching relations like to one-to-one matching relation in standard string matching, there are a number of algorithms which perform the matching in $O(m+n)$ time [KMP77, BM77]. For the slightly more complicated matching relation of the string matching with "don't cares" problem, no linear time algorithm is known.

1.1. Summary of Results

Our contribution is threefold.

(1) We show that:

(a) All possible matching relations fall into one of two classes. String matching under a matching relation in the first class can be done in $O(n+m)$ time. String matching under a matching relation in the second class is at least as hard as (and possibly harder than) the problem of computing the boolean convolution of two vectors of length n and m, respectively.

(b) A simple structural property of the matching relation determines the class it lies in.

Thus the complexity of boolean convolution is a lower bound for string matching under a matching relation in the second class. Since boolean convolution is a well studied problem [W86, AHU74] with the best known upper bound being $O(n \log m)$, it would be rather difficult to obtain linear time algorithms for these string matching problems.

(2) As an application of the above, we show that the matching relations for a number of independently studied string matching problems fall into the second (harder) of the two classes. These problems include all non-trivial versions of the following problems (defined later):

(a) String matching with "don't cares" problem [FP74].

(b) The smaller matching problem [AF91].

(c) Distance matching problems [DLG91].

(d) The alignment problem [A92].

The existence of linear time algorithms for these problems has been open. Our result shows that designing linear time

* A preliminary version of this paper appears as [MR92].

† The work of this author was initiated while visiting IBM T. J. Watson Research Center, Yorktown Heights, and supported in part by NSF/DARPA Grant CCR-89-06949 and NSF Grant CCR-91-03953.

‡ The work of this author was supported in part by NSF Grants CCR-8902221 and CCR-8906949.

0890-5401/95 $12.00

algorithms for these problems is as hard as designing a linear time algorithm for boolean convolution.

(3) We initiate the study of upper bounds for the *general string matching problem* that takes as input an arbitrary matching relation in addition to a text and a pattern. We assume that the matching relation is specified by a graph along with two oracles; one which tells whether two given symbols match each other and another which returns the degree of any given vertex in the above graph. Note that with these oracles, the size of the matching graph is not a lower bound for this problem. Further, for most natural problems, these oracles are very simple.

This problem is useful in situations in which the match criteria are rather fuzzy and one may wish to try a number of match criteria to determine the one that performs best. In such a situation, one would rather have a generic matching algorithm that works for a range of matching relations rather than writing a new algorithm for each matching relation. This problem can be solved in time $O(|\Sigma| B(n, m))$ by essentially employing Fischer and Paterson's reduction [FP74] to boolean convolution where $B(n, m)$ is the complexity of boolean convolution and Σ is the alphabet set from which the symbols are drawn. When Σ is large, i.e., $\Omega(m)$, this is no better than the trivial $O(nm)$ bound. We give an algorithm that runs in time $O(n(sm)^{1/3}\,\text{polylog}(m))$, where s is a factor related to the size of the matching relation and is $O(m^2)$. For $s = o(m^2/\log^2 m)$ this gives a $o(nm)$ time algorithm.

1.2. Previous Work

To our knowledge, this is the first attempt to study general matching relations in string matching. The motivation behind generalizing the string matching problem to allow for arbitrary matching relations arose from the study of the following problems. In each problem, all positions in the text t where pattern p matches t under a matching relation specific to the problem are sought.

String Matching with "Don't Cares"

This problem has the following matching relation: each symbol matches itself and in addition, a special "don't care" symbol ϕ matches all other symbols. So pattern $a\phi b$ matches text *acbacc* only at the leftmost position.

In their seminal paper [FP74], Fischer and Paterson introduced this problem and gave an algorithm for solving this problem by reducing it to boolean convolution (defined later). Their algorithm takes $O(|\Sigma| B(n, m))$ time where $|\Sigma|$ is the alphabet size and $B(n, m)$ is the complexity of boolean convolution. In fact, they were further able to replace the factor of $|\Sigma|$ by a $\log |\Sigma|$ factor by simply encoding each alphabet symbol as a binary string of length $\log |\Sigma|$. They also showed that the problem of string matching with "don't cares" in both the pattern and the text is as hard as boolean convolution. All our reductions are generalizations of the reductions of Fischer and Paterson.

Galil [Ga85] raised the question of whether one could do any better if the "don't care" symbols were restricted to the pattern alone. Pinter [Pi85] considered this scenario and gave an algorithm that ran in $O(n + m)$ time in a model that permitted any subset of m counters to be incremented *simultaneously*. In the more realistic RAM model, the complexity of this algorithm is $O(nd)$ where d is the number of "don't care" symbols in the pattern. Our result shows that this problem is as hard as boolean convolution even when the "don't cares" are restricted to the pattern or to the text.

The Smaller Matching Problem

This problem comes up in approximate matching of two-dimensional non-rectangular figures and was defined by Amir and Farach [AF91]. Here, the symbols form a partially or totally ordered set. The matching relation is defined as follows: a pattern symbol matches all text symbols greater or equal to it. For example, consider the total order $a < b < c < d$. Then pattern *abc* matches text *aaadaab* at the leftmost and the rightmost positions.

Amir and Farach [AF91] raise the question of whether the smaller matching problem can be solved in linear time. Our result shows that designing a linear time algorithm for this problem is at least as hard as designing a linear time algorithm for boolean convolution.

Distance Matching Problems

This family of problems arises in vision related pattern matching and the existence of linear time algorithms for these problems has been open for some time. Suppose a distance function d is defined between each pair of symbols in the alphabet.

In what we call the *threshold matching* problem [DLG91], the matching relation is as follows: symbols a and b match if and only if $d(a, b) \leqslant k$, for some specified constant k. For example, suppose $k = 2$ and the symbols are single digit numbers with distances defined in the obvious manner, then pattern 145 matches the text 327175 only at the leftmost position.

A related problem, but in a slightly different vein, is what we call the *min-threshold matching* problem. Here, the pattern is said to match the text at a particular position if and only if for some pair of aligned symbols a and b, $d(a, b) \leqslant k$, for some specified constant k. So note that in threshold matching, the distance between *each* pair of aligned characters must be at most k, and in min-threshold matching, the distance between *some* pair of aligned characters must be at most k.

The Alignment Problem

This is in some sense the "simplest" string matching problem. The alphabet is binary. The pattern is said to match the text at a particular position if some 1 in the pattern is aligned with a 1 in the text. In other words, the matching relation for this problem is as follows: a 1 in the pattern matches only a 1 in the text while a 0 in the pattern matches neither a 0 nor a 1 in the text. We show that even this "simple" problem is as hard as boolean convolution. In fact, we observe that this problem is a close variant of the boolean convolution problem itself!

This paper is organized as follows. Section 2 gives relevant definitions; Sections 3 and 4 establish the main result; Section 5 gives applications of the main result; and Section 6 deals with upper bounds for the general matching problem.

2. DEFINITIONS AND PRELIMINARIES

Suppose that the text and pattern strings are drawn from an alphabet set Σ_t and Σ_p, respectively. The two alphabet sets are assumed to be disjoint throughout this paper.

The Matching Problem

The *matching problem* is defined by a tuple $(M, \oplus)$, where M is a *match relation* and $\oplus$ is a boolean operator. The match relation M: $\Sigma_t \times \Sigma_p \to \{0, 1\}$ specifies all matching pairs of text–pattern symbols; i.e., $\sigma_t \in \Sigma_t$ matches $\sigma_p \in \Sigma_p$ if and only if $M(\sigma_t, \sigma_p) = 1$. The problem of $(M, \oplus)$-matching is to find all locations i, in a text string of length n, $1 \leqslant i \leqslant n - m + 1$, where for a given pattern string p of length m, $\bigotimes_{1 \leqslant j \leqslant m} (M(t_{i+j-1}, p_j) = 1)$. Intuitively, the pattern is placed on the text beginning at the position i, and we apply the $\oplus$ operator to the result of individual matches of the *aligned* text and pattern characters.

Depending on the operator $\oplus$, there are different versions of the matching problem. In this paper, we only deal with two operators, namely, logical AND (denoted by $\wedge$) and logical OR (denotes by $\vee$). The corresponding versions of the matching problem are referred to as the *and-version* and the *or-version*, respectively. While other operators are conceivable, these two operators are the most natural. In fact, we will also deal with the + operator in Section 6. The and-version seeks all positions in the text where every character in the pattern matches the aligned text character. Observe that the *standard string matching problem* is simply the matching problem $(M, \wedge)$, where M is a permutation matrix. The or-version seeks those positions in the text where at least one of the pattern characters matches its aligned text character. As we shall demonstrate later in Section 5, these two versions of the matching problem, for appropriately defined match relations, capture several well studied string matching problems.

The match relation M can be specified in the form of a boolean matrix of size $\Sigma_p \times \Sigma_t$. Associated with this matrix is a bipartite graph $G(M)$, called the *matching graph*. Σ_t and Σ_p constitute the two vertex set of $G(M)$. There is an edge between two nodes in the bipartite graph if and only if the corresponding symbols from the alphabets match. For the sake of convenience, we sometimes refer to the relation M and its associated matrix by the same name.

In the standard string matching scenario, the text and pattern share alphabet symbols. Our assumption regarding disjointness of the text and pattern alphabet preserves generality because each such shared symbol σ can be replaced by a text character σ_t and a pattern character σ_p with $M(\sigma_t, \sigma_p) = 1$.

The Boolean Convolution Problem and Its Variants

The problem of multiplying two binary numbers is well known and well studied. We consider the boolean convolution problem which is the problem of multiplying two binary vectors over the operators $(\wedge, \vee)$ instead of the usual arithmetic addition and multiplication $(\times, +)$. Formally, given two boolean vectors $a_1 a_2 \cdots a_n$ and $b_1 b_2 \cdots b_m$, $n \geqslant m$, we define their convolution to be the vectors $c_2 c_3 \cdots c_{n+m}$, where

$$c_k = \bigvee_{1 \leqslant i \leqslant m} (a_{k-i} \wedge b_i)$$

for $2 \leqslant k \leqslant n + m$, with any out of range references assumed to be 0. For example, the boolean convolution of vectors 100101110 and 101 yields the vector 10110111110.

Actually, we are interested only in the $n - m + 1$ bits $c_{m+1}, \ldots, c_{n+1}$. This is because the output of a matching problem consists of $n - m + 1$ text positions and we wish to relate these positions to bits in the output of the boolean convolution problem. With this in view, we define the *short-boolean-convolution problem*. The output of short-boolean-convolution is the same as the output of boolean convolution except that the first and last $m - 1$ bits are dropped. We define the *reverse-short-boolean-convolution problem* for two boolean vectors $a_1 \cdots a_n$ and $b_1 \cdots b_m$, as the problem of computing the short-boolean-convolution of the first vector with the vector resulting from reversing the bits of the second vector.

LEMMA 1. *The following problems are equivalent in the sense that they are reducible to each other in linear, i.e., $O(n + m)$, time:*

1. *Boolean convolution.*
2. *Short-boolean-convolution.*
3. *Reverse-short-boolean-convolution.*

Proof. Clearly, the latter two problems are equivalent and each can be reduced to the first problem in linear time. It suffices to show that the first problem is reducible to the second in linear time. To see this, note that the boolean convolution of vectors a and b, of length n and m, respectively, $n \geq m$, equals the short-boolean-convolution of vectors a' and b, where a' is obtained from a by padding it with $m-1$ 0's at each end. ∎

In what follows, *all references to the boolean convolution problem refer to the reverse-short-boolean-convolution problem unless stated otherwise.* The connection between the boolean convolution problem and the matching problem $(M, \oplus)$ is as follows. It can be intuitively seen that boolean convolution corresponds to aligning the vector b beginning at some location in the vector a, applying the operation $\wedge$ to each pair of bits in a and b aligned with each other, and then applying the operation $\vee$ to the resulting bits. $\wedge$ and $\vee$ in the boolean convolution problem are thus the counterparts of M and $\oplus$ in the matching problem.

Let $B(n, m)$ refer to the complexity of computing the convolution of two boolean vectors over $(\wedge, \vee)$. $B(n, m) = O(n \log m)$ [AHU74].

3. THE AND-VERSION OF THE MATCHING PROBLEM

In this section, we explore the computational complexity of the and-version of the matching problem $(M, \wedge)$.

Define a *complete bipartite graph* to be a bipartite graph such that each vertex of one side has an edge to each of the vertices on the other side. Note that the number of vertices on the two sides need not be the same. A bipartite graph is said to be *simple* if each connected component is either a single vertex or a complete bipartite graph.

For the rest of this section, *fix some matching problem* $(M', \wedge)$. Consider the complexity of matching pattern p of length m against a text t of size n under this matching relation. Note that this fixes the sizes of the alphabet sets from which t and p are drawn, i.e., n and m can be arbitrarily long compared to the size of the matching relation M' which is $O(|\Sigma_t| \cdot |\Sigma_p|)$. We explore the nature of the matching graph $G(M')$. Recall that the matching graph $G(M')$ is a bipartite graph on the vertices corresponding to the possible text symbols and the vertices corresponding to the possible pattern symbols. The edges in $G(M')$ denote matching symbols. Consider the connected components of $G(M')$.

LEMMA 2. *If $G(M')$ is simple, $(M', \wedge)$ can be solved for any t and p in $O(n+m)$ time.*

Proof. Find the connected components of $G(M')$. Associate with each connected component C a distinct symbol σ_C. Generate new text and pattern strings in which each character is replaced by the symbol of the connected component to which it belongs. Now, $(M', \wedge)$ is reduced to the standard string matching problem which can be solved in linear time using one of the many well known algorithms for standard string matching, e.g., the Knuth–Morris–Pratt algorithm [KMP77], The Boyer–Moore algorithm [BM77], etc. ∎

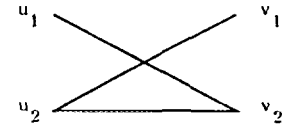

FIG. 1. A crucial subgraph.

Remark. In fact, if $G(M')$ is simple, then $(M', \wedge)$ can be solved for any t and p in $O(n+m)$ time even when the alphabet sets are not fixed given an oracle for determining whether two given symbols *match* each other. Here we say that two symbols *match* if their corresponding vertices are in the same connected component of $G(M')$. It can easily be seen that the Knuth–Morris–Pratt algorithm works unchanged in this case, with symbol comparisons being answered by the oracle.

An *induced subgraph* of a bipartite graph G' consists of a subset of the vertices of G' together with all edges of G' which have both endpoints in this subset of vertices. Define a *crucial subgraph* of a bipartite graph G' to be a induced subgraph of G' consisting of four vertices, two on either side, such that one vertex on each side has degree 2 and one vertex on each side has degree 1. A crucial subgraph is shown in Fig. 1.

LEMMA 3. *If $G(M')$ is not simple, it contains a crucial subgraph.*

Proof. There exists a connected component C of $G(M')$ such that C is neither a singleton vertex nor a complete bipartite graph. This implies that there exist two vertices $l_1, r_1 \in C$ such that l_1 and r_1 are on opposite sides and no edge connects l_1 and r_1. Since C is connected, there exists a path from l_1 to r_1 containing only vertices of C. Let $l_1, r_2, l_2, \ldots, r_k, l_k, r_1$ be such a path. Consider two cases now. These cases are illustrated in Fig. 2. If l_1 is connected by an edge to each of $r_2, \ldots, r_k$ then the subgraph induced by the vertices l_1, l_k, r_1, r_k is a crucial subgraph. Otherwise, let i, $3 \leq i \leq k$, be the least index such that l_1 and r_i are not connected by an edge. Then the subgraph induced by the vertices $l_1, l_{i-1}, r_{i-1}, r_i$ is a crucial subgraph. ∎

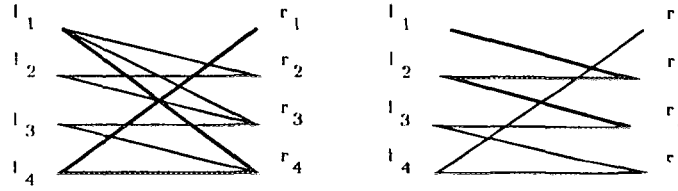

FIG. 2. Crucial subgraphs of $G(M')$.

LEMMA 4. *Fix a matching relation M'. If $G(M')$ has a crucial subgraph, then solving $(M', \wedge)$ for a text t and pattern p is at least as hard as performing the boolean convolution of two vectors of length n and m.*

Proof. We reduce any instance of the boolean convolution problem to an instance of the $(M', \wedge)$ problem with text t and p. Let $a_1 \cdots a_n$ and $b_1 \cdots b_m$ be the two vectors in the convolution problem. First a crucial subgraph of $G(M')$ is found in constant (i.e., independent of n, m) time. Let (u_1, u_2, v_1, v_2) be the vertices in the crucial subgraph such that u_1 and u_2 are on the side that corresponds to text symbols and v_1 and v_2 are on the side that corresponds to pattern symbols. Let u_2 and v_2 be of degree 2 and u_1 and v_1 be of degree 1. We refer to symbols associated with these vertices by the same name as the vertices themselves.

Generate a text $t_1 \cdots t_n$ from the vector a by replacing a 1 by the symbol u_1 and a 0 by the symbol u_2. Similarly, generate a pattern $p_1 \cdots p_m$ from the vector b by replacing a 1 by the symbol v_1 and a 0 by the symbol v_2. The positions at which the pattern does not match the text are *precisely* those positions at which the output of the convolution is a 1. To see this, note that the pattern does not match the text at position i if and only if some v_1 in the pattern is aligned with some u_1 in the text when the left end of the pattern is aligned with text position i. This implies that the pattern does not match the text at position i if and only if some 1 in b is aligned with some 1 in a when the left end of the vector b is aligned with a_i, or equivalently the ith output bit of the convolution problem has value 1.

The time taken for the reduction is $O(n+m)$. ∎

From the above, we draw the following main result.

THEOREM 1 (The Matching Complexity Theorem). *Fix a matching relation M. The matching problem $(M, \wedge)$ falls into one of two categories.*

1. *$(M, \wedge)$ can be solved for a pattern p of length m and a text t of length n in linear, i.e., $O(n+m)$, time.*
2. *$(M, \wedge)$ is at least as hard as boolean convolution.*

Let $|G(M)|$ be the number of edges and vertices in the match graph G. Then:

THEOREM 2. *Given any matching relation M, it can be determined in $O(|G(M)|$ time whether $(M, \wedge)$ is as hard as boolean convolution or solvable in linear time.*

Note that the crucial subgraph is quite simple; it can be found in a matching relation over text and pattern alphabets of size 2 each. Hence, some string matching problems are as hard as boolean convolution even when restricted to a binary alphabet.

4. THE OR-VERSION OF THE MATCHING PROBLEM

The or-version of the matching problem $(M, \vee)$ seeks those occurrences of the pattern in the text where at least one pair of the aligned characters match under the relation M. Consider the complementary relation specified by $\bar{M}$ where $\bar{M}(\sigma_t, \sigma_p) = 1$ if and only if the two symbols mismatch.

LEMMA 5. *Pattern p matches t at position i under $(\bar{M}, \wedge)$ if and only if p does not match t at position i under $(M, \vee)$.*

Proof. If there is a match under $(\bar{M}, \wedge)$ at a position i, then all pairs of aligned characters mismatch when the pattern is placed on the text at position i. This implies that there cannot be a match under $(M, \vee)$ since no pair of aligned characters matches. The other direction is argued analogously. ∎

THEOREM 3. *Fix a matching relation M. The matching problem $(M, \vee)$ falls into one of two categories.*

1. *Suppose $G(\bar{M})$ is simple. Then $(M, \vee)$ can be solved for a pattern p of length m and a text t of length n is linear, i.e., $O(n+m)$, time.*
2. *Suppose $G(\bar{M})$ is not simple. $(M, \vee)$ is at least as hard as boolean convolution.*

Proof. Follows from Lemma 5, the Matching Complexity Theorem, and the fact that $\bar{M}$ can be generated from M in constant time since M is fixed. ∎

5. APPLICATIONS

We have addressed the complexity of matching problems given a general match relation. Now, we consider some specific matching relations that correspond to well known problems and derive corollaries of our main result. In these problems, the pattern and the text character sets need not be disjoint. While applying the matching complexity theorem to these problems, the two character sets are made disjoint as described earlier in Section 2.

As already mentioned, the standard string matching problem has a diagonal or a permutation matching matrix. The matching graph for this matching relation does not have a crucial subgraph and this problem can be solved in linear time.

String Matching with "Don't Cares"

Let ϕ be the "don't care" symbol. As in standard string matching, any symbol matches itself. In addition, ϕ can match any of the symbols in the text and the pattern alphabet including itself. However, any one occurrence of ϕ

in the strings can match only one symbol. Assume that ϕ occurs only in the pattern. The argument for the case when ϕ occurs only in the text or in both the pattern and the text is similar. Let the text and pattern be strings over alphabet $\{\sigma_1, \sigma_2\}$. The matching relation for this problem is shown below.

	σ_1	σ_2	ϕ
σ_1	1	0	1
σ_2	0	1	1

Observe that the columns corresponding to σ_2 and ϕ yield a submatrix that corresponds to a crucial subgraph.

LEMMA 6. *The matching with "don't cares" problem is at least as hard as boolean convolution, when "don't cares" occur only in the text, only in the pattern or both.*

The Smaller Matching Problem

If the symbols are totally ordered, the matching matrix for this problem is simply an upper triangular matrix. For $|\Sigma| > 1$, this matrix will always have a 2×2 submatrix consisting of three 1's and a 0. This submatrix corresponds to a crucial subgraph. Next, consider the case when the symbols are partially ordered. Let C be the set of connected components of the collection of the directed acyclic graphs DAGs formed by the partial order. In any DAG in this collection, the character corresponding to a particular node matches exactly those characters which are associated with its descendant nodes.

LEMMA 7. *The smaller matching problem is at least as hard as boolean convolution for $|\Sigma| > 1$ whenever the connected components of the collection of DAGs of the partial order C are not all singleton vertices.*

Proof. We show that if some connected component in C has more than one vertex then the matching graph for this matching relation contains a crucial subgraph. Consider any component with at least two vertices and pick any two vertices a and b connected by an edge. Let a be the parent of b. We denote occurrences of character a in the pattern and the text by a_p and a_t, respectively. Similarly, b_p and b_t denote occurrences of the character b in the pattern and the text, respectively. Clearly, a_p matches a_t and b_t, while b_p matches only b_t. The matching subgraph induced by these four characters yields the required crucial subgraph. ∎

Distance Matching Problems

Consider the threshold-matching problem first. Let the text and the pattern be strings over some alphabet Σ, where $|\Sigma| \geq 2$. Let the specified threshold be k, $0 \leq k \leq |\Sigma| - 1$. The matching matrix for this problem is a Toeplitz matrix of order k. As an example, we show the matrix for $k = 1$. It is easy to see that if $k \neq 0$ and $k \neq |\Sigma| - 1$, any such matrix has a 2×2 submatrix containing three 1's and a 0 that corresponds to a crucial subgraph.

	σ_1	σ_2	σ_3	...	$\sigma_{\lvert\Sigma\rvert}$
σ_1	1	1	0	...	0
σ_2	1	1	1	...	0
σ_3	0	1	1	...	0
σ_4	0	0	1	...	0
...	...	...	...	...	...
$\sigma_{\lvert\Sigma\rvert}$	0	0	0	...	1

LEMMA 8. *The threshold matching problem is at least as hard as boolean convolution when $|\Sigma| - 1 > k > 0$ and $|\Sigma| > 1$.*

Consider the min-threshold matching problem next. The minimum distance between some pattern symbol and its aligned text symbol is required to be less than some threshold value k, $|\Sigma| - 1 \geq k \geq 0$, $|\Sigma| > 1$. This problem has the same matching matrix as the threshold problem and the matching problem we solve is the or-version of the threshold matching problem. By Theorem 3:

LEMMA 9. *The min-threshold matching problem is at least as hard as boolean convolution when $|\Sigma| - 1 > k > 0$ and $|\Sigma| > 1$.*

Analogously, we can consider the max-threshold problem where we seek those text positions in which the distance between some pair of aligned symbols is at least a threshold k. This problem is now easily seen to be at least as hard as boolean convolution.

The Alignment Problem

The matching matrix for this problem is given below.

	0	1
0	0	0
1	0	1

This problem is an or-version problem. The complement of the matching matrix itself is a crucial graph. So this problem is at least as hard as the boolean convolution problem. In fact, this problem is exactly the convolution problem redefined in string matching terms.

6. UPPER BOUNDS FOR THE GENERAL MATCHING PROBLEM

We define the *general string matching problem* as follows. This problem takes as input a matching relation M, a text

t, and a pattern p and locates all positions in the text at which the pattern matches under the matching relation M. We assume that the matching relation is specified by a bipartite graph along with two oracles, one which tells whether two given symbols match each other and another which returns the degree of any given vertex in the above graph. Note that with these oracles, the size of the matching graph is not a lower bound for this problem. Further, for most natural problems, these oracles are very simple. It is also required that the bipartite graph specify the matches between only those symbols which actually occur in the text or in the pattern.

Let n and m denote the sizes of the text and the pattern respectively. We assume that $n \leqslant 2m$. The case $n > 2m$ can be handled as follows. The text is divided into $\lfloor n/m \rfloor$ overlapping pieces, each of length $2m$ (except possibly the last piece which is at most $2m$ in length), with each pair of adjacent pieces overlapping in exactly m characters. The pattern is then matched against each of these pieces of the text separately.

The matching relation has size $|\Sigma_t| \cdot |\Sigma_p|$, where Σ_t is the set of those symbols which appear in the text and Σ_p is the set of those symbols which appear in the pattern. By the above assumption, $|\Sigma_t| \leqslant 2m$. We define a parameter s to be the *subproblem matching size*, the maximum, over all $\lfloor n/m \rfloor$ problems of the number of edges in the subgraph of $G(M)$ induced by those symbols that actually appear in the text and pattern of that problem. Since each text string has length at most $2m$, each subproblem has at most $2m$ different text symbols and at most m different pattern symbols. Thus $s \leqslant 2m^2$. Note that for a given t, p, and M, the precise value of s can be computed in $O(n)$ time by simply adding up the degrees, in $G(M)$, of the distinct symbols which appear in t.

We assume that the matching graph $G(M)$ has the crucial subgraph, hence the general matching problem we consider is at least as hard as boolean convolution. If the matching graph does not have a crucial subgraph, the general matching problem can be solved in linear time as described in Section 3.

Let $|\Sigma| = \min\{|\Sigma_t|, |\Sigma_p|\}$. The general matching problem can be solved in $O(|\Sigma| B(n, m))$ time using Fischer and Paterson's method [FP74]. Therefore, if the text or the pattern were strings over a constant-sized alphabet, it follows additionally from the proof of the Matching Complexity Theorem that any string matching problem is *exactly* as hard as boolean convolution if the matching graph for that problem has a crucial subgraph. However, for $|\Sigma| = \Omega(m)$, this is worse than the trivial $O(nm)$ bound. We give an algorithm that is $o(nm)$ whenever the matching relation is not too dense. This algorithm actually solves a more general problem. The definition of this problem and the corresponding convolution problem follow.

Counting Mismatches Problem

Given the text and the pattern, the counting mismatches problem seeks for each position i in the text, the number of mismatches under the relation M when the pattern is placed beginning at i. This problem, denoted by $(M, +)$, takes a text t and a pattern p and evaluates $+_{1 \leqslant j \leqslant m} M(t_{i+j-1}, p_j)$ for each index i, $1 \leqslant i \leqslant n - m + 1$. The operator $+$ is the arithmetic addition. Note that if there were no mismatches at a position i, then the pattern matches the text at i. Hence, solving $(M, +)$ is *sufficient* to solve the general string matching problem.

The convolution problem that corresponds to the counting mismatches problem is the integer convolution.

Integer Convolution. We define the *integer convolution* over binary vectors in the same way as the boolean convolution, but with the arithmetic addition and multiplication operators $(\times, +)$ replacing the boolean operators $(\vee, \wedge)$ respectively. Observe that there are no carry-overs as in conventional integer multiplication. Henceforth, we will refer to *reverse-short-integer-convolution* as the integer convolution. The property of the integer convolution of two binary vectors which relates to the counting mismatches problem is that the integer convolution computes, for all positions i in the longer vector, the number of 1's in the two vectors which are aligned when the smaller vector is placed on the larger one beginning at position i. This would be analogous to counting the mismatches in general string matching.

The integer convolution can be performed in time $I(n, m) = O(n \log m)$ [AHU74].

ALGORITHMS. The naive algorithm for the counting mismatches problem takes time $O(nm)$. We present two other algorithms for the counting mismatches problem $(M, +)$. In what follows, we assume that of the text and the pattern, the pattern has fewer distinct symbols and Σ denotes the set of these symbols. The case wen the text has fewer distinct symbols is solved similarly.

Algorithm **A** is a simple generalization of the standard Fischer–Paterson reduction for string matching with "don't cares" [FP74]. Algorithm **A** takes time $O(|\Sigma| I(n, m))$. Observe that in the worst case, $|\Sigma| = m$ and this complexity is worse than the naive complexity of $O(nm)$.

In Algorithm **B**, a divide-and-conquer strategy is adopted to reduce the counting mismatches problem to subproblems over a smaller alphabet size. This divide-and-conquer paradigm has been used in [Ab87, K89, AF91], to yield algorithms with worst case performance $o(nm)$ for special string matching problems. Our strategy is similar to the one in [Ab87]; however, the general nature of the matching relation we consider poses some additional problems. Algorithm B achieves $o(nm)$ time whenever the subproblem matching size s is not too large, i.e., $s = o(m^2/\log^2 m)$.

6.1. Algorithm A: Alphabet-Dependent Mismatch Counting

We generalize the Fischer–Paterson reduction to get Algorithm **A**. Consider each symbol that occurs in the pattern in turn. For each such symbol, generate the following two binary vectors and compute their integer convolution. The first binary vector has length n and contains a 1 at those positions at which the corresponding text character mismatches (under M), the symbol being considered. The second binary vector has length m and contains a 1 at those positions such that the corresponding position in the pattern is occupied by the symbol under consideration. Add the ith term in each convolution to get the number of mismatches when the left end of the pattern is aligned with position i in the text.

Algorithm **A** is correct since by considering each symbol in the pattern in turn, each mismatch is counted exactly once. This algorithm takes time $O(|\Sigma|\, I(n, m))$.

6.2. Algorithm B: Dividing Strategy

We divide the counting mismatches problem into subproblems over smaller alphabet sets. This strategy is similar to the one in [Ab87]. The description is presented in terms of two real parameters q and r. The values for these parameters will be fixed later.

Let the result of projecting a string x onto an alphabet set Σ be the string x_Σ which is x except that each symbol not in the set Σ is replaced by a special "don't care" symbol ϕ.

Divide the set Σ_p into two sets: Σ_p^1 is the set of all symbols in p which appear at least q times in p and $\bar{\Sigma}_p^1$ is its complement. The set Σ_p^1 can be easily identified in $O(m \log |\Sigma|)$ time by scanning the pattern once left to right. Divide Σ_t into two sets as follows: Σ_t^1 is the set of all text symbols which match at least r distinct pattern symbols under the match relation M (i.e., the set of text symbols with degree at least r in $G(M)$) and $\bar{\Sigma}_t^1$ is its complement. The set Σ_t^1 can be identified in $O(n)$ time since the degree of each node is available as part of the input. Our overall algorithm has four steps:

1. Solve the counting mismatches problem on t and $p_{\Sigma_p^1}$ using Algorithm **A**.
2. Solve the counting mismatches problem on $t_{\Sigma_t^1}$ and $p_{\bar{\Sigma}_p^1}$ using Algorithm **A**.
3. Solve the counting mismatches problem on $t_{\bar{\Sigma}_t^1}$ and $p_{\Sigma_p^1}$ as described below.
4. Sum the mismatches in Steps 1–3 for each position in the text.

Step 3 needs to be done carefully. The non-ϕ symbols in the pattern $p_{\Sigma_p^1}$ appear few times ($\leqslant q$) and the non-ϕ symbols in the text $t_{\Sigma_t^1}$ match few symbols ($\leqslant r$) each. However, there could be a large number of symbols in bolt the text and the pattern and matching them by Algorithm **A** could be prohibitive. We solve this problem in three steps:

3A. For each position in the text, count the number of locations at which a non-ϕ symbol in the text is aligned (but not necessarily matched) with a non-ϕ symbol in the pattern when the pattern is placed beginning at that position. This is performed by generating the following two binary vectors and computing their integer convolution. The first binary vector has size n and contains a 1 at those positions at which the text contains a non-ϕ character and a 0 at other positions. The second binary vector of size m is generated similarly from the pattern.

3B. For each position in the text, count the number of locations at which some non-ϕ character in the pattern matches the aligned non-ϕ character in the text when the pattern is placed beginning at that position.

This is done as follows. First, we compute for each symbol $\sigma \in \bar{\Sigma}_p^1$ a list of the locations in the pattern in which σ appears. This list can be constructed in $O(m \log |\Sigma|)$ time by scanning the pattern left to right once. Following this, consider each position i in the text in turn which is occupied by a non-ϕ character c_i. Consider each non-ϕ symbol σ that matches c_i and is in $\bar{\Sigma}_p^1$. Increment a counter $C[i-j]$ (initialized to 0, $1 \leqslant i-j \leqslant n-m+1$) for each j such that σ is the $j+1$ th character in the pattern.

3C. For each position in the text, subtract the number in Step 3B from the number in Step 3A.

That completes the description of the algorithm.

EXAMPLE OF STEP 3. Let $t = aaba\phi b\phi\phi ba$ and $p = \phi b\phi c$ be the text and the pattern, respectively, for Step 3, with the standard notion that two symbols match if and only if they are identical. (That is, t and p are respectively $t_{\bar{\Sigma}_t^1}$ and $p_{\Sigma_p^1}$, for appropriate $\bar{\Sigma}_t^1 (= \{a, b\})$, $\bar{\Sigma}_p^1 (= \{b, c\})$, and input to some counting mismatches problem.) We solve the counting mismatches problem on t and p in Step 3.

Step 3A. The output is 212021 obtained by computing the integer convolution of 111101011 and 0101 obtained from t and p, respectively.

Step 3B. The output is 010010 obtained as described below. Consider C initialized to 000000. Note that b appears in p at location 2 and its appears in t at locations 3, 6, and 8. Corresponding to locations 3 and 6 in t, locations $[3-1]$ and $[6-1]$ in C get incremented. That gives the output claimed above. Note that even though b appears at location 8 in t, it does not affect C.

Step 3C. The output is 202011 which is the correct output for the counting mismatches problem on t and p.

We now consider the complexity of this algorithm. There are at most m/q symbols in the pattern which each appear at least q times. Thus, Step 1 takes time $O((m/q)\, I(n, m))$

using Algorithm **A**. There are at most s/r different symbols in the text which match at least r pattern symbols, where s is the subproblem matching size. Thus, Step 2 takes time $O((s/r)\, I(n, m))$. In Step 3, each position in the text with a non-ϕ symbol could match at most r pattern symbols each of which could appear at most q times. Hence, Step 3 takes time $O(I(n, m) + nqr)$.

THEOREM 4. *Algorithm B takes time* $O(nqr + (m/q)\, I(n, m) + (s/r)\, I(n, m))$ *for two parameters q and r.*

To balance the complexity, set $q = ((m^2 \log m)/s)^{1/3}$ and $r = ((s^2 \log m)/m)^{1/3}$. Then Algorithm **B** takes $O(n(sm \log^2 m)^{1/3})$ time. This gives $o(nm)$ time when s is $o(m^2/\log^2 m)$.

Remarks. For any given problem, one might choose alternate values of q and r to achieve better worst-case behavior. For instance, consider a matching relation in which each text symbol matches at most a constant number of pattern symbols. Then, we can get an $O(n\sqrt{m}\,\text{polylog}\, m)$ time algorithm by omitting Step 2 and balancing the complexity with the parameter q. Also, note that the k-mismatch problem under any general matching relation can be solved using Algorithm **B**. Finally, the complexity of Algorithms **A** and **B** are incomparable in the following sense: depending upon the relative values of $|\Sigma|$ and s, either could be the more efficient of the two.

7. CONCLUSIONS

We have examined the computational complexity of string matching over a general matching relation. We have divided the set of matching relations into two classes, one for which string matching can be done in linear time and another for which string matching is at least as hard as boolean convolution. A simple property decides to which class a matching relation belongs. We show that the matching relations for the smaller matching problem, the distance problems, the alignment problem and all versions of the matching with "don't cares" problem (i.e., "don't cares" in text, pattern, or both) fall into the second class. We also define a general string matching problem which takes a matching relation in addition to a pattern and a text as input. We give an upper bound for this problem that is $o(nm)$, whenever the subproblem matching size of the matching relation is $o(m^2/\log^2 m)$. The existence of a $o(nm)$ upper bound in general remains open and efficient algorithms for this problem would be of interest.

ACKNOWLEDGMENTS

The authors thank Richard Cole, Gad Landau, Amihood Amir, and Martin Farach for discussions and references. The first author thanks Krishna Palem for discussions on tree pattern matching which led to the investigation of this problem.

Received September 30, 1992; final manuscript received December 9, 1994

REFERENCES

[A92] Amir, A. (1992), Open Problems Session, Combinatorial Pattern Matching Conference, Tucson, AZ.

[Ab87] Abrahamson, K. (1987), Generalized string matching, *SIAM J. Comput.* 1039–1051.

[AF91] Amir, A., and Farach, M. (1991), Efficient 2-dimensional approximate matching of non-rectangular figures, *in* "Proceedings of the Second ACM–SIAM Symposium on Discrete Algorithms," pp. 34–457.

[AHU74] Aho, A., Hopcroft, J., and Ullman, J. (1974), "The Design and Analysis of Algorithms," Addison-Wesley, Reading, MA.

[BM77] Boyer, R., and Moore, S. (1977), A fast string matching algorithm, *Comm. ACM* **20**, 762–772.

[DLG91] Dinstein, I., Landay, G., and Guy, G. (1991), Parallel (PRAM EREW) algorithms for contour-based 2-D shape recognition. *Pattern Recognition* **24**, No. 10, 929–942.

[FP74] Fischer, M., and Paterson, M. (1974), String matching and other products, *SIAM–AMS Proc.* **7**, 113–125.

[Ga85] Galil, Z. (1985), Open problems in stringology, *in* "Combinatorial Algorithms on Words" (A. Apostolico and Z. Galil, Eds.), pp. 1–8, NATO-ASI Series, Kluwer, Dordrecht, 1985.

[K89] Kosaraju, S. R. (1989), Efficient tree pattern matching, *in* "Proceedings of the 30th IEEE Symposium on Foundations of Computer Science," pp. 178–183.

[KMP77] Knuth, D. E., Morris, J., and Pratt, V. (1977), Fast pattern matching in strings, *SIAM J. Comput.* 323–350.

[MR92] Muthukrishnan, S., and Ramesh, H. (1992), String matching under a general matching relation, *in* "Proceedings of the 18th Annual Symposium on Foundations of Software Technology and Theoretical Computer Science," Lecture Notes in Computer Science, Vol. 652, pp. 356–367, Springer-Verlag, Berlin/New York.

[Pi85] Pinter, R. Y. (1985), Efficient string matching with don't-care patterns, *in* "Combinatorial Algorithms on Words" (A. Apostolico and Z. Galil, Eds.), NATO-ASI Series, pp. 11–29, Kluwer, Dordrecht.

[W86] Wegener, I. (1986), "The Complexity of Boolean Functions," Wiley–Teubner Series in Computer Science, pp. 168–169, Wiley, New York.

Printed in Belgium