Alphabet Dependence in Parameterized Matching

Amihood Amir[*] Georgia Tech

Martin Farach[†] DIMACS & Rutgers U.

S. Muthukrishnan[‡] Courant Institute

July 14, 1993

Abstract

The classical pattern matching paradigm is that of seeking occurrences of one string in another, where both strings are drawn from an alphabet set Σ. A recently introduced model is that of parameterized pattern matching; the main motivation for this scheme lies in software maintenance where programs are considered "identical" even if variables names are different. Besides the fixed symbols from Σ, strings, under this model, have additional symbols from a variable set Π and occurrences of one string in the other are sought, where renaming of the variables from Π is allowed in a match.

In this paper we show that finding the occurrences of a m-length string in a n-length string under the parameterized pattern matching paradigm can be done in time $O(n \log \pi)$, where $\pi = \min(m, |\Pi|)$, that is, independent of $|\Sigma|$. Additionally, we show that in general this dependence on $|\Pi|$ is inherent to any algorithm for this problem in the comparison model – that is, our algorithm is optimal.

1 Introduction

In the classical pattern matching model, we seek occurrences of a string, or more generally a set of strings, in a distinguished string, where all strings are comprised of symbols from an alphabet set Σ. The basic problem in this paradigm is that of *standard string matching*, that is, finding all occurrences of a pattern string of length m in a text string of length n. This problem is known to be solvable in $O(n + m)$ time independent of the alphabet size $|\Sigma|$ [4, 7].

A related model of parameterized pattern matching was recently introduced by Baker [2]. The main motivation for this scheme lies in software maintenance, where programs are to be considered

[*]College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280; (404) 853-0083; amir@cc.gatech.edu; Partially supported by NSF grant CCR-92-23699 and IRI-90-13055.

[†]DIMACS, Box 1179, Rutgers University, Piscataway, NJ 08855; (908) 932-5928; farach@dimacs.rutgers.edu; Supported by DIMACS under NSF contract STC-88-09648.

[‡]Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012; (212) 998-3061; muthu@cs.nyu.edu Partially supported by NSF/DARPA grant CCR-89-06949 and by NSF grant CCR-91-03953.

"identical" even if variable names are different. Therefore, strings under this model are comprised of symbols from two disjoint sets Σ and Π containing fixed symbols and variable/parameter symbols respectively. In this paradigm, we seek *parameterized occurrences* i.e., occurrences up to renaming of the variable symbols, of a string in another.

Formally, parameterized pattern matching is as follows. A *p-string* is a string over $\Sigma \cup \Pi$. Two p-strings s_1 and s_2 of same length are said to *p-match* if there exists a bijection $f : \Pi_1 \leftrightarrow \Pi_2$, where Π_1 and Π_2 are the symbols from Π in s_1 and s_2 respectively, such that the following holds: s_1 (s_2, respectively) equals s_2 (s_1, respectively) when any occurrence $x \in \Pi_1$ (Π_2, respectively) is replaced by $f(x)$ ($f^{-1}(x)$, respectively).

Given two strings s_1 and s_2, there is a *p-occurrence* of s_1 in s_2 at i, if s_1 p-matches s_2 beginning at the ith position from the left in s_2. The problem of *standard p-string matching* is the following: given the pattern p-string P of length m and the text p-string T of length n, determine all p-occurrences of P in T.

Baker [2] investigated the problem of finding repeated maximal parameterized occurrences of substrings in a string. This is naturally done by an appropriate suffix tree and for this purpose, Baker developed algorithms to build *parameterized suffix trees*. Using the parameterized suffix trees, Baker derived an algorithm for the standard p-string matching problem. Her algorithm essentially takes $O(n(|\Pi| + \log(|\Sigma| + |\Pi|)) + m \log(|\Sigma| + |\Pi|))$ time. Note that while standard string matching can be solved in linear, that is, $O(n+m)$ time, the Baker algorithm for standard p-string matching has an overhead dependent on the sizes of *both* the alphabet sets. This overhead appears as a result of the complexity of constructing the parameterized suffix tree. Idury and Schäffer [6] considered a generalization of the standard p-string matching, namely, dictionary matching under the parameterized pattern matching model. They used a modified Aho–Corasick automaton [1] that, again, has a $\log(|\Sigma| + |\Pi|)$ multiplicative factor.

In this paper, we investigate the alphabet-dependence in the complexity of the standard p-string matching problem. We provide an algorithm for p-string matching that takes time $O(n \log \pi)$, where $\pi = \min(m, |\Pi|)$. Therefore, its complexity is independent of $|\Sigma|$. We further show that $\log \pi$ factor is *inherent* to any algorithm for standard p-string matching in the comparison model. Therefore, our algorithm is optimal.

2 Upper Bound

We start by simplifying Baker's definition of parameterized pattern matching.

Definition 2.1 (**Mapped-Matching**) Let Π be an alphabet set, $T = t_1 \cdots t_n$ the *text* and $P = p_1 \cdots p_m$ the *pattern*, $t_i, p_j \in \Pi$, $\quad i = 1, \ldots, n; j = 1, \ldots, m$. We say that P *mapped-matches* or simply *m-matches* T in location j if $p_i \cong t_{j+i-1}, \quad i = 1, \ldots, m$, where $p_i \cong t_j$ if and only if one of the following two conditions hold:

1. $p_i \neq p_1, \ldots, p_{i-1}$ and $t_j \neq t_{j-i+1}, \ldots, t_{j-1}$.

2. for every $k = 1, \ldots, i-1, \quad p_i = p_{i-k}$ if and only if $t_j = t_{j-k}$.

The *m-matching problem* is to determine all m-matches of P in T. Two p-strings S_1 and S_2 of same length are said to *mapped-match* or simply *m-match* if $S_1[i] \cong S_2[i]$ for all i.

That m-matching is a special case of p-string matching is seen as follows. Define Π in m-matching to be the parameter set in p-string matching. The two conditions in the definition of m-matching essentially ensure that there exists a bijection between the symbols from Π in the pattern and overlapping text, when they p-match. (Our definition of this mapping is in a computationally suitable form, as will be evident shortly.) Intuitively, our definition of m-matching is a projection of p-string matching on to p-strings consisting of the parameter symbols alone. Correspondingly, the matching captures only the notion of one-to-one mapping between the parameter symbols. The notion of matching fixed symbols is absent.

Since m-matching is a special case of p-string matching, it can be solved by any algorithm for p-string matching. The following lemma shows that the opposite is also true.

Lemma 2.2 There is a linear-time reduction from the standard p-string matching problem to the m-matching problem.

Proof: Let, T, P be a text and pattern over alphabets Σ and Π, as defined in the standard p-string matching problem. Let $a \notin \Sigma, \quad b \notin \Pi$. Define strings T', T'' as follows:

$$T'[i] = \begin{cases} t_i & \text{if } t_i \in \Sigma; \\ a & \text{if } t_i \notin \Sigma. \end{cases}$$

$$T''[i] = \begin{cases} t_i & \text{if } t_i \in \Pi; \\ b & \text{if } t_i \notin \Pi. \end{cases}$$

Define P' and P'' similarly to T' and T'', respectively.

The strings T' and P' are over alphabet set $\Sigma' = \{a\} \cup \Sigma$. Solve the *standard string matching problem* for T' and P' by any $O(n)$ time algorithm (e.g. [7]). Let S_1 be all locations of T' where P' matches. The strings T'' and P'' are p-strings over $\Sigma'' = \phi$ and parameter alphabet set $\Pi'' = \{b\} \cup \Pi$. Solve the *m-matching problem* for T'' and P'' and let S_2 be all locations of T'' where there is a p-appearance of P''. We claim: $S_3 = S_1 \cap S_2$ is the set of all locations of T where P p-matches in the standard p-string matching problem. To prove this claim, we must show that there exists a matching bijection iff the two conditions for m-matching hold. One direction is trivial: if there is a p-matching bijection on the characters at each matching location then there is an m-matching over the parameter characters.

We now show, by induction, that if both m-matching conditions are satisfied, then there is a parameter character bijection. Suppose we have an m-match at position i of the text. Clearly, then we have an m-match of any prefix of the pattern at position i. How suppose that for the kth prefix of the pattern, we have defined a bijection f_k, between the parameter characters in $T_i, \ldots, T_{i+k-1}$ and $P_1, \ldots, P_k$. Now we have two cases. If P_{k+1} has not occured before, then we extend f_k to f_{k+1}

be mapping P_{k+1} to T_{i+k}. By condition (1) of m-matching, f_{k+1} is also a bijection. The other case is that P_{k+1} occured most recently at position k' of P. Then $f_k(P_{k'}) = T_{i+k'-1}$, by induction, and by condtion (2), we know that $T_{i+k} = T_{i+k'-1}$, thus we can set $f_{k+1} = f_k$. ▮

We modify the KMP algorithm to solve the m-matching problem simply by replacing every equality comparison "$x = y$" by "$x \cong y$".

Implementation of "$x \cong y$"

Construct table $A[1], \ldots, A[m]$ where $A[i] =$ the largest k, $1 \leq k < i$, such that $p_k = p_i$. If no such k exists then $A[i] = i$.

The following subroutines compute "$p_i \cong t_j$" for $j \geq i$, and "$p_i \cong p_j$" for $j \leq i$.

```
Compare(p_i, t_j)
    if A[i] = i and t_j ≠ t_{j-1}, ..., t_{j-i+1} then return equal
    if A[i] ≠ i and t_j = t_{j-i+A[i]} then return equal
    return not equal
end

Compare(p_i, p_j)
    if i - A[i] < j - 1 and p_j ≠ p_1, ..., p_{j-1} then return equal
    if i - A[i] ≥ j - 1 and p_j = p_{j-i+A[i]} then return equal
    return not equal
end
```

Theorem 2.3 The m-matching problem, and therefore, the standard p-string matching problem, can be solved in $O(n \log \pi)$ time, where $\pi = \min(m, |\Pi|)$.

Proof: We can assume that $n = 2m$, since we can break the text up into overlapping pieces of length $2m$. Then we will find any match of P in T within some such segment.

The table A can be constructed in $O(m \log \pi)$ time as follows: scan the pattern left to right keeping track of the distinct symbols from Π in the pattern in a balanced tree, along with the last occurrence of each such symbol in the portion of the pattern scanned thus far. When the symbol at location i is scanned, look up this symbol in the tree for the immediately preceding occurrence; that gives $A[i]$. Again, Compare can clearly be implemented in time $O(\log \pi)$ – for this, the immediately preceding occurrence of each text symbol from Π is maintained as above while scanning the text left to right. Therefore, the automaton construction in KMP algorithm with replacing every equality comparison "$x = y$" by "$x \cong y$" takes time $O(m \log \pi)$ and the text scanning takes time $O(n \log \pi)$, giving a total of $O(n \log \pi)$ time.

As for the correctness of our algorithm, we only need to show that the failure link in automaton node i produces the largest prefix of $p_1 \cdots p_i$ that m-matches the suffix of $p_1 \cdots p_i$. Our implementation of Compare(p_i,p_j) assures this by preserving the following invariant: *The largest prefix of $p_1 \cdots p_{i+1}$ that m-matches the suffix of $p_1 \cdots p_{i+1}$ is the largest prefix $p_1 \cdots p_k$ of $p_1 \cdots p_i$ that m-matches $p_{i-k+1} \cdots p_i$ and also satisfies $p_{k+1} \cong p_{i+1}$.* ▮

3 Lower Bound

In the preceding section, we derived a simple algorithm for the standard p-string matching problem that was independent of the Σ, but dependent on Π. In this section, we show that the $\log \pi$ factor in the complexity of our algorithm is *inherent* to the complexity of any comparison based algorithm for the standard p-string matching algorithm. We do this by showing a reduction to the standard p-string matching problem from the *Element Distinctness Problem* defined as follows: Given set S of n real numbers, decide if every number in S is distinct.

Lemma 3.1 The element distinctness problem is reducible to the standard p-string matching problem in linear time.

Proof: Let S contain n elements $a_1, a_2, \ldots, a_n$. First check if a_1 is a unique element; this can be done in $O(n)$ time. Assume that it is unique. Then set $T = a_1 a_2 \cdots a_n$ and set $P = a_2 a_3 \cdots a_n a_1$, that is, T is obtained by concatenating the symbols in S and P is obtained by a cyclic shift of T by one position anticlockwise. Let $\Sigma = \phi$ and $\Pi = S$, that is, T and P are p-strings over parameter alphabet set S. We claim that P p-matches T if and only if all elements of S are unique.

Assume P p-matches T. We prove our claim by induction on the prefixes of P. We know, by checking, that a_1 is unique. Now suppose all a_i are unique, for $i < k$. Then in particular a_{k-1} is unique. But in the p-matching of P with T, a_{k-1} p-matched a_k. So a_k must be unique since otherwise the next occurrence of a_k in P would mismatch.

Now assume that all elements of A are unique. Then we trivially get a p-match of P in T. ▮

Essentially as corollaries of this reduction, the theorems below follow.

Theorem 3.2 Any algorithm that solves the standard p-string matching problem over an unbounded set Π, takes time $\Omega(n \log |\Pi|)$ on the comparison model.

Proof: In the comparison model, it is a folklore result that element distinctness problem over n elements requires $\Omega(n \log n)$ time. The theorem follows from the reduction in Lemma 3.1. ▮

Note that the lower bound in the preceding theorem holds only for Π of unbounded size. If Π were, say, polynomial in n and m, this lower bound does not hold; in fact, our algorithm takes $O(n)$ time in this case since by utilizing an array of linear size can be used in place of the binary tree, Compare can be performed in $O(1)$ time. For Π of unbounded size, Theorem 3.2 can be extended to any algebraic model or to even randomized algorithm, by utilizing appropriate results for the element distinctness problem.

Theorem 3.3 Let $n = 2m$. For any algorithm that solves the standard p-string matching problem on a comparison-based branching program in time T and space S, $TS = \Omega(m^{2-\epsilon(m)})$ where $\epsilon(m) = O(1/(\log m)^{1/2})$.

Proof: This follows from the reduction in Lemma 3.1 and the fact that for any algorithm that solves the element distinctness problem on a comparison-based branching program in time T and space S, $TS = \Omega(m^{2-\epsilon(m)})$ where $\epsilon(m) = O(1/(\log m)^{1/2})$. The reduction in Lemma 3.1 is performed on the comparison-based branching program model (See [3] for the details of the model). To obtain a comparsion branching program for the element distinctness program, consider the program for the standard p-string matching problem. Add a path of length $m-1$ at the top, corresponding to the comparison of a_1 with each of the other elements in the text string. In the process, the capacity S of the branching program and the length T of the longest path in it, increase by $\log(m-1)$ and $m-1$ respectively. But for any comparison branching program for the standard p-string matching problem, $S \geq \log m$ and $T \geq m$; therefore, S and T remain asymptotically unchanged. That completes the reduction. ▮

In contrast to this theorem, standard string matching is known to be performable using time T and space S, such that $TS = O(m)$ [5].

4 Conclusions

For the standard p-string matching problem, we have derived an algorithm whose complexity is independent of $|\Sigma|$, the size of the set of fixed symbols in the p-strings. We have also demonstrated that the factor of $\log |\Pi|$ in the complexity of our algorithm corresponding to the set of parameter symbols in Π, is inherent in general in any algorithm. This we do by a reduction from the Element Distinctness Problem to the standard p-string matching problem. As a corollary of this reduction, a near-quadratic time-space tradeoff follows for the standard p-string matching problem.

References

[1] A.V. Aho and M.J. Corasick. Efficient string matching. *C. ACM*, 18(6):333–340, 1975.

[2] B. S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proc. 25th STOC*, pages 71–80. ACM, May 1993.

[3] A. Borodin, F. Fich, F. Meyer auf der Heide, E. Upfal, and A. Wigderson. A time-space tradeoff for element distinctness. *SIAM J. Computing*, 16(1):97–99, 1987.

[4] R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Comm. ACM*, 20:762–772, 1977.

[5] Z. Galil and J.I. Seiferas. Time-space-optimal string matching. *J. Computer and System Science*, 26:280–294, 1983.

[6] R.M. Idury and A.A Schäffer. multiple matching of parameterized patterns. submitted for publication, June 1993.

[7] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Comp.*, 6:323–350, 1977.