On the Decision Tree Complexity of String Matching

Xiaoyu He
Institute of Computing Technology, Chinese Academy of Sciences, China, and
University of Chinese Academy of Sciences, China
hexiaoyu14@mails.ucas.ac.cn

Neng Huang
University of Chinese Academy of Sciences, China
huangneng14@mails.ucas.ac.cn

Xiaoming Sun
Institute of Computing Technology, Chinese Academy of Sciences, China, and
University of Chinese Academy of Sciences, China
sunxiaoming@ict.ac.cn

Abstract

String matching is one of the most fundamental problems in computer science. A natural problem is to determine the number of characters that need to be queried (i.e. the decision tree complexity) in a string in order to decide whether this string contains a certain pattern. Rivest showed that for every pattern p, in the worst case any deterministic algorithm needs to query at least $n - |p| + 1$ characters, where n is the length of the string and $|p|$ is the length of the pattern. He further conjectured that this bound is tight. By using the adversary method, Tuza disproved this conjecture and showed that more than one half of binary patterns are *evasive*, i.e. any algorithm needs to query all the characters (see Section 1.1 for more details).

In this paper, we give a query algorithm which settles the decision tree complexity of string matching except for a negligible fraction of patterns. Our algorithm shows that Tuza's criteria of evasive patterns are almost complete. Using the algebraic approach of Rivest and Vuillemin, we also give a new sufficient condition for the evasiveness of patterns, which is beyond Tuza's criteria. In addition, our result reveals an interesting connection to *Skolem's Problem* in mathematics.

2012 ACM Subject Classification Theory of computation → Oracles and decision trees

Keywords and phrases String Matching, Decision Tree Complexity, Boolean Function, Algebraic Method

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.45

Related Version Full version is available at `https://arxiv.org/abs/1712.09738`

Funding This work was supported in part by the National Natural Science Foundation of China Grant 61433014, 61502449, 61602440, and the 973 Program of China Grants No.2016YFB1000201.

1 Introduction

The string matching problem is one of the most fundamental problems in computer science. The goal of string matching problem is to find one or all occurrences of a pattern in an input string. Lots of efficient algorithms have been discovered in the 20th century. For example, the KMP algorithm [7], discovered by Knuth, Morris and Pratt, is able to locate all occurrences of a pattern of length m in a string of length n in $O(n+m)$ time. This is essentially the best

possible since every algorithm needs $\Omega(n+m)$ time to process the input. Another elegant algorithm is the Karp-Rabin algorithm [6], which uses hashing and can be used to search for a set of patterns. A detailed treatment of these algorithms can be found in [3]. However, the problem becomes subtler when we adopt a different complexity measure, which is the number of characters that the algorithm has to examine in the input string given the prior knowledge of the pattern string. When we confine the alphabet to $\{0,1\}$, this measure is exactly the *decision tree complexity* of boolean string matching problem. Recall that for a binary function f, its decision tree complexity is the number of bits that we have to examine in the worst case in any input x in order to compute $f(x)$.

1.1 Notations and Previous Work

Let p be a pattern over alphabet Σ with $|\Sigma| = \sigma$. Throughout the paper, let $p[i]$ be the i-th character in p and $p[i..j]$ be the substring of p indexed from i to j. Let A_p be a deterministic string searching algorithm which searches for p in any given string s. Following Rivest [9], we denote $w(A_p, n)$ to be the maximum number of characters that A_p examines for any s of length n. Let $D_p(n) = \min_{A_p} w(A_p, n)$, where A_p is taken over all deterministic string searching algorithms. When the alphabet $\Sigma = \{0,1\}$, D_p is exactly the boolean decision tree complexity of string searching algorithm with pattern p. It is clear that this function is monotone, since we can simply add some redundant characters at the end of the searched text. We state this as the following proposition.

▶ **Proposition 1.** *For every pattern p and $n \in \mathbb{N}$, $D_p(n) \leq D_p(n+1)$.*

We define the evasiveness of a pattern as follows.

▶ **Definition 2.** A pattern p is called *evasive* if there exists $N_0 \in \mathbb{N}$ such that for all $n > N_0$, $D_p(n) = n$.

By this definition, a pattern p is evasive if for every algorithm A and every sufficiently large n, there is a string s of length n such that A has to query every character of s in order to determine whether s contains p as a substring. We are interested in determining what patterns are evasive and what patterns are not.

Let $|p|$ denote the length of a pattern p. Rivest gave the following linear lower bound on $D_p(n)$:

▶ **Theorem 3** ([9])**.** *For every pattern p, $D_p(n) \geq n - |p| + 1$ for all $n \in \mathbb{N}$.*

To prove this theorem, Rivest showed that for every $n \in \mathbb{N}$, there exists an integer i between 0 and $|p|$ such that $D_p(n+i) = n+i$, then combined with Proposition 1, Theorem 3 follows. Based on this result, we define *non-evasiveness* as follows.

▶ **Definition 4.** A pattern p is called *non-evasive* if for every $N_0 \in \mathbb{N}$ there exists $n > N_0$ such that $D_p(n) = n - |p| + 1$.

As discussed above, what Rivest proved in fact implies that it is impossible for a pattern to achieve the lower bound in Theorem 3 on consecutive integers, which is the reason we define non-evasiveness in this way. Rivest showed that the pattern $p = 1^k$ (and therefore 0^k) is non-evasive. He further conjectured that all patterns are non-evasive. However, this conjecture was later disproved by Tuza in [11]. We briefly summarize Tuza's work here. Given a string b, let $BE(b)$ denote the set of patterns prefixed and suffixed by b, but other than b. Also, for patterns u and v, let uv denote their concatenation. If $p \in BE(b)$, then let $p(b)$ be the string ubv where, $ub = bv = p$. Tuza proved the following result.

▶ **Theorem 5** ([11]). *Let $p \in BE(b)$. If*

1. *$p(b)$ does not contain a substring p' of length $|p|$ other than prefix or suffix of $p(b)$ such that p' and p differ from each other in at most two characters, and*
2. *the pattern pp does not contain a substring p' of length $|p|$ other than prefix or suffix of pp such that p' and p differ from each other in at most four characters,*

and $n \geq |p|(2|p| - |b|)/\gcd(|p|, |b|)$, then $D_p(n) \geq n - k$, where $k = n \mod \gcd(|p|, |b|)$.

If a pattern string p satisfies the conditions in Theorem 5 and $\gcd(|p|, |b|) = 1$, then one would have $D_p(n) = n$ for all sufficiently large n. This implies that p is evasive and therefore serves as a counterexample of Rivest's conjecture. Tuza estimated the proportion of pattern strings which satisfy the conditions in Theorem 5 and proved that when $\Sigma = \{0, 1\}$, there exists more than $0.5061 \cdot 2^m$ evasive patterns of length m.

Beyond the worst case complexity, the average-case complexity has also been studied previously, that is, finding out the numbers of characters that need to be examined on average assuming that the input string is sampled from the uniform distribution. Yao [12] showed that, for almost all patterns of large enough length m, an algorithm needs to examine $\Theta(\frac{n \log_q m}{m})$ characters on a uniformly random input string of length $n > 2m$, here q is the size of the alphabet.

1.2 Our Contributions

In this paper we settle the decision tree complexity for almost every string except an $o(1)$ fraction. More precisely, we prove that Tuza's lower bound, which is developed combinatorially, is in fact tight for almost every string, by showing an algorithm which achieves this lower bound. This algorithm is based on the periods of the pattern string.

▶ **Definition 6** (Periods). Let p be a pattern of length m and k be a positive integer no larger than m. We say that p is k-periodic, or p has a period k, if $p[i] = p[i + k]$ for all $1 \leq i \leq n - k$. Let $\mathbf{Period}(p) = \{k | p \text{ is } k\text{-periodic}\}$ be the set of all periods of p.

The definition here is the same as in [4], in which it was used to develop a time-space optimal algorithm. A similar idea can also be found in [11]. For set $S \subset \mathbb{N}$, let $\gcd(S)$ denote the greatest common divisor of all elements in S. Here is our main theorem.

▶ **Theorem 7** (Main). *Let p be a pattern of length m and $c = \gcd(\mathbf{Period}(p))$ be the greatest common divisor of all p's periods, then $D_p(n) = n - (n \mod c)$, except for an $O(m^5\sigma^{-m/2})$ fraction of patterns.*

Here, the fraction of patterns is computed in the following way. We first fix a pattern length m, and then count the number of patterns of length m that satisfy some certain properties, then compute its ratio to the total number of length-m patterns, which is σ^m. We then investigate the asymptotic behavior of this ratio as m goes to infinity.

By Theorem 7, the fraction of patterns whose decision tree complexity we don't know goes to 0 as the pattern length goes to infinity.

Besides this result, we also use the algebraic approach to show the evasiveness of certain family of patterns, for which Tuza's method does not work. This algebraic approach was first developed by Rivest and Vuillemin [10], and we extend it to our problem. Interestingly, we find that this approach reveals a relation between our problem and the *Skolem's Problem*. We also define the *characteristic polynomial* of a pattern, which is again closely related to the pattern's periodic behaviors. This polynomial, besides its application in this problem, is of independent interest on its own.

2 Upper Bounds

In this section, we prove one direction of Theorem 7, which can be stated as the following lemma.

▶ **Lemma 8.** *Let p be a pattern of length m and $c = \gcd(\mathbf{Period}(p))$ be the greatest common divisor of all p's periods, then $D_p(n) \leq n - (n \mod c)$.*

To show this lemma, we will develop an algorithm whose behavior depends on the periods of the pattern string.

2.1 Non-evasiveness of Bifix-free Patterns

We first look at the simple case where our pattern is bifix-free.

▶ **Definition 9.** A string s is called a *bifix* of a string t if s is both a prefix and a suffix of t. A pattern p is called *bifix-free* if p has no bifix other than itself.

▶ Remark (Relations to combinatorics on words). The concepts of periods and bifixes are also studied in the field of combinatorics on words under possibly different names. Bifixes are usually referred to as *borders* in combinatorics on words, and bifix-free strings are usually called *unbordered words*. For more details from viewpoint of combinatorics on words, see [2].

Bifix-free patterns have the following property in terms of periods.

▶ **Lemma 10.** *A pattern p of length n has a bifix of length $k < n$ if and only if it is $(n-k)$-periodic. Furthermore, p is bifix-free if and only if it has only one period, which is n.*

Proof. If a pattern p has a period $k < n$, then $p[1..(n-k)] = p[(k+1)..n]$. This is equivalent to say that p has a prefix of length $n-k$ which is equal to p's suffix of length $n-k$. The "furthermore" part follows directly. ◀

Then, for a bifix-free pattern p we have $|p| = \gcd(\mathbf{Period}(p))$. According to Lemma 8, the following result is expected.

▶ **Lemma 11.** *Let p be a bifix-free pattern of length m, then $D_p(n) \leq n - (n \mod m)$ and p is non-evasive.*

Proof. Consider the algorithm in Figure 1. We claim that this algorithm can produce the correct output after $n - (n \mod m)$ queries to the string. Suppose that in Line 11, we find that $s[i..j]$ is not equal to p, otherwise we can stop and output this occurrence. Note that until Line 11 we have only queried m characters in s, which are $s[i], s[i+1], \ldots, s[j]$. We show that for indices l with $1 \leq l \leq m$, we have $s[l..(l+m-1)] \neq p$.

- $1 \leq l < i$. In this case, there exists an index t with $i \leq t \leq l+m-1$ such that $s[t..(l+m-1)]$ is not a suffix of p, since otherwise $s[l+m]$ would not be queried, contradicting the fact that $j = i+m-1 \geq l+m$. And therefore we have $s[l..(l+m-1)]$ is not suffix of p.
- $l = i$. In this case we have by assumption that $s[l..(l+m-1)] = s[i..j] \neq p$.
- $i < l \leq m$. Assume that $s[l..(l+m-1)]$ equals to p. Then for all indices t with $l \leq t \leq j$, $s[l..t]$ is a prefix of p, and therefore by bifix-freeness, is not a suffix of p. However, since $i < l$, $s[l-1]$ is queried, so there must exists such an index t that $s[l..t]$ is a suffix of p, which is a contradiction. Hence $s[l..(l+m-1)]$ does not equal to p.

```
Input: string s of length n, bifix-free pattern p of length m
Output: whether p is a substring of s
function FIND(s, p)
    if n < m then
        return false
    i ← m, j ← m
    query(s[m])
    while j − i ≠ m − 1 do
        if s[i..j] is a suffix of p then
            query(s[i − 1]), i ← i − 1
        else
            query(s[j + 1]), j ← j + 1
    if s[i..j] = p then
        return true
    else
        return FIND(s[m + 1..n], p)
end function
```

Figure 1 Algorithm for bifix-free patterns.

Table 1 The table for the first three significant digits for b_∞^σ when $\sigma \leq 6$.

σ	2	3	4	5	6
b_∞^σ	0.268	0.557	0.688	0.760	0.801

This shows that after querying m characters, we either find an occurrence of p in s, or reduce the size of s by m. When the size of s is smaller than m, the algorithm trivially stops. Therefore after $n - (n \mod m)$ queries, we will be able to determine whether s contains p. This establishes an upper bound on $D_p(n)$, namely $D_p(n) \leq n - (n \mod m)$, which matches Rivest's lower bound. We conclude that bifix-free patterns are non-evasive. ◀

We note that the above algorithm is in fact applicable for all finite alphabets. For an alphabet Σ of size σ, we define b_m^σ to be the proportion of bifix-free strings in strings of length m, that is,

$$b_m^\sigma = \frac{|\{p \in \Sigma^m | p \text{ is bifix-free}\}|}{\sigma^m}, |\Sigma| = \sigma.$$

Nielsen [8] showed that the sequence $\{b_m^\sigma\}_{m=1}^\infty$ converges. Furthermore, he proved that

$$b_\infty^\sigma := \lim_{m\to\infty} b_m^\sigma \geq 1 - \sigma^{-1} - \sigma^{-2}.$$

Table 1 (from [8]) shows the first three significant digits for b_∞^σ when $\sigma \leq 6$.

From this we obtain that more than 26.7% of binary pattern strings of length m are non-evasive, where m is sufficiently large. We also note that, as the size of the alphabet increases, the percentage of patterns that are non-evasive tends to 1.

Algorithm 1 Algorithm for general patterns.

Input: string s of length n, pattern p of length m
Output: whether p is a substring of s

```
1: function FIND(s, p)
2:     if n < m then return false
3:     i ← m, j ← m
4:     query(s[m])
5:     while j − i ≠ m − 1 do
6:         if s[i..j] is a suffix of p then
7:             query(s[i − 1]), i ← i − 1
8:         else
9:             query(s[j + 1]), j ← j + 1
10:    if s[i..j] = p then
11:        return true
12:    l ← m + c
13:    while l ≤ n do
14:        i ← l, j ← l
15:        query(s[l])
16:        repeat
17:            if s[i..j] is a suffix of p then
18:                query(s[i − 1]), i ← i − 1
19:            else
20:                query(s[j + 1]), j ← j + 1
21:        until c new characters have been queried OR j − i = m − 1
22:        if s[(j − m + 1)..j] = p then
23:            return true
24:        l ← l + c
25:    return false
26: end function
```

2.2 The General Case

In the previous section, we used bifix-freeness as a crucial tool in our algorithm. The property stated in Lemma 10 is in fact playing an important role here. It is natural to ask that what if a pattern has periods other than its own length? An intuition is that if a pattern has good periodic behaviors, then a well-behaved algorithm must exist as well. We therefore formalize this intuition and give the proof of Lemma 8.

Proof of Lemma 8. Let's consider the algorithm in Algorithm 1, which is a generalization of the algorithm for bifix-free patterns. Intuitively, this algorithm examines the string by blocks of size c, which is the greatest common divisor of p's periods. Note that for simplicity we formulate this algorithm in a way that it may query the same character more than once. In such cases, we can reuse the previous result and need not really query that character. Our algorithm might also query a character in s with index larger than n. In such cases, we assume that we obtain a character different than $p[m]$, such that it cannot form the pattern p with previous characters. We also assume that $c > 1$.

First of all, it is easy to see that this algorithm queries at most $n - (n \mod c)$ characters in s. We now show that this algorithm returns the answer correctly. Our algorithm only

returns *true* when it really see the pattern p, so it suffices to show that if there are occurrences of p in s, then our algorithm will always be able to find one. Here we prove that it will always find the first occurrence.

Assume that the first occurrence of p in s is $s[k-m+1..k]$ and $k = hc + t$ for some $0 \leq t < c$. We want to show that, when our algorithm starts to examine the (hc)-th character of the string at Line 15 (it could be that our algorithm will be able to locate p in the while loop beginning at Line 5, but that case is even simpler), there are at most c characters in $s[k-m+1..k]$ which have not been queried yet. If this holds, then our algorithm will be able to identify $s[k-m+1..k]$ as p in at most c queries.

In fact, we prove a strong claim that whenever $k-m < l \leq k$, in order for the repeat-until loop at Line 16-21 to stop, we either either query c new characters in the range $s[k-m+1..k]$, or we have queried every character in the range $s[k-m+1..l+t]$.

Suppose our algorithm is going to query a character with index smaller than $k-m+1$ when $k-m < l \leq k$, then at some point our algorithm will query $s[k-m]$ at Line 18. Clearly, $i = k-m+1$ at that moment. Also, it must be that $j = l+t$, for when Line 18 is executed, $s[i..j]$ must be a suffix of p. But we also know that $s[i..j]$ is a prefix of p. Thus the length of $s[i..j]$, which is $j-k+m$, must be a multiple of c, implying that $j = l+t$ (since l is always a multiple of c). If our algorithm do not query a character with index smaller than $k-m+1$ when $k-m < l \leq k$, then in order for the repeat-until loop at Line 16 to end, we have all our c new characters in $s[k-m+1..k]$. This proves what we need, and the correctness of our algorithm follows. ◀

3 Proof of Theorem 7

In this section, we give the proof of our main theorem. We have proved one direction in Section 2. For the other direction, we use a similar analysis to Tuza's in [11]. We first restate (a stronger version of) Tuza's theorem here.

▶ **Theorem 12** ([11]). *Assume that $p \in BE(b_1), p \in BE(b_2), \ldots, p \in BE(b_l)$. If*

1. *for every $1 \leq i \leq l$, $p(b_i)$ does not contain a substring p' of length $|p|$ other than prefix or suffix of $p(b_i)$ such that p' and p differ from each other in at most two characters, and*
2. *the pattern pp does not contain a substring p' of length $|p|$ other than prefix or suffix of pp such that p' and p differ from each other in at most four characters,*

then for sufficiently large n, $D_p(n) \geq n-k$, where $k = n \mod \gcd(\{|p|, |b_1|, |b_2|, \ldots, |b_l|\})$.

We note that in Tuza's language, $p \in BE(b)$ essentially means that p has a bifix b. As is shown in Lemma 10, it is equivalent to say that p is $(|p|-|b|)$-periodic. Thus the condition $p \in BE(b_1), p \in BE(b_2), \ldots, p \in BE(b_l)$ is simply saying that p has periods $|p|-|b_1|, |p|-|b_2|, \ldots, |p|-|b_l|$, other than its own length $|p|$, and the expression $\gcd(\{|p|, |b_1|, |b_2|, \ldots, |b_l|\}$ is equivalent to $\gcd(\mathbf{Period}(p))$. To prove Theorem 7, we need the following two lemmas. These two lemmas are generalizations of Lemma 11 and Lemma 12 in [11].

▶ **Lemma 13.** *Let $B_1(n)$ be the set of patterns p such that $|p| = n$, $p \in BE(b)$ for some b and $p(b)$ contains a substring p' of length n other than prefix or suffix of $p(b)$ such that p' and p differ from each other in at most two characters. Then $|B_1(n)| = O(n^4\sigma^{n/2})$.*

▶ **Lemma 14.** *Let $B_2(n)$ be the set of patterns p such that $|p| = n$ and the pattern pp contains a substring p' of length n other than prefix or suffix of pp such that p' and p differ from each other in at most four characters. Then $|B_2(n)| = O(n^5\sigma^{n/2})$.*

Proof of Theorem 7. Let p be a pattern of length m. If $p \notin B_1(m) \cup B_2(m)$, then by Theorem 12, $D_p(n) \geq n - k$, where $k = n \mod \gcd(\mathbf{Period}(p))$. Also, by Lemma 8, $D_p(n) \leq n - k$. Therefore $D_p(n) = n - k$ for all $p \notin B_1(m) \cup B_2(m)$. By Lemma 13 and Lemma 14, $|B_1(m) \cup B_2(m)| = O(m^5\sigma^{m/2})$, and hence Theorem 7 follows. ◀

For simplicity, from now on we say $p(b)$ *has property 1* if $p \in BE(b)$ and $p(b)$ contains a substring p' of length n other than prefix or suffix of $p(b)$ such that p' and p differ from each other in at most two characters, and we say p *has property 2* if the pattern pp contains a substring p' of length n other than prefix or suffix of pp such that p' and p differ from each other in at most four characters.

3.1 Proofs of the Two Lemmas

Now we prove Lemma 13 and Lemma 14. Tuza proved the case when $\Sigma = \{0, 1\}$ in [11]. We will adapt his proof to handle the case where Σ is any finite alphabet.

▶ **Lemma 15.** *If $p(b)$ has property 1 for some $|b| > |p|/2$, then we can find b' with length at most $|p|/2$ such that $p(b')$ has property 1 as well.*

Proof. If $p \in BE(b)$ for some $|b| > |p|/2$, then by definition, $p[i] = p[i + |p| - |b|]$ for every $1 \leq i \leq |b|$. Assume that $k(|p| - |b|) < |p| \leq (k + 1)(|p| - |b|)$ for some k, then let $b' = p[k(|p| - |b|) + 1..|p|]$. It is straightforward to check that b' is also a bifix of p and $p(b')$ contains $p(b)$ as a substring. Therefore $p(b')$ has property 1 if $p(b)$ has property 1. ◀

Proof of Lemma 13. Let $p \in B_1(n)$. By definition and Lemma 15, for some $|b| \leq |p|/2$, $p(b)$ contains a substring p' of length n other than prefix or suffix of $p(b)$ such that p' and p differ from each other in at most two characters. These at most two characters can be chosen in $(\sigma - 1)^2 n(n - 1)/2 + (\sigma - 1)n + 1$ different ways. Assume that p' starts in the $(i + 1)$-th character in $p(b)$, then after we fix these two erroneous locations, the first $\gcd(i, n - |b|)$ characters in $p(b)$ will uniquely determine $p(b)$. Therefore we have

$$\begin{aligned}
|B_1(n)| &\leq \sum_{|b|=1}^{n/2} \sum_{i=1}^{n-|b|-1} ((\sigma - 1)^2 n(n - 1)/2 + (\sigma - 1)n + 1)\sigma^{\gcd(i,n-|b|)} \\
&\leq n^2\sigma^2 \sum_{|b|=1}^{n/2} \sum_{i=1}^{n-|b|-1} \sigma^{\gcd(i,n-|b|)} \\
&\leq n^2\sigma^2 \cdot \frac{n}{2} \cdot n\sigma^{n/2} \\
&= \frac{n^4}{2}\sigma^{n/2+2}.
\end{aligned}$$

◀

Proof of Lemma 14. The proof is similar to that of Lemma 13. Let $p \in B_2(n)$. Then the pattern pp has a substring p' that differs from p in at most four characters. These at most four characters can be chosen in at most $|\Sigma|^4 n^4$ different ways. Assume that p' starts in the $(i + 1)$-th position, then the first $\gcd(i, n)$ characters in p uniquely determines p. Therefore we have

$$|B_2(n)| \leq \sum_{i=1}^{n-1} \sigma^4 n^4 \sigma^{\gcd(i,n)} \leq \sum_{i=1}^{n-1} \sigma^4 n^4 \sigma^{n/2} \leq n^5\sigma^{n/2+4}.$$

◀

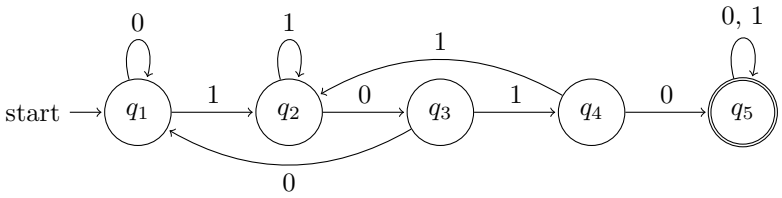

Figure 2 The finite state automaton for pattern $p = 1010$.

4 A Sufficient Condition for Evasiveness

Now that we have finished the proof of Theorem 7, a natural question to ask is what patterns lie outside the scope of Theorem 7? Rivest has given an example in [9], by showing that the pattern 1^n is non-evasive while $\gcd(\mathbf{Period}(1^n)) = 1$ since every integer between 1 and n is a period of 1^n. In this section, we will use the algebraic method to develop a new sufficient condition for evasiveness. We will assume that the alphabet $\Sigma = \{0, 1\}$. We first introduce the notion of characteristic polynomial in Section 4.1 and then state our theorem in Section 4.2. In Section 4.3, we will show the relationship between a pattern's periods and its characteristic polynomial, which allows for a convenient way to calculate the polynomial.

4.1 The KMP Automaton and the Transition Matrix

Following Rivest [9] we will make use of the finite state automaton constructed by the Knuth-Morris-Pratt algorithm. Let p be a pattern string of length m, then the automaton constructed will have $m + 1$ states, where state q_1 is the initial state and state q_{m+1} is the only accepting state. The automaton reaches state q_i if the previous $i - 1$ characters is a prefix of p where i is the largest possible among such ones, and the pattern p is not found already. The automation reaches state q_{m+1} as soon as the pattern p is found, and stays there forever. See Figure 2 for an example of the KMP automaton when the pattern $p = 1010$.

Let $U_p(n, i)$ be the set of strings of length n on which the automaton ends in state q_i. Let $g_p(n, i) := \sum_{s \in U_p(n,i)} x^{wt(s)}$, where $wt(s)$ is the number of 1's in s. The following lemma is used in [10] to show evasiveness of boolean functions.

▶ **Lemma 16** ([10]). *If $D_p(n) \leq n - l$ for some integer $1 \leq l \leq n$, then $(x + 1)^l$ divides $g_p(n, m + 1)$.*

A useful consequence of this lemma is the following corollary.

▶ **Corollary 17.** *If there exists $N_0 \in \mathbb{N}$ such that $g_p(n, m + 1) \not\equiv 0 \mod (x + 1)$ for all $n > N_0$, then $D_p(n) = n$, i.e. p is evasive.*

By Lemma 16, we are only interested in the value of $g_p(n, m + 1)$ modulo $x + 1$. Note that we always have

$$g_p(n + 1, m + 1) = (x + 1)g_p(n, m + 1) + y \cdot g_p(n, m),$$

where y equals 1 or x depending on the last bit of the pattern string. Taking modulus of $(x + 1)$ on both sides, we obtain

$$g_p(n + 1, m + 1) \equiv y \cdot g_p(n, m) \mod (x + 1).$$

Since $y \equiv \pm 1 \mod (x + 1)$ (with the sign determined by the last bit of the pattern string), we obtain the following lemma.

▶ **Lemma 18.** *$g_p(n+1, m+1) \equiv 0 \mod (x+1)$ if and only if $g_p(n, m) \equiv 0 \mod (x+1)$. Moreover, if there exists $N_0 \in \mathbb{N}$ such that $g_p(n, m) \not\equiv 0 \mod (x+1)$ for all $n > N_0$, then p is evasive.*

Now we define the transition matrix. Given a pattern string p of length m, we can express $g_p(n+1, 1), \ldots, g_p(n+1, m)$ in terms of $g_p(n, 1), \ldots, g_p(n, m)$. For example, when $p = 1010$, according to the automata in Figure 2, we can write

$$\begin{pmatrix} g_p(n+1,1) \\ g_p(n+1,2) \\ g_p(n+1,3) \\ g_p(n+1,4) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ x & x & 0 & x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & x & 0 \end{pmatrix} \begin{pmatrix} g_p(n,1) \\ g_p(n,2) \\ g_p(n,3) \\ g_p(n,4) \end{pmatrix}$$

Since we are only interested in these values modulo $x+1$, we may plug in $x = -1$ into all these terms. We denote $\overline{g}_p(n, i)$ to be the value obtained by plugging $x = -1$ into $g_p(n, i)$. In the previous example where $p = 1010$, we will obtain

$$\begin{pmatrix} \overline{g}_p(n+1,1) \\ \overline{g}_p(n+1,2) \\ \overline{g}_p(n+1,3) \\ \overline{g}_p(n+1,4) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ -1 & -1 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} \overline{g}_p(n,1) \\ \overline{g}_p(n,2) \\ \overline{g}_p(n,3) \\ \overline{g}_p(n,4) \end{pmatrix}$$

We call the matrix on the right hand side of the above equation the *transition matrix* of the pattern string $p = 1010$. In general, given a pattern string p, we write down the recurrence relation of $g_p(n, i)$ in the matrix form and plug in $x = -1$, and the resulting matrix will be our transition matrix. Let T_p denote the transition matrix for pattern p.

We will see later that the eigenvalues of T_p are of great use to us. We establish the following lemma using the characteristic polynomial of T_p. In the remaining part of this paper we will refer to the characteristic polynomial of T_p as characteristic polynomial of the pattern p.

▶ **Lemma 19.** *Let p be a pattern of length m. Let $P(\lambda) = \lambda^m + c_{m-1}\lambda^{m-1} + \cdots + c_0$ be the characteristic polynomial of p, then we have the recurrence relation*

$$\overline{g}_p(n+m, m) + c_{m-1}\overline{g}_p(n+m-1, m) + \cdots + c_0\overline{g}_p(n, m) = 0. \tag{1}$$

Proof. By the Cayley-Hamilton theorem (see Theorem 5.2.3 in [1]), we have

$$T_p^m + c_{m-1}T_p^{m-1} + \cdots + c_0 I = 0,$$

where I is the identity matrix. Right multiply both sides by column vector

$$\overline{g}_p(n) = (\overline{g}_p(n,1),\ \overline{g}_p(n,2),\ \ldots,\ \overline{g}_p(n,m))^t,$$

we obtain

$$\overline{g}_p(n+m) + c_{m-1}\overline{g}_p(n+m-1) + \cdots + c_0\overline{g}_p(n) = 0,$$

since $T_p\overline{g}_p(n) = \overline{g}_p(n+1)$. Both sides of the equation above are m-dimensional column vectors, and we get the desired recurrence relation by looking at the last row of both vectors. ◀

4.2 The Skolem Problem and Finite Zeroes

Lemma 19 gives us a tool to get around $\overline{g}_p(n,1),\ldots,\overline{g}_p(n,m-1)$ and focus only on $\overline{g}_p(n,m)$. Now we are faced with the following problem:

> Let $\{u_n\}$ be a linear recurrent sequence. Does there exist N_0 such that u_n is non-zero for all $n > N_0$?

This problem is very similar to the *Skolem's Problem*, which can be stated as follows:

> Let $\{u_n\}$ be a linear recurrent sequence. Does there exist n such that $u_n = 0$?

For a detailed survey of the Skolem's problem, readers are referred to [5]. We will use the following result from [5], which partially solved our problem.

▶ **Lemma 20** ([5]). *Assume sequence* $\{u_n\}_{n=1}^{\infty}$ *satisfies*

$$u_n = a_{m-1}u_{n-1} + \cdots + a_1 u_{n-m+1} + a_0 u_{n-m},$$

where $a_0, a_1, \ldots, a_{m-1}$ *are fixed integers. Also assume that* $p(\lambda) = \lambda^m - a_{m-1}\lambda^{m-1} - \cdots - a_1\lambda - a_0$ *has the decomposition*

$$p(\lambda) = (\lambda-\lambda_1)^{m_1}(\lambda-\lambda_2)^{m_2}\cdots(\lambda-\lambda_r)^{m_r},$$

where $\lambda_1,\ldots,\lambda_r \in \mathbb{C}$ *are distinct roots of* $p(\lambda)$ *and* $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_r|$. *Then there exists* $N_0 \in \mathbb{N}$ *such that* u_n *is non-zero for all* $n > N_0$ *if one of the following cases holds:*

1. $|\lambda_1| > |\lambda_2|$.
2. $|\lambda_1| = |\lambda_2| > |\lambda_3|,\ \lambda_1 = \overline{\lambda_2}$.
3. $|\lambda_1| = |\lambda_2| = |\lambda_3| > |\lambda_4|,\ \lambda_1 \in \mathbb{R},\ \lambda_2 = \overline{\lambda_3}$.

The proof of this lemma can be found in Proposition 4.1 in [5]. Note that our statement is a little bit different. In [5] it is proved that the Skolem's problem is decidable in these cases, by showing that there exists an algorithmically computable constant N_0 such that $u_n \neq 0$ for all $n \geq N_0$, and therefore an algorithm for deciding the Skolem's problem only needs to check whether there are zeroes below the bound N_0.

Using this lemma, we can show the evasiveness of some pattern strings. As an example, we prove the following proposition.

▶ **Proposition 21.** *The pattern* $p = 10^k1$ *is evasive when* $k > 0$.

Proof. To begin with, we calculate its characteristic polynomial, which is

$$p(\lambda) = \det(\lambda I - T_p) = \lambda^{k+2} - \lambda + 1.$$

We note that this is also the characteristic polynomial for the recurrence of $\overline{g}(n, k+2)$ (here $|p| = k+2$). Now assume $z = re^{i\theta}$ is a root of $p(\lambda) = 0$. Then we have $|z^{k+2}| = |z-1|$, which implies

$$r^{k+2} = \sqrt{r^2 + 1 - 2r\cos\theta}.$$

This shows that for every r the value of $\cos\theta$ is determined, and therefore there can be at most 2 choices of θ. Thus, the pattern p either satisfies condition 1 or condition 2 in Lemma 20. We conclude that p is evasive. ◀

▶ Remark. The evasiveness of pattern $p = 10^k1$ is not covered by Tuza's Theorem. Though $p \in BE(b)$ where $b = 1$, the pattern 10^k110^k1 has a substring $p' = 0^k11$ which differs from p in only two positions, and thus violates the condition (b) in Theorem 5.

4.3 The Characteristic Polynomial and Periods

Writing down the characteristic polynomial through the transition matrix can sometimes be inefficient. Here we develop a faster way to calculate a pattern's characteristic polynomial and show some interesting connection to the periodic behavior of the pattern.

We give the following formula for a pattern's characteristic polynomial in terms of the pattern's periods. The proofs of results in this subsection can be found in the full version of this paper.

▶ **Theorem 22.** *Let p be a pattern of length m. Let $P(\lambda)$ be the characteristic polynomial of p, then we have $P(\lambda) = \lambda^m + c_{m-1}\lambda^{m-1} + \cdots + c_1\lambda + c_0$, where for $1 \le k \le m$,*

$$c_{m-k} = \begin{cases} (-1)^{wt(p[1..k])}, & \text{if } k \text{ is a period of } p, \\ 0, & \text{otherwise.} \end{cases}$$

In proving the above theorem, the following two lemmas will be useful.

▶ **Lemma 23.** *Let p be a pattern of length m. Assume that state q_m of the KMP automaton for p has a transition back to state q_{m-k+1} where $k \le m$.*

- *If $k = m$, then all patterns of $p[1..m-1]$ are preserved. That is to say, if $p[1..m-1]$ is l-periodic for some l, then $p[1..m]$ is also l-periodic.*
- *If $k < m$, then k is the smallest of the periods of $p[1..m-1]$ which are destroyed. That is to say, $p[1..m-1]$ is k-periodic while $p[1..m]$ is not k-periodic, and furthermore, if $p[1..m-1]$ is l-periodic for some $l < k$, then $p[1..m]$ is also l-periodic.*

▶ **Lemma 24.** *Let p be a pattern of length m. Let $P_i(\lambda)$ be the characteristic polynomial of the pattern $p[1..i]$. Assume that state q_m of the KMP automaton for p has a transition back to state q_{m-k+1} where $1 \le k \le m$. Then*

$$P_m(\lambda) = \begin{cases} \lambda P_{m-1}(\lambda) - (-1)^{wt(p[1..k])} P_{m-k}(\lambda), & \text{if } k < m, \\ \lambda P_{m-1}(\lambda) + (-1)^{wt(p[1..m])}, & \text{if } k = m. \end{cases}$$

5 Conclusions

In this paper, we determined the decision tree complexity of string matching problem for almost every string, except for those string the adversary method fails to give a lower bound, whose fraction is negligible.

The algebraic approach in Section 4 further proves that a few of these strings are evasive. One open problem is to resolve the remaining cases.

The characteristic polynomial of a pattern p, which we encountered in Section 4.3, might be of independent interest itself. We have shown that this polynomial is related to the pattern's periodic behaviour, and it will be interesting to investigate whether other properties of strings can be related to it.

Another natural extension is to consider randomized algorithms. All algorithms proposed in this paper are deterministic, and randomized complexity is still widely open.

References

1 M. Artin. *Algebra*. Pearson Prentice Hall, 2011.

2 J. Berstel and J. Karhumäki. Combinatorics on words - a tutorial. *Bulletin EATCS*, pages 178–228, February 2003.

3 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 2001.

4 Z. Galil and J. Seiferas. Time-space-optimal string matching. *Journal of Computer and System Sciences*, 26(3):280–294, 1983. doi:10.1016/0022-0000(83)90002-8.

5 V. Halava, T. Harju, M. Hirvensalo, and J. Karhumäki. Skolem's problem - on the border between decidability and undecidability. *TUCS Technical Reports 683*, 2005.

6 R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.

7 D. Knuth, J. Morris, and V. Pratt. Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6(2):323–350, jun 1977. doi:10.1137/0206024.

8 P. Nielsen. A note on bifix-free sequences (Corresp.). *IEEE Transactions on Information Theory*, 19(5):704–706, 1973. doi:10.1109/TIT.1973.1055065.

9 R. L. Rivest. On the Worst-Case Behavior of String-Searching Algorithms. *SIAM Journal on Computing*, 6(4):669–674, 1977. doi:10.1137/0206048.

10 R. L. Rivest and J. Vuillemin. A Generalization and Proof of the Aanderaa-Rosenberg Conjecture. In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, STOC '75, pages 6–11, New York, NY, USA, 1975. ACM. doi:10.1145/800116.803747.

11 Z. Tuza. Worst-case behavior of string-searching algorithms. *Journal of Statistical Planning and Inference*, 6(1):99–103, 1982. doi:10.1016/0378-3758(82)90060-X.

12 A. Yao. The Complexity of Pattern Matching for a Random String. *SIAM Journal on Computing*, 8(3):368–387, 1979. doi:10.1137/0208029.