Simple KMP Pattern-Matching on Indeterminate Strings

Neerja Mhaskar and W. F. Smyth

Neerja Mhaskar

Dept. of Computing and Software, McMaster University, Canada

Prague Stringology Conference, August 31 – September 2, 2020
Outline

- Introduction
- Encoding
- KMP algorithm
- KMP style algorithm for indeterminate strings
- Open problems
Introduction

Given a fixed finite alphabet $\Sigma = \{\lambda_1, \lambda_2, \ldots, \lambda_\sigma\}$.

A **regular letter**, also called a **character**, is any single element of $\Sigma$.

For example, for the DNA alphabet $\Sigma_{DNA} = \{a, c, g, t\} = a, c, g, t$ are all regular letters.

An **indeterminate letter** is any subset of $\Sigma$ of cardinality greater than one.

Some examples of an indeterminate letter over $\Sigma_{DNA} = \{a, c, g, t\}$ are $\{a, c\}$, $\{a, g, t\}$, and $\{a, c, g, t\}$. 
Introduction

A regular string $x = x[1..n]$ on $\Sigma$ is an array of regular letters drawn from $\Sigma$.

An indeterminate string $x[1..n]$ on $\Sigma$ is an array of letters drawn from $\Sigma$, of which at least one is indeterminate.

Whenever entries $x[i]$ and $x[j]$, $1 \leq i, j \leq n$, both contain the same character (possibly other characters as well), we say that $x[i]$ matches $x[j]$ and write $x[i] \approx x[j]$. 
Encoding for Indeterminate Strings

- We propose a new encoding for indeterminate strings using prime numbers and the GCD operation.

- We make use of a mapping $f : \Sigma \rightarrow P$, where $P$ is the set of the first $|\Sigma| = \sigma$ prime numbers, such that each element of $\Sigma$ uniquely maps to an element of $P$.

For example, for $\Sigma_{DNA} = \{a, c, g, t\}$, a possible mapping is $f : a \rightarrow 2, c \rightarrow 3, g \rightarrow 5, t \rightarrow 7$. 
Then given $x = x[1..n]$ on $\Sigma$ (the source string), we apply the mapping $f$ to compute $y = y[1..n]$ (the mapped string) according to the following rule:

**(R)** For every $x[i] = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$, $1 \leq k \leq \sigma$, $1 \leq i \leq n$, where $\lambda_h \in \Sigma$, $1 \leq h \leq k$, set

$$y[i] \leftarrow \prod_{h=1}^{k} f(\lambda_h), \text{ where } \lambda_h \in x[i].$$

For example, consider a source string $x = a\{a, c\}g\{a, t\}t\{c, g\}$, over $\Sigma_{DNA}$, and $\sigma = 4$. Let the mapping be $f : a \rightarrow 2, c \rightarrow 3, g \rightarrow 5, t \rightarrow 7$.

Applying Rule (R) for $1 \leq k \leq 4$, we compute the mapped string $y = 2/6/5/14/7/15$. 
The mapping $f$ and Rule (R) allows an ordering on the indeterminate letters drawn from $\Sigma$.

For example, for the above example and mapping,

$$a = 2 < g = 5 < \{a, c\} = 6 < t = 7 < \{a, t\} = 14 < \{c, g\} = 15.$$

On the other hand, for the same example, a different mapping (say, $f : t \rightarrow 2, c \rightarrow 3, a \rightarrow 5, g \rightarrow 7$) yields $y = 5/15/7/10/2/21$ with a quite different ordering,

$$t = 2 < a = 5 < g = 7 < \{a, t\} = 10 < \{a, c\} = 15 < \{c, g\} < 21.$$
Lemma (1)

If \( y \) is computed from \( x \) by Rule (R), then for every \( i_1, i_2 \in 1..n \), \( x[i_1] \approx x[i_2] \) if and only if \( \gcd(y[i_1], y[i_2]) > 1 \).

Two strings \( x_1 \) and \( x_2 \) of equal length \( n \) are said to be **isomorphic** if and only if for every \( i, j \in \{1, \ldots, n\} \),

\[
x_1[i] \approx x_1[j] \iff x_2[i] \approx x_2[j].
\]  

(1)

We thus have the following observations:

Observation (1)

If \( x \) is an indeterminate string on \( \Sigma \), and \( y \) is the numerical string constructed by applying Rule (R) to \( x \), then \( x \) and \( y \) are isomorphic.

Observation (2)

By virtue of Lemma 1 and (1), \( y \) can overwrite the space required for \( x \) (and vice versa) with no loss of information.
Observation (3)

Suppose $\ell_1$ and $\ell_2$ are integers representable in at most $B$ bits. Then $\gcd(\ell_1, \ell_2)$ can be computed in $O(M_B \log B)$ time, where $M_B$ denotes the maximum time required to compute $\ell_1 \ell_2$ over all such integers.

Then for example when $\sigma = 4$, corresponding to $\Sigma_{DNA}$, $2 \times 3 \times 5 \times 7 = 210 < 256$, and so $B = 8$ and the matching time is $O(M_8 \log 8) = O(3M_8)$. Similarly for $\sigma = 9$ the time required to match any two indeterminate letters is $O(5M_{32})$.

Observation (4)

We assume therefore that, for $\sigma \leq 9$, computing a match between $x[i_1]$ and $x[i_2]$ on $\Sigma$ (that is, between $y[i_1]$ and $y[i_2]$ computed using Rule (R)) requires time bounded above by a (small) constant.
Pattern matching in strings

A **border array** \( \beta_x = \beta_x[1..n] \) of \( x \) is an integer array where for every \( i \in [1..n] \), \( \beta_x[i] \) is the length of the longest border of \( x[1..i] \).

A **prefix array** \( \pi_x = \pi_x[1..n] \) of \( x \) is an integer array where for every \( i \in [1..n] \), \( \pi_x[i] \) is the length of the longest substring starting at position \( i \) that matches a prefix of \( x \).

<table>
<thead>
<tr>
<th>1 2 3 4 5 6 7 8 9 10 11 12 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>x a a b a a b a a a {a, b} b a a a {a, c}</td>
</tr>
<tr>
<td>( \beta_x ) 0 1 0 1 2 3 4 5 6 3 4 5 2</td>
</tr>
<tr>
<td>( \pi_x ) 13 1 0 6 1 0 3 5 1 0 2 2 1</td>
</tr>
</tbody>
</table>

**Figure 1:** Border array \( \beta_x \) and prefix array \( \pi_x \) computed for the string \( x = aabaabaa\{a, b\}baa\{a, c\} \).
Lemma ([AHU74])

The border array and prefix array of a regular string of length $n$ can be computed in $O(n)$ time.

Lemma ([Smy03, SW08])

The border array and prefix array of an indeterminate string of length $n$ can be computed in $O(n^2)$ time in the worst-case, $O(n)$ in the average case.

Lemma ([IR16])

The prefix array of an indeterminate string of length $n$ over a constant-sized alphabet can be computed in $O(n\sqrt{n})$ time and $O(n)$ space.
The Knuth-Morris-Pratt (KMP) Algorithm

- The most famous pattern-matching algorithm.
- It computes the **border** of every prefix of $p$; that is, computes the border array of $p$ ($BA_p$) to compute the shift.

![Diagram of KMP algorithm](image)

```
1    i    n
x . . . a . . .
1    j    m
p   b . . .
match
```
The KMP Algorithm - 2

The Longest border of $p[1..j-1]$ equals $x[i..n]$. 

Diagram shows: 
- String $x$ with characters $i$ to $n$.
- Pattern $p$ with characters $j$ to $m$.
- Longest border of $p[1..j-1]$ is shaded.
The KMP Algorithm - 3
KMP\textsubscript{Indet} - Simple KMP style algorithm for indeterminate strings

- KMP\textsubscript{Indet} is a hybrid algorithm - works for both regular and indeterminate strings.
- If input is regular, KMP\textsubscript{Indet} is the classical KMP algorithm, and uses the border array of $p$ to compute shifts.
- Otherwise, it checks if the matched prefix of $p$ and the matched substring of $x$ are regular.
  - If yes, it uses the border array of $p$ to compute the shift.
  - Otherwise, it constructs a new string $p'$ from the longest proper prefix of the matched pattern $p$ and the longest proper suffix of the matched substring of the text $x$, and computes the prefix array of $p'$ to compute the shift.
If the matched prefix of $p$ and the matched substring of $x$ are both regular, KMP\_Indet uses the border array of $p$ ($\beta_p$) to compute the shift.
If \( p' \) is indeterminate, \texttt{KMP\_INDET} constructs the prefix array of \( p' \) (\( \pi_{p'} \)) to compute the shift.

The shift is the maximum value in the second half of the prefix array (\( \pi_{p'} \)), say at position \( k \), such that a prefix of \( p' \) matches the entire suffix at \( k \).
The example below simulates the execution of KMP\textsubscript{Indet} on the text $x = aabaabaa\{a, b\}baa\{a, c\}$ and pattern $p = aabaa$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td></td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td></td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

When pattern is aligned at positions 1 and 4, KMP\textsubscript{Indet} uses the $BA_p$ to compute the shift.

When pattern is aligned at position 7, a mismatch occurs at index 10. Also, $p' = aba\{a, b\}$ is indeterminate. Therefore, we compute the prefix array of $p'$ ($\pi_{p'} = (4, 0, 2, 1)$). Since the shift is 2, pattern is aligned at position 8.

After execution, KMP\textsubscript{Indet} returns the list of positions $\{1, 4, 8\}$ at which $p$ occurs in $x$. 
Running time of KMP\_Indet

Theorem (1)

Given text $y = y[1..n]$ and pattern $q = q[1..m]$ on an alphabet of constant size $\sigma$, KMP\_Indet executes in $O(n)$ time when $y$ and $q$ are both regular; otherwise, when both are indeterminate, the worst-case upper bound is $O(m^2n)$. The algorithm’s additional space requirement is $O(m)$, for the pattern $q'$ and corresponding arrays $\beta_{q'}$ and $\pi_{q'}$.

Using Lemma [IR16] we restate Theorem (1) resulting in an improved run time complexity for KMP\_Indet.

Theorem (1)

Given text $y = y[1..n]$ and pattern $q = q[1..m]$ on an alphabet of constant size $\sigma$, KMP\_Indet executes in $O(n)$ time when $y$ and $q$ are both regular; otherwise, when both are indeterminate, the worst-case upper bound is $O(nm^2\sqrt{m})$. The algorithm’s additional space requirement is $O(m)$, for the pattern $q'$ and corresponding arrays $\beta_{q'}$ and $\pi_{q'}$. 
Pattern matching in conservative indeterminate strings

A **conservative indeterminate string** is an indeterminate string in which the number of indeterminate letters is bounded above by a constant \( k \geq 0 \).

- Crochemore et. al in [CIK+16] proposed an \( O(nk) \) algorithm which uses suffix trees and other auxiliary data structures to search for pattern \( p \) in the text \( x \). The number of indeterminate letters in \( x \) and \( p \) is bounded by a constant \( k \).

- Daykin et. al in [DGG+19], proposed a pattern matching algorithm by first constructing the Burrows Wheeler Transform (BWT) of \( x \) in \( O(mn) \) time, and use it to find all occurrences of \( p \) in \( x \) in \( O(km^2 + q) \) time, where \( q \) is the number of occurrences of the pattern in \( x \).

- \texttt{KMP\_Indet} on the other hand, requires \( O(n + km^2) \) time in the best case and requires \( O(nm^2) \) in the worst case.
In the paper, we present a simple KMP style pattern matching algorithm (KMP\textsubscript{Indet}) for indeterminate strings that is very efficient in cases that arise in practice.

Further, the algorithm uses negligible $\Theta(m)$ space in all cases.

We conjecture that a similar approach is feasible for the Boyer-Moore algorithm [BM77], together with its numerous variants (BM-Horspool, BM-Sunday, BM-Galil, Turbo-BM): see [Smy03, Ch. 8] and

https://www-igm.univ-mlv.fr/~lecroq/string/

As a future research problem, we intend to optimize KMP\textsubscript{Indet} for the conservative indeterminate strings.

We also intend to perform experimental comparison of the running times of existing indeterminate pattern-matching algorithms with those of KMP\textsubscript{Indet}, assuming various frequencies of indeterminate letters.
References I

*The Design and Analysis of Computer Algorithms.*  

A fast string searching algorithm.  

Linear algorithm for conservative degenerate pattern matching.  
References II


In CPM, 2016.

References III

New perspectives on the prefix array.

Thank you!