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 Σ.

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 Σ 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** $\boldsymbol{x} = \boldsymbol{x}[1..n]$ on Σ is an array of regular letters drawn from Σ.

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

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

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 Σ 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 $\boldsymbol{x} = \boldsymbol{x}[1..n]$ on Σ (the **source string**), we apply the mapping f to compute $\boldsymbol{y} = \boldsymbol{y}[1..n]$ (the **mapped string**) according to the following rule:

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

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

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

  Applying **Rule (R)** for $1 \leq k \leq 4$, we compute the mapped string $\boldsymbol{y} = 2/6/5/14/7/15$.

- The mapping f and Rule (R) allows an ordering on the indeterminate letters drawn from Σ.
- 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, $f : t \to 2, c \to 3, a \to 5, g \to 7$) yields $\boldsymbol{y} = 5/15/7/10/2/21$ with a a quite different ordering,

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

Lemma (1)

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

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

$$\boldsymbol{x_1}[i] \approx \boldsymbol{x_1}[j] \Longleftrightarrow \boldsymbol{x_2}[i] \approx \boldsymbol{x_2}[j]. \tag{1}$$

We thus have the following observations:

Observation (1)

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

Observation (2)

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

Observation (3)

Suppose ℓ_1 and ℓ_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 Σ_{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 $\boldsymbol{x}[i_1]$ and $\boldsymbol{x}[i_2]$ on Σ (that is, between $\boldsymbol{y}[i_1]$ and $\boldsymbol{y}[i_2]$ computed using Rule (R)) requires time bounded above by a (small) constant.

Pattern matching in strings

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

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

	1	2	3	4	5	6	7	8	9	10	11	12	13
$\boldsymbol{x}$	a	a	b	a	a	b	a	a	{a, b}	b	a	a	{a, c}
$\boldsymbol{\beta_x}$	0	1	0	1	2	3	4	5	6	3	4	5	2
$\boldsymbol{\pi_x}$	13	1	0	6	1	0	3	5	1	0	2	2	1

Figure 1: Border array $\boldsymbol{\beta_x}$ and prefix array $\boldsymbol{\pi_x}$ computed for the string $\boldsymbol{x} = aabaabaa\{a,b\}baa\{a,c\}$.

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])

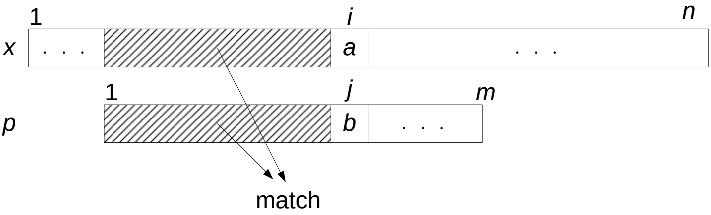
The border array and prefix array of an indeterminate string of length n can be computed in $O(n^2)$ time in the worst-case, $O(n)$ in the average case.

Lemma ([IR16])

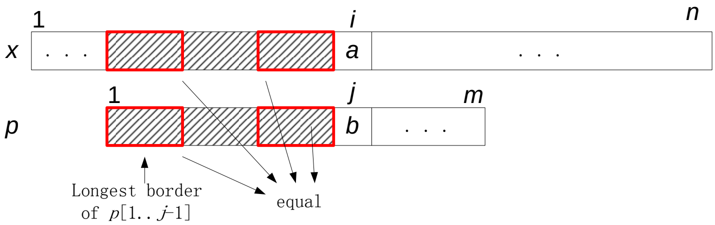
The prefix array of an indeterminate string of length n over a constant-sized alphabet can be computed in $O(n\sqrt{n})$ time and $O(n)$ space.

The Knuth-Morris-Pratt (KMP) Algorithm

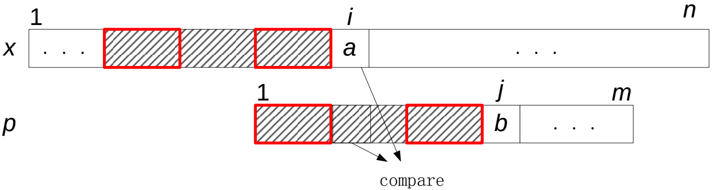
- The most famous pattern-matching algorithm.
- It computes the border of every prefix of $\boldsymbol{p}$; that is, computes the border array of $\boldsymbol{p}$ $(BA_{\boldsymbol{p}})$ to compute the shift.



The KMP Algorithm - 2

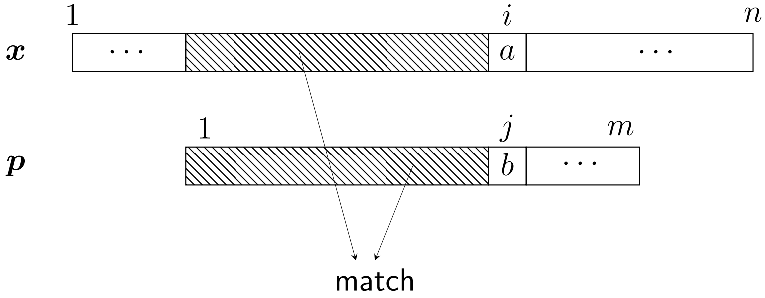


The KMP Algorithm - 3

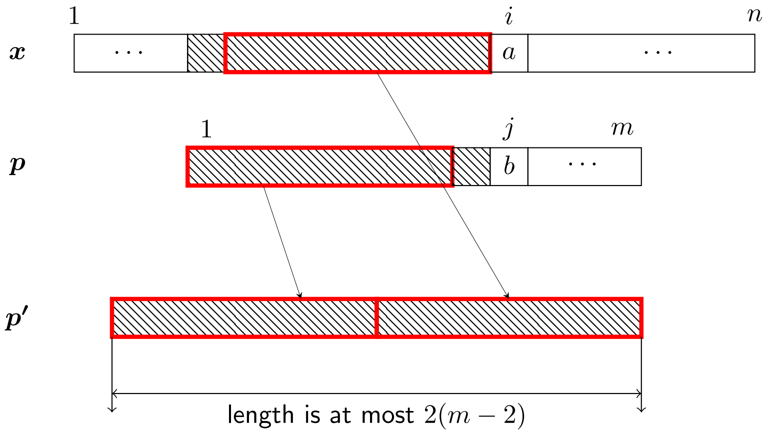


KMP_INDET - Simple KMP style algorithm for indeterminate strings

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



If the matched prefix of $\boldsymbol{p}$ and the matched substring of $\boldsymbol{x}$ are both regular, KMP_INDET uses the border array of $\boldsymbol{p}$ ($\beta_{\boldsymbol{p}}$) to compute the shift.



If $\boldsymbol{p'}$ is indeterminate, KMP_INDET constructs the prefix array of $\boldsymbol{p'}$ ($\pi_{\boldsymbol{p'}}$) to compute the shift.

The shift is the maximum value in the second half of the prefix array ($\pi_{\boldsymbol{p'}}$), say at position k, such that a prefix of $\boldsymbol{p'}$ matches the entire suffix at k.

- The example below simulates the execution of $\text{KMP\_I}\textsc{ndet}$ on the text $\boldsymbol{x} = aabaabaa\{a,b\}baa\{a,c\}$ and pattern $\boldsymbol{p} = aabaa$.

	1	2	3	4	5	6	7	8	9	10	11	12	13
$\boldsymbol{x}$	a	a	b	a	a	b	a	a	{a, b}	b	a	a	{a, c}
	a	a	b	a	a								
				a	a	b	a	a					
							a	a	b	x			
								a	a	b	a	a	

- When pattern is aligned at positions 1 and 4, $\text{KMP\_I}\textsc{ndet}$ uses the $BA_{\boldsymbol{p}}$ to compute the shift.
- When pattern is aligned at position 7, a mismatch occurs at index 10. Also, $\boldsymbol{p'} = aba\{a,b\}$ is indeterminate. Therefore, we compute the prefix array of $\boldsymbol{p'}$ ($\pi_{p'} = (4,0,2,1)$). Since the shift is 2, pattern is aligned at position 8.
- After execution, $\text{KMP\_I}\textsc{ndet}$ returns the list of positions $\{1,4,8\}$ at which $\boldsymbol{p}$ occurs in $\boldsymbol{x}$.

Running time of KMP_INDET

Theorem (1)

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

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

Theorem (1)

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

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 $\boldsymbol{p}$ in the text $\boldsymbol{x}$. The number of indeterminate letters in $\boldsymbol{x}$ and $\boldsymbol{p}$ 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 $\boldsymbol{x}$ in $O(mn)$ time, and use it to find all occurrences of $\boldsymbol{p}$ in $\boldsymbol{x}$ in $O(km^2 + q)$ time, where q is the number of occurrences of the pattern in $\boldsymbol{x}$.
- 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.

Conclusions

- In the paper, we present a simple KMP style pattern matching algorithm (KMP_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_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_INDET, assuming various frequencies of indeterminate letters.

References I

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman.
The Design and Analysis of Computer Algorithms.
Addison–Wesley, 1974.

[BM77] Robert S. Boyer and J. Strother Moore.
A fast string searching algorithm.
Communications of the ACM, 20(10):762–772, 1977.

[CIK$^+$16] Maxime Crochemore, Costas S. Iliopoulis, Ritu Kundu, Manal Mohamed, and Fatima Vayani.
Linear algorithm for conservative degenerate pattern matching.
Eng. Appls. of Artificial Intelligence, 51:109–114, 2016.

References II

[DGG+19] J. W. Daykin, R. Groult, Y. Guesnet, T. Lecroq, A. Lefebvre, M. Léonard, L. Mouchard, É. Prieur-Gaston, and B. Watson.
Efficient pattern matching in degenerate strings with the Burrows-Wheeler transform.
Information Processing Letters, 147:82–87, 2019.

[IR16] Costas S. Iliopoulos and Jakub Radoszewski.
Truly subquadratic-time extension queries and periodicity detection in strings with uncertainties.
In *CPM*, 2016.

[Smy03] Bill Smyth.
Computing Patterns in Strings.
Pearson/Addison–Wesley, 2003.

References III

[SW08] W. F. Smyth and Shu Wang.

New perspectives on the prefix array.

Proc. 15th String Processing & Inform. Retrieval Symp. (SPIRE), 5280:133–143, 2008.

Thank you!