Faster Subsequence and Don’t-Care Pattern Matching on Compressed Texts

Takanori Yamamoto, Hideo Bannai,
Shunsuke Inenaga, Masayuki Takeda

Department of Informatics,
Kyushu University, JAPAN

Originally presented at CPM 2011

Self introduction

- Name: Shunsuke Inenaga （稲永 俊介）
- Affiliation: Kyushu University, Japan
- Research interests: String matching, Text compression, Algorithms, Data structures

Agenda

- Subsequence Pattern Matching
- Compressed String Processing
- Straight Line Program (SLP)
- Algorithms
  - **<u>Minimum Subsequence Occurrences on SLP</u>**
  - Fixed Length Don't Care Matching on SLP
  - Variable Length Don't Care Matching on SLP
- Summary

Subsequences

- String P of length m is a *subsequence* of string T of length N
  $\Leftrightarrow\ \exists\, i_0, \ldots, i_{m-1}$ s.t.
  $0 \leq i_0 < \ldots < i_{m-1} \leq N\text{-}1$ and
  $P[j] = T[i_j]$ for all $j = 0, \ldots, m-1$

Example

	0	1	2	3	4	5	6	7	8	9
$T =$	a	c	c	b	a	b	b	c	a	b

$P =$ abc

Example

`0 1 2 3 4 5 6 7 8 9`

T = `a c c b a b b c a b`

P = `a b c`

(`0`, `3`, `7`)

Example

```
     0123456789
T =  accbabbcab

P =  abc
```

(0, 3, 7)

(0, 5, 7)

Example

```
     0123456789
T =  accbabbcab

P =  abc
```

(0, 3, 7)
(0, 5, 7)
(0, 6, 7)

Example

```
    0 1 2 3 4 5 6 7 8 9
T = a c c b a b b c a b

P = a b c
```

(0, 3, 7)
(0, 5, 7)
(0, 6, 7)
(4, 5, 7)

Example

```
     0123456789
T =  accbabbcab

P =  abc
```

(0, 3, 7)
(0, 5, 7)
(0, 6, 7)
(4, 5, 7)
(4, 6, 7)

There can be too many occurrences

```
     0 1 2 3 4 5 6 7 8 9
T =  a b a b a b a b a b        P = a a a

     a   a   a
     a   a       a
     a       a   a
     a       a       a
         a   a   a
             ⋮
```

# of choices of indices is $O\left(\binom{N}{m}\right)$

Consider only start & end

```
     0 1 2 3 4 5 6 7 8 9
T =  a b a b a b a b a b        P = a a a

     a   a   a
     a   a       a
(0, 6)
     a       a   a
     a       a       a
         a   a   a
           ⋮
```

two occurrences are equivalent
⇔ they start and end at the same positions

⇓

there still exist $O(N^2)$ non-equivalent occurrences

Minimal Subsequence Occurrences

- An occurrence (i_0, i_{m-1}) of subsequence P in T is <u>*minimal*</u>, if there is *no* occurrence of P in $T[i_0 : i_{m-1}-1]$ or $T[i_0+1 : i_{m-1}]$.

- In other words, (i_0, i_{m-1}) is minimal, if there is no other occurrence of P within $T[i_0 : i_{m-1}]$.

Minimal Subsequence Occurrences

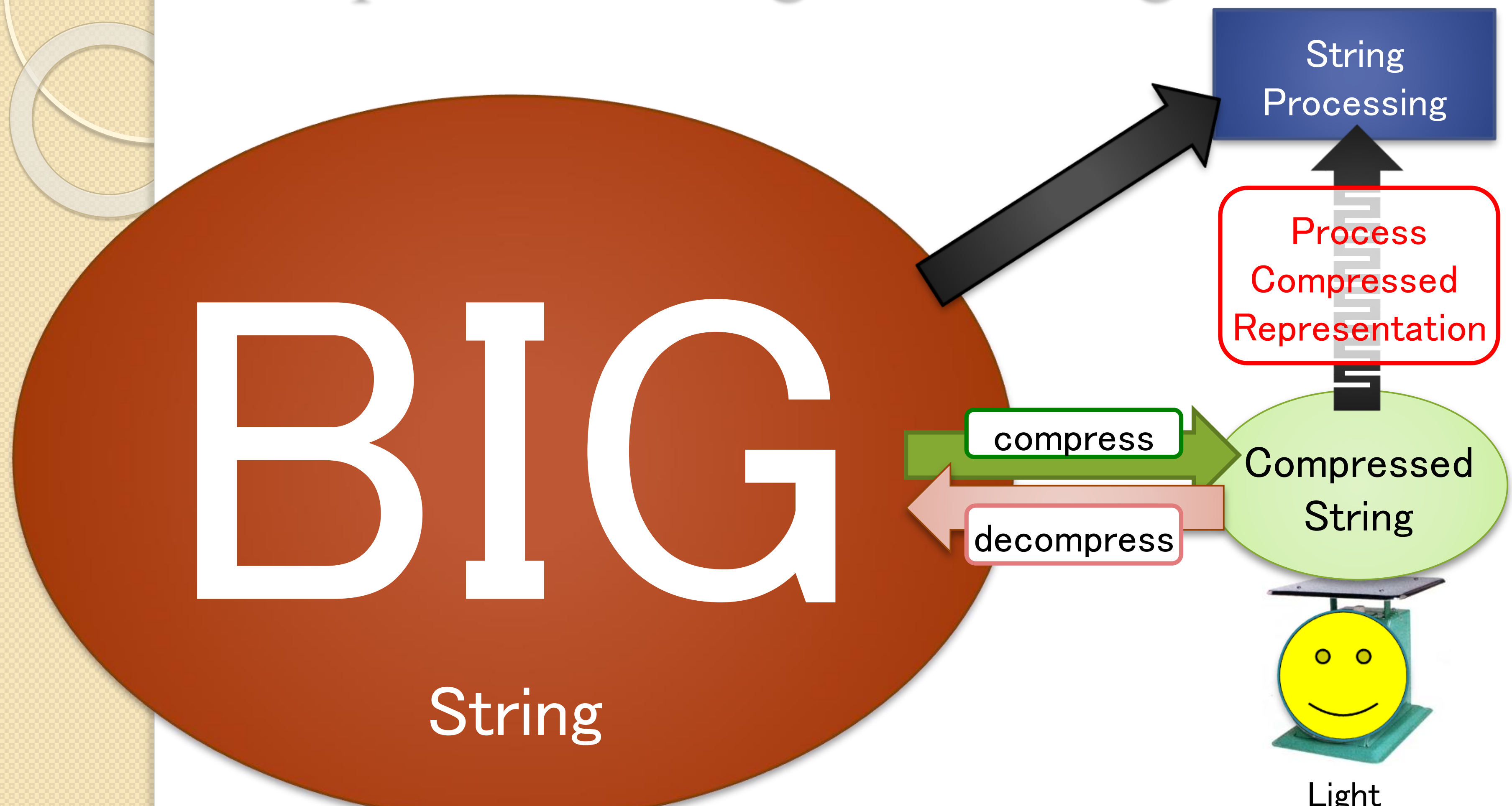
```
       0 1 2 3 4 5 6 7 8 9
T =    a b a b a b a b a b        P = a a a
(0, 4) a   a   a
       a   a       a
       a       a   a
       a       a       a
(2, 6)     a   a   a
               ⋮
```

there are only $O(N)$ minimal occurrences

Problem setting

- We want to solve the problem of computing minimal occurrences of a query pattern when a text is given in a *compressed form*.

Compressed String Processing



Processing *without* explicit decompression can dramatically save time and space

Straight Line Program [1/2]

An SLP $\mathcal{S}$ is a sequence of n assignments

$$X_1 = expr_1\ ;\ X_2 = expr_2;\ \ldots\ ;\ X_n = expr_n;$$

X_k : variable,

$$expr_k : \begin{cases} \boldsymbol{a} & (\ \boldsymbol{a} \in \boldsymbol{\Sigma}\ ) \\ X_i X_j & (\ i, j < k\ ). \end{cases}$$

SLP $\mathcal{S}$ for string T is a context free grammar in the Chomsky normal form s.t. $L(\mathcal{S}) = \{T\}$.

Straight Line Program [2/2]

SLP S

n

$X_1 = \mathrm{a}$

$X_2 = \mathrm{b}$

$X_3 = X_1X_2$

$X_4 = X_3X_1$

$X_5 = X_3X_4$

$X_6 = X_5X_5$

$X_7 = X_4X_6$

$X_8 = X_7X_5$

$T =$ a b a a b a b a a b a b a a b a b a

N

$N = O(2^n)$

Straight Line Program [2/2]

SLP S

n

$X_1 = a$
$X_2 = b$
$X_3 = X_1X_2$
$X_4 = X_3X_1$
$X_5 = X_3X_4$
$X_6 = X_5X_5$
$X_7 = X_4X_6$
$X_8 = X_7X_5$

$T = \mathtt{a\ b\ a\ a\ b\ a\ b\ a\ a\ b\ a\ b\ a\ a\ b\ a\ b\ a}$

N

$N = O(2^n)$

SLP: Abstract model of compression

- Output of grammar-based compression algorithms (e.g., Re-pair, Sequitur, LZ78) of size n can be trivially converted to SLPs of size $O(n)$ in $O(n)$ time.

- Output of LZ77 of size r can be converted to an SLP of size $O(r \log N)$ in $O(r \log N)$ time.

- Therefore, algorithms working on SLPs are so useful that they can be applied to various types of compressed strings.

Our contribution

Given an SLP-compressed text and an uncompressed pattern, we propose $O(nm)$ algorithms for:

- Subsequence pattern matching
- FLDC (fixed length don't care) pattern matching
- VLDC (variable length don't care) pattern matching

n = size of SLP
m = length of pattern

Subsequence matching

Subsequence Problems on SLP [Cégielski *et al*. 2006]

Minimal Subsequence Occurrences

Input : SLP of size n representing string T, string P
Output : # of minimal subsequence occurrences of P in T

Several variations, e.g.:

Bounded Minimal Subsequence Occurrences

Input : SLP of size n representing T, string P, integer w
Output : # of minimal subsequence occurrences (i_0, i_{m-1}) of P in T satisfying $i_{m-1} - i_0 \leq w$

Comparison to previous work

Decomp.& [Troníček 2001] ➔ $O(Nm) = O(2^n m)$

[Cégielski *et al*. 2006] ➔ $O(nm^2 \log m)$

[Tiskin 2009] ➔ $O(nm^{1.5})$

[Tiskin 2011] ➔ $O(nm \log m)$

This Work

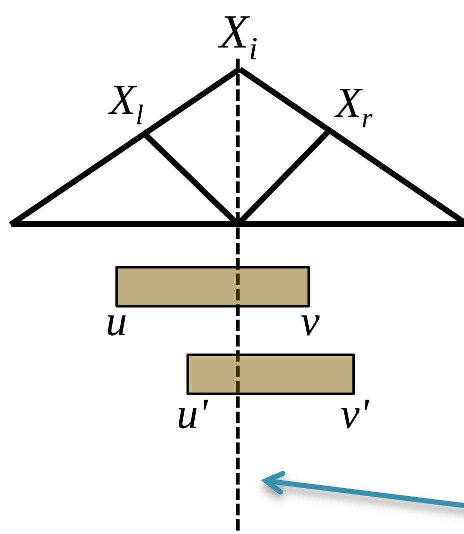
Subsequence problems ➔ $O(nm)$

Extensions to pattern matching with Fixed/Variable Length Don't Care Symbols ➔ $O(nm)$

串：Stabbed occurrences

For $X_i = X_l X_r$, an occurrence (u, v) of P is said to be a *stabbed occurrence* in X_i if :

$$0 \leq u < |X_l| \leq v \leq |X_i| - 1.$$



串

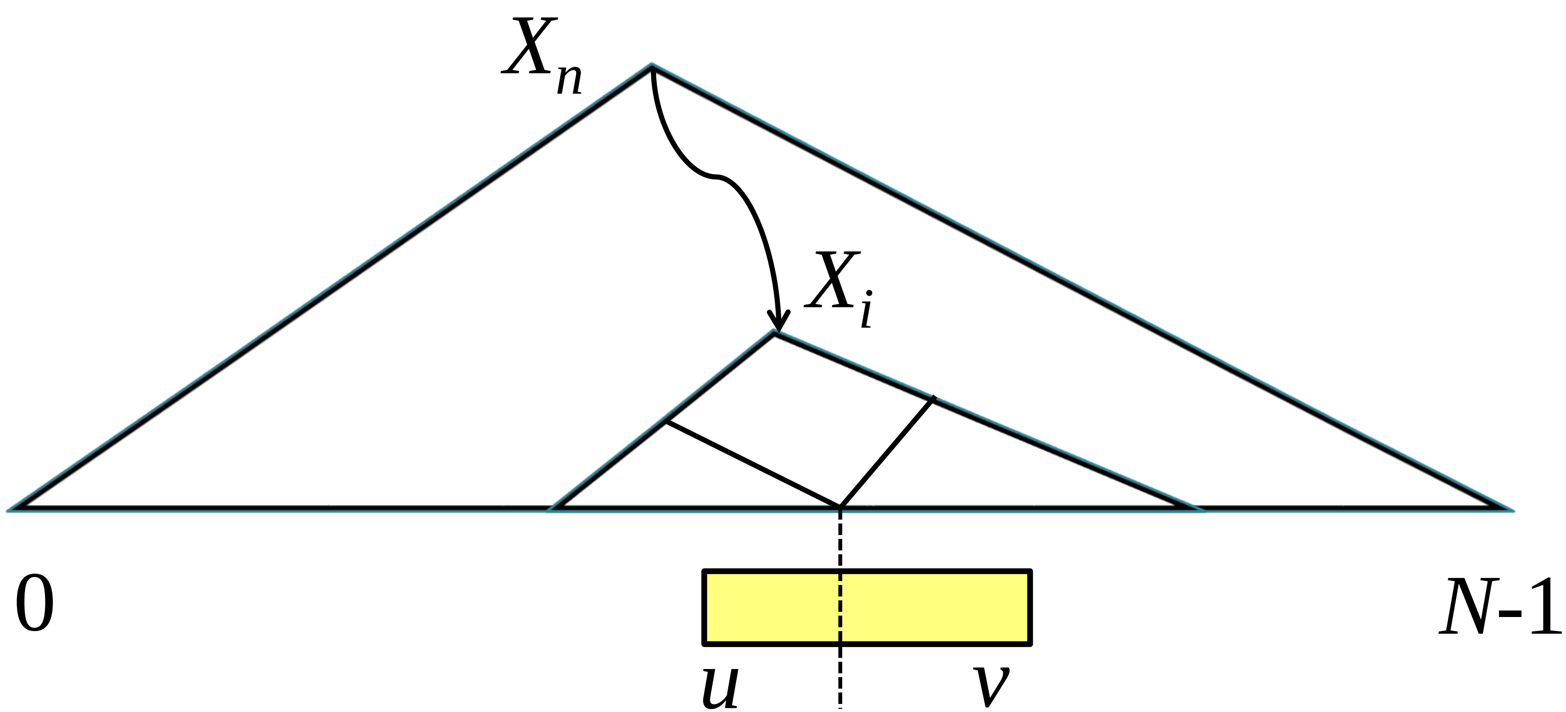
おでん
“ODEN”

串 (KUSHI) is a Kanji character meaning “skewer”, used to stab food.

Every occurrence is stabbed

Observation

For any interval $[u, v]$ with $0 \leqq u \leqq v \leqq N\text{-}1$, there exists a variable X_i which stabs $[u, v]$.



Counting minimal occurrences

- M_i : # of minimal occurrences of P in X_i
- $M^{\text{串}}(l, r)$: # of *stabbed* minimal occurrences of P in $X_i = X_l X_r$

M_n is the solution to our Problem

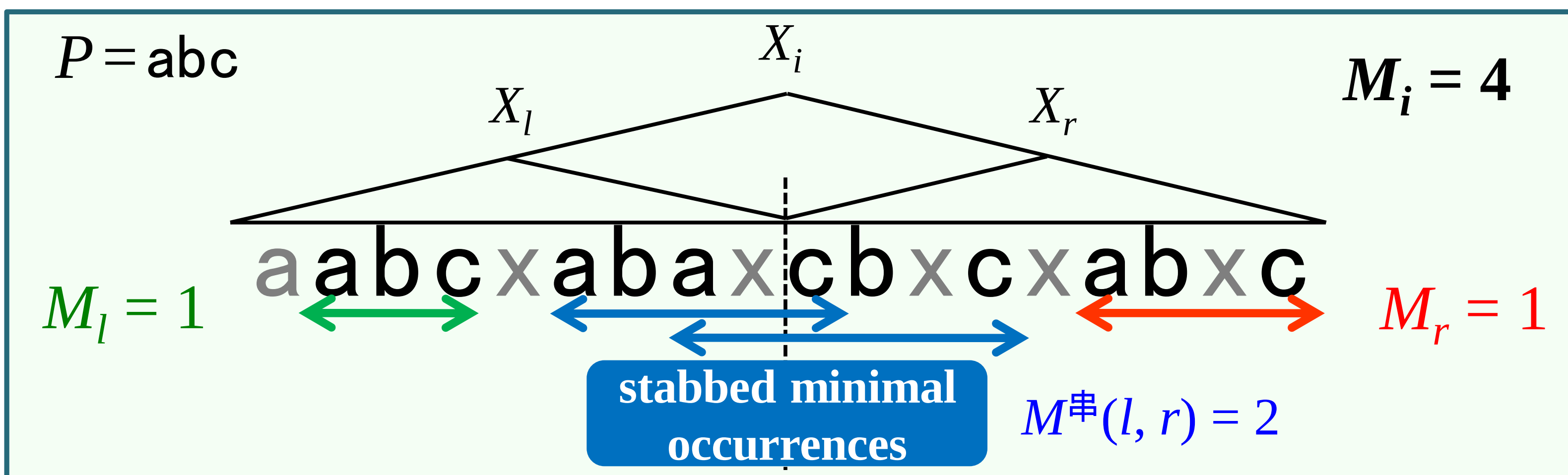
Computing M_i

- If $X_i = \mathrm{a} \quad (\mathrm{a} \in \Sigma)$

$$M_i = \begin{cases} 0 & \text{if } m \neq 1 \text{ or } P \neq \mathrm{a} \\ 1 & \text{if } m = 1 \text{ and } P = \mathrm{a} \end{cases}$$

- If $X_i = X_l X_r \quad (l, r < i)$

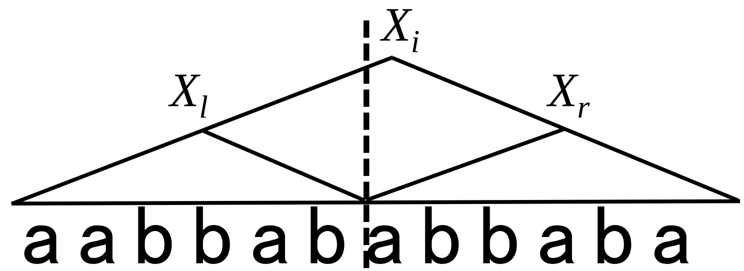
$$M_i = M_l + M_r + M^{\text{串}}(l, r)$$



Computing $M^{串}(l, r)$

there are at most $m-1$ stabbed minimal occurrences

$P = \text{abbba}$



k	$R(l,k)$			$L(r,m-k)$
0	∞			–
1	5	a b b b	a	1
2	5	a b b	b a	4
3	2	a b	b b a	4
4	2	a	b b b a	6
5	-		a b b b a	6

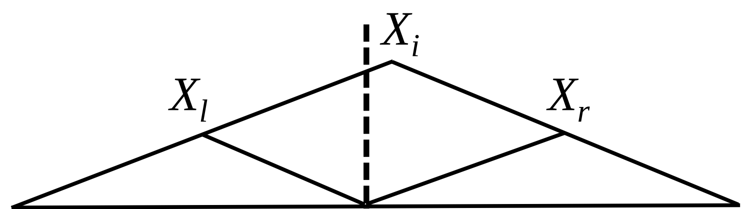
shortest suffix of X_l containing $P[0{:}m\text{-}k\text{-}1]$

shortest prefix of X_r containing $P[m\text{-}k{:}m\text{-}1]$

Computing $M^{串}(l, r)$

there are at most $m-1$ crossing minimal occurrences

$P =$ abbba



k	$R(l,k)$	a a b b a b	a b b a b a	$L(r, m-k)$
0	∞			–
1	5	a b b b	a	1
2	5	a b b	b a	4
3	2	a b	b b a	4
4	2	a	b b b a	6
5	-		a b b b a	6

shortest suffix of X_l containing $P[0{:}m\text{-}k\text{-}1]$

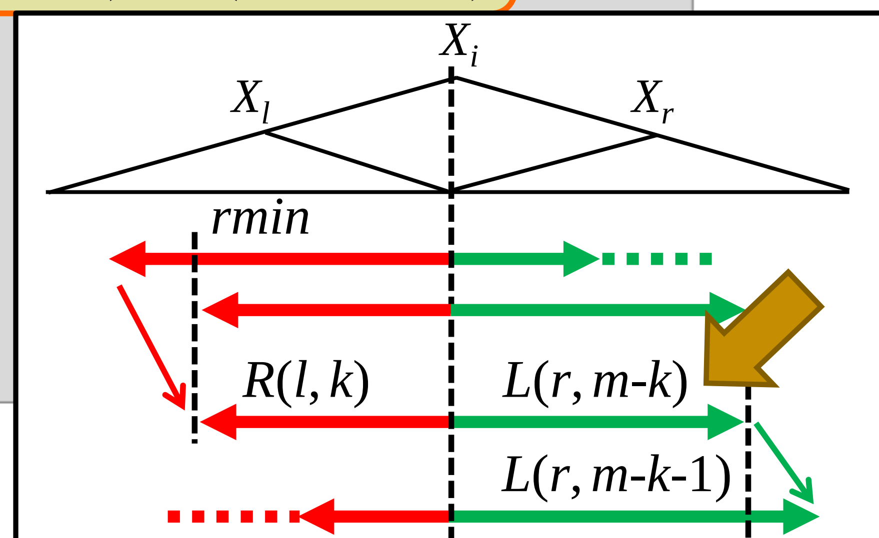
shortest prefix of X_r containing $P[m\text{-}k{:}m\text{-}1]$

Computing $M^{串}(l, r)$

Lemma

$M^{串}(l,r)$ for all $X_i = X_l X_r$ can be computed in a total of $O(nm)$ time using L and R.

```
C := 0, rmin := R(l, 0)
for k := 1 to m – 1
  if rmin > R(l,k) and L(r,m-k) < L(r,m-k-1) then
    C := C + 1
    rmin := R(l, k)
  end if
end for
M串(l, r) := C
```



$L(i,j)$: Length of shortest prefix of X_i s.t. $P[j{:}m\text{-}1]$ is subsequence

$R(i,j)$: Length of shortest suffix of X_i s.t. $P[0{:}m\text{-}j\text{-}1]$ is subsequence

Computing Q (to compute L)

$Q(i,j)$: length of longest prefix of $P[j{:}]$ which is also a subsequence of X_i. ($i = 1, \ldots, n,\ j = 0, \ldots, m$)

Computing $Q(i, j)$

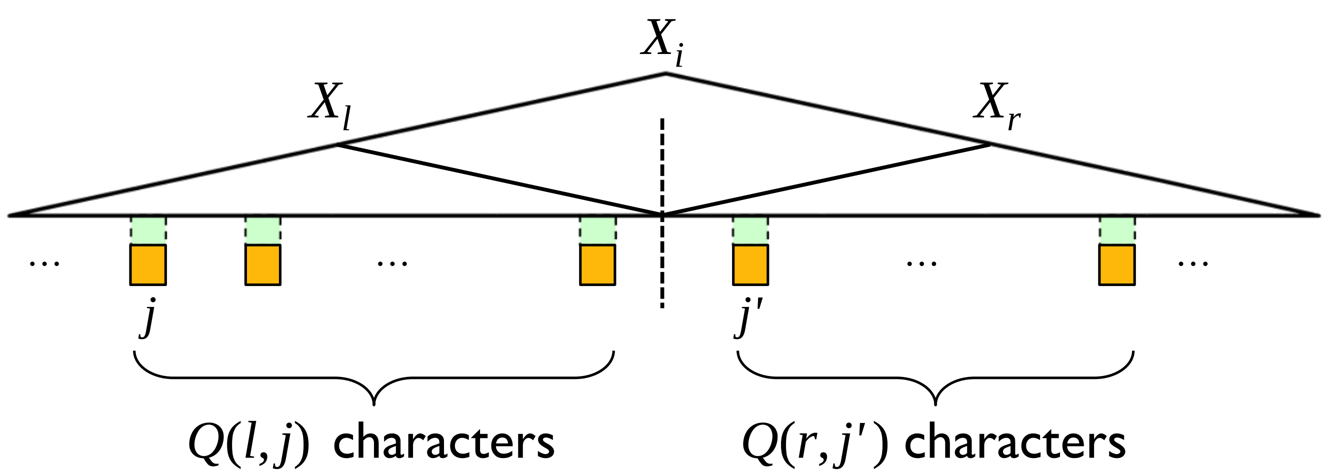
- If $X_i = \mathrm{a}\ \ (\mathrm{a} \in \Sigma)$

$$Q(i, j) = \begin{cases} 0 & \text{if } P[j] \neq \mathrm{a} \\ 1 & \text{if } P[j] = \mathrm{a} \end{cases}$$

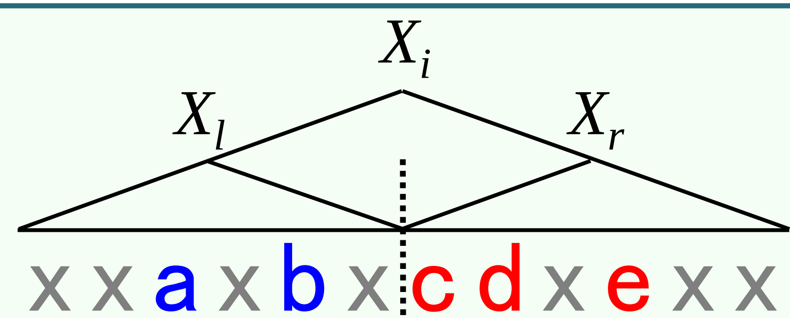
- If $X_i = X_l X_r\ \ (l, r < i)$

$$Q(i, j) = Q(l, j) + Q(r, j')$$

$$\underline{(j' = j + Q(l, j))}$$



Computing Q (to compute L)



$P[j:] = $ abcdef

$P[j':] = $ cdef

$Q(l,j) = 2$

$Q(r,j') = 3$

$j' := j + Q(l,j)$

$Q(i,j) := Q(l,j) + Q(r,j')$

$= 2 + 3 = 5$

Lemma [Cégielski *et al.*]

For all $i = 1, \ldots, n$ and $j = 0, \ldots, m$

$Q(i, j)$ can be calculated in $O(nm)$ time using DP.

Computing L

$L(i, j)$: length of shortest prefix of X_i s.t. $P[j:]$ is subsequence

$(i = 1,\ldots,n,\ j = 0,\ldots,m)$ (∞ if $P[j:]$ is not subsequence of X_i)

Computing $L(i, j)$

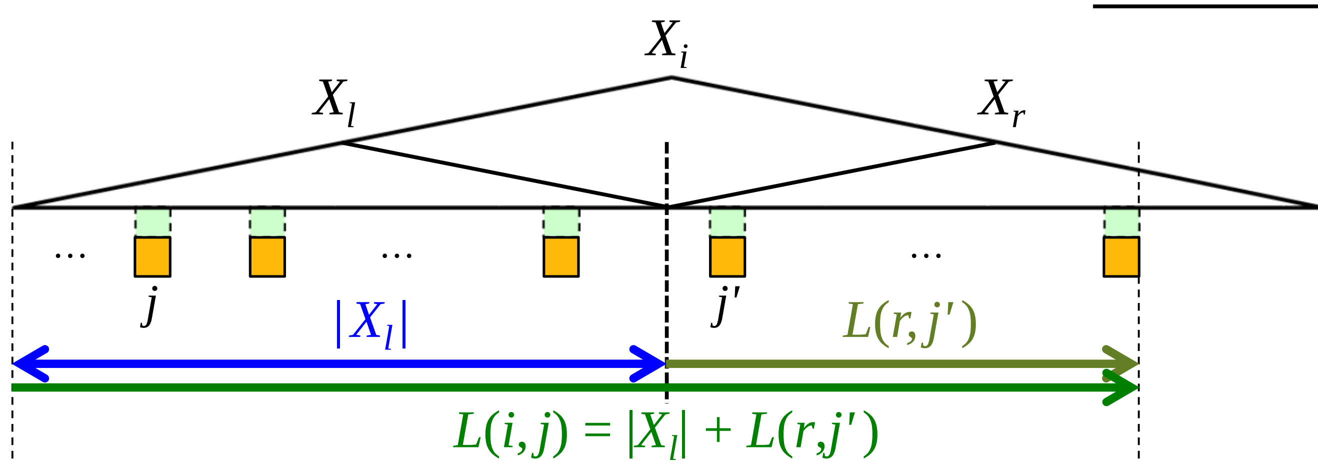
- If $X_i = \mathrm{a}$ ($\mathrm{a} \in \Sigma$)

$$L(i, j) = \begin{cases} 0 & \text{if } j = m \\ 1 & \text{if } P[j:] = \mathrm{a} \\ \infty & \text{if } P[j:] \neq \mathrm{a} \end{cases}$$

- If $X_i = X_l X_r$

$$L(i, j) = \begin{cases} L(l, j) & \text{if } j' = m \\ |X_l| + L(r, \underline{j'}) & \text{if } j' < m \end{cases}$$

$$\underline{(j' = j + Q(l, j))}$$

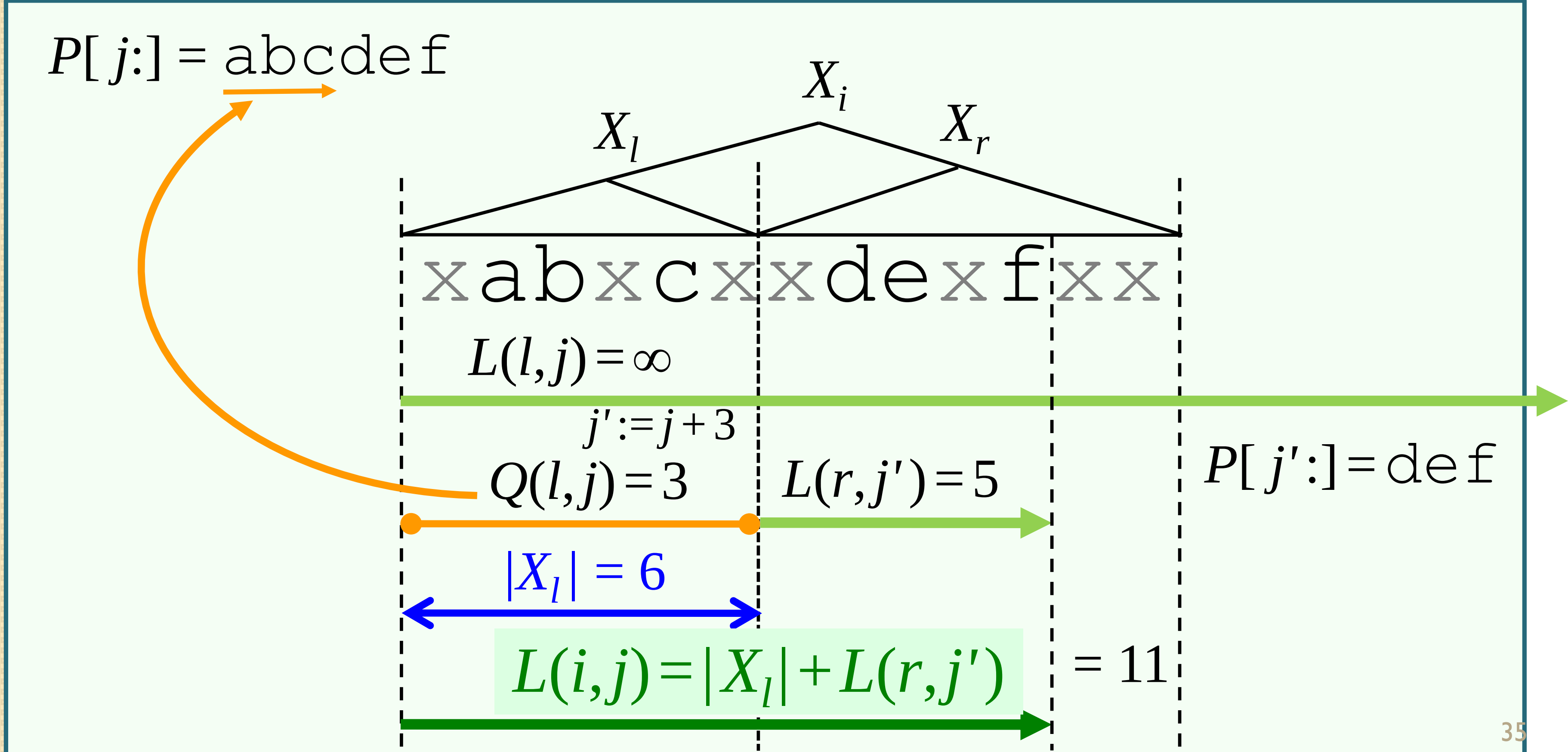


Computing L

[Cégielski *et al.*, 2007]
$O(nm^2 \log m)$

Lemma

$L(i, j)$ can be computed for all $i = 1, \ldots, n$, $j = 0, \ldots, m$, in a total of $O(nm)$ time using $Q(i,j)$.



Result

Minimal Subsequence Occurrences Problem

Input : SLP of size n representing string T, string P
Output : # of minimal occurrences of subsequence P in T

Theorem

Given an SLP of size n and a pattern of length m, minimal subsequence occurrences can be computed in $\underline{O(nm)}$ <u>time and space</u>.

$O(Nm) = O(2^n m)$ Decomp.&[Troníček 2001]

$O(nm^2 \log m)$ [Cégielski et al. 2007]

$O(nm^{1.5})$ [Tiskin 2009]

$O(nm \log m)$ [Tiskin 2011]

$\Rightarrow$ $\boldsymbol{O(nm)}$

FLDC matching

Fixed Length Don't Care Pattern

- We allow pattern P to contain special *don't-care* symbol ○ that matches any single character.

$P =$ ab○d○a

$T =$ xaabcdabddx

ab○d○b

Fixed Length Don't Care

- We can apply the subsequence matching algorithm to FLDC matching!

Bounded Minimal Subsequence Occurrences Problem

Input : SLP of size n representing T, string P, integer w
Output : # of minimal occurrences (i_0, i_{m-1}) of subsequence P in T, where $i_{m-1} - i_0 \leq w$

Observation

Bounded Minimal Subsequence Occurrences Problem with window size $w = |P|$ ⇔ substring matching

Fixed Length Don’t Care

Solution

- Set the window-size to m (= $|P|$)
- Extend algorithm to handle don't care symbol ‘○’
  ➔ Just modify base cases for Q and L,
  computation of M and $M^{串}$ are the same.

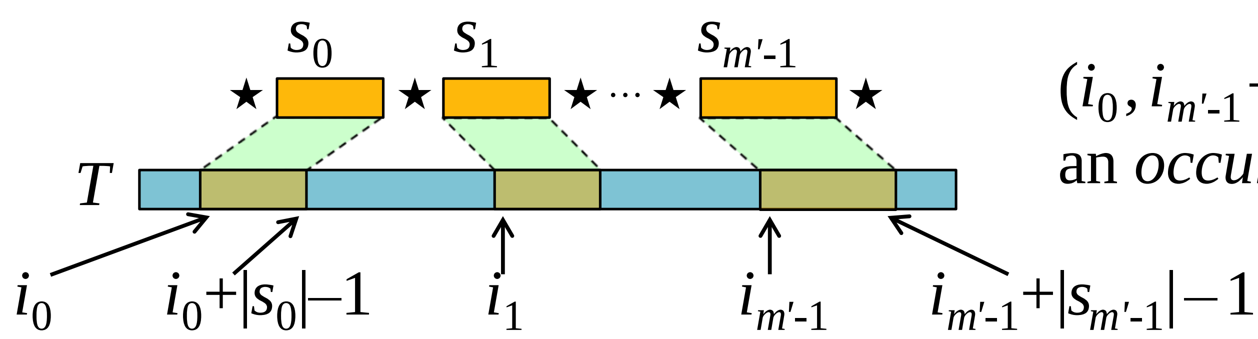
Theorem

Given SLP of size n and an FLDC pattern of size m, the FLDC matching problem can be solved in <u>$O(nm)$ time and space</u>.

VLDC matching

Variable Length Don't Care

P	VLDC Pattern ($\star\ s_0 \star s_1 \star \cdots \star s_{m'-1} \star$)
$\star$	VLDC symbol that matches *any string*
s_j	segment ($s_j \in \Sigma^+$, $j = 0, \ldots, m'-1$)
m'	# of segments
m	pattern length ($m = \|s_0\| + \|s_1\| + \cdots + \|s_{m'-1}\|$)



$(i_0, i_{m'-1}+|s_{m'-1}|-1)$ is an *occurrence* of P in T

Example

```
    0123456789
T = accbabbcab

P = ★ab★c★
```

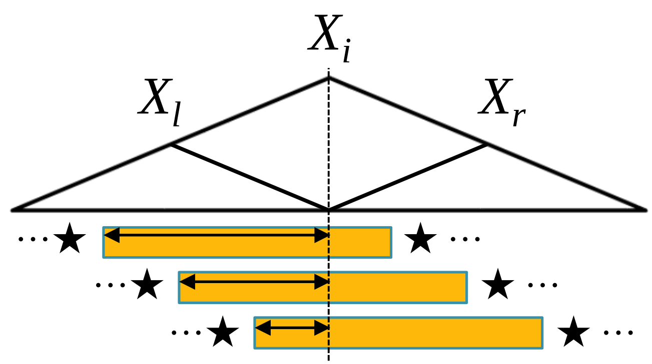
Example

$$\begin{array}{l}
\phantom{T = {}}\texttt{0123456789} \\
T = \texttt{accb}\boxed{\texttt{ab}}\texttt{b}\boxed{\texttt{c}}\texttt{ab} \\
P = \star\boxed{\texttt{ab}}\star\boxed{\texttt{c}}\star
\end{array}$$

(4,7)

VLDC Pattern Matching

Let $Occ^{\text{串}}(X_i, s_j)$ denote the stabbed occurrences of segment s_j in X_i $(i = 1, \ldots, n,\ j = 0, \ldots, m')$



Theorem [Kida *et al*., 2003]

All $Occ^{\text{串}}(X_i, s_j)$ can be computed in a total of $O(nm)$ time. Each $Occ^{\text{串}}(X_i, s_j)$ forms a single arithmetic progression, which can be represented in $O(1)$ space

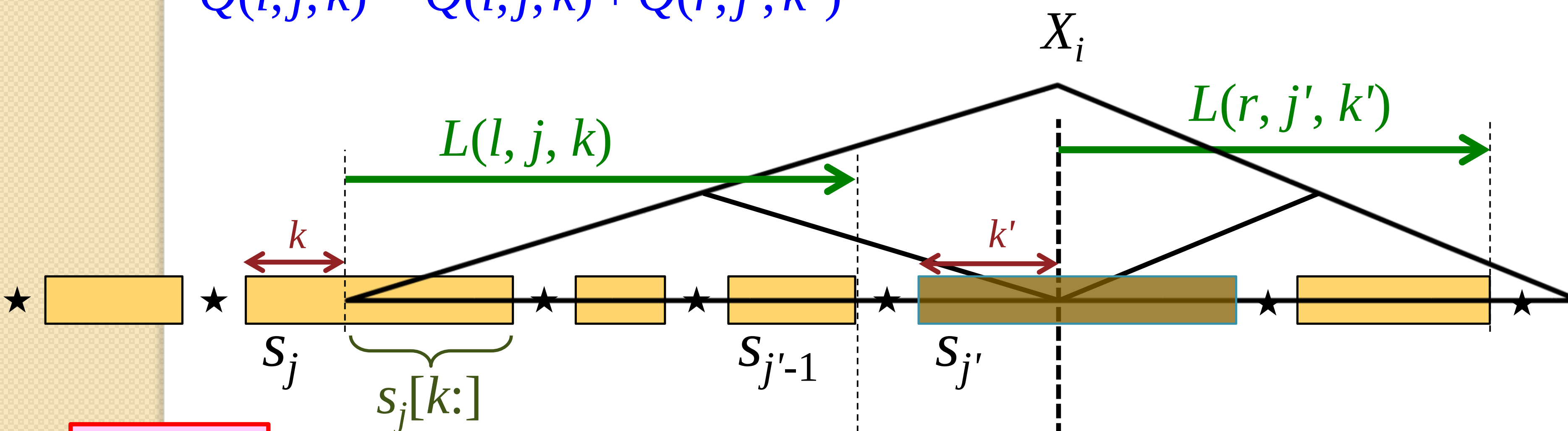
Computing Q and L for VLDC

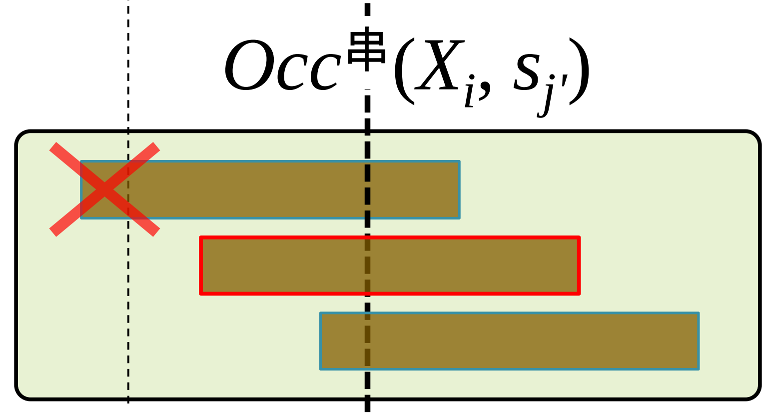
case: $Q(l, j, k) \geq 1$ or $k = 0$

$j' := j + Q(l, j, k)$

$k' := \max\{x \mid x \in Occ^{串}(X_i, s_{j'}),\ x + L(l,j,k) \leq |X_l|\}$

$Q(i,j,k) = Q(l,j,k) + Q(r,j',k')$





Lemma

$Q(i,j,k)$ and $L(i,j,k)$ can be computed in $O(nm)$ time using $Occ^{串}(X_i, s_j)$

Conclusion

- We proposed $O(nm)$ algorithms on SLP*s* for:
  - Subsequence matching
  - Fixed/Variable Length Don't Care matching

- Existing best algorithm for computing minimal subsequence occurrences on *uncompressed* text of length N takes $O(Nm)$ time [Troníček 2001].
  - Since $n = O(N)$, our $O(nm)$ solution is at least as efficient as the $O(Nm)$ solution, and is faster when the text is compressible.

Open Problems

- Matching for patterns which contain *both* FLDC & VLDC symbols.
- Bounding minimum & maximum lengths for VLDCs.
- Faster longest common subsequence (LCS)?
  - Tiskin's $O(nm \log m)$ subsequence matching algorithm can be used to compute LCS.
- Succinct index for subsequence matching?