Contextual Pattern Matching in Less Space

Paniz Abedin,[1] Oliver Chubet[2] , Daniel Gibney[3], Sharma V. Thankachan[2]

[1]Florida Polytechnic University
[2]North Carolina State University
[3]Georgia Institute of Technology

Outline

Outline

Pattern Matching

Preprocess: A Text T$[1, n]$
Query: A Pattern $P[1, m]$
Output: The occurrences of P in T

Optimal Solution: $O(n)$ space and $O(m + occ)$ query time

Motivation

Why Contextual Pattern Matching?

Motivation

Why Contextual Pattern Matching?

If the substring T' is repeated 1,000 times within T and P occurs within T', should our algorithm report all of these 1,000 occurrences of P?

This motivated the Contextual Pattern Matching problem introduced by Navarro, which is likely better suited for such situations.

Problem Definition

Contextual Pattern Matching Problem

Introduced by Navarro (SPIRE 2020)

Preprocess: A text T$[1, n]$
Query: (P, ℓ) A string $P[1, m]$ and a length ℓ
Output: All c distinct strings XPY where $|X| = |Y| = \ell$

Problem Definition

Contextual Pattern Matching Problem

Introduced by Navarro (SPIRE 2020)

Preprocess: A text T$[1, n]$

Query: (P, ℓ) A string $P[1, m]$ and a length ℓ

Output: All c distinct strings XPY where $|X| = |Y| = \ell$

Solution: $O(n)$ space and $O(m + c)$ query time using suffix trees

Problem Definition

Contextual Pattern Matching Problem

Introduced by Navarro (SPIRE 2020)

Preprocess: A text T$[1, n]$
Query: (P, ℓ) A string $P[1, m]$ and a length ℓ
Output: All c distinct strings XPY where $|X| = |Y| = \ell$

Solution: $O(n)$ space and $O(m + c)$ query time using suffix trees

Can we solve this problem using space proportional to a compressed form of T?

Results

Preprocess: A text $T[1, n]$
Query: (P, ℓ) A string P and a length ℓ
Output: All c distinct strings XPY where $|X| = |Y| = \ell$

Publications	Space	Time
Navarro [SPIRE 2020]	$O(\bar{r} \log(n/\bar{r}))$	$O(\|P\| + c \log n)$
Our result	$O(r \log(n/r))$	$O(\|P\| + c \log \ell \cdot \log(n/r))$

r : The number of runs in the BWT of T
$\bar{r}$: The maximum of the number of runs in the BWT of T and its reverse

Results

Preprocess: A text T$[1, n]$
Query: (P, ℓ) A string P and a length ℓ
Output: All c distinct strings XPY where $|X| = |Y| = \ell$

Publications	Space	Time
Navarro [SPIRE 2020]	$O(\bar{r}\log(n/\bar{r}))$	$O(\lvert P\rvert + c\log n)$
Our result	$O(r\log(n/r))$	$O(\lvert P\rvert + c\log\ell\cdot\log(n/r))$

r : The number of runs in the BWT of T
$\bar{r}$: The maximum of the number of runs in the BWT of T and its reverse

$\bar{r} = O(r\log^2 n)$ ⇨ Space complexity could potentially be $O(r\log^3 n)$

Outline

Suffix Trees, Suffix Arrays and BWT

Let $\mathrm{T} = \mathrm{T}[1, n]$ be a text over an alphabet Σ:

- **Suffix Tree:** The suffix tree ST of T is a compact trie over all strings in the set $\{\mathrm{T}[i, n] \mid i \in [1, n]\}$

Suffix Trees, Suffix Arrays and BWT

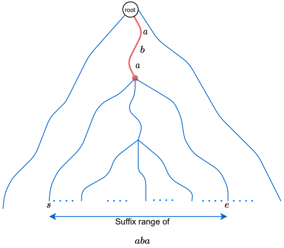
Let $\mathsf{T} = \mathsf{T}[1, n]$ be a text over an alphabet Σ:

- **Suffix Tree:** The suffix tree ST of T is a compact trie over all strings in the set $\{\mathsf{T}[i, n] \mid i \in [1, n]\}$
- **Suffix array:** The suffix array SA and the inverse suffix array ISA of T are arrays of length n, such that $\mathsf{SA}[i] = j$ and $\mathsf{ISA}[j] = i$ iff $\mathsf{T}[j, n]$ is the ith smallest suffix of T in lexicographic order

Suffix Trees, Suffix Arrays and BWT

Let $T = T[1, n]$ be a text over an alphabet Σ:

- **Suffix Tree:** The suffix tree ST of T is a compact trie over all strings in the set $\{T[i, n] \mid i \in [1, n]\}$
- **Suffix array:** The suffix array SA and the inverse suffix array ISA of T are arrays of length n, such that $SA[i] = j$ and $ISA[j] = i$ iff $T[j, n]$ is the ith smallest suffix of T in lexicographic order
- **Suffix range** for a substring $T[i, j]$: The interval of the suffix array whose values contain the starting indices of all suffixes of T having $T[i, j]$ as a prefix



Suffix Trees, Suffix Arrays and BWT

Let $\mathrm{T} = \mathrm{T}[1, n]$ be a text over an alphabet Σ:

- **Suffix Tree:** The suffix tree ST of T is a compact trie over all strings in the set $\{\mathrm{T}[i, n] \mid i \in [1, n]\}$
- **Suffix array:** The suffix array SA and the inverse suffix array ISA of T are arrays of length n, such that $\mathrm{SA}[i] = j$ and $\mathrm{ISA}[j] = i$ iff $\mathrm{T}[j, n]$ is the ith smallest suffix of T in lexicographic order
- **Suffix range** for a substring $\mathrm{T}[i, j]$: The interval of the suffix array whose values contain the starting indices of all suffixes of T having $\mathrm{T}[i, j]$ as a prefix
- LCP **array:** Stores the length of the longest common prefix between consecutive suffixes in SA, $\mathrm{LCP}[i] = \mathrm{LCP}(\mathrm{T}[\mathrm{SA}[i], n], \mathrm{T}[\mathrm{SA}[i-1], n])$

Suffix Trees, Suffix Arrays and BWT

Let $\mathrm{T} = \mathrm{T}[1, n]$ be a text over an alphabet Σ:

- **Suffix Tree:** The suffix tree ST of T is a compact trie over all strings in the set $\{\mathrm{T}[i, n] \mid i \in [1, n]\}$
- **Suffix array:** The suffix array SA and the inverse suffix array ISA of T are arrays of length n, such that $\mathrm{SA}[i] = j$ and $\mathrm{ISA}[j] = i$ iff $\mathrm{T}[j, n]$ is the ith smallest suffix of T in lexicographic order
- **Suffix range** for a substring $\mathrm{T}[i, j]$: The interval of the suffix array whose values contain the starting indices of all suffixes of T having $\mathrm{T}[i, j]$ as a prefix
- LCP **array:** Stores the length of the longest common prefix between consecutive suffixes in SA, $\mathrm{LCP}[i] = \mathrm{LCP}(\mathrm{T}[\mathrm{SA}[i], n], \mathrm{T}[\mathrm{SA}[i-1], n])$
- BWT: is a permutation of T such that $\mathrm{BWT}[i] = \mathrm{T}[\mathrm{SA}[i] - 1]$

Suffix Trees, Suffix Arrays and BWT

Let $\mathrm{T} = \mathrm{T}[1, n]$ be a text over an alphabet Σ:

- **Suffix Tree:** The suffix tree ST of T is a compact trie over all strings in the set $\{\mathrm{T}[i, n] \mid i \in [1, n]\}$
- **Suffix array:** The suffix array SA and the inverse suffix array ISA of T are arrays of length n, such that $\mathrm{SA}[i] = j$ and $\mathrm{ISA}[j] = i$ iff $\mathrm{T}[j, n]$ is the ith smallest suffix of T in lexicographic order
- **Suffix range** for a substring $\mathrm{T}[i, j]$: The interval of the suffix array whose values contain the starting indices of all suffixes of T having $\mathrm{T}[i, j]$ as a prefix
- LCP **array:** Stores the length of the longest common prefix between consecutive suffixes in SA, $\mathrm{LCP}[i] = \mathrm{LCP}(\mathrm{T}[\mathrm{SA}[i], n], \mathrm{T}[\mathrm{SA}[i-1], n])$
- BWT: is a permutation of T such that $\mathrm{BWT}[i] = \mathrm{T}[\mathrm{SA}[i] - 1]$
- r: The number of equal-letter runs in the BWT. It is known that r is within logarithmic factors from several other popular compression measures, including the LZ77 parse size

Fully Functional Suffix Trees in BWT-Runs Bounded Space

Gagie et al. provide a data structure of size $O(r \log(n/r))$ for a given text $\mathrm{T}[1, n]$ that can simulate a suffix tree ST

- Find the suffix range of any pattern $P[1, m]$ in $O(m)$ time.
- Find $\mathrm{SA}[i]$, $\mathrm{ISA}[i]$, or $\mathrm{LCP}[i]$ in $O(\log(n/r))$ time for any $i \in [1, n]$.
- Find the string depth of any node in the suffix tree in $O(\log(n/r))$ time.
- Find $\mathrm{RMQ}_{\mathrm{LCP}}(a, b) = \arg\min_{a \leq k \leq b} \mathrm{LCP}[k]$ in $O(\log(n/r))$ time for any $a \leq b \in [1, n]$.
- Find lowest common ancestor of two nodes u and v, $\mathrm{LCA}(u, v)$, in $O(\log(n/r))$ time.
- Compute the Weiner-link, $\mathrm{WLink}(v, a)$, i.e., if v represents string α then the node that represents string $a \circ \alpha$, where $\circ$ denotes concatenation, in time $O(\log \log_w(n/r))^1$.

---

[1] w represents the machine word length.

Outline

Main Theorem

Let T be a text of length n, and r be the maximum of the number of equal letter runs of its BWT:

There is a data structure requiring $O(r \log(n/r))$ space that finds the c contextual pattern matches of $(P[1,m],\ell)$ in time $O(m + c \log \ell \cdot \log(n/r))$.

Our Algorithm - Overview

1. Find the suffix range of occurrences of pattern P in SA, denoted as $[sp, ep]$.

Our Algorithm - Overview

1. Find the suffix range of occurrences of pattern P in SA, denoted as $[sp, ep]$.
2. Partition $[sp, ep]$ into k maximal intervals s.t suffixes within each interval have a lcp $\geq m + \ell$.

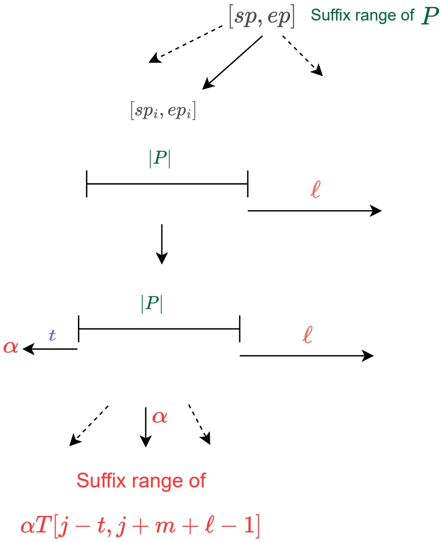
Our Algorithm - Overview

1. Find the suffix range of occurrences of pattern P in SA, denoted as $[sp, ep]$.
2. Partition $[sp, ep]$ into k maximal intervals s.t suffixes within each interval have a lcp $\geq m + \ell$.
3. For every interval $[sp_i, ep_i]$, let $j \in \mathrm{SA}[sp_i, ep_i]$ be chosen arbitrarily. Find the length t of the longest string that precedes all occurrences of $\mathrm{T}[j, j + m + \ell - 1]$.

Our Algorithm - Overview

1. Find the suffix range of occurrences of pattern P in SA, denoted as $[sp, ep]$.
2. Partition $[sp, ep]$ into k maximal intervals s.t suffixes within each interval have a lcp $\geq m + \ell$.
3. For every interval $[sp_i, ep_i]$, let $j \in \mathrm{SA}[sp_i, ep_i]$ be chosen arbitrarily. Find the length t of the longest string that precedes all occurrences of $\mathrm{T}[j, j + m + \ell - 1]$.
    - If $t \geq \ell$, find the suffix range for $\mathrm{T}[j - t, j + m + \ell - 1]$ and add it to the solution and finish with interval $[sp_i, ep_i]$
    - Else, for each distinct $\alpha \circ \mathrm{T}[j - t, j + m + \ell - 1]$ in T , find the suffix range of them and recursively apply step 3 on them

Our Algorithm - Overview



$[sp, ep]$ Suffix range of P

$[sp_i, ep_i]$

$|P|$ ℓ

$|P|$ α t ℓ

α

Suffix range of

$\alpha T[j-t, j+m+\ell-1]$

Outline

Our Algorithm - Example

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

T = $cabcdabacabcaababda$

$$(P, \ell) = (ab, 1)$$

$P = ab$

sp *ep*

SA	1	2	3	**4**	**5**	**6**	**7**	**8**	9	10	11	12	13	14	15	16	17	18	19	20
	20	19	13	14	6	10	2	16	8	15	7	11	3	17	9	1	12	4	18	5

LCP	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	0	0	1	1	3	2	3	2	1	0	2	1	2	1	0	4	2	1	0	2

Our Algorithm - Example

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

T = $cabcdabacabcaababda$

$$(P, \ell) = (ab, 1)$$

$P = ab$

sp ep

SA	1	2	3	**4**	**5**	**6**	**7**	**8**	9	10	11	12	13	14	15	16	17	18	19	20
	20	19	13	14	6	10	2	16	8	15	7	11	3	17	9	1	12	4	18	5

LCP	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	0	0	1	1	3	**2**	3	**2**	1	0	2	1	2	1	0	4	2	1	0	2

sp_1 $ep_1 sp_2$ ep_2 $sp_3 ep_3$

Our Algorithm - Example

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

T = $cabcdabacabcaababda$

$$(P, \ell) = (ab, 1)$$

$$\overbrace{sp \quad\quad\quad\quad ep}^{P = ab}$$

SA	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	20	19	13	14	6	10	2	16	8	15	7	11	3	17	9	1	12	4	18	5

LCP	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	0	0	1	1	3	2	3	2	1	0	2	1	2	1	0	4	2	1	0	2

$$\underbrace{sp_1 \quad ep_1}\ \underbrace{sp_2 \quad ep_2}\ \underbrace{sp_3 \quad ep_3}$$

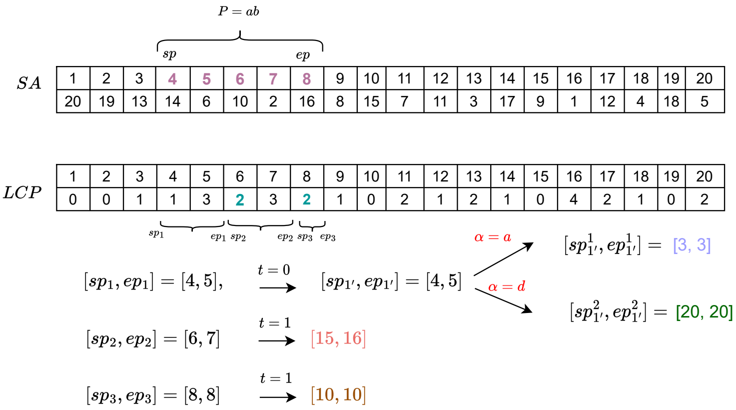
$$\begin{tikzcd}
 & & {[sp^1_{1'}, ep^1_{1'}] = [3,3]} \\
{[sp_1, ep_1] = [4,5],} \arrow[r, "t = 0"] & {[sp_{1'}, ep_{1'}] = [4,5]} \arrow[ur, "\alpha = a"] \arrow[dr, "\alpha = d"'] & \\
 & & {[sp^2_{1'}, ep^2_{1'}] = [20,20]}
\end{tikzcd}$$

Our Algorithm - Example

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

T = $cabcdabacabcaababda$

$(P, \ell) = (ab, 1)$



$P = ab$

sp ep

SA	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	20	19	13	14	6	10	2	16	8	15	7	11	3	17	9	1	12	4	18	5

LCP	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	0	0	1	1	3	2	3	2	1	0	2	1	2	1	0	4	2	1	0	2

sp_1 ep_1 sp_2 ep_2 sp_3 ep_3

$[sp_1, ep_1] = [4, 5], \xrightarrow{t=0} [sp_{1'}, ep_{1'}] = [4, 5]$

$\xrightarrow{\alpha = a} [sp^1_{1'}, ep^1_{1'}] = [3, 3]$

$\xrightarrow{\alpha = d} [sp^2_{1'}, ep^2_{1'}] = [20, 20]$

$[sp_2, ep_2] = [6, 7] \xrightarrow{t=1} [15, 16]$

$[sp_3, ep_3] = [8, 8] \xrightarrow{t=1} [10, 10]$

Our Algorithm - Step 1

1. Find the suffix range of occurrences of pattern P in SA, denoted as $[sp, ep]$.

Solution: $O(m)$ time using $O(r \log(n/r))$ space data structures of Gagie et al.

Our Algorithm - Step 2

2. Partition $[sp, ep]$ into k maximal intervals s.t within each interval suffixes have lcp of length $\geq m + \ell$.

Solution: $O(k \log(n/r))$ time using $O(r \log(n/r))$ space data structures of Gagie et al.

Lemma

Given a suffix range, $[sp, ep]$, and a length t, we can partition $[sp, ep]$ into k maximal intervals $[sp_1, ep_1]$, $[sp_2, ep_2], \cdots$ and $[sp_k, ep_k]$ where $sp_i = ep_{i-1} + 1$, such that suffixes within each interval have a longest common prefix of length $\geq t$ in $O(k \log(n/r))$ total time.

Proof: At most k number of $\mathrm{RMQ}_{\mathrm{LCP}}(sp, ep)$

Our Algorithm - Step 3

Let $l = m + \ell$

3. For every interval, $[sp_i, ep_i]$ resulting from Step 2:

- Find the length t of the longest string that precedes all occurrences of T$[j, j + l - 1]$ for any arbitrary $j \in [sp_i, ep_i]$

Our Algorithm - Step 3

Let $l = m + \ell$

3. For every interval, $[sp_i, ep_i]$ resulting from Step 2:

- Find the length t of the longest string that precedes all occurrences of $T[j, j + l - 1]$ for any arbitrary $j \in [sp_i, ep_i]$
  - If $t \geq l$ ⇒ Find the suffix range for $T[j - t, j + l - 1]$

Our Algorithm - Step 3

Let $l = m + \ell$

3. For every interval, $[sp_i, ep_i]$ resulting from Step 2:

- Find the length t of the longest string that precedes all occurrences of $\mathrm{T}[j, j + l - 1]$ for any arbitrary $j \in [sp_i, ep_i]$
  - If $t \geq l$ ⇨ Find the suffix range for $\mathrm{T}[j - t, j + l - 1]$
  - Else, for all distinct $\alpha \in \Sigma$, where $\alpha \circ \mathrm{T}[j - t, j + l - 1]$ is in T
    - Find the suffix ranges of all distinct $\alpha \circ \mathrm{T}[j - t, j + l - 1]$ and recursively apply step 3 on them

Our Algorithm - Step 3

Let $l = m + \ell$

3. For every interval, $[sp_i, ep_i]$ resulting from Step 2:

- Find the length t of the longest string that precedes all occurrences of $\mathrm{T}[j, j + l - 1]$ for any arbitrary $j \in [sp_i, ep_i]$
    - If $t \geq l$ ⇒ Find the suffix range for $\mathrm{T}[j - t, j + l - 1]$
    - Else, for all distinct $\alpha \in \Sigma$, where $\alpha \circ \mathrm{T}[j - t, j + l - 1]$ is in T
        - Find the suffix ranges of all distinct $\alpha \circ \mathrm{T}[j - t, j + l - 1]$ and recursively apply step 3 on them

Next, we show how to solve this step efficiently

Our Algorithm - Step 3

- Find the length t of the longest string that precedes all occurrences of T$[j, j + l - 1]$ for any arbitrary $j \in [sp_i, ep_i]$

Our Algorithm - Step 3

- Find the length t of the longest string that precedes all occurrences of $\mathsf{T}[j, j + l - 1]$ for any arbitrary $j \in [sp_i, ep_i]$

Lemma

We can find the length t in $O(\log \ell \cdot \log(n/r))$ time.

Our Algorithm - Step 3

- Find the length t of the longest string that precedes all occurrences of $\mathrm{T}[j, j+l-1]$ for any arbitrary $j \in [sp_i, ep_i]$

Lemma

We can find the length t in $O(\log \ell \cdot \log(n/r))$ time.

We first prove the following lemma

Lemma

Let the suffix range $[sp, ep]$ and value t be given. Let l be the string depth of the node for $[sp, ep]$. We can check in $O(\log(n/r))$ time whether all substrings $\mathrm{T}[\mathrm{SA}[i], \mathrm{SA}[i]+\ell-1]$ for $i \in [sp, ep]$ are preceded by the same length t substring.

Our Algorithm - Step 3

- Find the length t of the longest string that precedes all occurrences of $\mathrm{T}[j, j+l-1]$ for any arbitrary $j \in [sp_i, ep_i]$

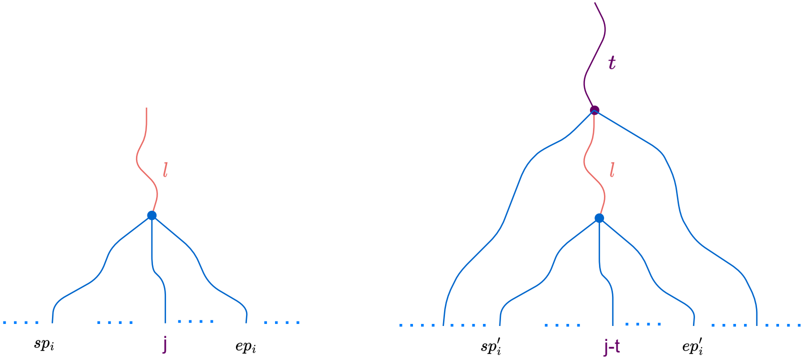
Let $[sp_i^{'}, ep_i^{'}]$ be the suffix range for $\mathrm{T}[j-t, j+l-1]$

Claim:

- $ep_i - sp_i = ep_i^{'} - sp_i^{'}$
- $|\mathrm{LCA}([sp', sp'], [ep', ep'])| \geq t + l.$

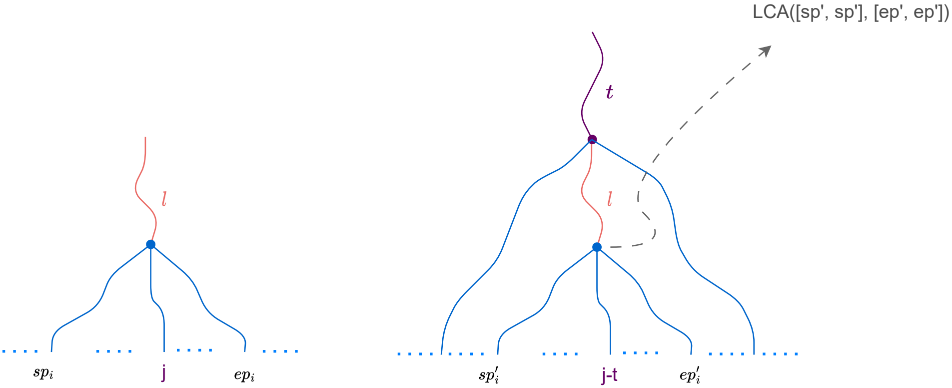
Our Algorithm - Step 3

- $ep_i - sp_i = ep_i^{'} - sp_i^{'}$



Our Algorithm - Step 3

- $|\mathrm{LCA}([sp', sp'], [ep', ep'])| \geq t + l$.



Our Algorithm - Step 3

Lemma

Let the suffix range $[sp, ep]$ *and value* t *be given. Let* l *be the string depth of the node for* $[sp, ep]$*. We can check in* $O(\log(n/r))$ *time whether all substrings* $\mathrm{T}[\mathrm{SA}[i], \mathrm{SA}[i] + \ell - 1]$ *for* $i \in [sp, ep]$ *are preceded by the same length* t *substring.*

Proof: Using fully functional suffix trees in BWT-runs bounded space for LCA and SA and ISA queries

Our Algorithm - Step 3

- Find the length t of the longest string that precedes all occurrences of $\mathrm{T}[j, j+l-1]$ for any arbitrary $j \in [sp_i, ep_i]$

Lemma

We can find the length t in $O(\log \ell \cdot \log(n/r))$ time.

Proof: Using exponential search to find the largest t such that all instances of the substring $\mathrm{T}[j, j+l-1]$ share the prefix $\mathrm{T}[j-t, j-1]$ and $t + l \leq 2\ell + m$

Our Algorithm - Step 3

3. For every interval, $[sp_i, ep_i]$ resulting from Step 2:

- Find the length t of the longest string that precedes all occurrences of $\mathrm{T}[j, j + l - 1]$ for any arbitrary $j \in [sp_i, ep_i]$
    - If $t \geq l$ → Find the suffix range for $\mathrm{T}[j - t, j + l - 1]$
    - Else, for all distinct $\alpha \in \Sigma$, where $\alpha \circ \mathrm{T}[j - t, j + l - 1]$ is in T
        - Find the suffix ranges of all distinct $\alpha \circ \mathrm{T}[j - t, j + l - 1]$ and recursively apply step 3 on them

Our Algorithm - Step 3

- For all distinct $\alpha \in \Sigma$, where $\alpha \circ \mathsf{T}[j - t, j + l - 1]$ is in T, find the suffix ranges of all distinct $\alpha \circ \mathsf{T}[j - t, j + l - 1]$

Our Algorithm - Step 3

- For all distinct $\alpha \in \Sigma$, where $\alpha \circ \mathsf{T}[j - t, j + l - 1]$ is in T, find the suffix ranges of all distinct $\alpha \circ \mathsf{T}[j - t, j + l - 1]$

How to find all distinct $\alpha \in \Sigma$, where $\alpha \circ \mathsf{T}[j - t, j + l - 1]$ is in T?

Our Algorithm - Step 3

Lemma

For a text T *with a* BWT_T *having* r *runs, there exists a data structure requiring* $O(r)$ *space such that given the suffix range* $[sp, ep]$ *corresponding to a substring* $\mathsf{T}[i, j]$ *and containing the start or end of at least one* BWT_T *run, reports all distinct* $\alpha \in \Sigma$ *such that* $\alpha \circ \mathsf{T}[i, j]$ *is a substring of* T*. This can be done in constant time per* α *reported.*

Our Algorithm - Step 3

Lemma

For a text T *with a* BWT_T *having* r *runs, there exists a data structure requiring* $O(r)$ *space such that given the suffix range* $[sp, ep]$ *corresponding to a substring* $\mathrm{T}[i, j]$ *and containing the start or end of at least one* BWT_T *run, reports all distinct* $\alpha \in \Sigma$ *such that* $\alpha \circ \mathrm{T}[i, j]$ *is a substring of* T*. This can be done in constant time per* α *reported.*

Proof: By 1-D color range reporting structure given by Nekrich and Vitter (2013), we obtain a data structure that requires $O(r)$ space that, given a query range, reports each distinct color in constant time

Our Algorithm - Step 3

- For all distinct $\alpha \in \Sigma$, where $\alpha \circ \mathsf{T}[j-t, j+l-1]$ is in T, find the suffix ranges of all distinct $\alpha \circ \mathsf{T}[j-t, j+l-1]$

Solution: For each $\alpha \in \Sigma'$, we find WLink(v, α). Each of these requires $O(\log\log(n/r))$ time. This does not affect the time spent on this interval asymptotically.

Outline

Our Algorithm - Complexity

1. Find the suffix range of P,$[sp, ep]$ ⇨ $O(m)$

Our Algorithm - Complexity

1. Find the suffix range of P,$[sp, ep]$ → $O(m)$
2. Partition $[sp, ep]$ into k maximal intervals s.t suffixes within each interval have a $lcp \geq m + \ell$. → $O(k \log(n/r))$, $k \leq c$

Our Algorithm - Complexity

1. Find the suffix range of P,$[sp, ep]$ ⇨ $O(m)$
2. Partition $[sp, ep]$ into k maximal intervals s.t suffixes within each interval have a $lcp \geq m + \ell$. ⇨ $O(k \log(n/r))$, $k \leq c$
3. For each $[sp_i, ep_i]$:
   - 3.1 Let $j \in \mathrm{SA}[sp_i, ep_i]$ be chosen arbitrarily. Find t,the length of the longest string that precedes all occurrences of $\mathrm{T}[j, j + l - 1]$. ⇨ $O(\log \ell \cdot \log(n/r))$
   - 3.2 If $t \geq l$. find the suffix range for $\mathrm{T}[j - t, j + l - 1]$ ⇨ $O(\log \ell \cdot \log(n/r))$

Our Algorithm - Complexity

1. Find the suffix range of P,$[sp, ep]$ $\Longrightarrow$ $O(m)$
2. Partition $[sp, ep]$ into k maximal intervals s.t suffixes within each interval have a $lcp \geq m + \ell$. $\Longrightarrow$ $O(k \log(n/r))$, $k \leq c$
3. For each $[sp_i, ep_i]$:
   - 3.1 Let $j \in \mathrm{SA}[sp_i, ep_i]$ be chosen arbitrarily. Find t,the length of the longest string that precedes all occurrences of $\mathrm{T}[j, j + l - 1]$. $\Longrightarrow$ $O(\log \ell \cdot \log(n/r))$
   - 3.2 If $t \geq l$. find the suffix range for $\mathrm{T}[j - t, j + l - 1]$ $\Longrightarrow$ $O(\log \ell \cdot \log(n/r))$
   - 3.3 Else, for each distinct $\alpha \circ \mathrm{T}[j - t, j + l - 1]$ in T , find the suffix range of them and recursively apply step 3 on them $\Longrightarrow$ $O(c \log \ell \log(n/r))$

Our Algorithm - Complexity

1. Find the suffix range of P,$[sp, ep]$ ⇒ $O(|P|)$
2. Partition $[sp, ep]$ into k maximal intervals s.t suffixes within each interval have a $lcp \geq m + \ell$. ⇒ $O(k \log(n/r))$, $k \leq c$
3. For each $[sp_i, ep_i]$:
   - 3.1 Let $j \in \mathrm{SA}[sp_i, ep_i]$ be chosen arbitrarily. Find t,the length of the longest string that precedes all occurrences of $\mathrm{T}[j, j + l - 1]$. ⇒ $O(\log \ell \cdot \log(n/r))$
   - 3.2 If $t \geq l$. find the suffix range for $\mathrm{T}[j - t, j + l - 1]$ ⇒ $O(\log \ell \cdot \log(n/r))$
   - 3.3 Else, for each distinct $\alpha \circ \mathrm{T}[j - t, j + l - 1]$ in T , find the suffix range of them and recursively apply step 3 on them ⇒ $O(c \log \ell \log(n/r))$

Total time complexity: $O(|P| + c \log \ell \cdot \log(n/r))$

Our Algorithm - Complexity

1. Find the suffix range of P,$[sp, ep]$ $O(|P|)$
2. Partition $[sp, ep]$ into k maximal intervals s.t suffixes within each interval have a $lcp \geq m + \ell$. $O(k \log(n/r))$, $k \leq c$
3. For each $[sp_i, ep_i]$:
    - 3.1 Let $j \in \mathrm{SA}[sp_i, ep_i]$ be chosen arbitrarily. Find t,the length of the longest string that precedes all occurrences of $\mathrm{T}[j, j+l-1]$. $O(\log \ell \cdot \log(n/r))$
    - 3.2 If $t \geq l$. find the suffix range for $\mathrm{T}[j-t, j+l-1]$ $O(\log \ell \cdot \log(n/r))$
    - 3.3 Else, for each distinct $\alpha \circ \mathrm{T}[j-t, j+l-1]$ in T , find the suffix range of them and recursively apply step 3 on them $O(c \log \ell \log(n/r))$

Total time complexity: $O(|P| + c \log \ell \cdot \log(n/r))$

Total space complexity: $O(r \log(n/r))$

Outline

Conclusion

- We revisited the Contextual Pattern Matching Problem introduced by Navarro
- We improved the space complexity without sacrificing the query time that much
- Our algorithm is independent to any data structures based on the reverse of T
- Our framework is based on the Fully Functional Suffix Trees for a compressed form of T introduced by Gagie et al.
- We provided an $O(r \log(n/r))$ space solution that answers queries in $O(|P| + c \log \ell \cdot \log(n/r))$ time

Final Remarks

- If $r' \leq r$, we can solve the problem in $O(r' \log(n/r'))$ space and $O(m + c \log \ell \cdot \log(n/r'))$ query time
- The same techniques can also be used when different lengths are desired for the strings X and Y occurring in the contextual pattern matches XPY. Letting $\ell_1 = |X|$ and $\ell_2 = |Y|$
- Open problem: How to efficiently answer counting queries?

Thank you!