Speeding up $q$-gram mining on grammar based compressed text

Kyushu University
○Keisuke Goto, Hideo Bannai, Shunsuke Inenaga, Masayuki Takeda
Background: Processing large scale string data

- Data compression allows large scale string data to be stored compactly
Background: Processing large scale string data

- In order to process such data, we usually decompress them, which requires a lot of space and time.
In order to process such data, we usually decompress them, which requires a lot of space and time.
Background: Processing large scale string data

- In order to process such data, we usually decompress them, which requires a lot of space and time.
Background: Processing large scale string data

- One solution is to process compressed strings without explicit decompression.
Background: Processing large scale string data

- One solution is to process compressed strings without explicit decompression.
Grammar-Based Compressed String Processing

<table>
<thead>
<tr>
<th><strong>Problem</strong></th>
<th><strong>Previous Work</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality Test</td>
<td>[Plandowski '94]; Lifshits '07]; [Schmidt-Schauss+ '09];</td>
</tr>
<tr>
<td>Pattern Match</td>
<td>[Karpinski+ '97], [Miyazaki+ '97], [Inenaga+ '04], [Lifshits '06], [Gawrychowski '11]</td>
</tr>
<tr>
<td>Approximate Pattern Match</td>
<td>[Matsumoto+ '00]; [Navarro+ '01]</td>
</tr>
<tr>
<td>Subsequence Match</td>
<td>[Cegielski+ '00]; [Tiskin '09]; [Yamamoto+ '11]</td>
</tr>
<tr>
<td>Longest Common Subsequence / Edit Distance</td>
<td>[Tiskin '07, '08]; [Hermelin + '09, ’11]</td>
</tr>
<tr>
<td>Pattern Discovery</td>
<td>[Inenaga+ ’09]; [Matsubara+ ’09]</td>
</tr>
<tr>
<td>q-gram Frequencies</td>
<td>[Goto+ ’11]; [Goto+ ’12]</td>
</tr>
</tbody>
</table>
## Grammar-Based Compressed String Processing

<table>
<thead>
<tr>
<th>Problem</th>
<th>Previous Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality Test</td>
<td>[Plandowski '94]; Lifshits '07]; [Schmidt-Schauss+ '09];</td>
</tr>
<tr>
<td>Pattern Match</td>
<td>[Karpinski+ '97], [Miyazaki+ '97], [Inenaga+ '04], [Lifshits '06], [Gawrychowski '11]</td>
</tr>
<tr>
<td>Approximate Pattern Match</td>
<td>[Matsumoto+ '00]; [Navarro+ '01]</td>
</tr>
<tr>
<td>Subsequence Match</td>
<td>[Cegielski+ '00]; [Tiskin '09]; [Yamamoto+ '11]</td>
</tr>
<tr>
<td>Longest Common Subsequence / Edit Distance</td>
<td>[Tiskin '07, '08]; [Hermelin + '09, '11]</td>
</tr>
<tr>
<td>Pattern Discovery</td>
<td>[Inenaga+ '09]; [Matsubara+ '09]</td>
</tr>
<tr>
<td>q-gram Frequencies</td>
<td>[Goto+ '11]; [Goto+ '12]</td>
</tr>
</tbody>
</table>
Main contribution

<table>
<thead>
<tr>
<th>$q$-gram Freq</th>
<th>Uncompressed String (SPIRE 2011)</th>
<th>SLP (This work)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(</td>
<td>T</td>
<td>) = O(2^n)$ time and space</td>
</tr>
</tbody>
</table>

$T$: uncompressed string, $n$: the size of SLP

$\text{dup}(q, D)$: a quantity that represents the amount of redundancy that the SLP D captures with respect to $q$-grams

The algorithm is asymptotically always at least as fast and better in many cases compared to working on the uncompressed string.
Definition

Input: string \( T \), positive integer \( q \)

Output: \( \{(P, \text{Freq}(T, P)) \mid P \in \Sigma^q, \text{Freq}(T, P) > 0\} \)

where \( \text{Freq}(T, P) \) is the number of occurrences of \( P \) in \( T \)
\textit{q-gram frequencies problem}

**Definition**

Input: string $T$, positive integer $q$

Output: $\{(P, \text{Freq}(T, P)) \mid P \in \Sigma^q, \text{Freq}(T, P) > 0\}$

where $\text{Freq}(T, P)$ is \# occurrences of $P$ in $T$

Example

$q = 3$

$T = \text{abaababaab}$
**q-gram frequencies problem**

**Definition**

Input: string \( T \), positive integer \( q \)

Output: \( \{ (P, \text{Freq}(T, P)) \mid P \in \Sigma^q, \text{Freq}(T, P) > 0 \} \)

where \( \text{Freq}(T, P) \) is # occurrences of \( P \) in \( T \)

Example

\[ q = 3 \]

\[ T = \text{abaababaab} \]

- aba
- aba
- aba
- aba
- bab
- bab
- aba
- aba
- baa
- aab
$q$-gram frequencies problem

**Definition**

Input: string $T$, positive integer $q$

Output: $\{(P, \text{Freq}(T, P)) \mid P \in \Sigma^q, \text{Freq}(T, P) > 0\}$

where $\text{Freq}(T, P)$ is # occurrences of $P$ in $T$

Example $q = 3$

$T = \text{abaababaab}$

$F \text{req}(T, \text{"aba"}) = 3$
**q-gram frequencies problem**

**Definition**

Input: string $T$, positive integer $q$

Output: $\{(P, \text{Freq}(T, P)) \mid P \in \Sigma^q, \text{Freq}(T, P) > 0\}$

where $\text{Freq}(T, P)$ is # occurrences of $P$ in $T$

Example

$q = 3$

$T = \text{abaababaab}$

$\text{Freq}(T, \text{“aba”}) = 3$

$\text{Freq}(T, \text{“baa”}) = 2$
q-gram frequencies problem

**Definition**

Input: string \( T \), positive integer \( q \)
Output: \( \{(P, \text{Freq}(T, P)) \mid P \in \Sigma^q, \text{Freq}(T, P) > 0\} \)

where \( \text{Freq}(T, P) \) is \# occurrences of \( P \) in \( T \)

Example  \( q = 3 \)

\[
T = \text{abaababaab}
\]

\[
\begin{align*}
\text{Freq}(T, "aba") & = 3 \\
\text{Freq}(T, "baa") & = 2 \\
\text{Freq}(T, "aab") & = 2
\end{align*}
\]
**q-gram frequencies problem**

**Definition**

Input: string $T$, positive integer $q$

Output: $\{(P, Freq(T, P)) \mid P \in \Sigma^q, Freq(T, P) > 0\}$

where $Freq(T, P)$ is the number of occurrences of $P$ in $T$

Example $q = 3$

$T = \text{abaababaab}$

- $Freq(T, \text{“aba”}) = 3$
- $Freq(T, \text{“baa”}) = 2$
- $Freq(T, \text{“aab”}) = 2$
- $Freq(T, \text{“bab”}) = 1$
Straight Line Program (SLP)

**Definition**

Straight Line Program is a context free grammar in the Chomsky normal form that derives a single string.

\[
X_1 = expr_1, X_2 = expr_2, \ldots, X_n = expr_n
\]

\[
expr_i \in \Sigma \text{ or } expr_i = X_l \cdot X_r (l, r < i)
\]

SLP can represent the output of well-known compression algorithms

- e.g. RE-PAIR, SEQUITUR, LZ78, LZW, LZ77, LZSS
Example of SLP

SLP: $D$

$X_1 = a$
$X_2 = b$
$X_3 = X_1 X_2$
$X_4 = X_1 X_3$
$X_5 = X_3 X_4$
$X_6 = X_4 X_5$
$X_7 = X_6 X_5$

$n = |D| = 7$

Derivation Tree of $D$

$T = \begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
a & a & b & a & b & a & b & a & b & a & b & a & b \\
\end{array}$
Example of SLP

SLP: \( D \)

\[
\begin{align*}
X_1 &= a \\
X_2 &= b \\
X_3 &= X_1 X_2 \\
X_4 &= X_1 X_3 \\
X_5 &= X_3 X_4 \\
X_6 &= X_4 X_5 \\
X_7 &= X_6 X_5
\end{align*}
\]

\( n = |D| = 7 \)

Derivation Tree of \( D \)

Length of the decompressed string can be \( \Theta(2^n) \)
Example of SLP

SLP: $D$

- $X_1 = a$
- $X_2 = b$
- $X_3 = X_1 X_2$
- $X_4 = X_1 X_3$
- $X_5 = X_3 X_4$
- $X_6 = X_4 X_5$
- $X_7 = X_6 X_5$

$n = |D| = 7$

Length of the decompressed string can be $\Theta(2^n)$
$O(qn)$ algorithm for $q$-gram frequencies problem on SLP

[Goto et al., SPIRE 2011]
Important Observation: stabbing

**Definition**

For $X_i = X_l X_r$, $X_i$ stabs an occurrence of $P$ $\iff$ $P$ starts in $X_l$ and ends in $X_r$.
Important Observation: stabbing

**Definition**

For $X_i = X_l X_r$, $X_i$ stabs an occurrence of $P \iff P$ starts in $X_l$ and ends in $X_r$. 

$q = 3$

$T = a\ a\ b\ a\ b\ a\ a\ \ b\ a\ a\ b\ a$
Important Observation: stabbing

**Definition**

For $X_i = X_l X_r$, $X_i$ stabs an occurrence of $P$ $\iff$ $P$ starts in $X_l$ and ends in $X_r$.

$q = 3$

$T = \begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
\text{a} & \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{b} & \text{b} \\
\end{array}$
Important Observation: stabbing

**Definition**

For $X_i = X_l X_r$, $X_i$ stabs an occurrence of $P \iff P$ starts in $X_l$ and ends in $X_r$.

**Observation**

For each occurrence of $q$-gram $P$, there exists a unique variable which stabs the occurrence of $P$. 

$q = 3$
Important idea: counting stabbed occurrences

We can compute $Freq(T, P)$ by counting the number of occurrences of $P$ stabbed by $X_i$, and summing them up for all $X_i$

$$Freq(T, P) = 2 \cdot 1 + 1 + 1$$

$q = 3$

$T = 1 \ a \ a \ b \ a \ b \ a \ a \ b \ a \ b \ a \ a \ b$
More formal description

**Definition**

For each variable $X_i$,
- $\text{Freq}_P(X_i, P)$: # occurrences of $P$ stabbed by $X_i$ in the string derived from $X_i$.
- $v\text{Occ}(X_i)$: # nodes labeled by $X_i$ in the derivation tree of the last variable $X_n$. 
More formal description

**Definition**

For each variable $X_i$,

- $Freq(X_i, P)$: # occurrences of $P$ stabbed by $X_i$ in the string derived from $X_i$.

- $vOcc(X_i)$: # nodes labeled by $X_i$ in the derivation tree of the last variable $X_n$.

串(Kushi): Japanese skewer, used to stab foods
More formal description

**Definition**

For each variable $X_i$,
- $Freq(X_i, P)$: # occurrences of $P$ stabbed by $X_i$ in the string derived from $X_i$.
- $vOcc(X_i)$: # nodes labeled by $X_i$ in the derivation tree of the last variable $X_n$.

串 (Kushi):  
Japanese skewer, used to stab foods

$X_n$
More formal description

**Definition**

For each variable $X_i$,
- $\text{Freq}(X_i, P) : \# \text{ occurrences of } P \text{ stabbed by } X_i \text{ in the string derived from } X_i$.
- $\text{vOcc}(X_i) : \# \text{ nodes labeled by } X_i \text{ in the derivation tree of the last variable } X_n$.

\[
\begin{align*}
\text{Freq}(X_i, P) &= 3, \\
\text{Freq}(X_j, P) &= 1, \\
\text{vOcc}(X_i) &= 2, \\
\text{vOcc}(X_j) &= 1.
\end{align*}
\]

Frequency of $P = 3 \cdot 2 + 1 \cdot 1 = 7$
More formal description

Definition

For each variable $X_i$,
- $Freq(X_i, P)$: \# occurrences of $P$ stabbed by $X_i$ in the string derived from $X_i$.
- $vOcc(X_i)$: \# nodes labeled by $X_i$ in the derivation tree of the last variable $X_n$.

Lemma

$$Freq(T, P) = \sum_{i=1}^{n} Freq(X_i, P) \cdot vOcc(X_i)$$

$Freq(X_i, P) = 3$, $Freq(X_j, P) = 1$
$vOcc(X_i) = 2$, $vOcc(X_j) = 1$
Frequency of $P = 3 \cdot 2 + 1 \cdot 1 = 7$
Computing $Freq^\text{串}(X_i, P)$
Computing $Freq_{	ext{串}}(X_i, P)$

$X_i$ stabs $P$ $\iff$ $P$ starts in $X_l$ and ends in $X_r$

$q$-grams stabbed by $X_i$
Computing $Freq(X_i, P)$

**Observation**

For any $P \in \Sigma^q$, $Freq(X_i, P) = Freq(t_i, P)$
Computing $Freq_{串}(X_i, P)$ by $Freq(t_i, P)$

**Lemma**

$$Freq(T, P) = \sum_{i=1}^{n} Freq(t_i, P) \cdot vOcc(X_i)$$

$q$-grams stabbed by $X_i$
Computing frequencies by $\text{Freq}(t_i, P)$ and $\nu\text{Occ}(X_i)$

Lemma

$$\text{Freq}(T, P) = \sum_{i=1}^{n} \text{Freq}(t_i, P) \cdot \nu\text{Occ}(X_i)$$

$O(n)$ time and space in total

$O(qn)$ time and space in total
Computing frequencies by $Freq(t_i, P)$ and $vOcc(X_i)$

**Lemma**

\[ Freq(T, P) = \sum_{i=1}^{n} Freq(t_i, P) \cdot vOcc(X_i) \]

**Theorem**

SLP $q$-gram Frequencies Problem can be solved in $O(qn)$ time and space.

**Sketch of proof:**
Using the suffix array of the concatenation of all $t_i$'s, we can compute all $q$-gram frequencies in $O(qn)$ time and space.
Efficiency & Inefficiency of $O(qn)$ algorithm

Total length of decompressed strings $t_i$

ENGLISH data of 200MB from pizza & chili corpus
Efficiency & Inefficiency of $O(qn)$ algorithm

ENGLISH data of 200MB from pizza & chili corpus

- when $q$ is small, the algorithm runs faster
Efficiency & Inefficiency of $O(qn)$ algorithm

- when $q$ is small, the algorithm runs faster
- when $q$ is large, the algorithm runs slower

ENGLISH data of 200MB from pizza & chili corpus
New algorithm
New Algorithm

Inefficiency of $O(qn)$ algorithm

- Total length of decompressed strings $t_i$ can be larger than $|T|$. 

$$T = \begin{array}{cccccccc}
T_i = & a & a & b & a & b & a & b & a & b & a & a & b & a & a & b
\end{array}$$

$q = 3$
New Algorithm

Inefficiency of $O(qn)$ algorithm

- There are overlaps between partially decompressed strings $t_i$

$q = 3$

$t_4$ and $t_6$ overlap with “ab”
New Algorithm

Inefficiency of $O(qn)$ algorithm

There are overlaps between partially decompressed strings $t_i$

$t_6$ and $t_5$ overlap with “ab”
New Algorithm

Identifying the redundancies

- Consider all partially decompressed strings $t_i$ in derivation tree
Removing overlaps of neighboring $t_i$’s

- Eliminate length-$(q-1)$ prefix of all $t_i$’s except for leftmost one
Removing overlaps of neighboring $t_i$’s

- Concatenation of remaining strings equals to $T$

$$T = \text{aababaababaab}$$
Removing duplicate $t_i$’s

- For all partially eliminated $t_i$, remove all but first occurrence
What we have left

- Compact representation of all $t_i$'s

$q = 3$

$T = \text{a a b a b a a b a b a b a b}$
What we have left

- Compact representation of all $t_i$’s
New Algorithm

What we have left

- Compact representation of all $t_i$'s

$q = 3$

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

$t_6$
New Algorithm

What we have left

- Compact representation of all $t_i$'s

$q = 3$

$T = \text{[sequence of symbols]}$

$t_5 \circ$
New Algorithm

What we have left

- Compact representation of all $t_i$'s

$q = 3$

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

$t_7$
Neighbor tree

- Edge from $X_i$ to $X_j \iff t_i$ and $t_j$ are neighboring

$q = 3$

$T = \begin{array}{cccccccc}
\text{a} & \text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a} & \text{b} \\
\end{array}$
New Algorithm

Neighbor tree

- Edge from $X_i$ to $X_j \Leftrightarrow t_i$ and $t_j$ are neighboring
New Algorithm

**Neighbor tree**

- Edge from $X_i$ to $X_j$ $\iff t_i$ and $t_j$ are neighboring

$$q = 3$$

![Diagram: Neighbor tree with nodes $X_1$ to $X_7$ connected by edges labeled with alphabet $a$ and $b$. The string $T = ababaa$ is traversed through the tree.]
New Algorithm

Neighbor tree

- Edge from \( X_i \) to \( X_j \) ⇔ \( t_i \) and \( t_j \) are neighboring
Size of neighbor tree

- Edge from $X_i$ to $X_j \iff t_i$ and $t_j$ are neighboring

**Lemma**

The total length of edge labels in neighbor tree of $G$ is

$$(q-1) + \sum \{|t_i| - (q-1) | |X_i| \geq q, i = 1, ..., n\}$$

$$= |T| - \text{dup}(q, D)$$

where $\text{dup}(q, D) = \sum \{(vOcc(X_i) - 1) \cdot (|t_i| - (q - 1)) | |X_i| \geq q, i = 1, \ldots, n\}$
Summary of Improved algorithm

**Lemma**

The neighbor tree from SLP $D$ can be constructed in $O(\min\{qn, |T|\cdot \text{dup}(q, D)\})$
Summary of Improved algorithm

Lemma
The neighbor tree from SLP $D$ can be constructed in $O(\min\{qn, |T|-\text{dup}(q, D)\})$

Lemma [Shibuya, 2003]
The suffix tree for a trie can be constructed in time linear in its size
Summary of Improved algorithm

**Lemma**

The neighbor tree from SLP $D$ can be constructed in $O(\min\{qn, |T|\text{-}\text{dup}(q, D)\})$

**Lemma [Shibuya, 2003]**

The suffix tree for a trie can be constructed in time linear in its size.

**Theorem**

The $q$-gram frequencies problem on a SLP $D$ of size $n$, representing string $T$ can be solved in $O(\min\{qn, |T|\text{-}\text{dup}(q, D)\})$ time and space.
Example of ENGLISH data of 200MB from pizza & chili corpus

New Algorithm

Preliminary Experiment (ENGLISH 200MB)
size of neighbor tree and $\Sigma |t_i|$
## Summary

<table>
<thead>
<tr>
<th>$q$-gram Freq</th>
<th>Uncompressed String (SPIRE 2011)</th>
<th>SLP (This work)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(</td>
<td>T</td>
<td>) = O(2^n)$ time and space</td>
</tr>
</tbody>
</table>

**Future work:**
Other applications of neighbor tree
(e.g. one paper accepted to SPIRE 2012)