New Graph Model for Consistent Superstring Problems

Kunsoo Park
Seoul National University

Joint work with J.C. Na, S.H. Cho, S. Choi, J.W. Kim and J.S. Sim
Outline

- Introduction
- Problem Definition
- New Graph Model
- Algorithms
- Conclusion
Common Superstring

- **Input:** string set \( P = \{x_1, x_2, \ldots, x_p\} \) over \( \Sigma \)

- **Common superstring of** \( P \)
  - String that includes every string \( x_i \) as a substring
  - Ex) \( P = \{ab, bb\} \) over \( \Sigma = \{a, b\} \)
  - Common superstrings of \( P \)
    \[
    \{abb, aabb, abba, abbb, babb, bbab, baabb, \ldots\}
    \]
Common Non-Superstring

- **Input**: string set \( N = \{y_1, y_2, ..., y_n\} \) over \( \Sigma \)

- **Common non-superstring (CNSS) of** \( N \)
  - String that does not include any string \( y_i \) as a substring
  - Ex) \( N = \{aaa, aba, bba, bbb\} \) over \( \Sigma = \{a, b\} \)
  - Common non-superstrings of \( N \)
    \[
    \{\lambda, a, b, aa, ab, ba, bb, aab, abb, baa, bab, aabb, baab, babb, baabb\}
    \]
Consistent Superstring

- **Input:** Positive string set $P = \{x_1, x_2, \ldots, x_p\}$ and negative string set $N = \{y_1, y_2, \ldots, y_n\}$ over $\Sigma$

- **Consistent superstring (CSS)** of $P$ and $N$
  - String that is both a common superstring of $P$ and a common non-superstring of $N$
Example of Consistent Superstrings

$P = \{ab, bb\}$, $N = \{aaa, aba, bba, bbb\}$ over $\Sigma = \{a, b\}$

The set of common superstrings of $P$: $\{aab, aabb, abba, abbb, babb, bbab, baabb, \ldots\}$

The set of common non-superstrings of $N$: $\{\lambda, a, b, aa, ab, ba, bb, aab, abb, baa, bab, aabb, baab, babb, baabb\}$

The set of consistent superstrings of $P$ and $N$: $\{aab, aabb, babb, baabb\}$
CSS Problems

**Input:** positive string set \( P = \{x_1, x_2, \ldots, x_p\} \) and negative string set \( N = \{y_1, y_2, \ldots, y_n\} \) over \( \Sigma \)

1. **Shortest Consistent Superstring (SCSS) Problem**
   
   Output: If \( CSS = \emptyset \), 'No SCSS exists.'
   
   otherwise, an SCSS of \( P \) and \( N \)

2. **Longest Consistent Superstring (LCSS) Problem**
   
   Output: If \( CSS = \emptyset \) or an arbitrarily long CSS can be made,
   
   'No LCSS exists.'
   
   otherwise, an LCSS of \( P \) and \( N \)
Assumptions

- \( P = \{x_1, x_2, ..., x_p\} \) and \( N = \{y_1, y_2, ..., y_n\} \)

1) For all \( x_i \) and \( x_j (i \neq j) \), \( x_i \) is not a substring of \( x_j \). (If \( x_i \) is a substring of \( x_j \), then any superstring of \( x_j \) is a superstring of \( x_i \). Hence, we can remove \( x_i \) from \( P \).)

2) For all \( y_i \) and \( y_j (i \neq j) \), \( y_i \) is not a substring of \( y_j \). (Otherwise, we can remove \( y_j \) from \( N \).)

3) For all \( x_i \) and \( y_j \), \( y_j \) is not a substring of \( x_i \). (Otherwise, no CSS exists.)

4) For all \( x_i \) and \( y_j \), \( x_i \) is not a substring of \( y_j \). (inclusion-free)
Previous Work

- **Jiang-Li (1993)** introduced the notion of CSS in the context of learning strings.

- **Jiang-Timkovsky (1995)**
  - Used a graph model based only on the strings in \( N \)
  - Assumed non-trivial conditions
  - Proposed polynomial time algorithms for finding SCSS and LCSS when \(|P|\) is bounded by a constant
Contributions

- **New graph model**
  - Based on the Aho-Corasick automaton using all the strings in $P$ and $N$
  - Does not assume non-trivial conditions
  - Is more intuitive and leads to simpler algorithms than Jiang-Timkovsky’s

- **Improved algorithms for SCSS and LCSS problems**
  - Our algorithms solve the CSS problems for more cases and/or more efficiently.
Our graph model is related to Aho-Corasick (AC) automaton for multiple pattern matching.

The AC automaton consists of vertices (states) and three functions (transitions): goto function, failure function, output function.

The AC automaton has its DFA version.
AC Automaton for \{aa, aba, abba, bb\}

- Goto function
- Failure function
- Output function

\[
Q(aa) = \{v3\} \\
Q(abba) = \{v8\} \\
Q(aba) = \{v6\} \\
Q(bb) = \{v5, v7\}
\]
DFA Version of AC Automaton

\[ Q(aa) = \{v3\} \]
\[ Q(abba) = \{v8\} \]
\[ Q(aba) = \{v6\} \]
\[ Q(bb) = \{v5, v7\} \]
AC automaton accepts all pattern strings

Finding all occurrences of pattern strings in a text string.

text string: \textit{baaabba}
AC Automaton

AC automaton accepts all pattern strings

Finding all occurrences of pattern strings in a text string.

text string: \textit{baabba}
AC automaton accepts all pattern strings

Finding all occurrences of pattern strings in a text string.

text string: \texttt{baabba}
AC Automaton

AC automaton accepts all pattern strings

Finding all occurrences of pattern strings in a text string.

text sting: baab bba
AC Automaton

AC automaton accepts all pattern strings

Finding all occurrences of pattern strings in a text string.

Text string: \textcolor{red}{baabba}
AC Automaton

AC automaton accepts all pattern strings

Finding all occurrences of pattern strings in a text string.

text string: \textcolor{red}{baabba}

\textcolor{red}{v0} \rightarrow \textcolor{red}{v1} \rightarrow \textcolor{red}{v3} \rightarrow \textcolor{red}{v6} \rightarrow \textcolor{red}{v7} \rightarrow \textcolor{red}{v8}
Our Graph Model

\[ P = \{aba, bb\}, \ N = \{aa, abba\} \]

- **Build AC automaton for** \( P \cup N \)

\[ Q(aa) = \{v3\} \]
\[ Q(abba) = \{v8\} \]
\[ Q(aba) = \{v6\} \]
\[ Q(bb) = \{v5, v7\} \]
Our Graph Model

Remove all negative output states

\[ P = \{aba, bb\}, N = \{aa, abba\} \]

\[ Q(aa) = \{v3\} \]

\[ Q(abba) = \{v8\} \]
We call this graph $G_{CSS}$

$P = \{aba, bb\}, N = \{aa, abba\}$

$Q(aba) = \{v6\}$

$Q(bb) = \{v5, v7\}$
\( P = \{aba, bb\}, N = \{aa, abba\} \)

\( \lambda\text{-path}: \) a path from \( v0 \)

\( \lambda\text{-path}(\alpha): \) a path from \( v0 \) representing string \( \alpha \)

\( \alpha \) is a CNSS of \( N \) \( \Leftrightarrow \) 
\( \lambda\text{-path}(\alpha) \) exists in \( \text{GCSS} \)

ex) \( abbb \) is a common non-superstring of \( N \)

longest CNSS of \( N \) exists \( \Leftrightarrow \) \( \text{GCSS} \) is acyclic
Q-path: a $\lambda$-path which passes at least one vertex in $Q(x_i)$ for every $x_i \in P$

$\alpha$ is a CSS of $P$ and $N$ $\Leftrightarrow$ $\lambda$-path($\alpha$) that is a Q-path exists in GCSS

ex) $ababb$ is a consistent superstring of $P$ and $N$

$Q(aba) = \{v6\}$

$Q(bb) = \{v5, v7\}$
Algorithm for CSS

1. Construct $G_{CSS}$.
2. Find shortest (longest) Q-path in $G_{CSS}$.
3. Compute SCSS (LCSS) if shortest (longest) Q-path is found in step 2.
\( P \cup N \) is inclusion-free

- \(|Q(x_i)| = 1\) for every positive string \( x_i \)
- Case \( G_{CSS} \) is acyclic
- If a Q-path exists, q-vertices must be in a path.
- Such a Q-path can be found by depth-first search (topological sort).

\[
\begin{align*}
v_0 & \rightarrow u_1 & u_2 & \rightarrow ur
\end{align*}
\]
$P \cup N$ is inclusion-free

- Case $G_{CSS}$ is cyclic
- Build $G_{QS}$:
  - vertices: $v_0$ and all q-vertices of $G_{CSS}$
  - Edge $(u, v)$ is defined if there is a path from $u$ to $v$ in $G_{CSS}$ and its weight is the length of shortest path from $u$ to $v$ in $G_{CSS}$
$P \cup N$ is inclusion-free

- Case $G_{CSS}$ is cyclic
- Shortest Q-path in $G_{CSS}$ is shortest path $A_s$ in $G_{QS}$ that starts at $v_0$ and passes over all vertices (SCSS is reduced to TSP)
- If $G_{QS}$ is acyclic, $A_s$ must pass over all vertices of $G_{QS}$ in topological order
$P \cup N$ is not inclusion-free

- Build $G_{QS}$ from $G_{CSS}$
- Shortest Q-path in $G_{CSS}$ is shortest path in $G_{QS}$ that starts at $v_0$ and passes over at least one vertex in every $Q(x_i)$ (SCSS is reduced to Generalized TSP)

$ababb$ is SCSS
Shortest CSS

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Cases</th>
<th>LCNSS of $N$ exists</th>
<th>No LCNSS of $N$ exists</th>
<th>Algorithms</th>
<th>Cases</th>
<th>LCNSS of $N$ exists</th>
<th>No LCNSS of $N$ exists</th>
</tr>
</thead>
<tbody>
<tr>
<td>JT95</td>
<td>IF &amp; final closure</td>
<td>$(O(P + N)^3)$</td>
<td>$(O(P + N)^5 + k^2 P^2)$</td>
<td>JT95</td>
<td>IF &amp; final closure</td>
<td>$(O(P + N)^3)$</td>
<td>$(O(P + N)^5 + k^2 P^2)$</td>
</tr>
<tr>
<td>JT95</td>
<td>IF &amp; final closure</td>
<td>$(O(P + N)^3)$</td>
<td>$(O(P + N)^5 + k^2 P^2)$</td>
<td>JT95</td>
<td>IF &amp; final closure</td>
<td>$(O(P + N)^3)$</td>
<td>$(O(P + N)^5 + k^2 P^2)$</td>
</tr>
</tbody>
</table>

- $O(P)$ is required since $O(P + N)$ is the input size.
- $k$ is the number of all $q$-vertices.
- Even though $P \cup N$ is not inclusion-free, $|Q(x_i)|$ can be 1 for every positive string $x_i$. In this case (Q1) we use the algorithm for case $P \cup N$ is inclusion-free.
Algorithm for LCSS

1. Construct $G_{CSS}$.
2. Find longest Q-path in $G_{CSS}$.
3. Compute LCSS if longest Q-path is found in step 2.
$P \cup N$ is inclusion-free

- Case $G_{CSS}$ is acyclic: similar to SCSS
- Case $G_{CSS}$ is cyclic
- Build $G_{QL}$:
  - vertices: $v_0$, all q-vertices of $G_{CSS}$, and $v_f$
  - Edge $(u, v)$ for $u, v \neq v_f$ is defined if there is a path from $u$ to $v$ in $G_{CSS}$ and its weight is $-1$ multiplied by the length of longest path from $u$ to $v$ in $G_{CSS}$
  - Edge $(u, v_f)$ is always defined and its weight is $-1$ multiplied by the length of longest path from $u$ to any vertex in $G_{CSS}$
\( P \cup N \) is inclusion-free

- Longest Q-path in \( G_{CSS} \) is shortest path in \( G_{QL} \) that starts at \( v_0 \) and passes over all vertices. (\( G_{QL} \) is acyclic or not)

(a) \( G_{CSS} \) and (b) \( G_{QL} \) for \( P = \{bba, bba\} \) and \( N = \{ab, bbb\} \)

Arbitrarily long CSS bbaaaaa ...
$P \cup N$ is not inclusion-free

- Build $G_{QL}$ from $G_{CSS}$
- Longest Q-path in $G_{CSS}$ is shortest path in $G_{QL}$ that starts at $v_0$, and passes over at least one vertex in every $Q(x_i)$, and ends at $v_f$ (LCSS is reduced to Generalized TSP)
Longest CSS

- \( k \) is the number of all q-vertices.
Conclusion

- Simple and intuitive graph model for CSS problems based on Aho-Corasick automaton
- Q-paths have a one-to-one correspondence with CSSs.
- Leads to improved algorithms for SCSS and LCSS problems.
Thank You