New Graph Model and Algorithms for Consistent Superstring Problems

Joong Chae Na

Sejong University

Joint work with S.H. Cho, S. Choi, J.W. Kim, K. Park and J.S. Sim

Outline

- Introduction and background
- Problem Definition
- New Graph Model
- Algorithms
- Conclusion

3 Introduction and Background

String

String

- Sequence of characters over an alphabet Σ
  - alphabet Σ : a set of characters in strings
    - Ex) ASCII, $\{0, 1\}$, $\{A, C, G, T\}$

- Examples
  - DNA sequences over $\Sigma = \{A, C, G, T\}$
  - Binary sequences over $\Sigma = \{0, 1\}$
  - $abaab$ over $\Sigma = \{a, b\}$

Substring and Superstring

- **Substring** of a string x
  - String that is included in x
  - Ex) aa is a substring of $x = baab$

    substrings of x : $\{\lambda, a, b, aa, ab, ba, \sout{bb}, aab, baa, baab\}$

- **Superstring** of a string x
  - String that includes x as a substring
  - Ex) $baab$ is a superstring of $x = aa$

    superstrings of x : $\{aa, aaa, aab, \sout{aba}, \sout{abb}, baa, aaaa, \ldots\}$

Common Substring

- **Input:** string set $P = \{x_1, x_2, \ldots, x_p\}$ over Σ

- **Common substring of P**
  - String that is included in every string x_i
  - Ex) $P = \{ababbabb, bbabaaba\}$ over $\Sigma = \{a, b\}$
  - Common substrings of P

    $\{\lambda, a, b, ab, ba, bb, aba, bab, bba, bbab\}$

- **Longest common substring problem**
  - Solvable in polynomial time

Common Superstring

- **Input:** string set $P = \{x_1, x_2, \dots, x_p\}$ over Σ

- **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

    $\{abb, aabb, abba, abbb, babb, bbab, baabb, \dots\}$

- **Shortest common superstring problem**
  - NP-hard

Common Non-Substring

- **Input:** string set $N = \{y_1, y_2, \dots, y_n\}$ over Σ

- **Common non-substring of N**
  - String that isn't included in any string y_i
  - Ex) $N = \{abb, baba\}$ over $\Sigma = \{a, b\}$
  - Common non-substrings of N

    $\{aa, aaa, aab, baa, bba, bbb, aaaa, aaab, aaba, aabb, abaa, \dots\}$

- **Shortest common non-substring problem**
  - Solvable in polynomial time

Common Non-Superstring

- **Input:** string set $N = \{y_1, y_2, \dots, y_n\}$ over Σ

- **Common non-superstring 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

$$\{\lambda, a, b, aa, ab, ba, bb, aab, abb, baa, bab, aabb, baab, babb, baabb\}$$

- **Longest common non-superstring problem**
  - Solvable in polynomial time

Inclusion or Non-Inclusion

- (Longest) common substrings
- (Shortest) common superstrings (NP-hard)
- (Shortest) common non-substrings
- (Longest) common non-superstrings

- **Applications**
  - Data compression, molecular biology, computer security

Inclusion and Non-Inclusion

- (Longest) common substrings
- **(Shortest) common superstrings (NP-hard)**
- Shortest common non-substrings
- **(Longest) common non-superstrings**

- Problem considering both inclusion and non-inclusion
  - ➔ **Consistent Superstring**

12

Problem Definition

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 Σ

- **Consistent superstring** (CSS) of P and N
  - String that is both a common superstring of P and a common non-superstring of N
  - Applications: DNA sequencing, data compression, security

Example of Consistent Superstrings

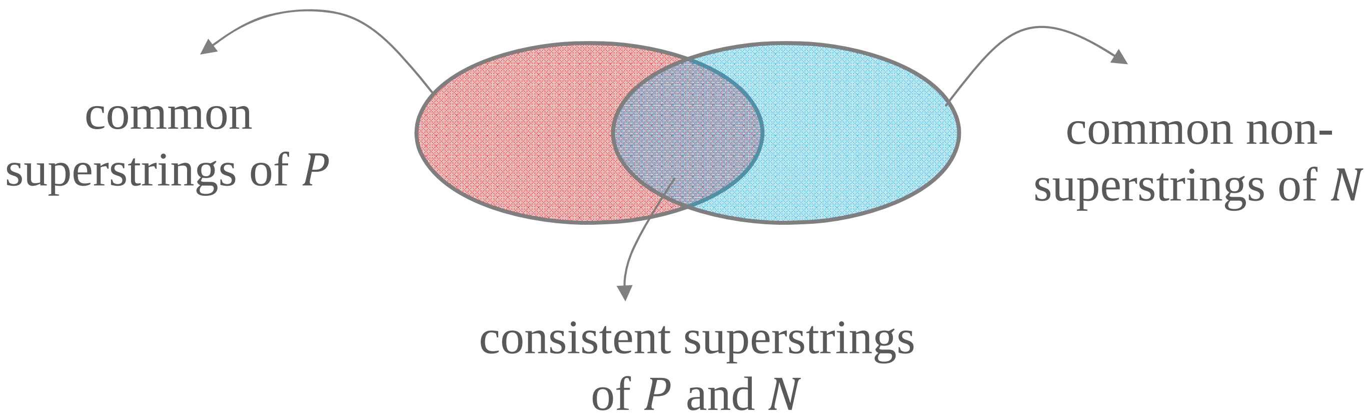
$P = \{ab, bb\}$, $N = \{aaa, aba, bba, bbb\}$ over $\Sigma = \{a, b\}$

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

$\cap$

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 Σ

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

- $P = \{x_1, x_2, \dots, x_p\}$ and $N = \{y_1, y_2, \dots, y_n\}$
  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 (1994)** introduced the notion of CSS in the context of learning strings (DNA sequencing, etc.)
- **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.

19

New Graph Model

Graph Model

- 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}

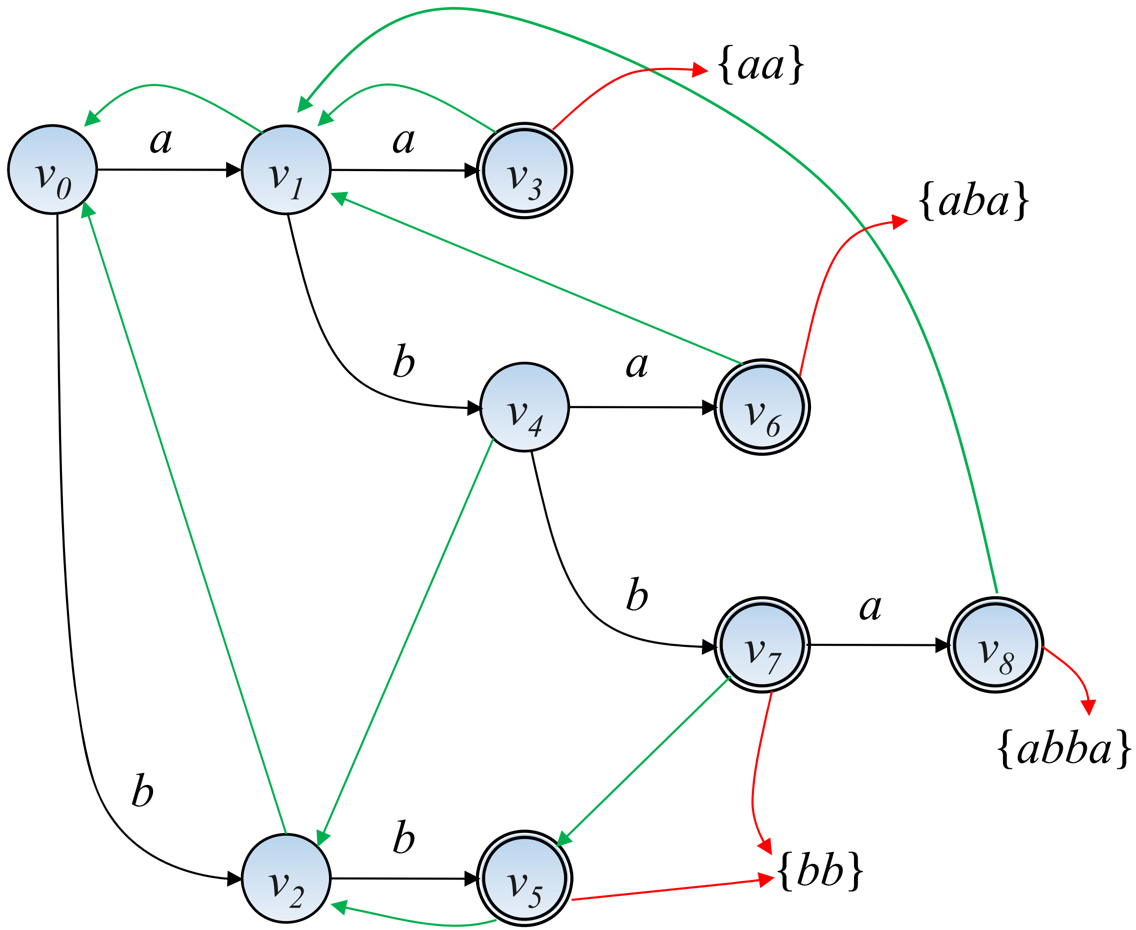
- Goto function
- Failure function
- Output function

$Q(aa) = \{v_3\}$

$Q(abba) = \{v_8\}$

$Q(aba) = \{v_6\}$

$Q(bb) = \{v_5, v_7\}$



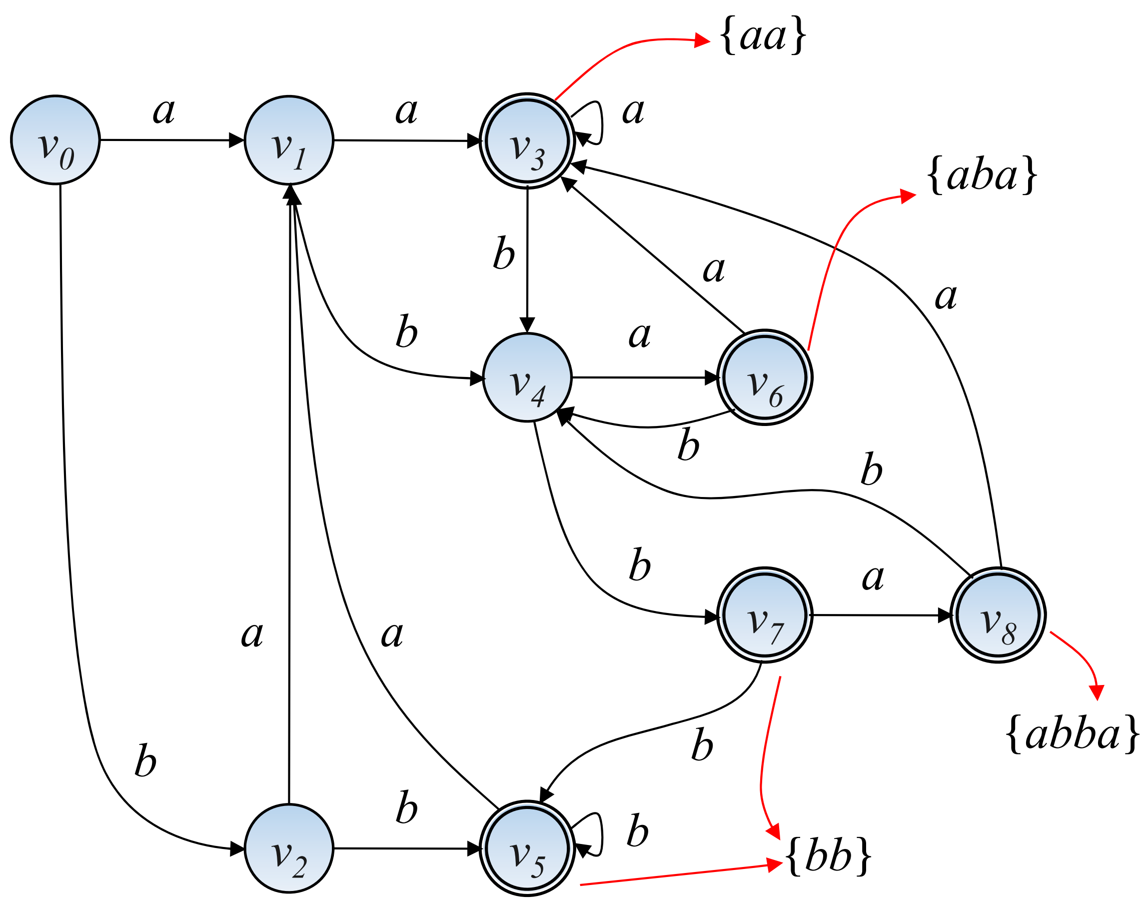
DFA Version of AC Automaton

$Q(aa) = \{v_3\}$

$Q(abba) = \{v_8\}$

$Q(aba) = \{v_6\}$

$Q(bb) = \{v_5, v_7\}$



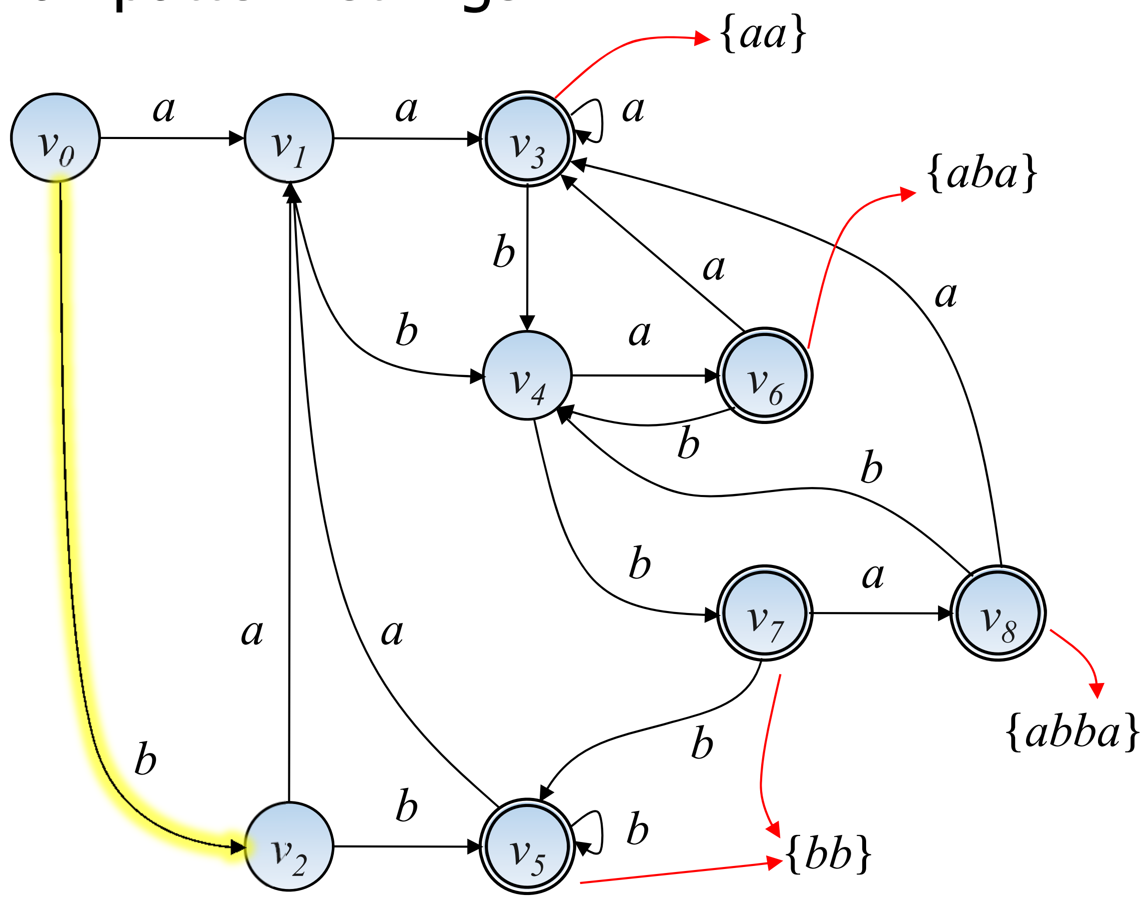
AC Automaton

AC automaton accepts all pattern strings

Finding all occurrences of pattern strings in a text string.

text sting:

b*aabba*



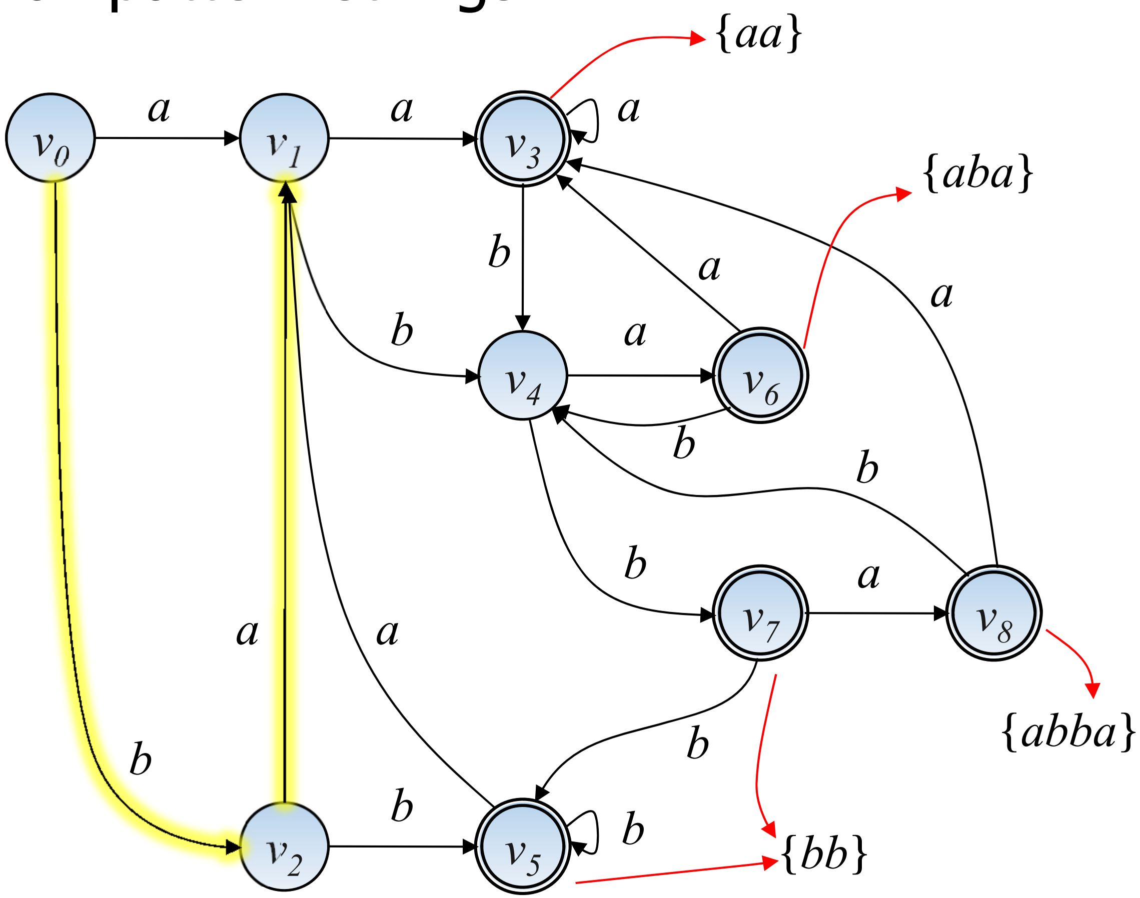
AC Automaton

AC automaton accepts all pattern strings

Finding all occurrences of pattern strings in a text string.

text sting:

***ba**abba*



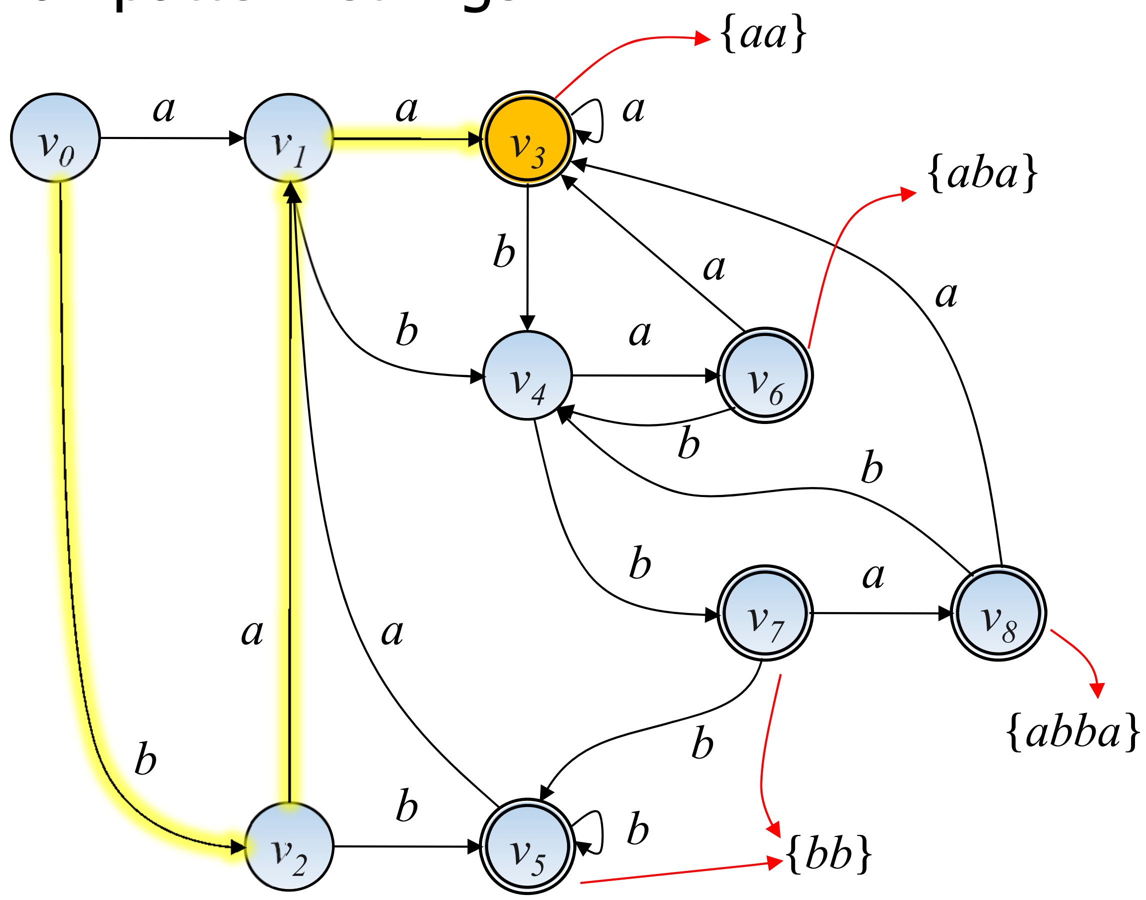
AC Automaton

AC automaton accepts all pattern strings

Finding all occurrences of pattern strings in a text string.

text sting:

ba*a*bba



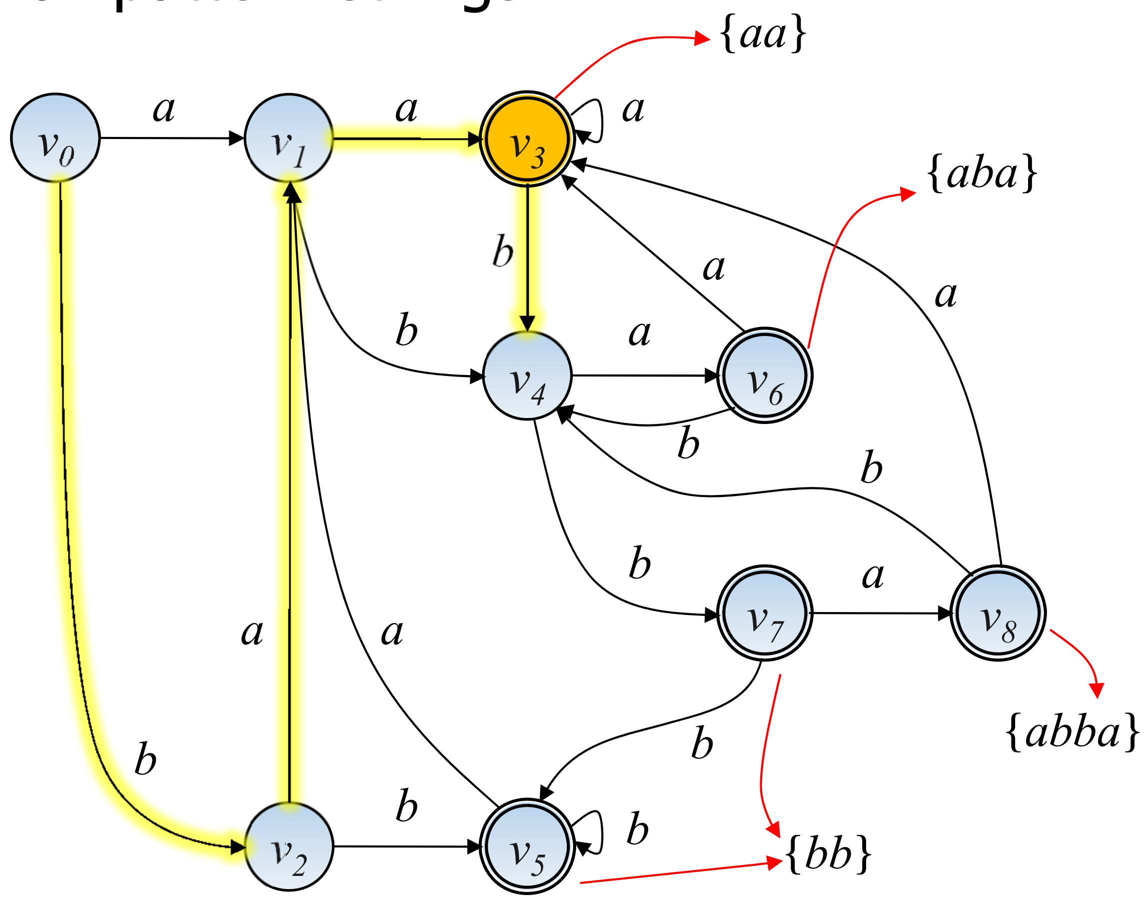
AC Automaton

AC automaton accepts all pattern strings

Finding all occurrences of pattern strings in a text string.

text sting:

***baab**ba*



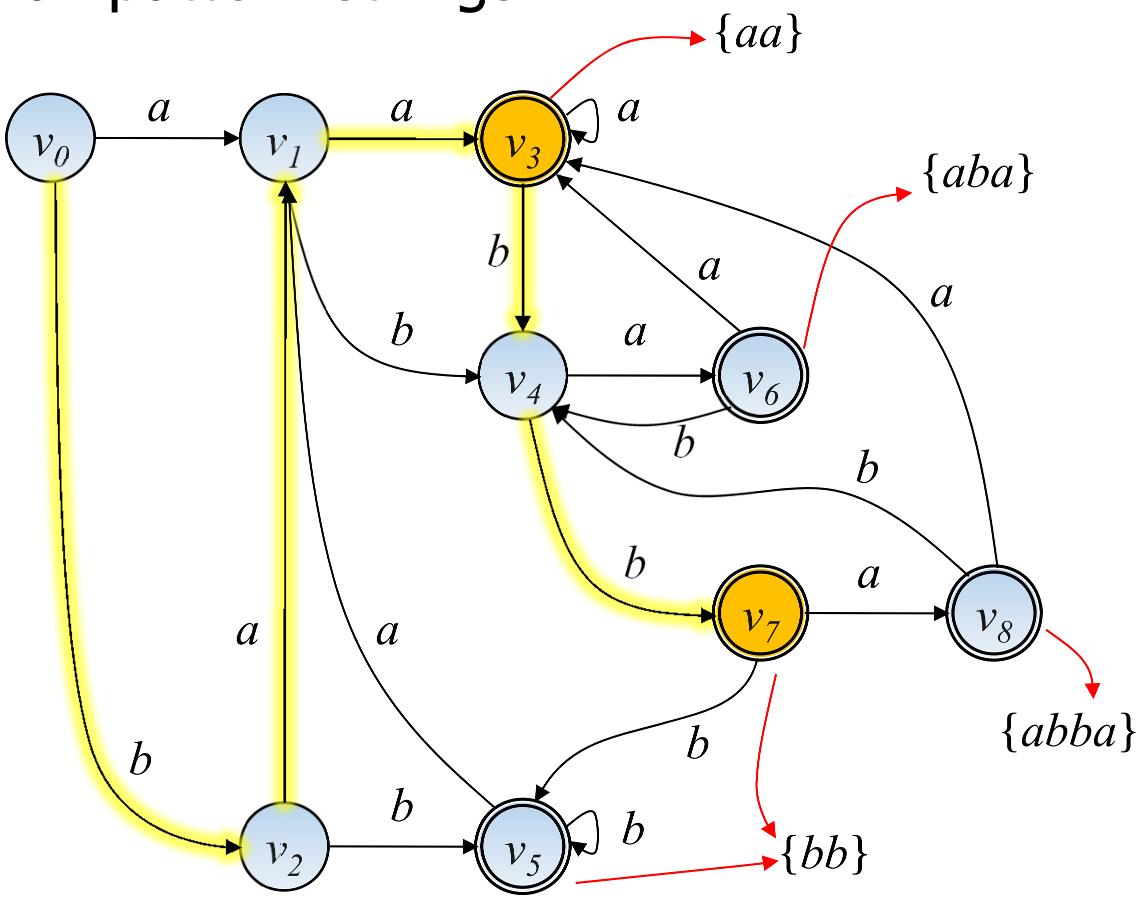
AC Automaton

AC automaton accepts all pattern strings

Finding all occurrences of pattern strings in a text string.

text sting:

*baab***b***a*



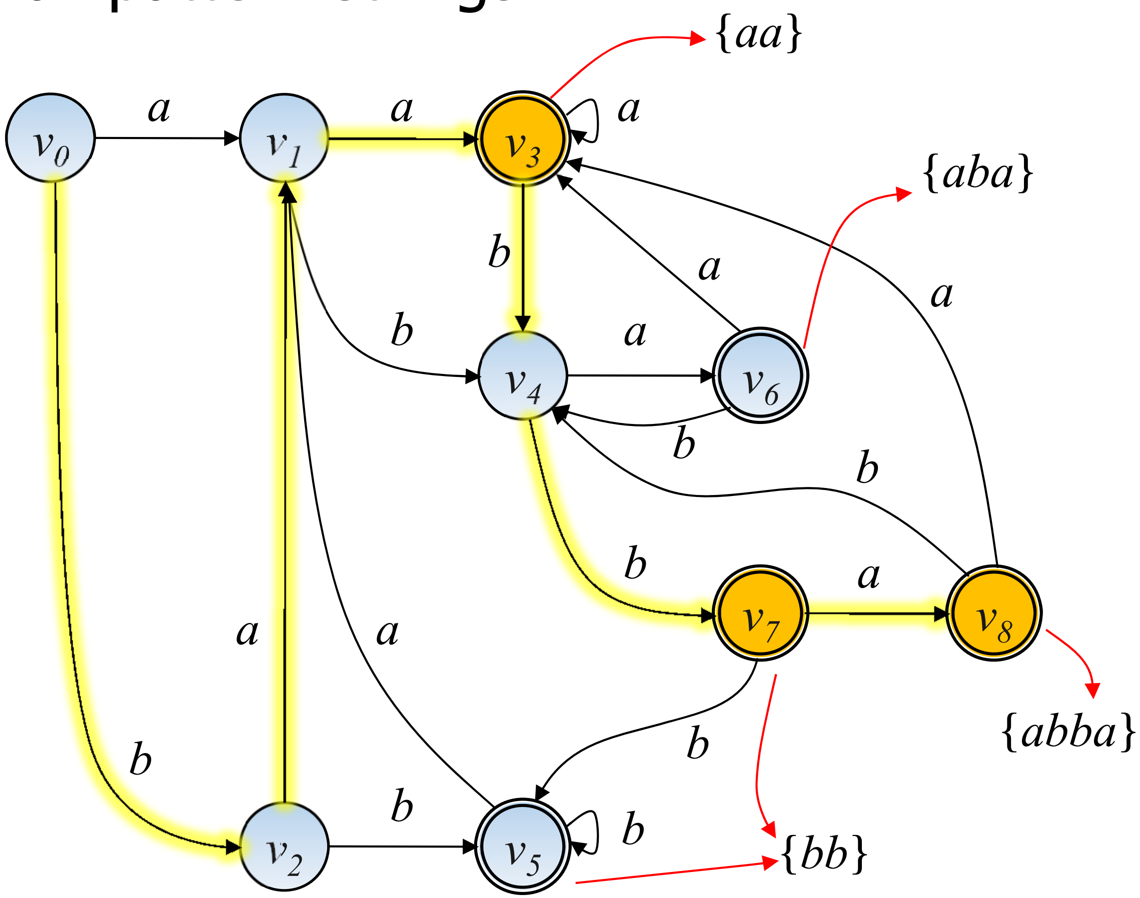
AC Automaton

AC automaton accepts all pattern strings

Finding all occurrences of pattern strings in a text string.

text sting:

baabba



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

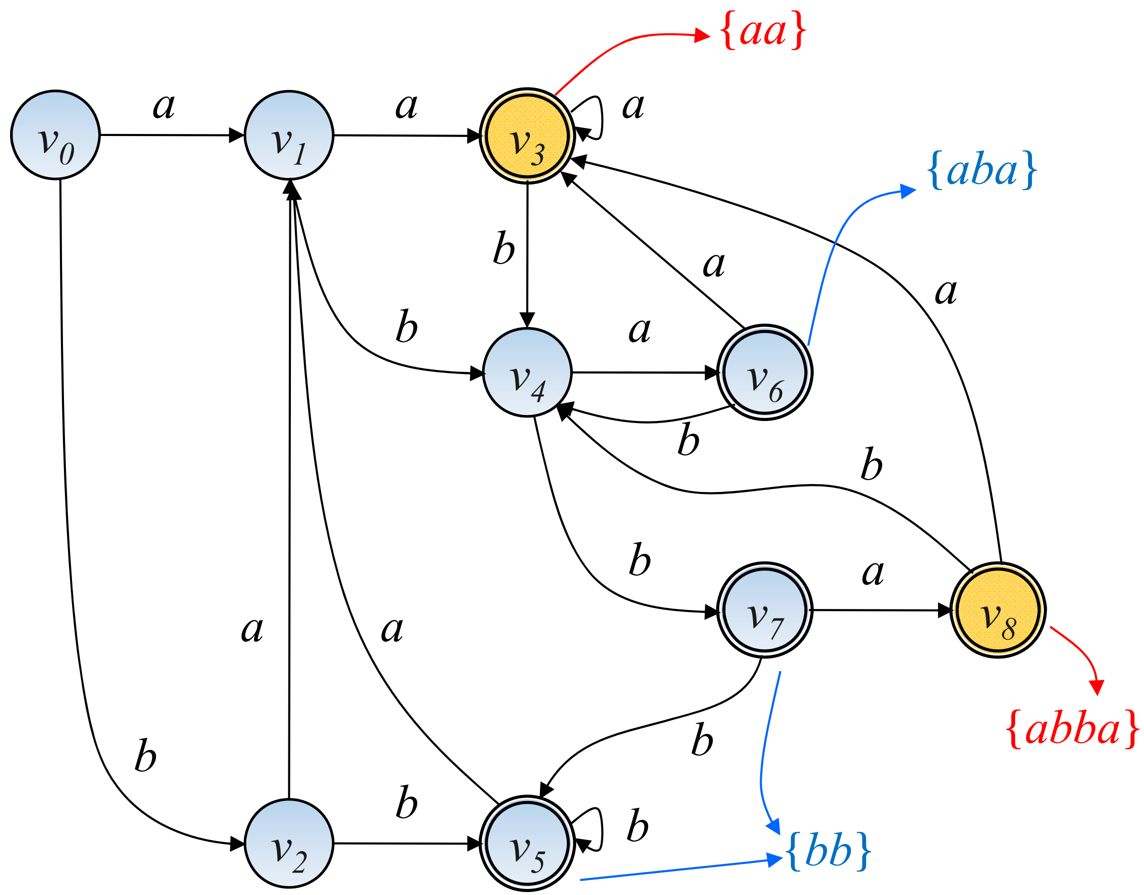
Build AC automaton for $P \cup N$

$Q(aa) = \{v_3\}$

$Q(abba) = \{v_8\}$

$Q(aba) = \{v_6\}$

$Q(bb) = \{v_5, v_7\}$

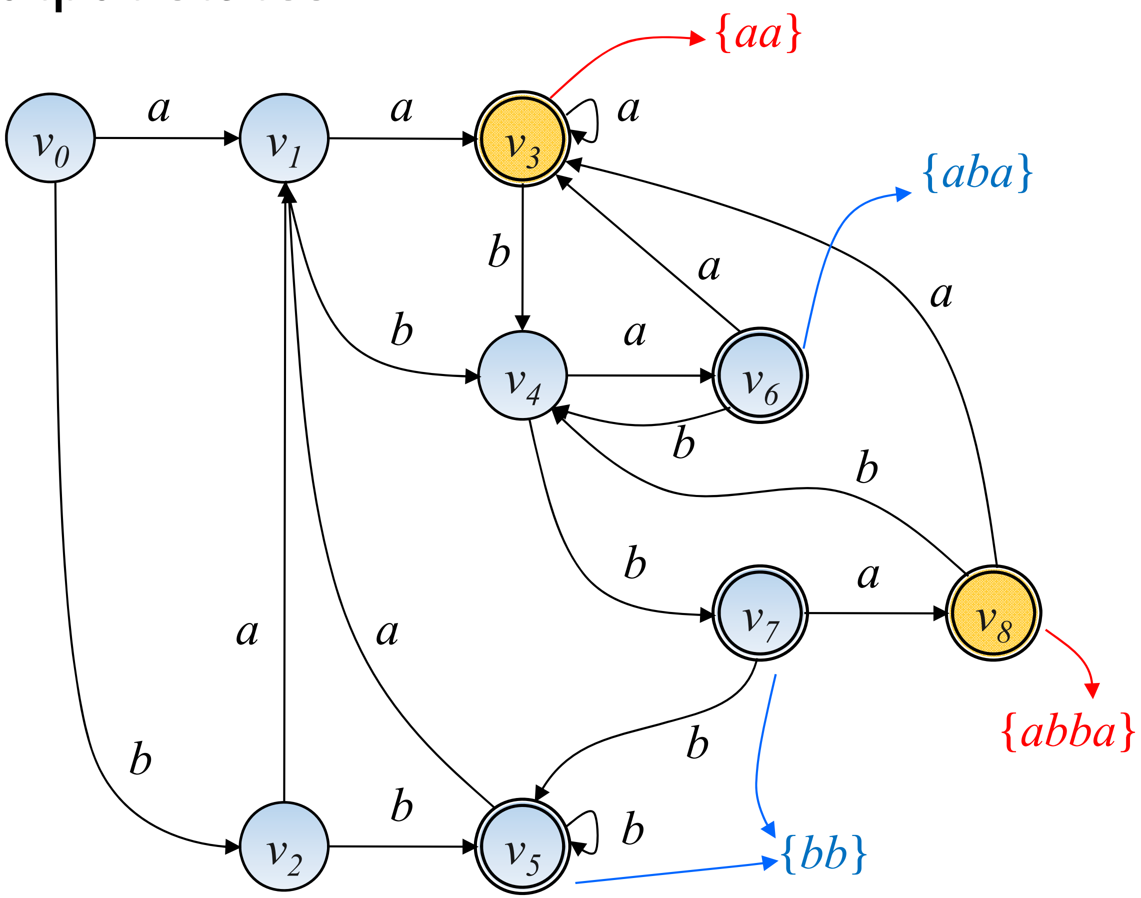


Our Graph Model $P = \{aba, bb\}, N = \{aa, abba\}$

Remove all negative output states

$Q(aa) = \{v_3\}$

$Q(abba) = \{v_8\}$



G_{CSS}

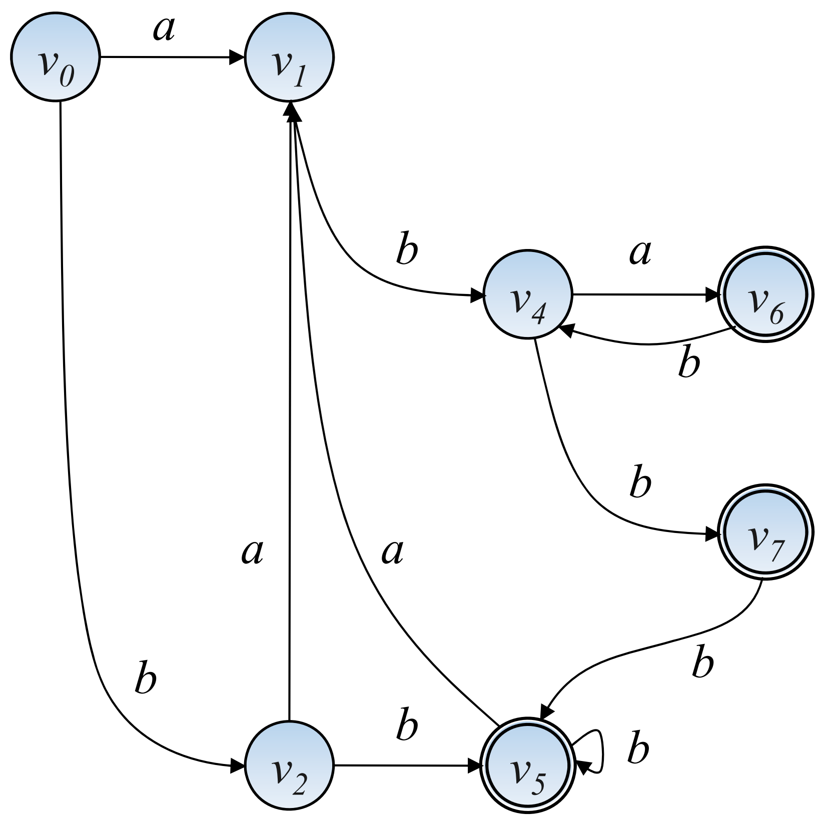
$P = \{aba, bb\}, N = \{aa, abba\}$

We call this graph G_{CSS}

q-vertex

$Q(aba) = \{v_6\}$

$Q(bb) = \{v_5, v_7\}$



G_{CSS}

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

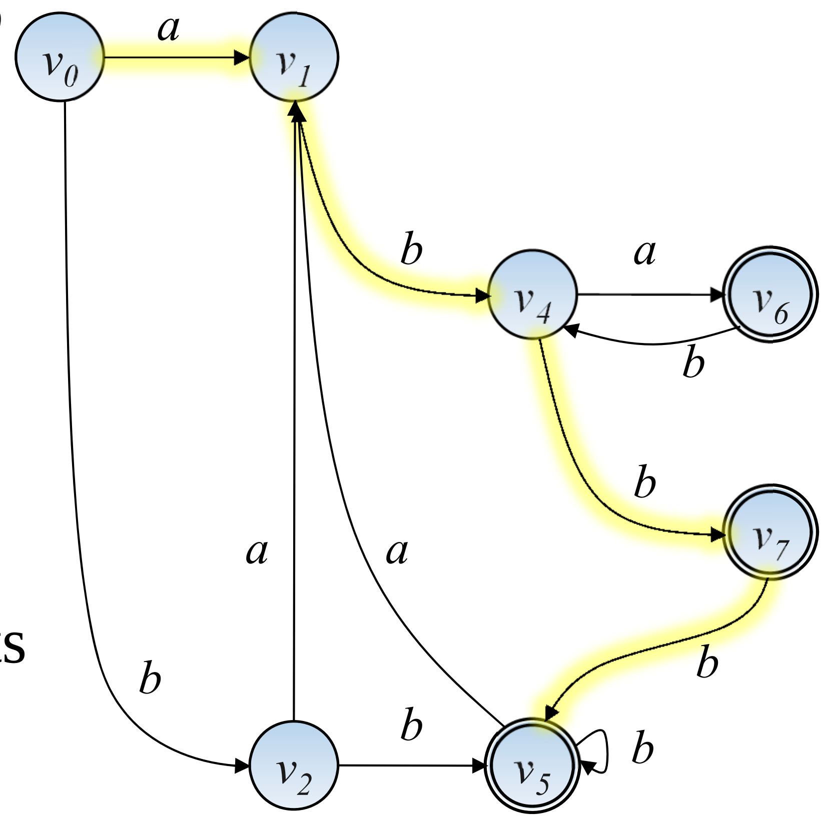
λ-*path*: a path from v_0

λ-*path*(α): a path from v_0 representing string α

α is a CNSS of $N \Leftrightarrow$ λ-*path*(α) exists in G_{CSS}

ex) *abbb* is a common non-superstring of N

longest CNSS of N exists $\Leftrightarrow$ G_{CSS} is acyclic



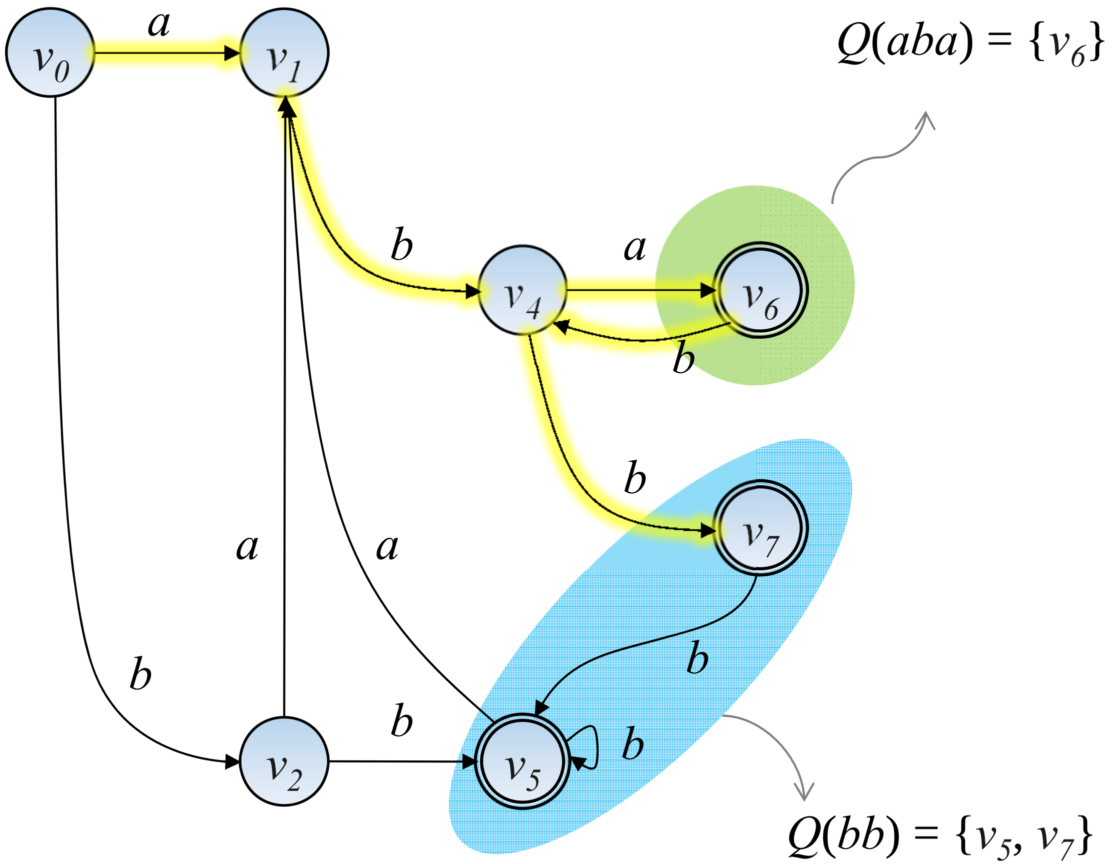
G_{CSS}

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

Q-path: a λ-path which passes at least one vertex in $Q(x_i)$ for every $x_i \in P$

α is a CSS of P and N $\Leftrightarrow$ $\lambda\text{-}path(\alpha)$ that is a Q-path exists in G_{CSS}

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

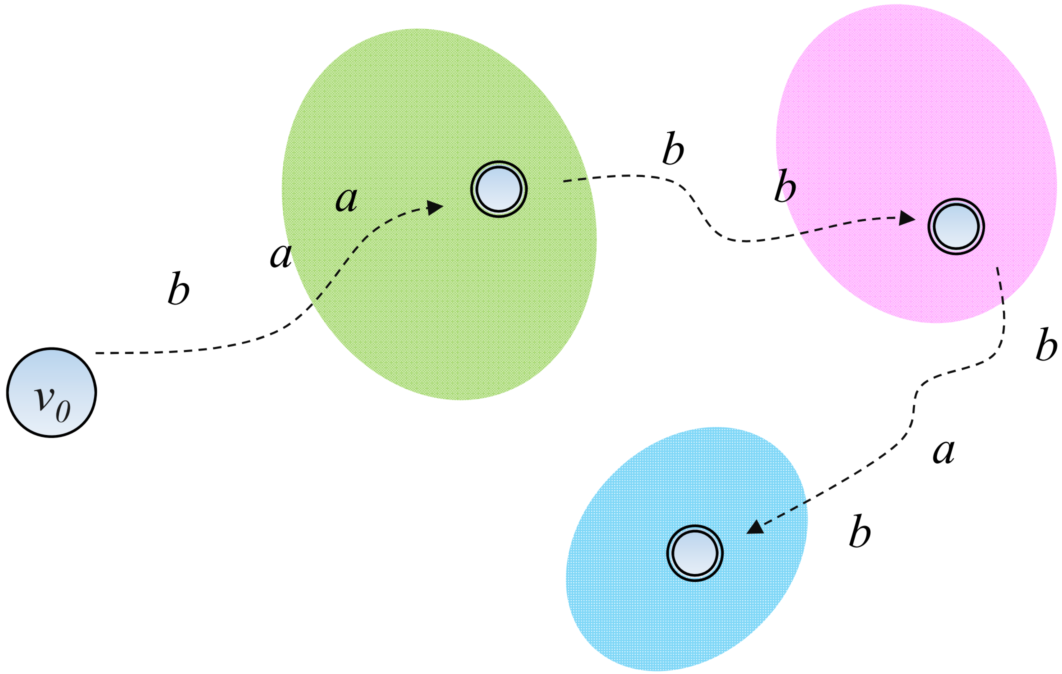


34

Improved Algorithms

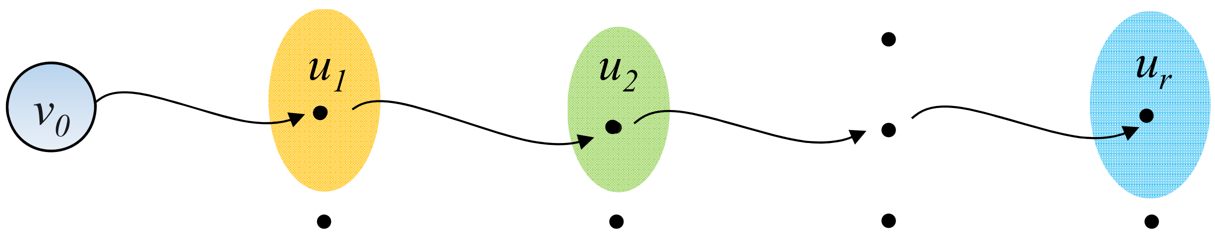
Algorithm for CSS

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.



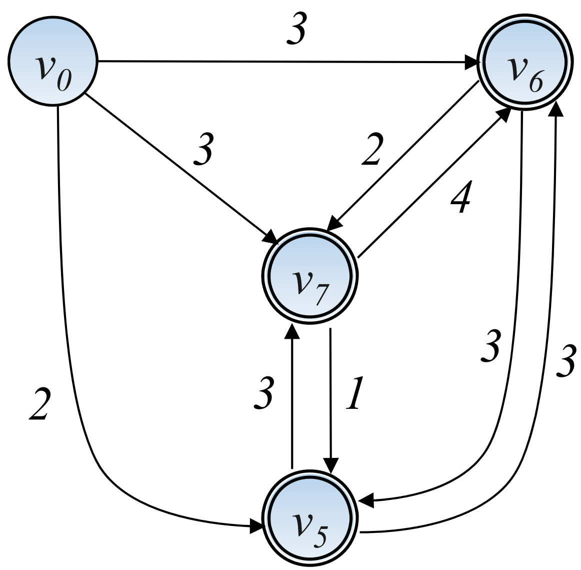
$P \cup N$ is inclusion-free

- $|Q(x_i)| = 1$ 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).



$P \cup N$ 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}

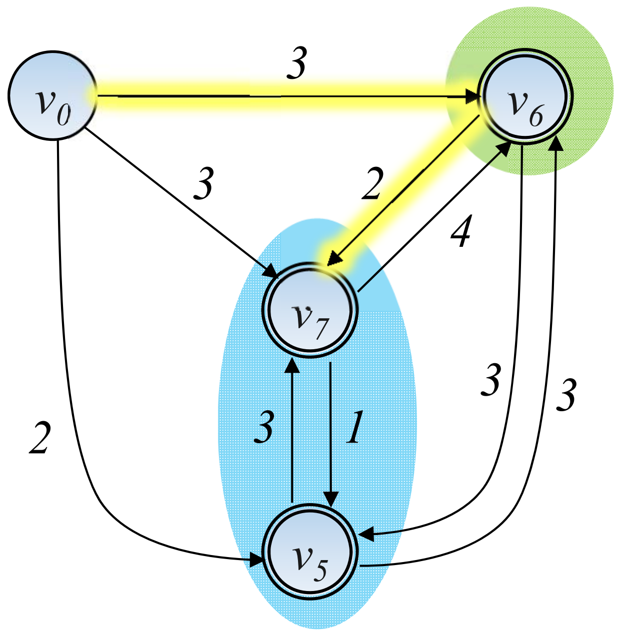


$P \cup N$ is inclusion-free

- 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

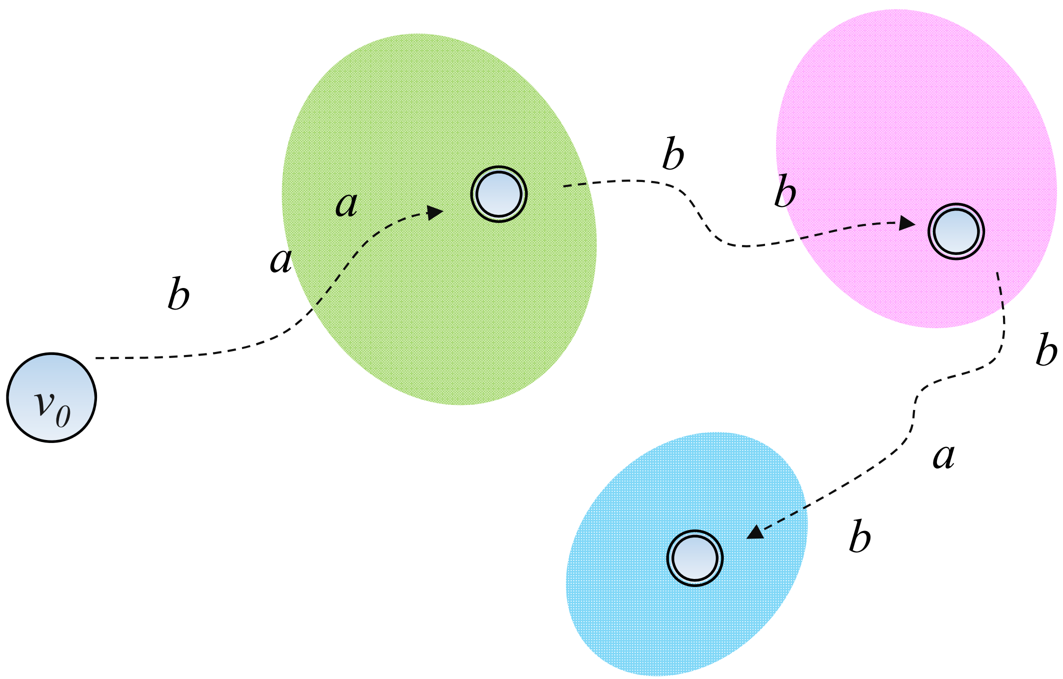
Shortest CSS

Algorithms	Cases		LCNSS of N exists	No LCNSS of N exists	
				G_{QS} is acyclic	G_{QS} is cyclic
JT95	IF & final closure		$O(p^2 + pN^2)^\dagger$	$O(pN^2 + p^2 2^p)^\dagger$	
Ours	IF	Q1	$O(P + N)$	$O(p(P + N))$	$O(p(P + N) + p^2 2^p)$
	~IF	Q1			
		~Q1	$O(k(P + N) + k^2 2^p)$		

- † $O(P)$ 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.

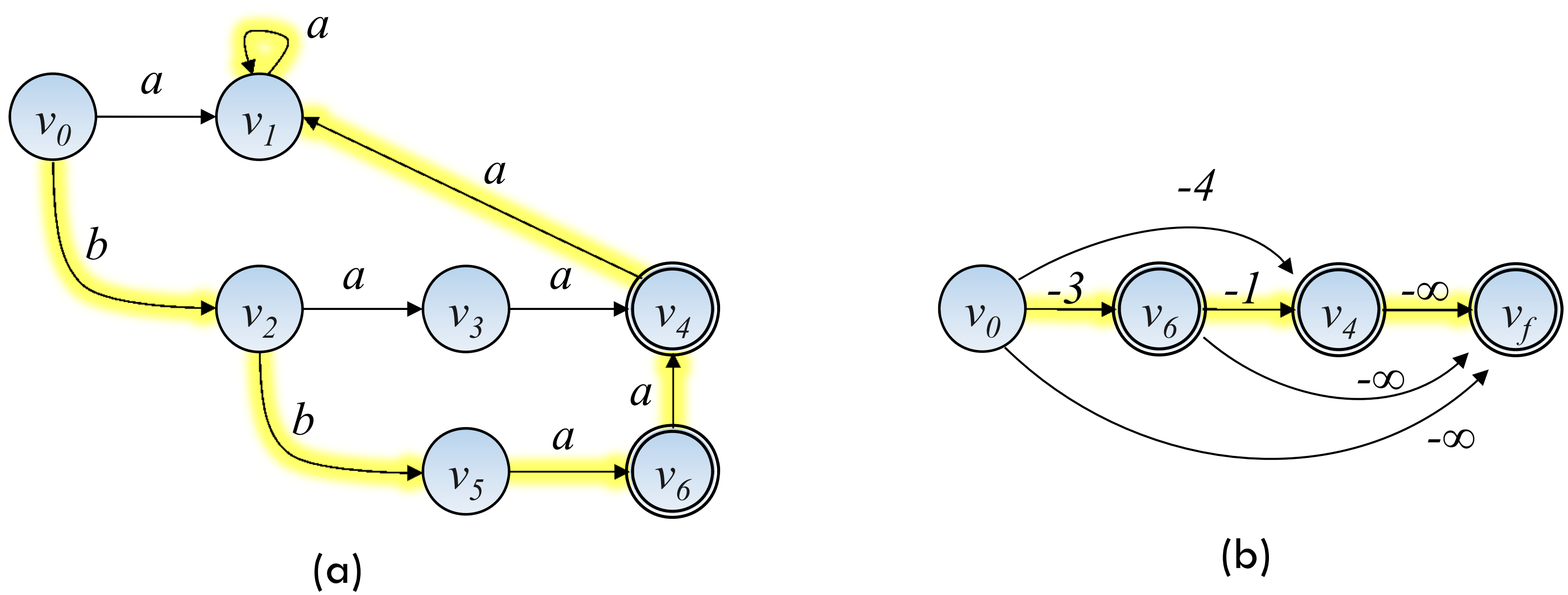


$P \cup N$ is inclusion-free

- 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 $P = \{baa, bba\}$ and $N = \{ab, bbb\}$
Arbitrarily long CSS $bbaaaa$ ...

$P \cup N$ 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

<table>
<tr><th>Algorithms</th><th colspan="2">Cases</th><th>LCNSS of N exists</th><th>No LCNSS of N exists</th></tr>
<tr><td>JT95</td><td colspan="2">IF & final closure</td><td>$O(p^2 + pN^2)$</td><td>-</td></tr>
<tr><td rowspan="3">Ours</td><td>IF</td><td rowspan="2">Q1</td><td rowspan="2">$O(P + N)$</td><td rowspan="2">$O(p(P + N)^2)$</td></tr>
<tr><td rowspan="2">~IF</td></tr>
<tr><td>~Q1</td><td colspan="2">$O(k(P + N)^2 + k^2 2^p)$</td></tr>
</table>

- 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