IWOCA 2024

Computing Minimal Absent Words and Extended Bispecial Factors with CDAWG Space

Shunsuke Inenaga[1], Takuya Mieno[2], Hiroki Arimura[3], Mitsuru Funakoshi[1], Yuta Fujishige[4]

[1] Kyushu University

[2] University of Electro-Communications

[3] Hokkaido University

[4] Fujitsu

Minimal Absent Words (MAWs) [1/2]

- A string w over an alphabet Σ is called a **Minimal Absent Word (MAW)** for a string S, if:
  1. w is a character from Σ not occurring in S, or
  2. $w = aub$ ($a, b \in \Sigma$, $u \in \Sigma^*$) does not occur in S, but both au and ub occur in S.

Example

$w =$ `b a b`

$S =$ `a b a a b`

Minimal Absent Words (MAWs) [1/2]

- A string w over an alphabet Σ is called a **Minimal Absent Word (MAW)** for a string S, if:
  1. w is a character from Σ not occurring in S, or
  2. $w = aub$ ($a, b \in \Sigma$, $u \in \Sigma^*$) does not occur in S, but both au and ub occur in S.

Example

$w =$ `b a b`

$S =$ `a b a a b`

`bab` is a MAW for `abaab`

Minimal Absent Words (MAWs) [2/2]

- $\mathrm{MAW}(S)$ denotes the set of MAWs for a string S.

Example

$$S = \mathtt{abaab} \qquad \Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$$

$$\mathrm{MAW}(S) = \{\mathtt{aaa}, \mathtt{aaba}, \mathtt{bab}, \mathtt{bb}, \mathtt{c}\}$$

- The <u>number $|\mathrm{MAW}(S)|$ of MAWs</u> for a string S of length n over an alphabet of size σ is $O(\sigma n)$, and there is a matching lower bound [Crochemore et al. 1998].

Motivations for Computing MAWs

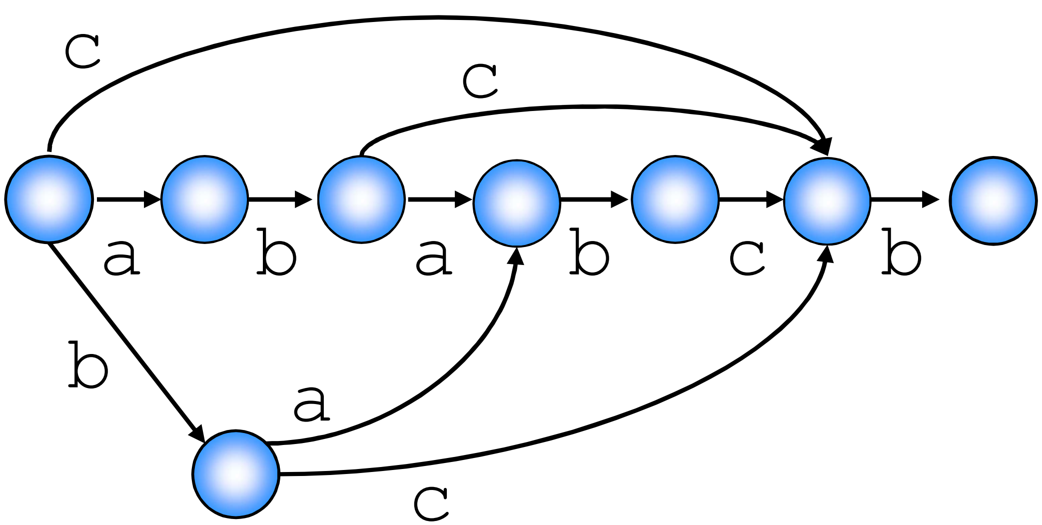
- Computing (minimal) absent words for a given string has various applications including:
  - **Data compression** (anti-dictionary)
    [Crochemore et al. 2000] [Crochemore & Navarro 2002]
    [Ayad et al. 2021]
  - **Sequence comparison**
    [Chairungsee & Crochemore 2012]
    [Charalampopoulos et al. 2018]
  - **Bioinformatics**
    [Almirantis et al. 2017] [Koulouras & Frith 2019]
    [Pratas & Silva 2020]

Computing MAWs with DAWG [1/3]

- Previous algorithms [Crochemore et al. 1998, Fujishige et al. 2023] for computing MAWs for a string S of length n use **DAWG** (**Directed Acyclic Word Graph**) for S, which is an $\mathrm{O}(n)$-size automaton representing all substrings of S.

[Blumer et al. 1985]

E.g. $S = \mathtt{ababcb}$



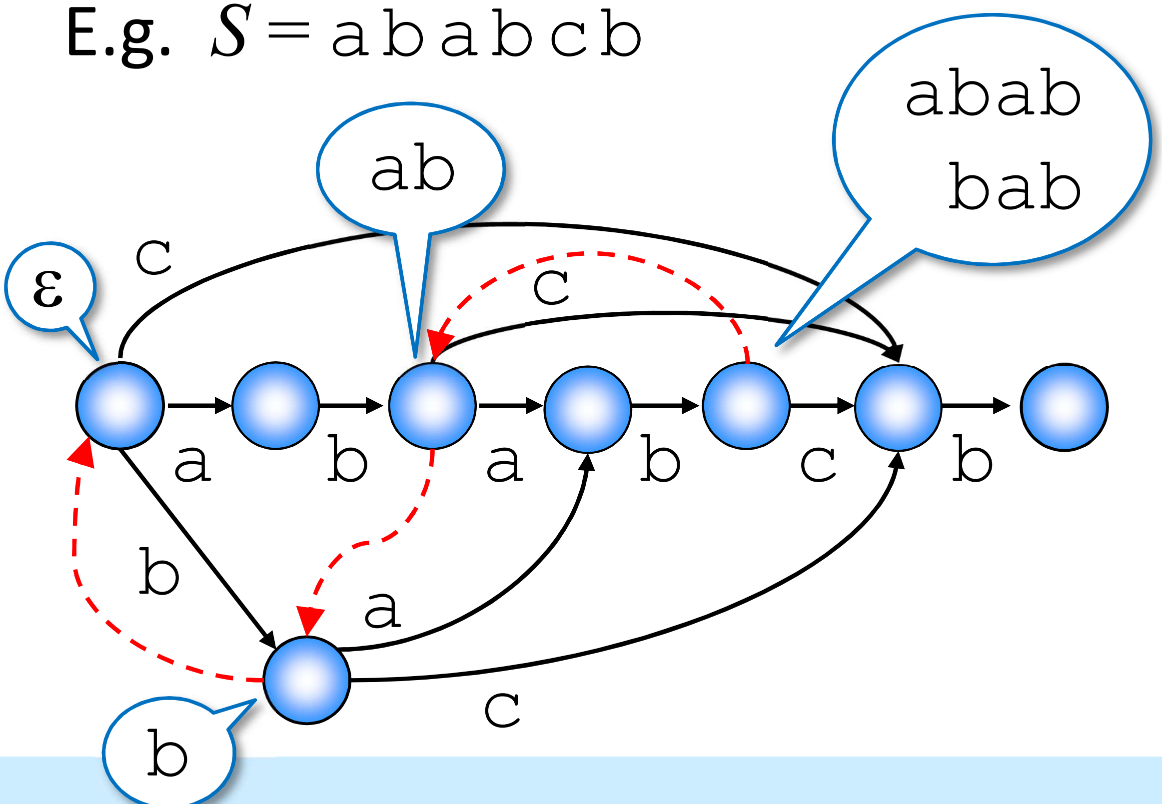
Substrings of S are represented by the same node of DAWG(S) iff they have the same ending position(s) in S.

Computing MAWs with DAWG [1/3]

- Previous algorithms [Crochemore et al. 1998, Fujishige et al. 2023] for computing MAWs for a string S of length n use **DAWG** (**Directed Acyclic Word Graph**) for S, which is an $\mathrm{O}(n)$-size automaton representing all substrings of S.

[Blumer et al. 1985]

E.g. $S = \texttt{ababcb}$



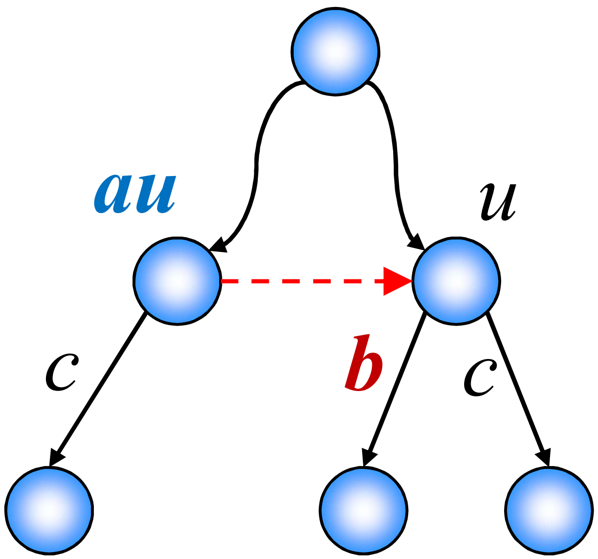
Substrings of S are represented by the same node of DAWG(S) iff they have the same ending position(s) in S.

suffix link

Computing MAWs with DAWG [2/3]

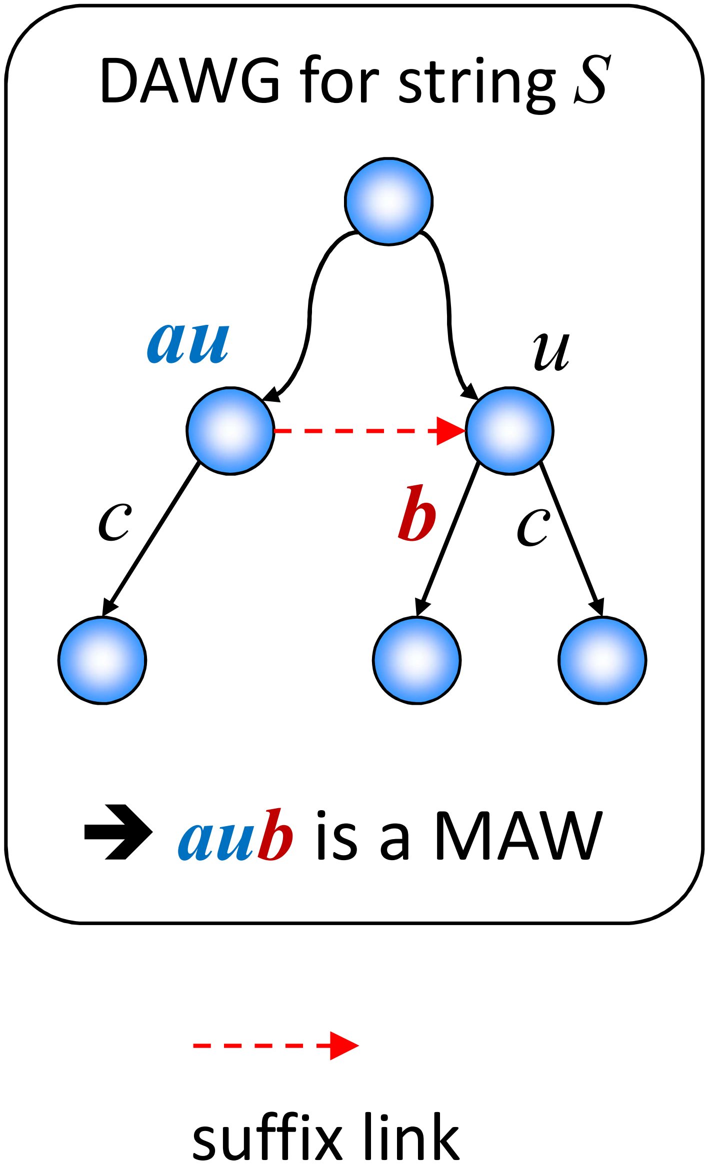
- If the edges of DAWG are sorted, then one can compute MAW(S) in $O(n + |\mathrm{MAW}(S)|)$ time [Fujishige et al. 2023].

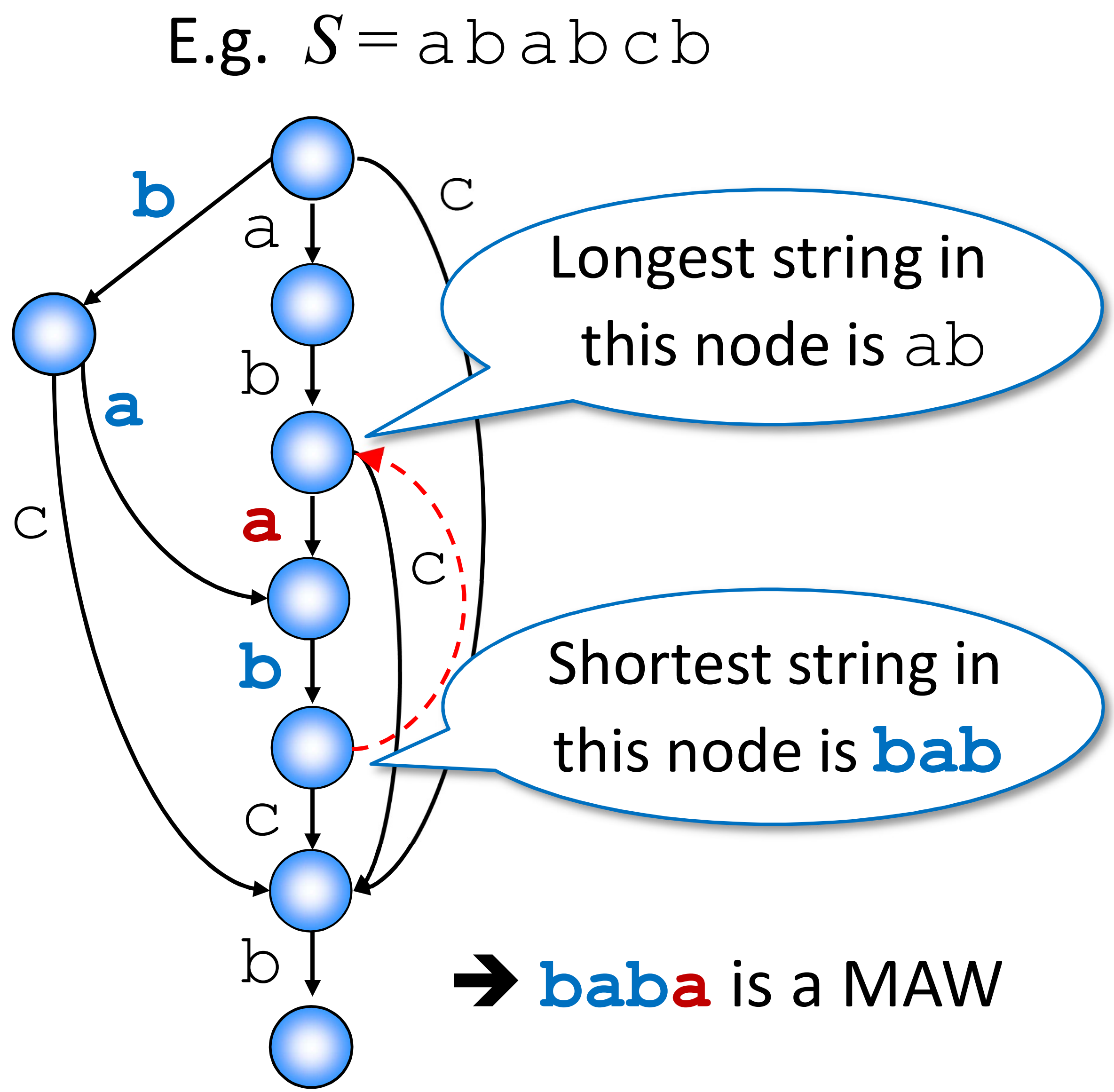
DAWG for string S



- Consider each pair of nodes au and u which are connected by a suffix link, where a is a character and u is a string.
- Compare the labels of the out-edges of nodes au and u in sorted order.
  - For b: au has no out-edge with b, but u has an out-edge with b.
    → aub is a MAW for the input string S.
  - For c: both au and u have out-edges with c
    → auc is not a MAW for the input string S, but this cost of character comparisons can be charged to this out-edge of au labeled c.

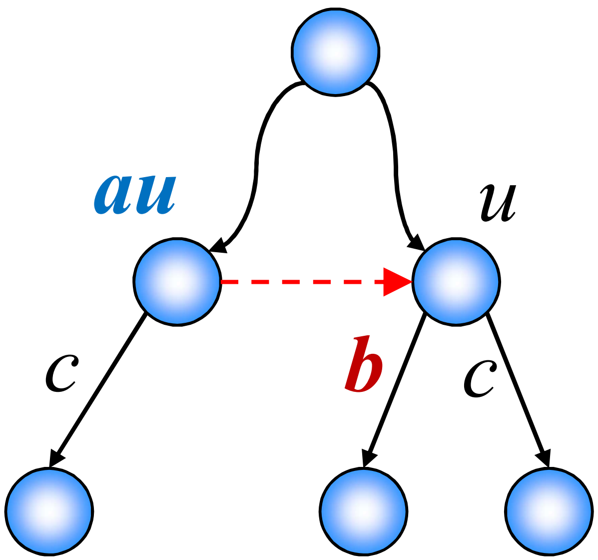
Computing MAWs with DAWG [3/3]





Redundancy in MAW computation with DAWG

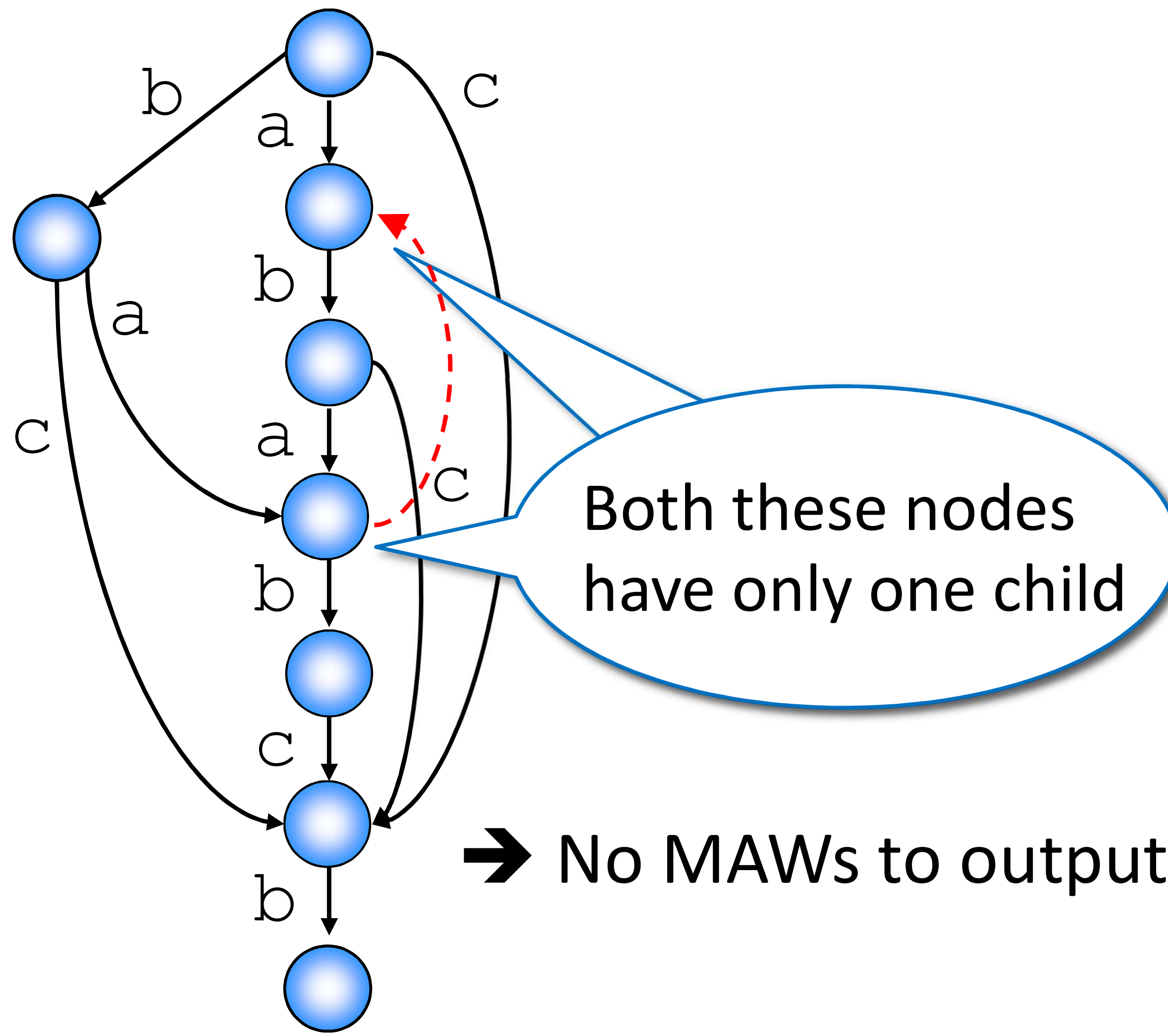
DAWG for string S



➔ ***aub*** is a MAW

suffix link

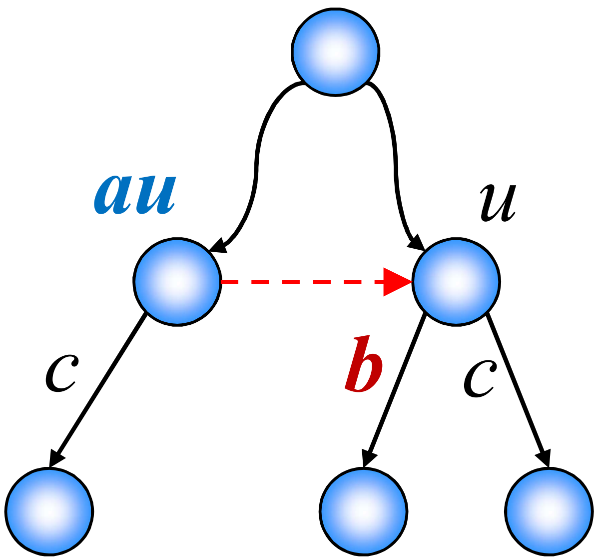
E.g. $S = \mathtt{ababcb}$



➔ No MAWs to output

Redundancy in MAW computation with DAWG

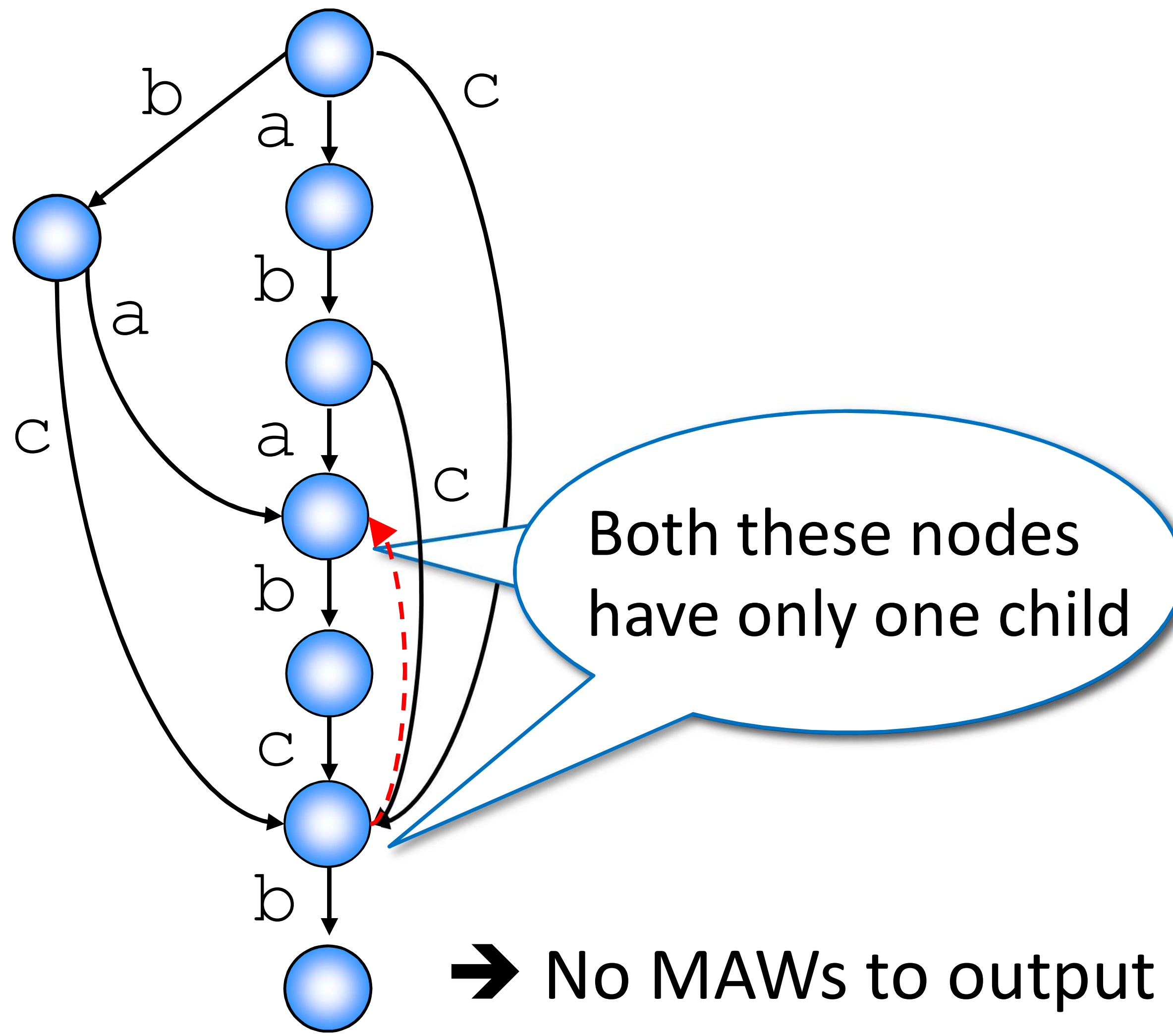
DAWG for string S



➔ aub is a MAW

suffix link

E.g. $S = \texttt{ababcb}$

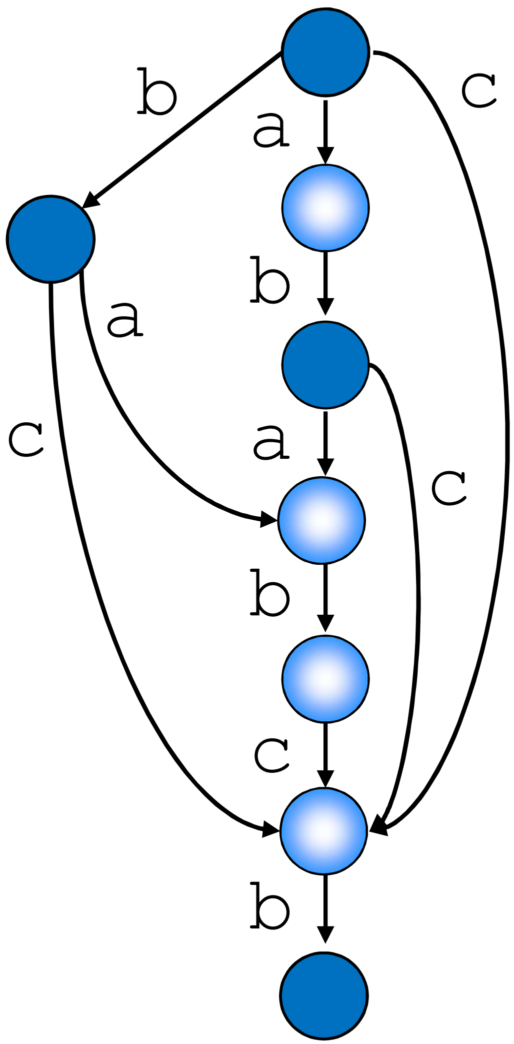


➔ No MAWs to output

From DAWG to CDAWG (Compact DAWG)

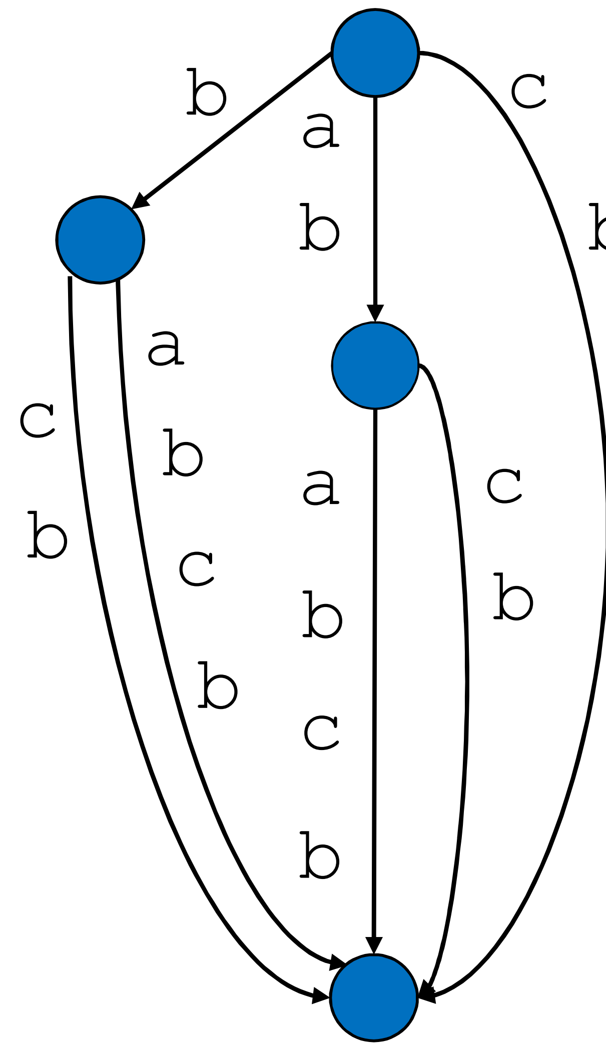
E.g. $S = \mathtt{ababcb}$

DAWG [Blumer et al. 1985]



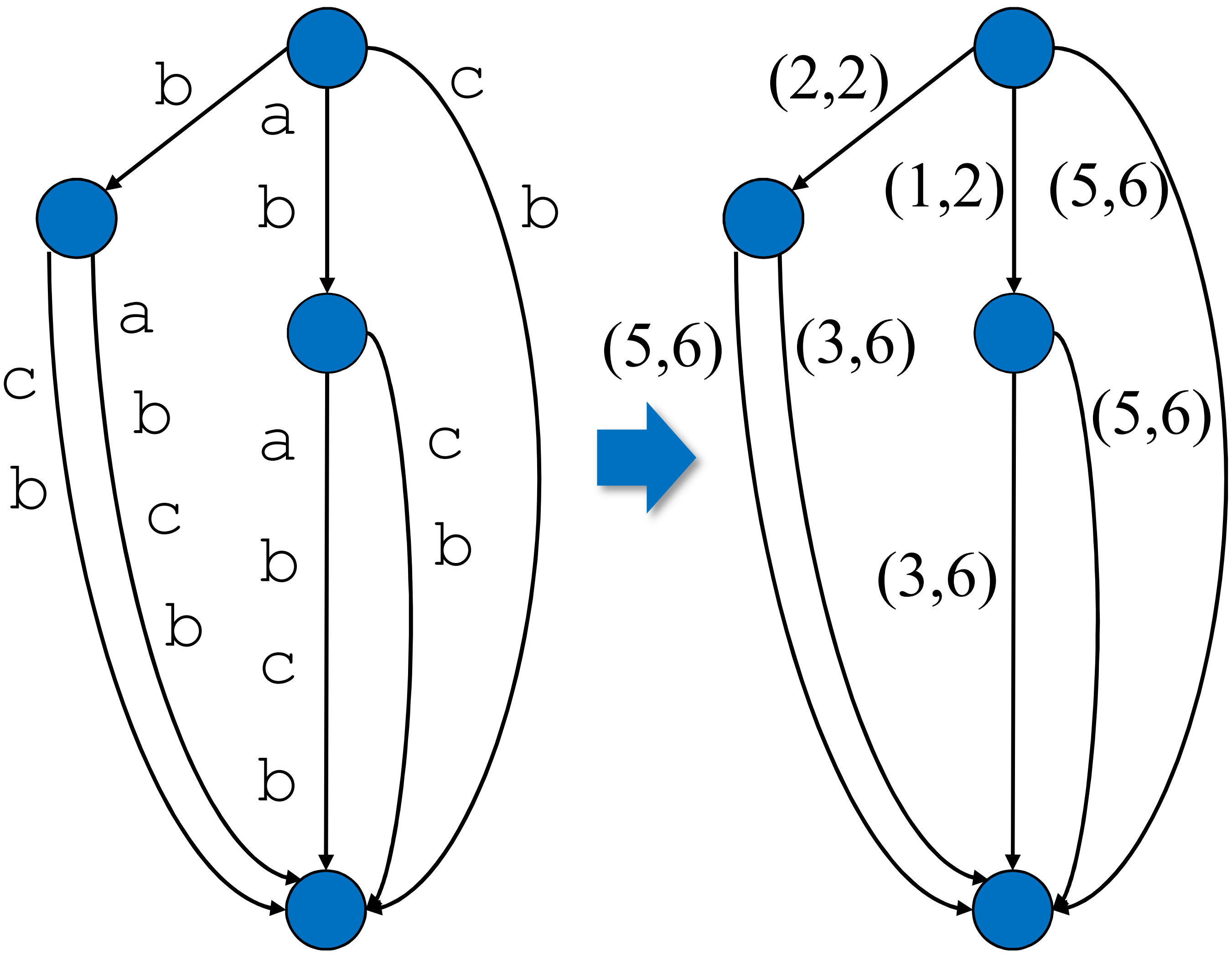
(1) Keep only branching nodes + source & sink
(2) Path contraction

CDAWG [Blumer et al. 1987]



Encoding CDAWG in O(e) space [1/2]

CDAWG for $S =$ $\overset{1}{\texttt{a}}\overset{2}{\texttt{b}}\overset{3}{\texttt{a}}\overset{4}{\texttt{b}}\overset{5}{\texttt{c}}\overset{6}{\texttt{b}}$

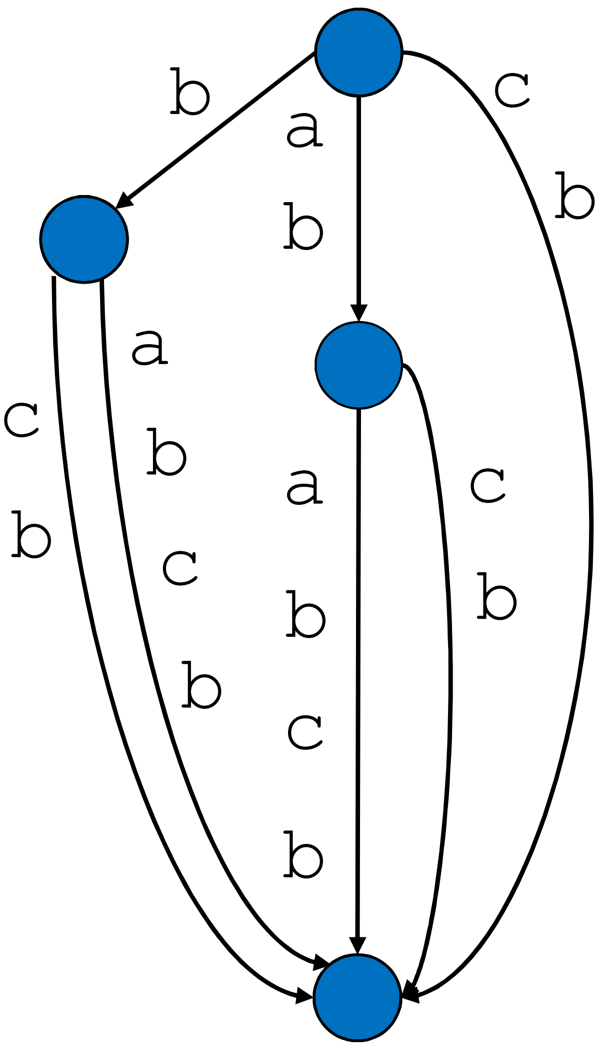


Each string edge label $x \in \Sigma^+$ can be encoded by a pair (i, j) of positions such that $x = S[i..j]$.

➔ CDAWG with e edges can be stored in O(e) space, plus the input string of length n.

Encoding CDAWG in $\mathrm{O}(e)$ space [2/2]

$S = \cancel{\texttt{ababcb}}$



Theorem 1
[Belazzougui & Cunial 2015, Inenaga 2024]

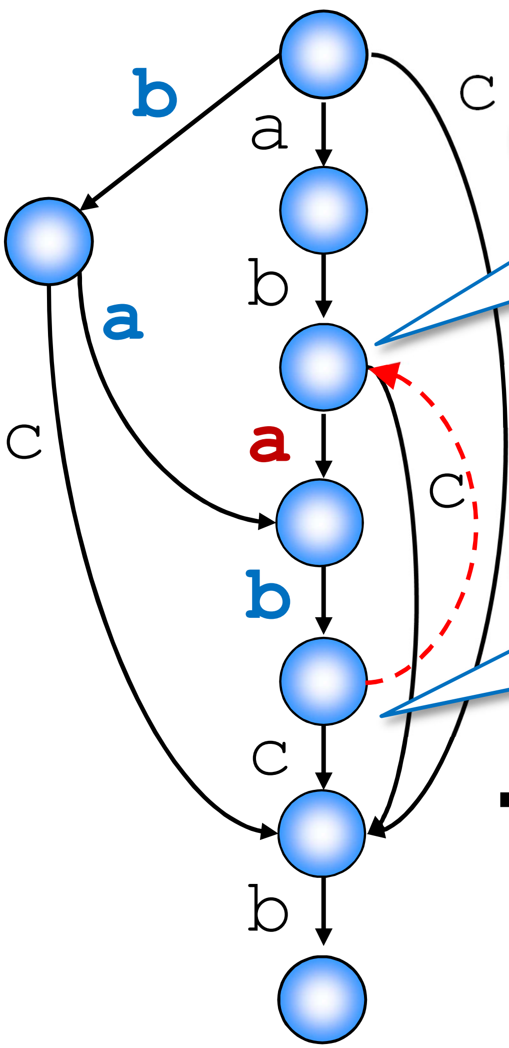
It is possible to store the edge labels of the CDAWG in $\mathrm{O}(e)$ total space, so that each edge string label x can be retrieved in $\mathrm{O}(|x|)$ time, where e is the number of edges in the CDAWG.

Note: e can be as small as $\mathrm{O}(\log n)$ for some highly repetitive strings.

Computing MAWs with CDAWG [1/3]

E.g. $S = \mathtt{ababcb}$

DAWG

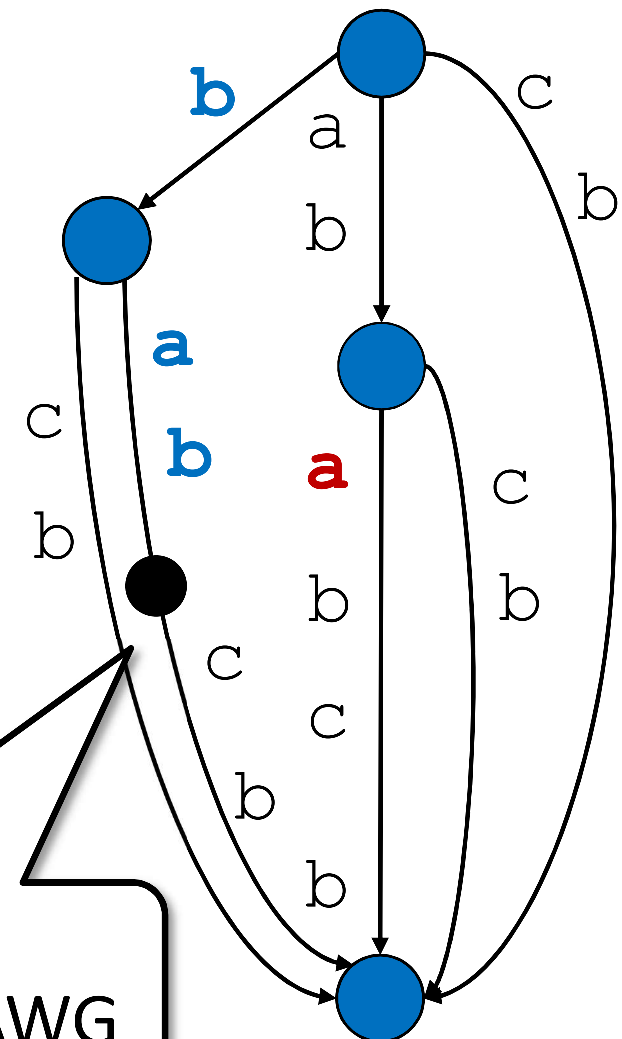


Longest string in this node is ab

Shortest string in this node is **bab**

➔ **baba** is a MAW

CDAWG



Node for **bab** is missing in the CDAWG

Computing MAWs with CDAWG [2/3]

Theorem 2 [Belazzougui & Cunial 2015]

The number of "missing" nodes in the CDAWG of string S for computing $\mathsf{MAW}(S)$ is at most the number $\bar{e}$ of edges of the CDAWG for the *reversed string* $\bar{S}$.

Theorem 3 [Belazzougui & Cunial 2015]

There exists a data structure of $\mathrm{O}(e + \bar{e})$ space that can output a representation of $\mathsf{MAW}(S)$ for the input string S in $\mathrm{O}(e + \bar{e} + |\mathsf{MAW}(S)|)$ time.

How much is the gap between e and $\bar{e}$?

Computing MAWs with CDAWG [3/3]

How much is the gap between e and $\bar{e}$?

Theorem 4 [Karkkainen 2017, Inenaga 2024]

There exists a family of strings of length n such that $\bar{e}/e = \Omega(\sqrt{n})$.

Theorem 5 [Inenaga, Kosolobov, Zuba 2024 (unpublished)]

For any string of length n, $\bar{e}/e = O(\sqrt{n})$.

Our Contribution

Theorem 6

There exists a data structure of $O(e_{\min})$ space that can output a representation of $\mathsf{MAW}(S)$ for string S in $O(|\mathsf{MAW}(S)|)$ time, where $e_{\min} = \min\{e, \bar{e}\}$.

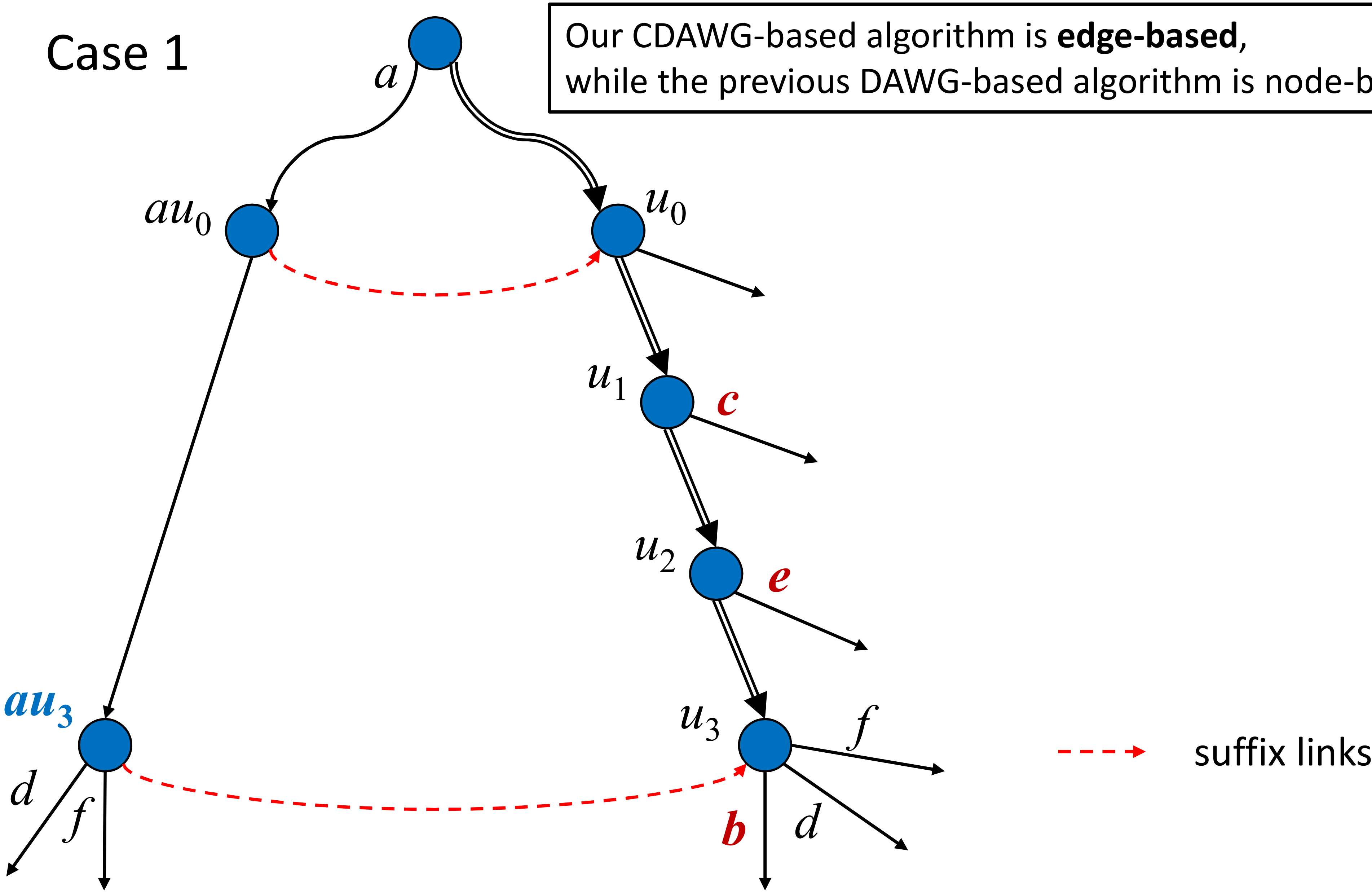
algorithm	time	space
Crochemore et al. 1998	$O(\sigma n)$	$O(n)$
Fujishige et al. 2023	$O(n + \|\mathsf{MAW}(S)\|)$	$O(n)$
Belazzougui & Cunial 2015	$O(e + \bar{e} + \|\mathsf{MAW}(S)\|)$	$O(e + \bar{e})$
Ours	$\mathbf{O(\|MAW(S)\|)}$	$\mathbf{O(e_{\min})}$

Note: e and $\bar{e}$ are at most $2n$-1, and can be as small as $O(\log n)$ for some highly repetitive strings (e.g. Fibonacci strings).

Computing MAWs in O($e_{\min}$) space [1/4]

Case 1

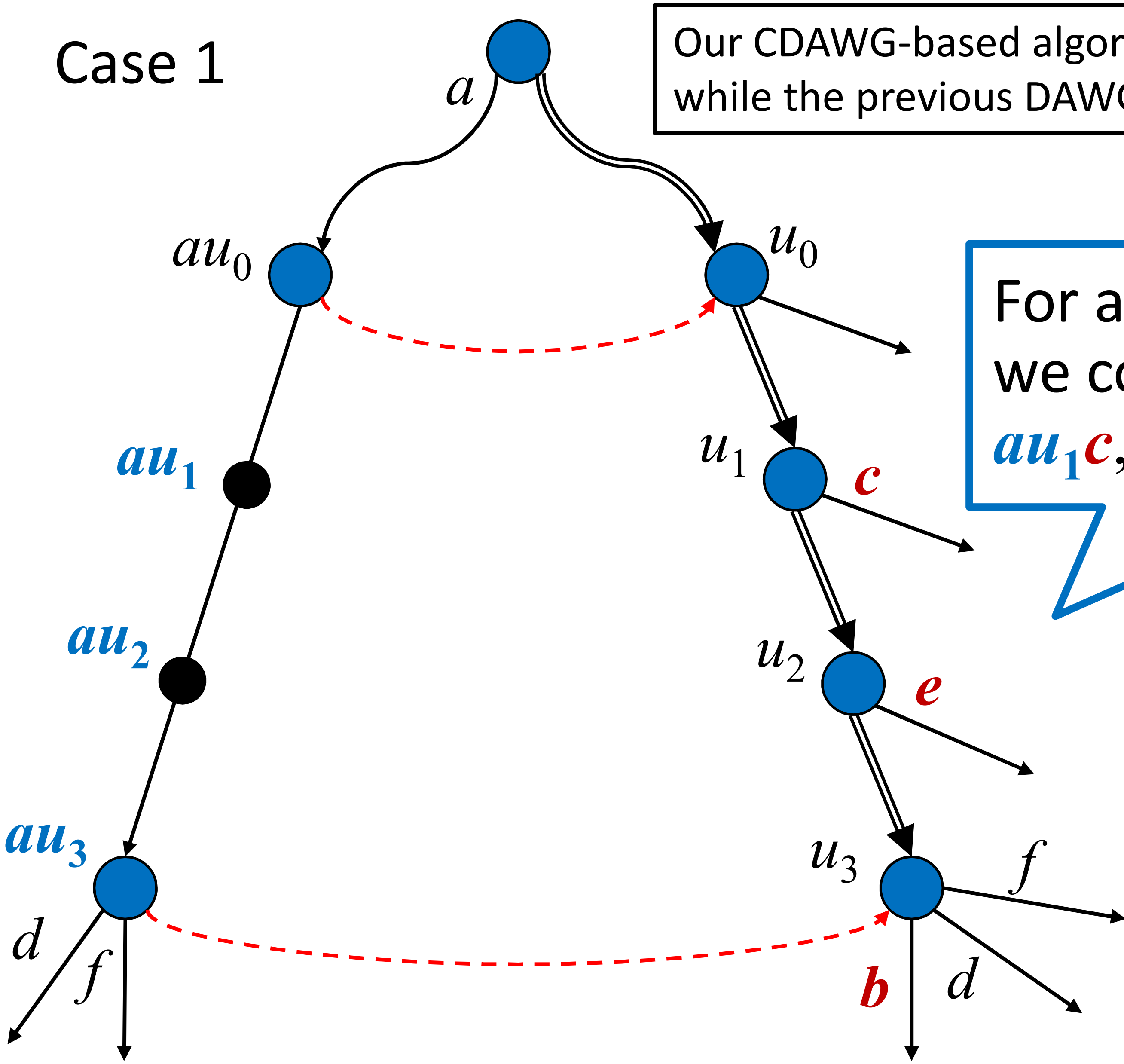
Our CDAWG-based algorithm is **edge-based**,
while the previous DAWG-based algorithm is node-based.



Computing MAWs in O($e_{\min}$) space [1/4]

Case 1

Our CDAWG-based algorithm is **edge-based**,
while the previous DAWG-based algorithm is node-based.



For a given edge of CDAWG, we compute MAWs au_1c, au_2e, au_3b in a batch.

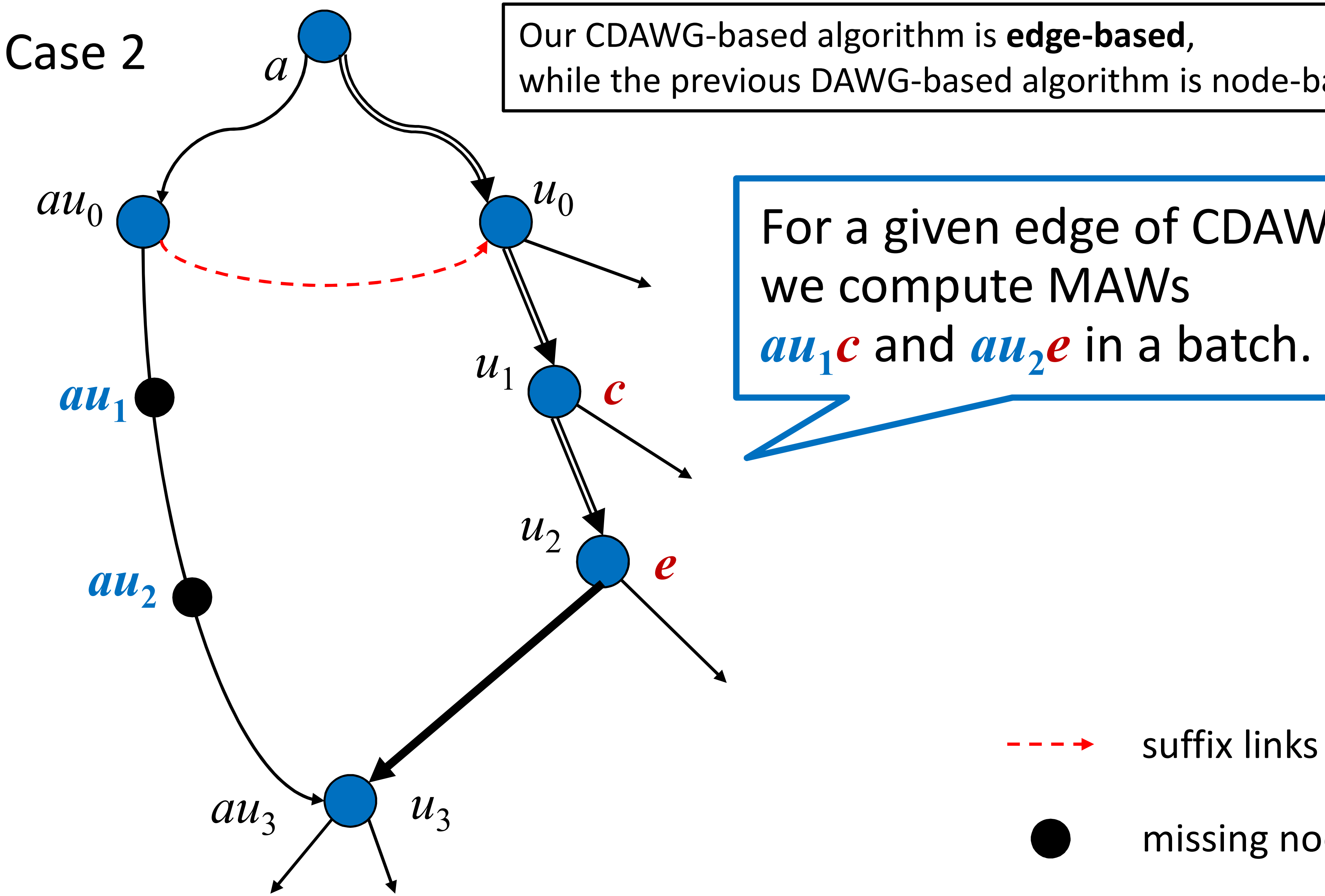
suffix links

missing nodes

Computing MAWs in $O(e_{\min})$ space [2/4]

Case 2

Our CDAWG-based algorithm is **edge-based**,
while the previous DAWG-based algorithm is node-based.

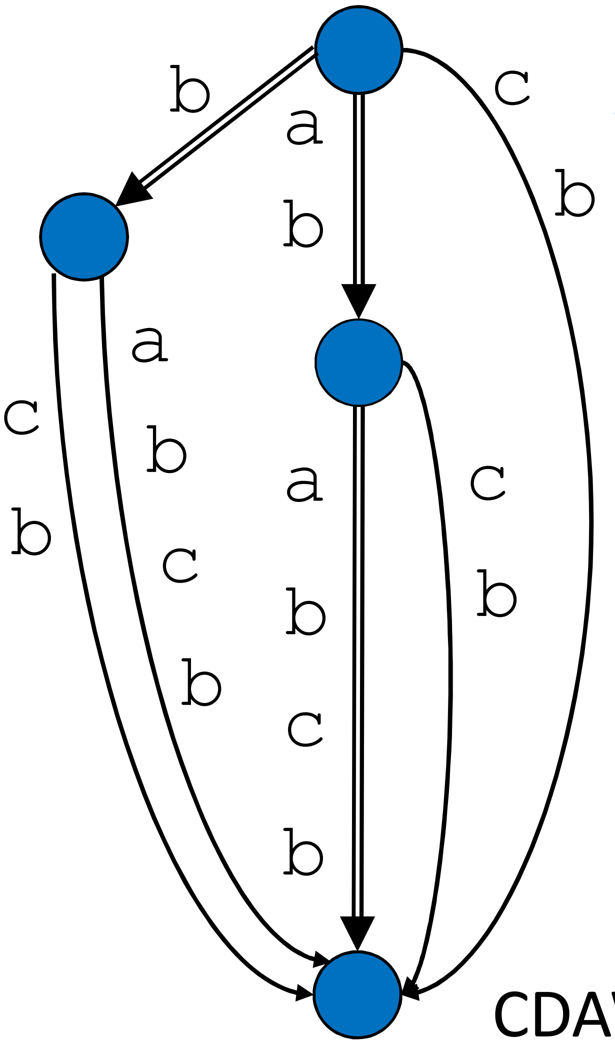


For a given edge of CDAWG, we compute MAWs au_1c and au_2e in a batch.

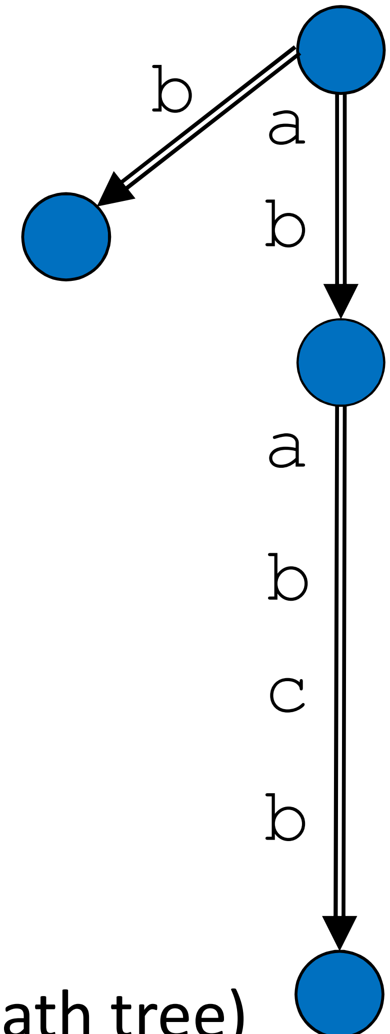
Computing MAWs in $O(e_{\min})$ space [3/4]

$S = \mathtt{ababcb}$

The **primary edges** are those that are in the **longest paths** from the source to the nodes.



CDAWG

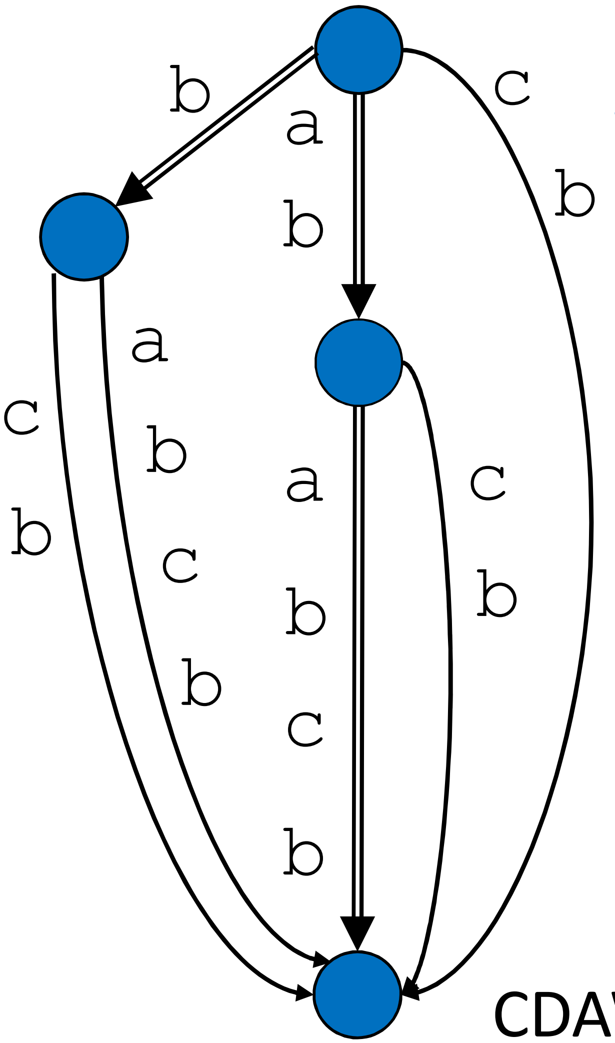


LPT
(longest path tree)

Computing MAWs in $O(e_{\min})$ space [3/4]

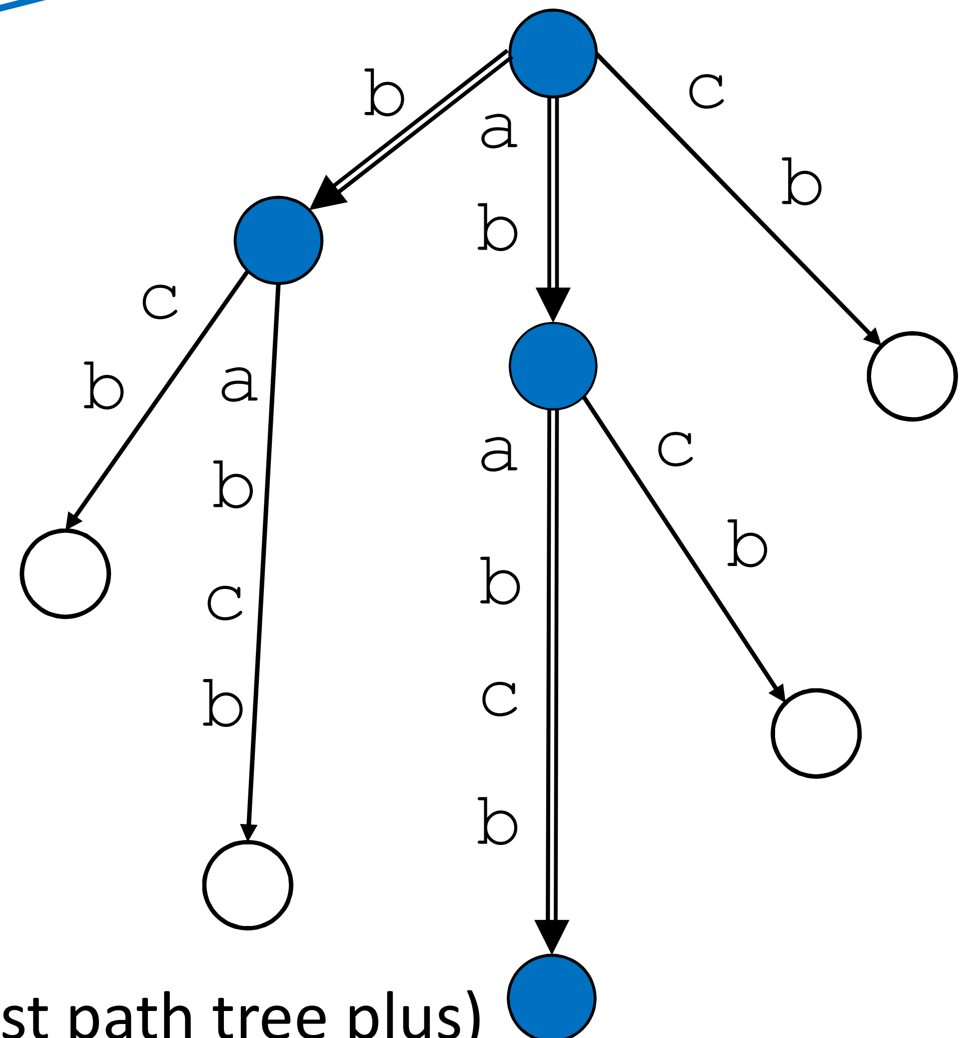
$S = \texttt{ababcb}$

The **primary edges** are those that are in the **longest paths** from the source to the nodes.



CDAWG

equivalent

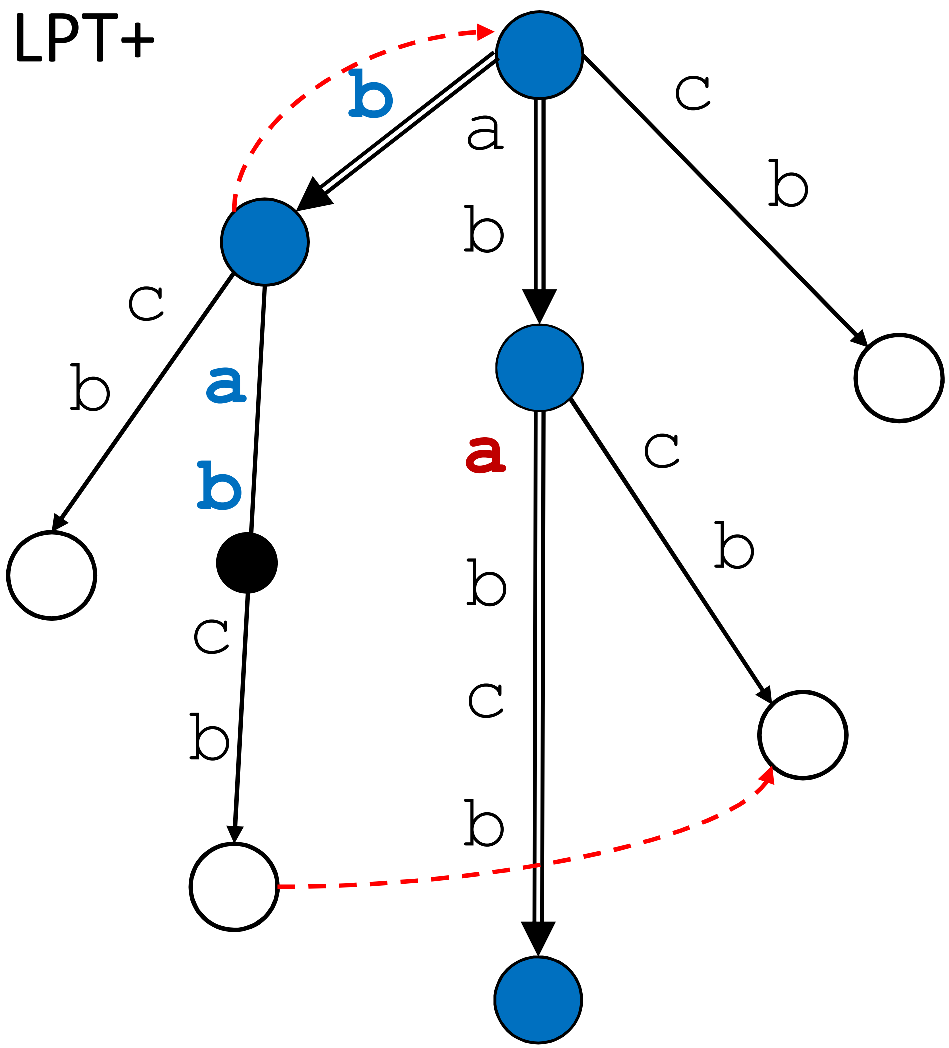


LPT+
(longest path tree plus)

Computing MAWs in $O(e_{\min})$ space [4/4]

$S =$ a b a b c b

LPT+



primary edges

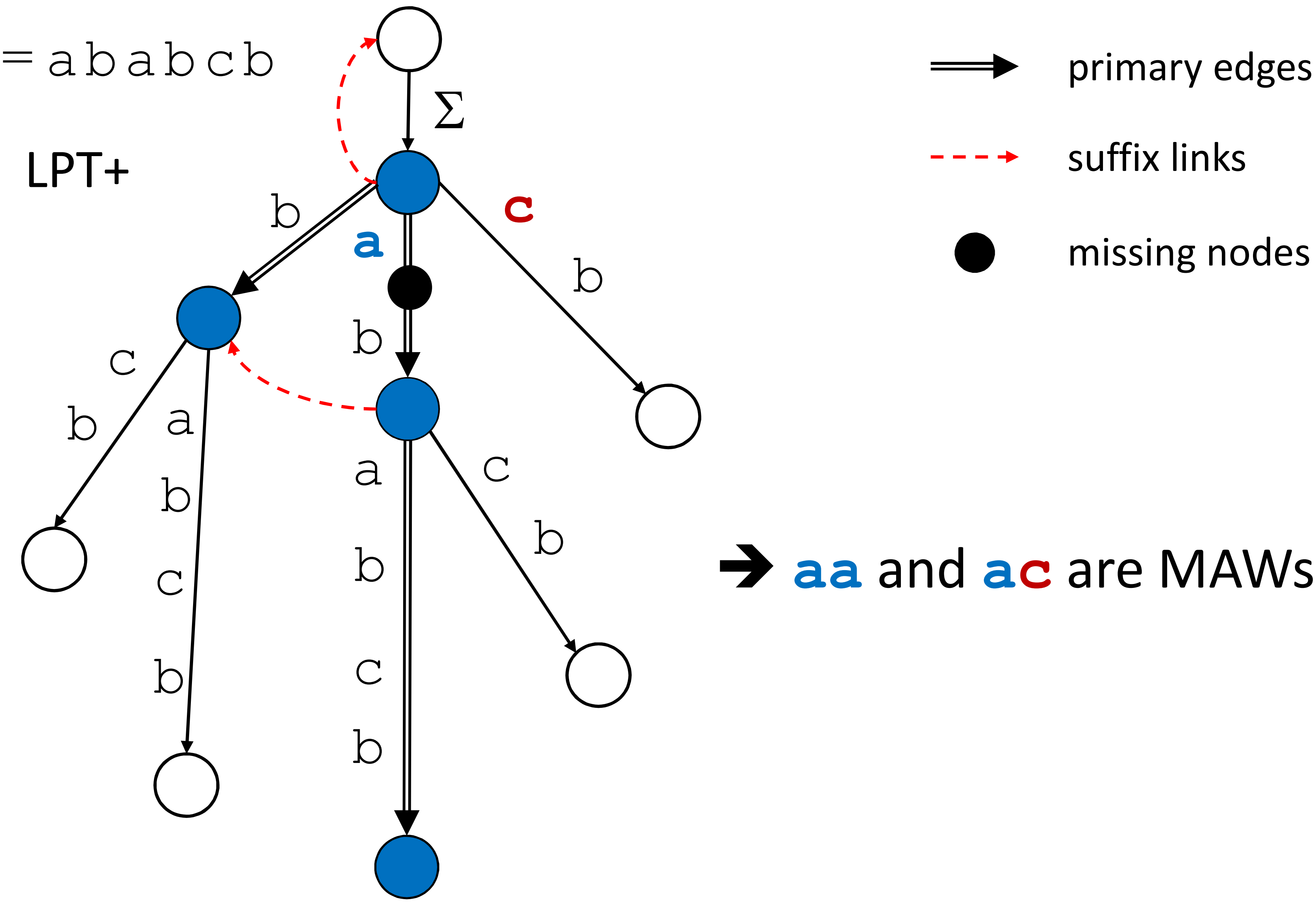
suffix links

missing nodes

➔ **baba** is a MAW

Computing MAWs in $O(e_{min})$ space [4/4]

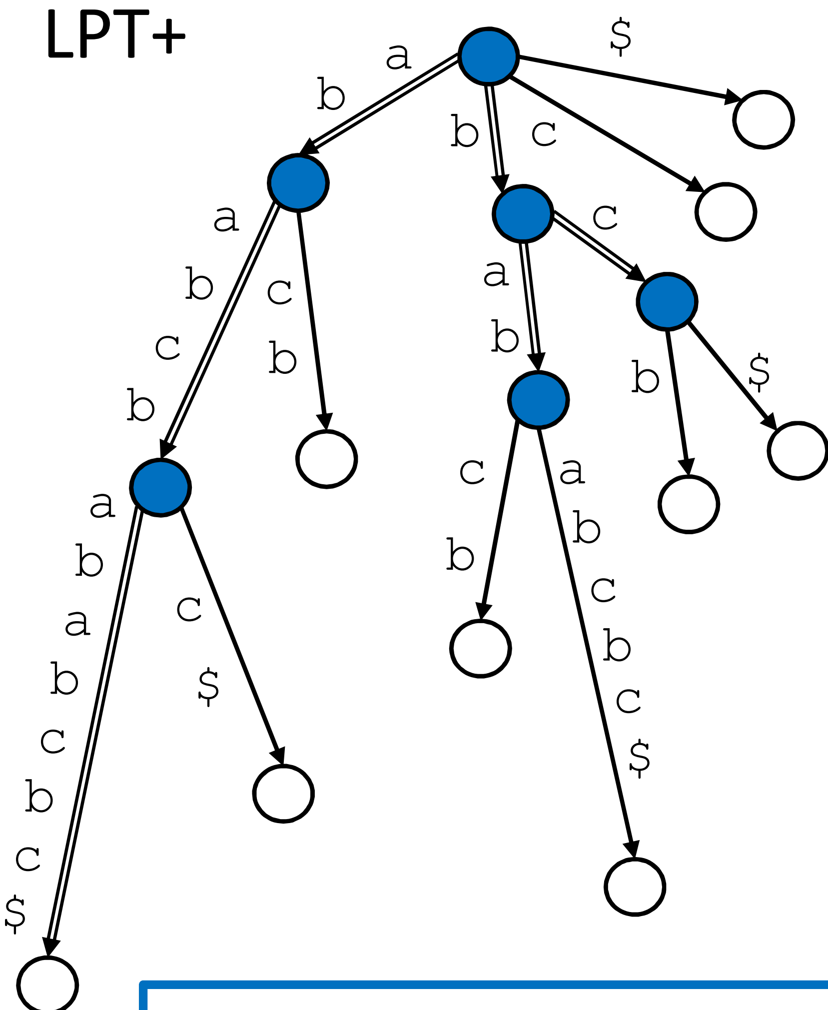
$S = \texttt{ababcb}$

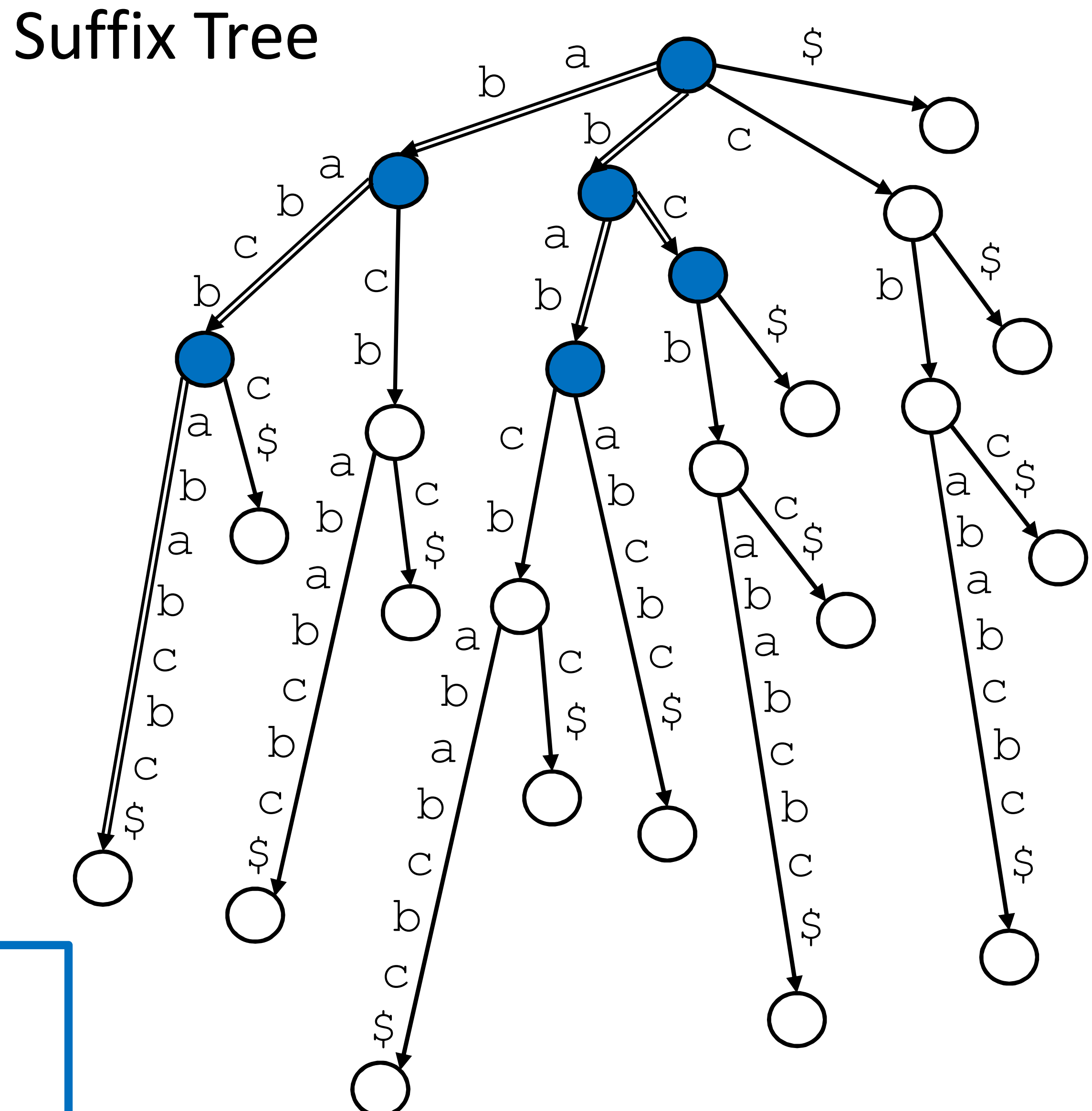


➔ **aa** and **ac** are MAWs

LPT+ $\neq$ Suffix Tree

$S = \texttt{ababcbababcbc\$}$





LPT+ is of size $\Theta(e)$ while Suffix Tree is of size $\Theta(n)$.

Conclusions

- We have shown a **CDAWG**-based data structure of $\mathrm{O}(e)$ space that can report all **MAWs** in optimal time.

- Our data structure can be extended to report **minimal rare words** (**MRW**) in optimal time:
A string aub is a k-MRW of a string S if aub occurs k times in S, while au and ub occur more than k times in S.
    - **0-MRWs are MAWs**.
    - **1-MRWs are MUSs** (minimal unique substrings) [Ilie & Smyth 2011]

Future Work

- Is there a data structure of $\mathbf{O}(\boldsymbol{r})$ space that reports all MAWs/MRWs in optimal time, where r is the number of equal-letter runs in the **BWT** (Burrows-Wheeler transform)?
- It is know that $r \leq e$ [Belazzougui & Cunial 2017].

➢ There is a data structure of $\mathrm{O}(r \operatorname{polylog}(n))$ space that can do this in $\mathrm{O}((e + \bar{e})\operatorname{polylog}(n) + \mathit{output})$ time with $\mathrm{O}(\sigma \log n)$ working space (under review).

➢ We want to do this in optimal $\mathrm{O}(\mathit{output})$ time with $\mathrm{O}(r + \bar{r})$ space.