Sparse Directed Acyclic Word Graphs

Shunsuke Inenaga
(*Kyushu University*)

Masayuki Takeda
(*Kyushu University & Japan Science Technology Agency*)

Contents

- *Basic Pattern Matching Problem*
  - Text Indexing Structures

- *Natural Language Text Search*
  - Phrase-level Pattern Matching Problem
  - Sparse Text Indexing Structures

- *Sparse Directed Acyclic Word Graphs*
  - Size
  - Construction

- *Summary and Future Work*

Basic Pattern Matching Problem

Input: text $\boldsymbol{T}$ from Σ^* and pattern $\boldsymbol{P}$ from Σ^*

Output: whether or not $\boldsymbol{P}$ occurs in $\boldsymbol{T}$

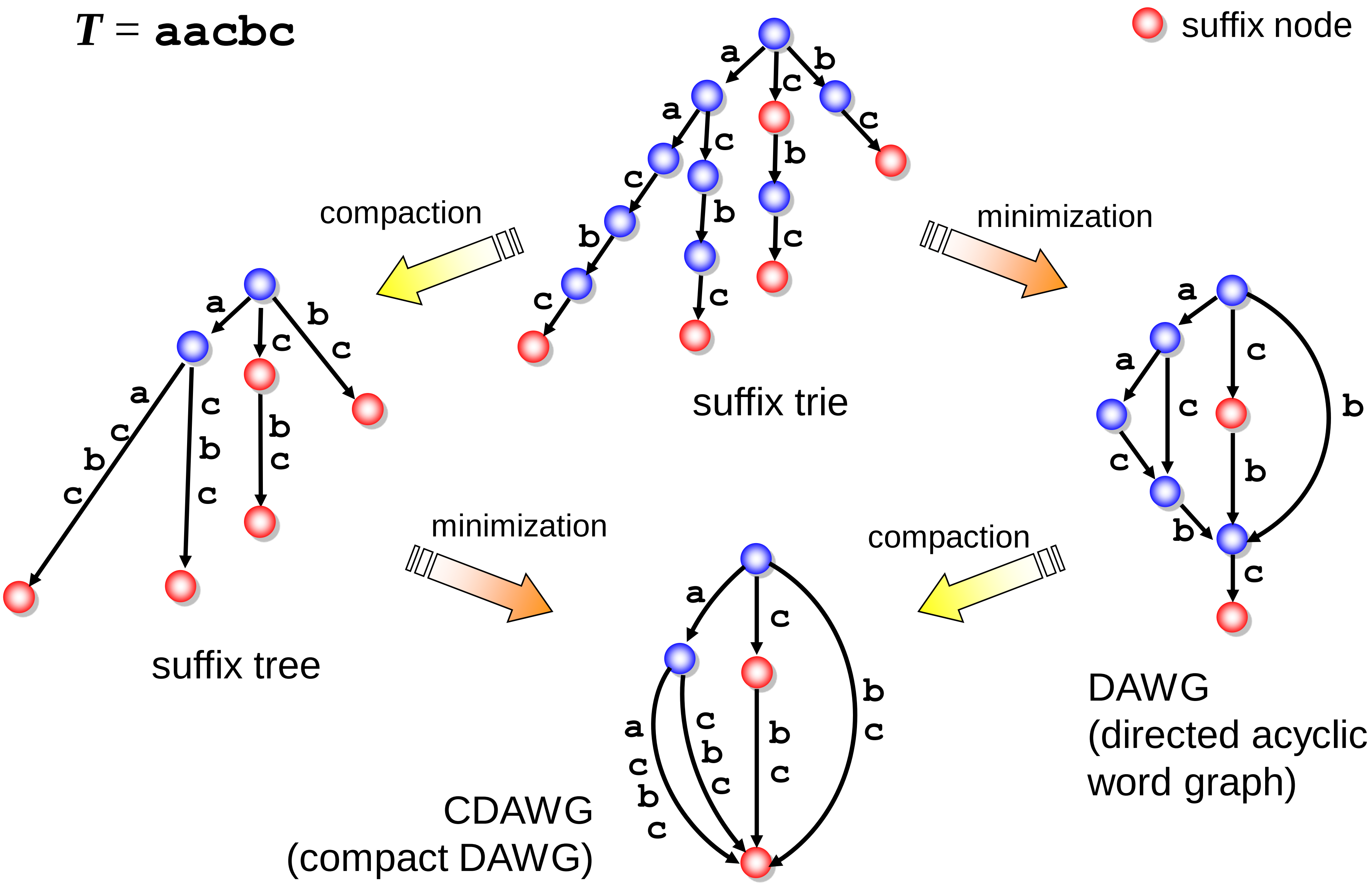
Σ : alphabet (set of *characters*)
Σ^* : set of *strings*

- Text indexing structures for $\boldsymbol{T}$ enable us to solve this problem in $\boldsymbol{O(m)}$ time (for fixed alphabet Σ).

$\boldsymbol{m}$: the length of pattern $\boldsymbol{P}$

Text Indexing Structures

T = **aacbc**



Sizes of Indexing Structures

	max num of nodes	max num of edges
suffix tries	$n(n+1)/2 + 1$	$n(n+1)/2$
suffix trees	$2n-1$	$2n-2$
DAWGs	$2n-1$	$3n-4$
CDAWGs	$n+1$	$2n-2$

n : text length

Contents

- *Basic Pattern Matching Problem*
  - Text Indexing Structures
- *Natural Language Text Search*
  - Phrase-level Pattern Matching Problem
  - Sparse Text Indexing Structures
- *Sparse Directed Acyclic Word Graphs*
  - Size
  - Construction
- *Summary and Future Work*

Considering Natural Language Texts

- T = “string processing and information retrieval”

- *We seldom want to search from the inside of words, we only want to search from the head of words.*

  - Indexing all the suffixes of text T is a waste of space
  - Unwanted matching (see below) should be avoided

  e.g. P = ring processing

Introducing Word Separator #

- # : word separator - special symbol *not* in Σ
- $D = \Sigma^* \#$: dictionary of *words*

- Let text T be an element of D^+
(*T* is a sequence $T_1 T_2 \ldots T_k$ of k words from D)

- e.g., T = **`This#is#a#pen#`**
  - Σ = {**`A`**, ..., **`z`**}
  - D = {..., **`This#`**, ..., **`a#`**, ..., **`is#`**, ..., **`pen#`**, ...}

Sizes of Sparse Indexing Structures

	max num of nodes	max num of edges
word suffix tries	$k(n+2)/2 + 1$	$k(n+2)/2$
word suffix trees	$2k-1$	$2k-2$
SDAWGs	**?**	**?**
SCDAWGs	$k+1$	$2k-2$

n : text length
k : number of words in text
Note that $k \leqq n$

Phrase-level Pattern Matching Problem

Input: text T from D^+ and pattern P from D^+

Output: whether or not P occurs at the head of a word in T

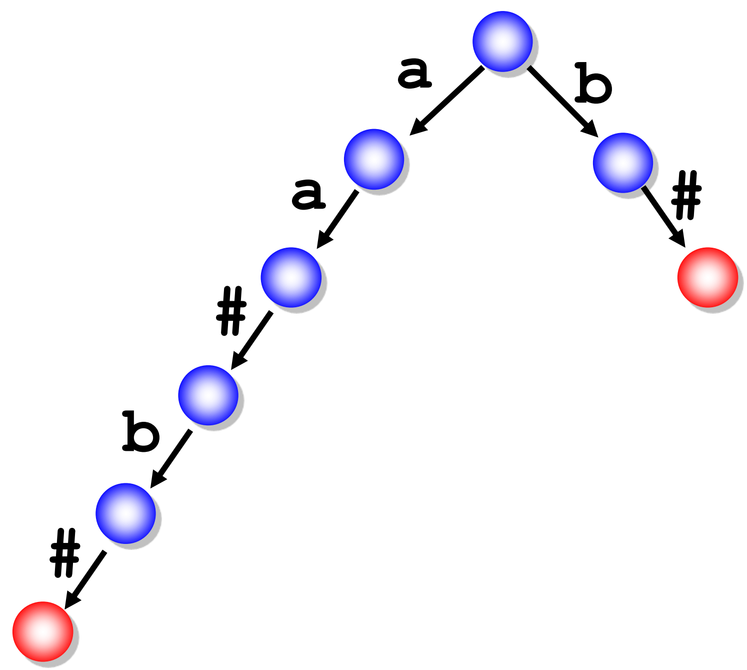
- To solve the above problem, we want a "sparse" text indexing structure that represents only the suffixes of T beginning at the head of a word in T.

Word Suffix Trie

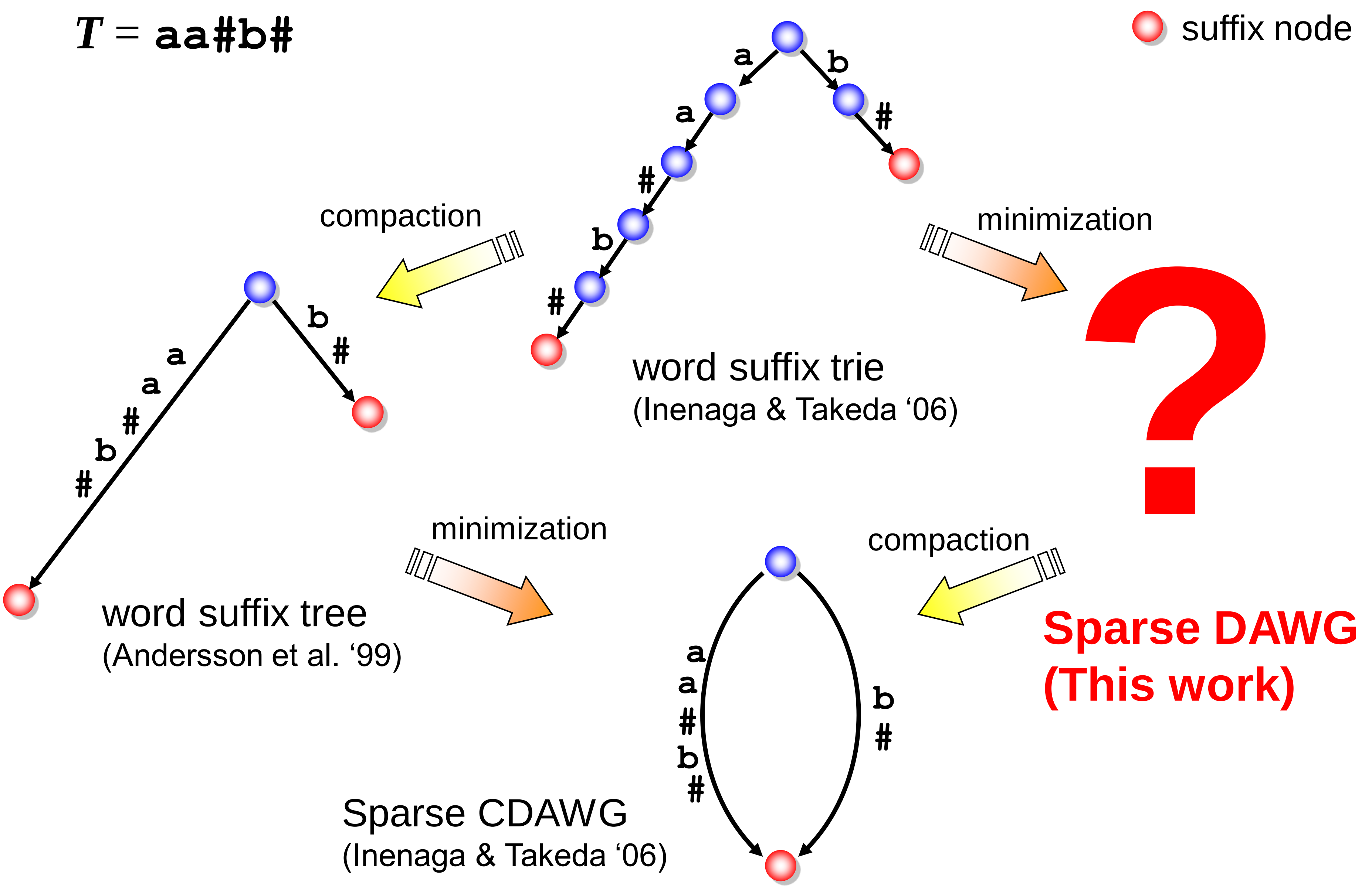
- A trie which represents only the suffixes of T beginning at the head of a word in T

T = **aa#b#**

aa#b#
a#b#
#b#
b#
#



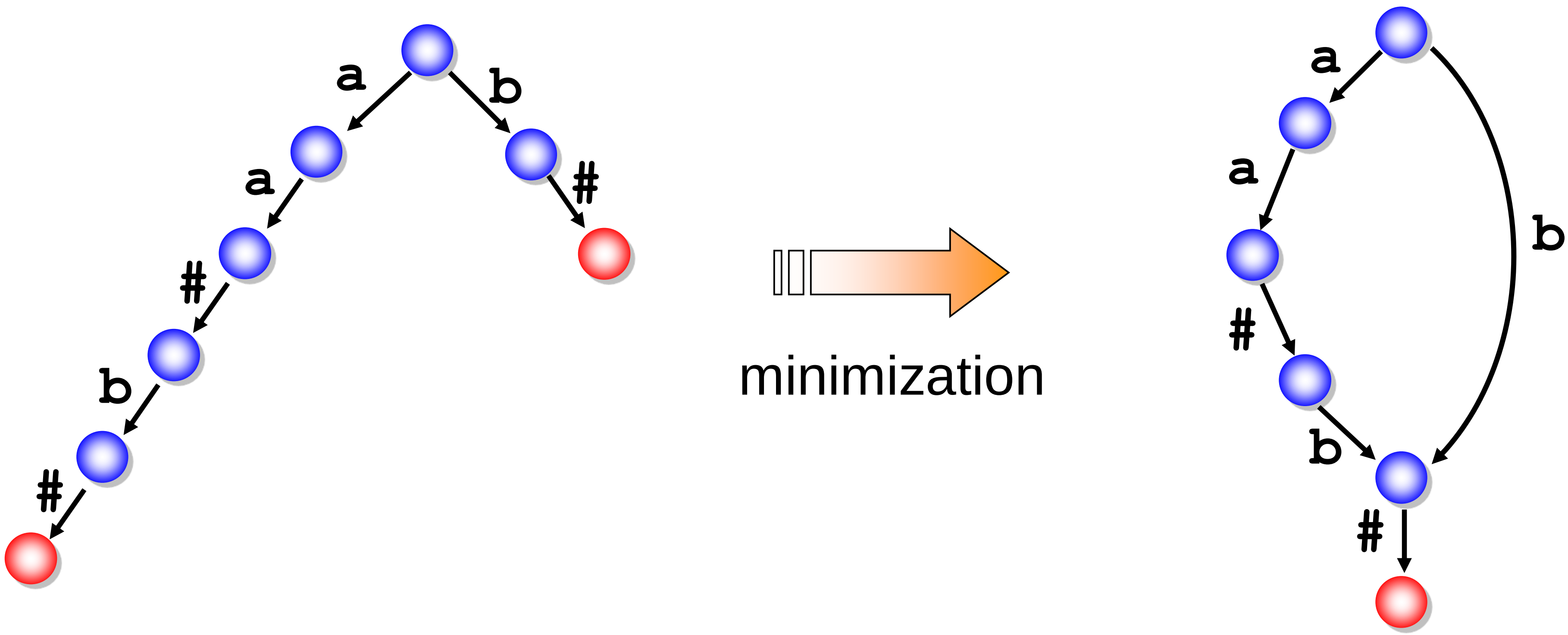
Sparse Text Indexing Structures



Contents

Sparse Directed Acyclic Word Graph

T = **aa#b#**



Word Suffix Trie

Sparse DAWG (SDAWG)

Comparing Normal and Sparse DAWGs

T = **aa#b#**

suffixes of *T*

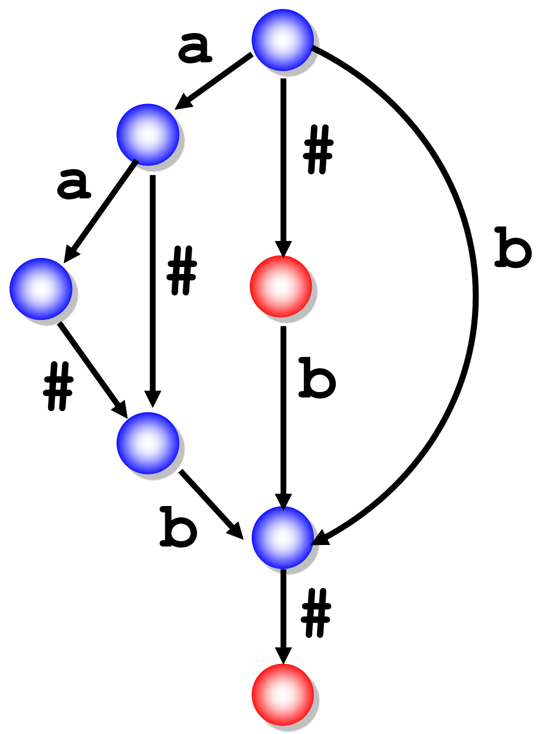
<u>**aa#b#**</u>

a#b#

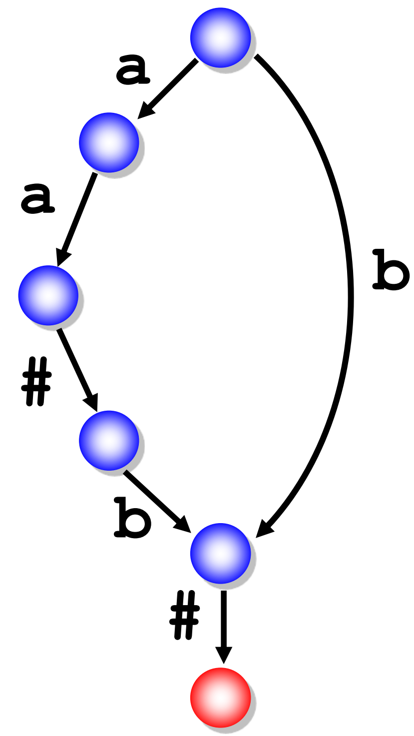
#b#

<u>**b#**</u>

#



DAWG

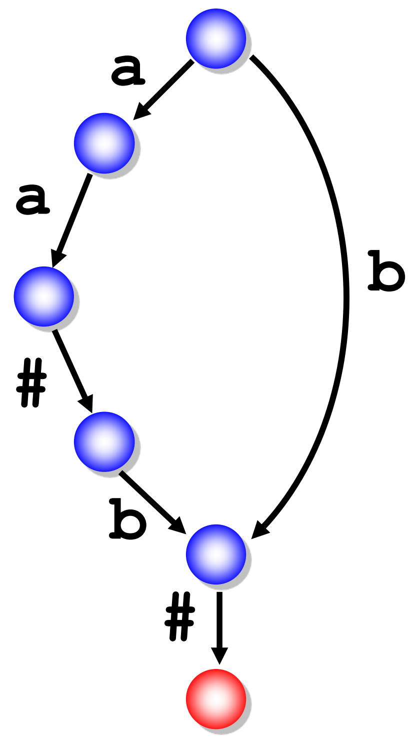


Sparse DAWG
(SDAWG)

Size of SDAWGs – Lower Bound

***Theorem*:**
The SDAWG of any text T of length n has at least $n+1$ nodes.

T = **aa#b#**



Size of SDAWGs – Upper Bound

Theorem:
The SDAWG of any text T of length n has $O(n)$ nodes and edges.

Shown by similar ideas to Blumer et al.
for the size of DAWGs (1985)

Sizes of Sparse Indexing Structures

	max num of nodes	max num of edges	total space complexity
word suffix tries	$k(n+2)/2 + 1$	$k(n+2)/2$	$O(kn)$
word suffix trees	$2k-1$	$2k-2$	$O(n)$
SDAWGs	$O(n)$	$O(n)$	$O(n)$
SCDAWGs	$k+1$	$2k-2$	$O(n)$

Word suffix trees and SCDAWGs need the original text to be kept

n : text length
k : number of words in text
Note that $k \leqq n$

SDAWG Construction

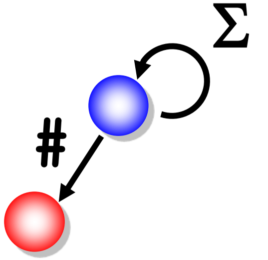
- SDAWGs can be constructed by minimizing word suffix tries in $O(kn)$ time.
  - using Revuz's DAG minimization algorithm (1992)

- Question : Is direct construction of SDAWGs possible?

- Answer : **YES!**
  Using minimum DFA accepting dictionary $D = \Sigma^*\#$, we can directly build SDAWGs in $O(n)$ time.

Minimum DFA Accepting Dictionary D

- The minimum DFA accepting $D = \Sigma^* \#$ clearly requires constant space (for fixed Σ).



Modification of DAWG Construction Algorithm

- Bluer et al. proposed an on-line $O(n)$-time algorithm to construct normal DAWGs (1985).

- We modify their algorithm by:
  - replacing the source node of the DAWG with the final state of the DFA;
  - setting the suffix link of the source node of the DAWG to the initial state of the DFA.

- Then the resulting algorithm constructs SDAWGs in on-line manner and in $O(n)$ time!

Modification of DAWG Construction Algorithm

Input: $w = w[1..n] \in D^+$ and M_D with initial state q_s and final state q_f.
Output: $SDAWG_D(w)$.

{
  $length(q_f) = 0$; $length(q_s) = -1$;
  $source = q_f$; $link(source) = q_s$;
  $sink = source$;
  for $(i = 1; i \leq n; i{+}{+})$ $sink = Update(sink, i)$;
}

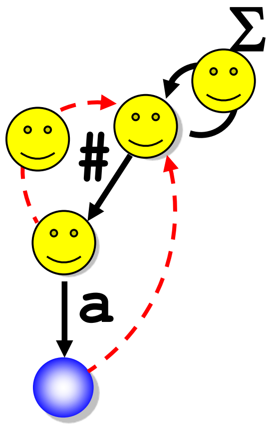
Just change here!!

node $Update(sink, i)$ {
  $c = w[i]$;
  create new node $newsink$; $length(newsink) = i$;
  create new edge $(sink, c, newsink)$;
  for $(s = link(sink)$; no c-edge from s; $s = link(s))$
    create new edge $(s, c, newsink)$;
  $s' = SplitNode(s, c)$;
  $link(newsink) = s'$;
  return $newsink$;
}

node $SplitNode(s, c)$ {
  let s' be the head of the c-edge from s;
  if $(length(s') == length(s) + 1)$ **return** s';
  create node r' as a duplication of s' with the out-going edges;
  $link(r') = link(s')$; $link(s') = r'$;
  $length(r') = length(s) + 1$;
  do {
    replace edge (s, c, s') by edge (s, c, r');
    $s = link(s)$;
  } **while** the head of the c-edge from s is s';
  return r';
}

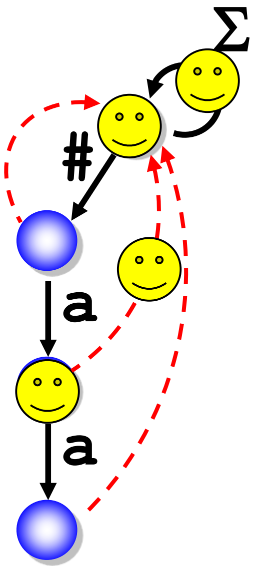
On-line Construction of SDAWG

T **= `aa#b#b...`**



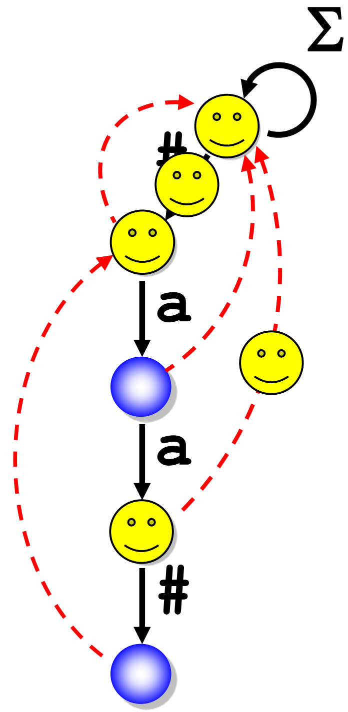
On-line Construction of SDAWG

T = aa#b#b...



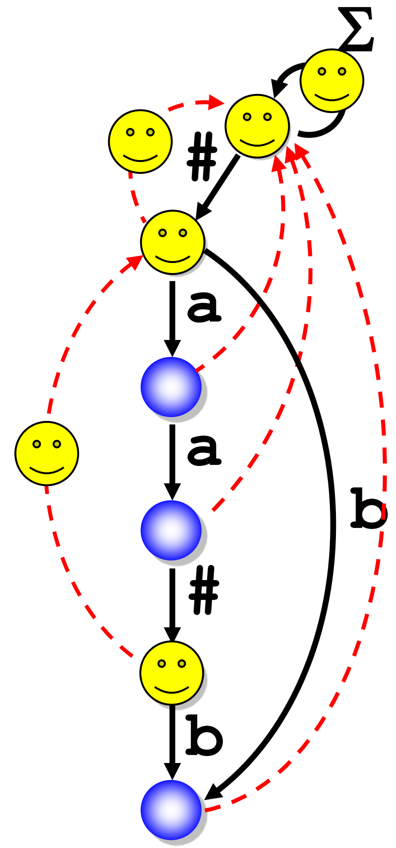
On-line Construction of SDAWG

T = aa#b#b...



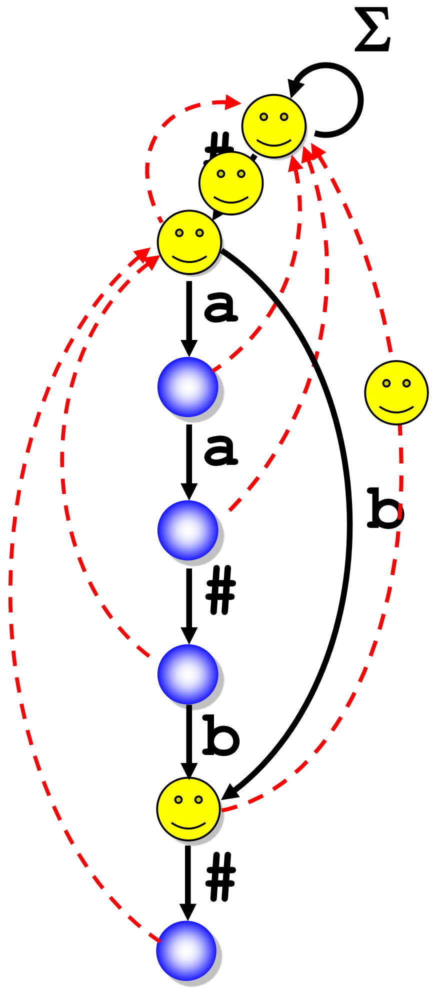
On-line Construction of SDAWG

T = aa#b#b. . .



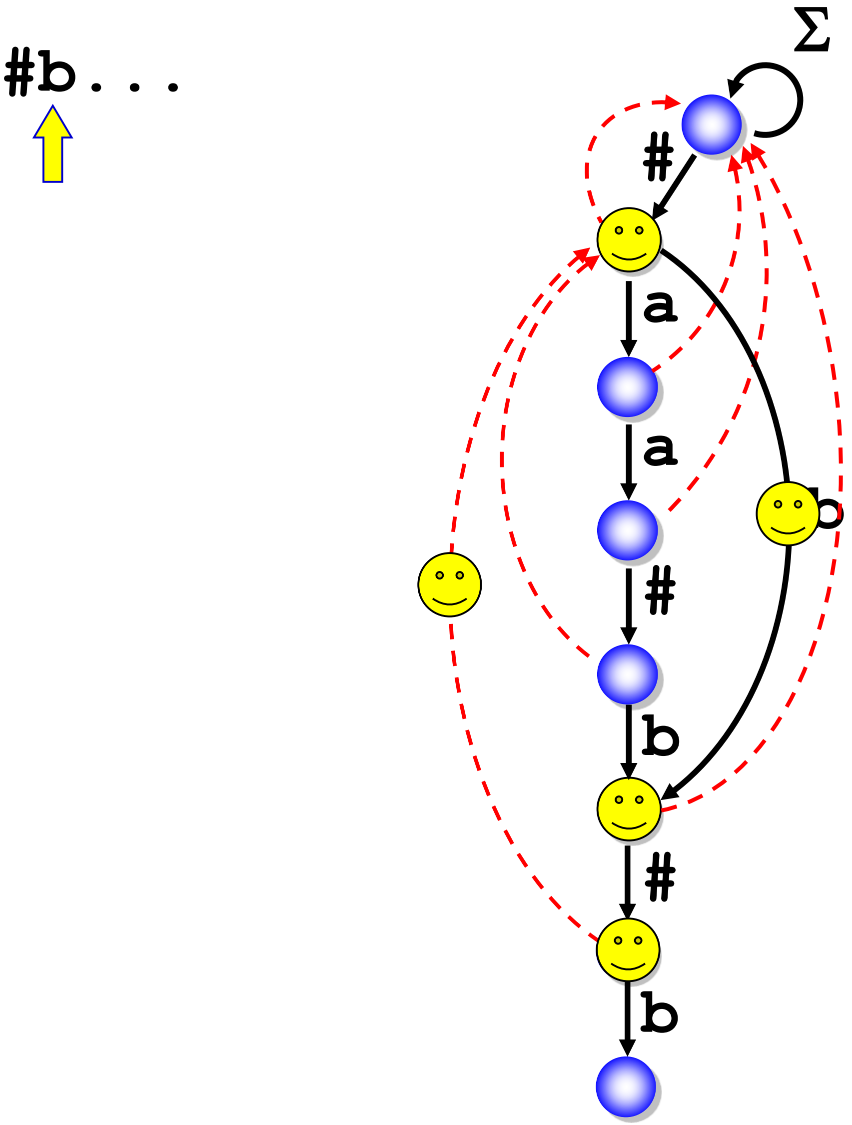
On-line Construction of SDAWG

T = **`aa#b#b...`**



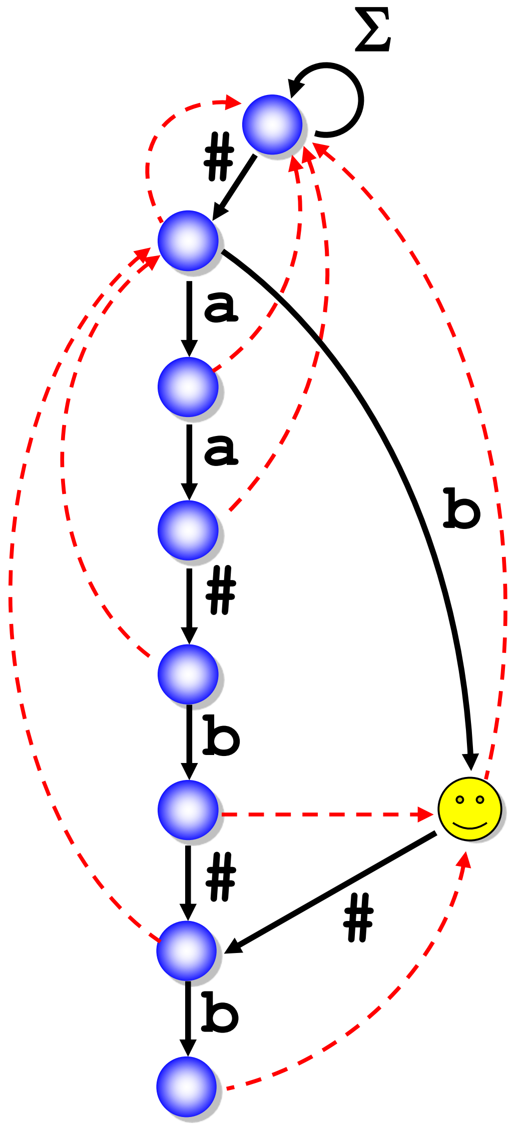
On-line Construction of SDAWG

T = aa#b#b...



On-line Construction of SDAWG

T = aa#b#b. . .



Contents

- *Basic Pattern Matching Problem*
  - Text Indexing Structures
- *Natural Language Text Search*
  - Phrase-level Pattern Matching Problem
  - Sparse Text Indexing Structures
- *Sparse Directed Acyclic Word Graphs*
  - Size
  - Construction
- *Summary and Future Work*

Summary

- We introduced new sparse text indexing structure, *Sparse Directed Acyclic Word Graphs* (*SDAWGs*), that are useful for word- and phrase-level search on natural language texts.
- We showed that SDAWGs require $\boldsymbol{O(n)}$ space.
- We developed an on-line SDAWG construction algorithm running in $\boldsymbol{O(n)}$ time and space (for fixed $\boldsymbol{\Sigma}$).

Future Work

- Exact max numbers of nodes and edges of SDAWGs.

	max num of nodes	max num of edges
word suffix tries	$k(n+2)/2 + 1$	$k(n+2)/2$
word suffix trees	$2k-1$	$2k-2$
SDAWGs	$O(n)$	$O(n)$
SCDAWGs	$k+1$	$2k-2$

- Experiments to evaluate practical space economy of SDAWGs in comparison to normal DAWGs and other sparse indexing structures.

Future Work [cont.]

- Constructing SDAWGs for an arbitrary subset of suffixes.
  - Given: text T and subset S of the positions in T
  - Construct: SDAWG representing only the suffixes starting from the positions in S