Efficiently Testing Simon's Congruence

Paweł Gawrychowski Florin Manea Maria Kosche
Tore Koß Stefan Siemer

Combinatorics on Words Online Seminar
22.02.2021

Subsequences

w $\boxed{b \quad a \quad c \quad b \quad a \quad a \quad b \quad a \quad d \quad a}$

Subsequences

ababa

d

bbb

ba

bc

w $\boxed{b \quad a \quad c \quad b \quad a \quad a \quad b \quad a \quad d \quad a}$

baab

a

bb

c

aaaaa

b

Subsequences

ababa

d

abc

bbb

ba

bc

w $\boxed{b \quad a \quad c \quad b \quad a \quad a \quad b \quad a \quad d \quad a}$

baab

a

bb

c

aaaaa

b

Subsequences

ababa

d

bbb

~~abc~~

ba

bc

w	b	a	c	b	a	a	b	a	d	a

baab

bb

a

c

aaaaa

b

Subsequences

Subsequence

We call w' a subsequence of length k of a word w, where $|w| = n$, if there exist positions $1 \leq i_1 < i_2 < \ldots < i_k \leq n$, such that $w' = w[i_1]w[i_2]\cdots w[i_k]$.

Set of Subsequences of length k

Let $\mathrm{Subseq}_k(i, w)$ denote the set of subsequences of length k of $w[i : n]$. Accordingly, the set of subsequences of length k of the entire word w will be denoted by $\mathrm{Subseq}_k(1, w)$.

Example: $\mathrm{Subseq}_2(1, abaca) = \{aa, ab, ac, ba, bc, ca\}$

Simon's Congruence

Simon's Congruence

(i) Let $w, w' \in \Sigma^*$. We say that w and w' are equivalent under Simon's congruence $\sim_k$ if $\mathsf{Subseq}_k(1, w) = \mathsf{Subseq}_k(1, w')$.

Simon's Congruence

Simon's Congruence

(i) Let $w, w' \in \Sigma^*$. We say that w and w' are equivalent under Simon's congruence $\sim_k$ if $\mathsf{Subseq}_k(1, w) = \mathsf{Subseq}_k(1, w')$.

Example: $w = abacab, w' = baacabba$

Simon's Congruence

Simon's Congruence

(i) Let $w, w' \in \Sigma^*$. We say that w and w' are equivalent under Simon's congruence $\sim_k$ if $\mathsf{Subseq}_k(1, w) = \mathsf{Subseq}_k(1, w')$.

Example: $w = abacab, w' = baacabba$

$$\mathsf{Subseq}_2(1, w) = \{aa, ab, ac, ba, bb, bc, ca, cb\}$$

Simon's Congruence

Simon's Congruence

(i) Let $w, w' \in \Sigma^*$. We say that w and w' are equivalent under Simon's congruence $\sim_k$ if $\mathsf{Subseq}_k(1, w) = \mathsf{Subseq}_k(1, w')$.

Example: $w = abacab, w' = baacabba$

$$\mathsf{Subseq}_2(1, w) = \{aa, ab, ac, ba, bb, bc, ca, cb\}$$
$$\mathsf{Subseq}_2(1, w') = \{aa, ab, ac, ba, bb, bc, ca, cb\}$$

Simon's Congruence

Simon's Congruence

(i) Let $w, w' \in \Sigma^*$. We say that w and w' are equivalent under Simon's congruence $\sim_k$ if $\mathrm{Subseq}_k(1, w) = \mathrm{Subseq}_k(1, w')$.

Example: $w = abacab, w' = baacabba$

$$\mathrm{Subseq}_2(1, w) = \{aa, ab, ac, ba, bb, bc, ca, cb\}$$
$$\mathrm{Subseq}_2(1, w') = \{aa, ab, ac, ba, bb, bc, ca, cb\}$$
$$\mathrm{Subseq}_2(1, w) = \mathrm{Subseq}_2(1, w') \Rightarrow w \sim_2 w'$$

Simon's Congruence

Simon's Congruence

(i) Let $w, w' \in \Sigma^*$. We say that w and w' are equivalent under Simon's congruence $\sim_k$ if $\mathrm{Subseq}_k(1, w) = \mathrm{Subseq}_k(1, w')$.

Example: $w = abacab, w' = baacabba$

Simon's Congruence

Simon's Congruence

(i) Let $w, w' \in \Sigma^*$. We say that w and w' are equivalent under Simon's congruence $\sim_k$ if $\mathsf{Subseq}_k(1, w) = \mathsf{Subseq}_k(1, w')$.

Example: $w = abacab, w' = baacabba$

$$bbb \notin \mathsf{Subseq}_3(1, w), bbb \in \mathsf{Subseq}_3(1, w')$$

Simon's Congruence

Simon's Congruence

(i) Let $w, w' \in \Sigma^*$. We say that w and w' are equivalent under Simon's congruence $\sim_k$ if $\mathsf{Subseq}_k(1, w) = \mathsf{Subseq}_k(1, w')$.

Example: $w = abacab, w' = baacabba$

$$bbb \notin \mathsf{Subseq}_3(1, w), bbb \in \mathsf{Subseq}_3(1, w')$$
$$\mathsf{Subseq}_3(1, w) \neq \mathsf{Subseq}_3(1, w') \Rightarrow w \nsim_3 w'$$

Simon's Congruence

Simon's Congruence

(i) Let $w, w' \in \Sigma^*$. We say that w and w' are equivalent under Simon's congruence $\sim_k$ if $\mathsf{Subseq}_k(1, w) = \mathsf{Subseq}_k(1, w')$.
(ii) Let $1 \leq i < j \leq |w|$. We define $i \sim_k j$ (w.r.t. w) if $w[i:n] \sim_k w[j:n]$, and we say that the positions i and j are k-equivalent.

Simon's Congruence

Simon's Congruence

(i) Let $w, w' \in \Sigma^*$. We say that w and w' are equivalent under Simon's congruence $\sim_k$ if $\mathsf{Subseq}_k(1, w) = \mathsf{Subseq}_k(1, w')$.
(ii) Let $1 \leq i < j \leq |w|$. We define $i \sim_k j$ (w.r.t. w) if $w[i : n] \sim_k w[j : n]$, and we say that the positions i and j are k-equivalent.
(iii) A word u of length k distinguishes w and w' w.r.t. $\sim_k$ if u occurs in exactly one of the sets $\mathsf{Subseq}_k(1, w)$ and $\mathsf{Subseq}_k(1, w')$.

Problem Definition

SimK

Given two words s and t over an alphabet Σ, with $|s| = n$ and $|t| = n'$, with $n \geq n'$, and a natural number k, decide whether $s \sim_k t$.

MaxSimK

Given two words s and t over an alphabet Σ, with $|s| = n$ and $|t| = n'$, with $n \geq n'$, find the maximum k for which $s \sim_k t$.

History

- Line of research originating in the PhD thesis of Imre Simon from 1972

History

- Line of research originating in the PhD thesis of Imre Simon from 1972
- Long history of algorithm designs and improvements for associated problems. State of the art:
  – $\textsc{SimK}$ linear time solution for constant alphabets, via shortlex form [Kufleitner, Fleischer, MFCS 2018]
  – $\textsc{SimK}$ optimal linear time solution for integer alphabets (and optimal solution in general), via shortlex form [DLT 2020]

History

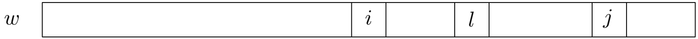
- Line of research originating in the PhD thesis of Imre Simon from 1972
- Long history of algorithm designs and improvements for associated problems. State of the art:
  – SimK linear time solution for constant alphabets, via shortlex form [Kufleitner, Fleischer, MFCS 2018]
  – SimK optimal linear time solution for integer alphabets (and optimal solution in general), via shortlex form [DLT 2020]
  – MaxSimK $O(n \log n)$ time [based on DLT 2020]
  – Simon claimed a linear time solution for MaxSimK in 2003, but never published it.

History

- Line of research originating in the PhD thesis of Imre Simon from 1972
- Long history of algorithm designs and improvements for associated problems. State of the art:
  – SimK linear time solution for constant alphabets, via shortlex form [Kufleitner, Fleischer, MFCS 2018]
  – SimK optimal linear time solution for integer alphabets (and optimal solution in general), via shortlex form [DLT 2020]
  – MaxSimK $O(n \log n)$ time [based on DLT 2020]
  – Simon claimed a linear time solution for MaxSimK in 2003, but never published it.
- Today: the first optimal linear-time algorithm for the MaxSimK problem. [STACS 2021]

Simon-tree

Equivalence Classes



$$\mathcal{SF}_k(i,w) \supset \mathcal{SF}_k(l,w) \supset \mathcal{SF}_k(j,w)$$

- Splitting a word suffixwise into blocks of equivalence classes w.r.t. $\sim_k$
- If $i \sim_k j$, then $\mathrm{Subseq}_k(i,w) = \mathrm{Subseq}_k(l,w) = \mathrm{Subseq}_k(j,w)$ and we say that i, l, and j are in the same *k-block*
- $\sim_{k+1}$ is a refinement of $\sim_k$
- Index i is a $(k+1)$*-splitting position* if $i \sim_k i+1$ but not $i \sim_{k+1} i+1$

Equivalence Classes

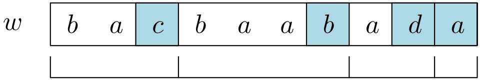
Use these properties to build a block structure for a word

1. $i \sim_1 j$ iff $\mathsf{alph}(w[i:n]) = \mathsf{alph}(w[j:n])$ for any $1 \leq i < j \leq |w|$
   $\rightarrow$ We can go from right to left through the word and determine 1-splitting positions

Equivalence Classes

Use these properties to build a block structure for a word

1. $i \sim_1 j$ iff $\mathsf{alph}(w[i:n]) = \mathsf{alph}(w[j:n])$ for any $1 \leq i < j \leq |w|$
   $\rightarrow$ We can go from right to left through the word and determine 1-splitting positions

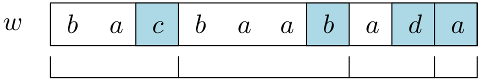


Equivalence Classes

Use these properties to build a block structure for a word

1. $i \sim_1 j$ iff $\mathrm{alph}(w[i:n]) = \mathrm{alph}(w[j:n])$ for any $1 \leq i < j \leq |w|$
   $\rightarrow$ We can go from right to left through the word and determine 1-splitting positions

1-blocks

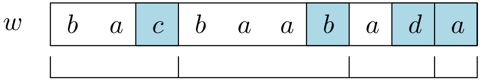


Equivalence Classes

Use these properties to build a block structure for a word

1. $i \sim_1 j$ iff $\mathsf{alph}(w[i:n]) = \mathsf{alph}(w[j:n])$ for any $1 \leq i < j \leq |w|$
   $\rightarrow$ We can go from right to left through the word and determine 1-splitting positions
2. Split a k-block $w[m_a : n_a]$ into:
   – the $(k+1)$-block containing n_a only and then
   – the $(k+1)$-blocks obtained by going from right to left through $w[m_a : n_a - 1]$ and determining the $(k+1)$-splitting positions **exactly** as for 1-splitting positions.

1-blocks

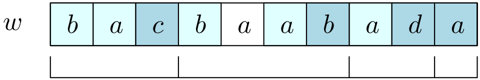


Equivalence Classes

Use these properties to build a block structure for a word

1. $i \sim_1 j$ iff $\mathsf{alph}(w[i:n]) = \mathsf{alph}(w[j:n])$ for any $1 \leq i < j \leq |w|$
   $\rightarrow$ We can go from right to left through the word and determine 1-splitting positions
2. Split a k-block $w[m_a : n_a]$ into:
   – the $(k+1)$-block containing n_a only and then
   – the $(k+1)$-blocks obtained by going from right to left through $w[m_a : n_a - 1]$ and determining the $(k+1)$-splitting positions **exactly** as for 1-splitting positions.

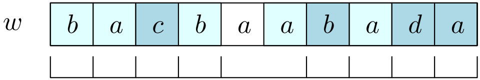
1-blocks



Equivalence Classes

Use these properties to build a block structure for a word

1. $i \sim_1 j$ iff $\mathrm{alph}(w[i:n]) = \mathrm{alph}(w[j:n])$ for any $1 \leq i < j \leq |w|$
   $\rightarrow$ We can go from right to left through the word and determine 1-splitting positions
2. Split a k-block $w[m_a : n_a]$ into:
   – the $(k+1)$-block containing n_a only and then
   – the $(k+1)$-blocks obtained by going from right to left through $w[m_a : n_a - 1]$ and determining the $(k+1)$-splitting positions **exactly** as for 1-splitting positions.

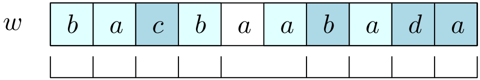


Equivalence Classes

Use these properties to build a block structure for a word

1. $i \sim_1 j$ iff $\mathrm{alph}(w[i:n]) = \mathrm{alph}(w[j:n])$ for any $1 \leq i < j \leq |w|$
   $\rightarrow$ We can go from right to left through the word and determine 1-splitting positions
2. Split a k-block $w[m_a : n_a]$ into:
   – the $(k+1)$-block containing n_a only and then
   – the $(k+1)$-blocks obtained by going from right to left through $w[m_a : n_a - 1]$ and determining the $(k+1)$-splitting positions **exactly** as for 1-splitting positions.

2-blocks

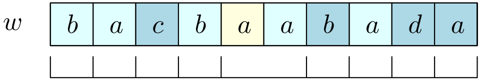


Equivalence Classes

Use these properties to build a block structure for a word

1. $i \sim_1 j$ iff $\mathrm{alph}(w[i:n]) = \mathrm{alph}(w[j:n])$ for any $1 \leq i < j \leq |w|$
   $\rightarrow$ We can go from right to left through the word and determine 1-splitting positions
2. Split a k-block $w[m_a : n_a]$ into:
   – the $(k+1)$-block containing n_a only and then
   – the $(k+1)$-blocks obtained by going from right to left through $w[m_a : n_a - 1]$ and determining the $(k+1)$-splitting positions **exactly** as for 1-splitting positions.

2-blocks



Equivalence Classes

Use these properties to build a block structure for a word

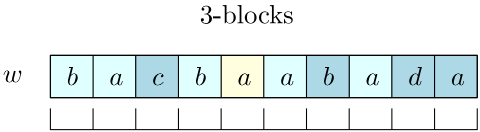
1. $i \sim_1 j$ iff $\mathsf{alph}(w[i:n]) = \mathsf{alph}(w[j:n])$ for any $1 \leq i < j \leq |w|$
   $\rightarrow$ We can go from right to left through the word and determine 1-splitting positions
2. Split a k-block $w[m_a : n_a]$ into:
   – the $(k+1)$-block containing n_a only and then
   – the $(k+1)$-blocks obtained by going from right to left through $w[m_a : n_a - 1]$ and determining the $(k+1)$-splitting positions **exactly** as for 1-splitting positions.

w	b	a	c	b	a	a	b	a	d	a

Equivalence Classes

Use these properties to build a block structure for a word

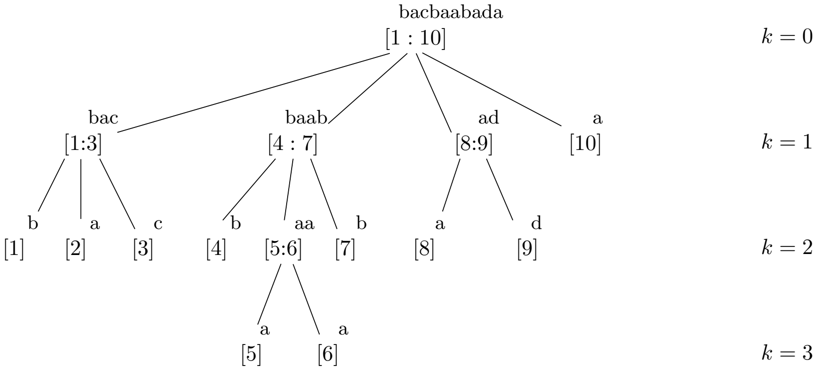
1. $i \sim_1 j$ iff $\mathsf{alph}(w[i:n]) = \mathsf{alph}(w[j:n])$ for any $1 \leq i < j \leq |w|$
   $\rightarrow$ We can go from right to left through the word and determine 1-splitting positions
2. Split a k-block $w[m_a : n_a]$ into:
   – the $(k+1)$-block containing n_a only and then
   – the $(k+1)$-blocks obtained by going from right to left through $w[m_a : n_a - 1]$ and determining the $(k+1)$-splitting positions **exactly** as for 1-splitting positions.



Simon-tree Definition

- New data structure: Simon-tree
- Represents presented block structure
- Efficiently partition positions of a given word
- Construction takes linear time

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



Simon-tree Definition

Simon-tree

The *Simon-tree* T_w associated to the word w, with $|w| = n$, is an ordered rooted tree. The nodes represent $k-$blocks of w, for $0 \leq k \leq n$, and are defined recursively.

- The root corresponds to the 0-block of the word w, i.e., the interval $[1 : n]$.
- For $k > 1$ and for a node b on level $k - 1$, which represents a $(k-1)$-block $[i : j]$ with $i < j$, the children of b are exactly the blocks of the partition of $[i : j]$ in k-blocks, ordered decreasingly by their starting position.
- For $k > 1$, each node on the level $k - 1$ which represents a $(k-1)$-block $[i : i]$ is a leaf.

Simon-tree Construction

- **Algorithm:** Build the Simon-tree right to left as the word is traversed right to left. Only the leftmost branch is edited during construction.
    1. Insert the new position/letter into the tree by moving up the leftmost branch from leaf to root.
    2. Find lowest node that is not split by this position (and close all the others on the way).
    3. Insert the new position/letter as a leftmost child of this node.

Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞

Simon-tree Construction

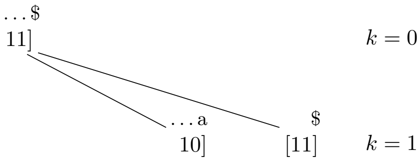
position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞

$$\begin{array}{c} \ldots\$ \\ 11] \end{array}$$

$k = 0$

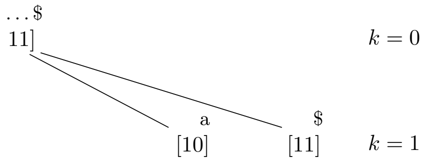
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



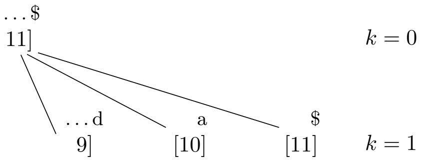
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	\$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



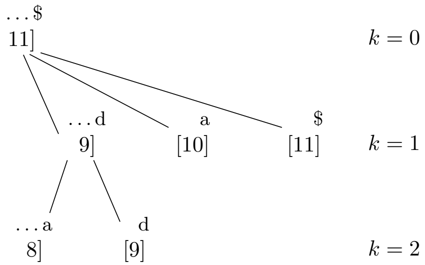
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	\$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



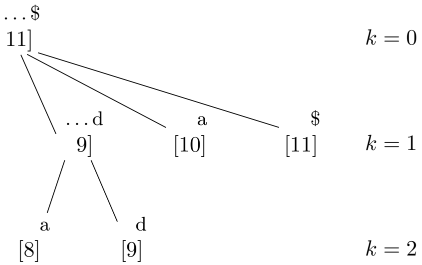
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	\$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



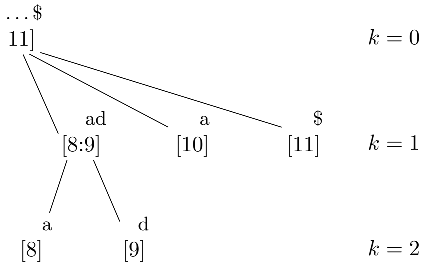
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	\$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



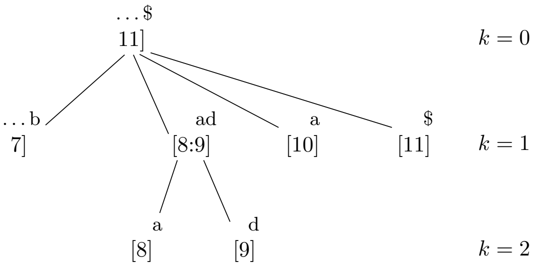
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



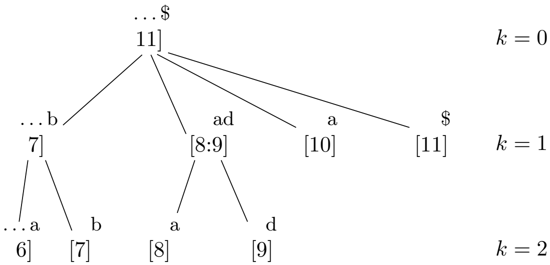
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	\$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



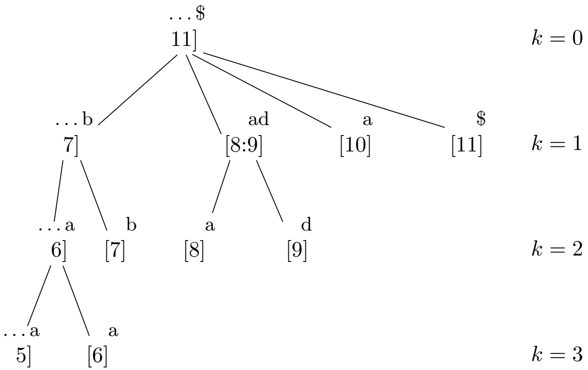
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



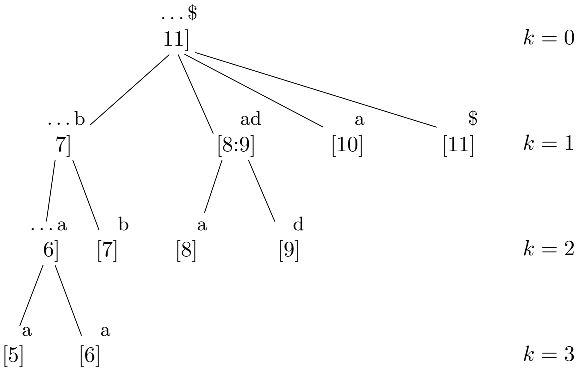
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



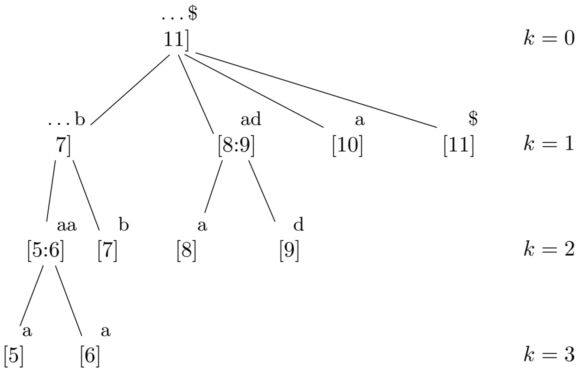
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	\$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



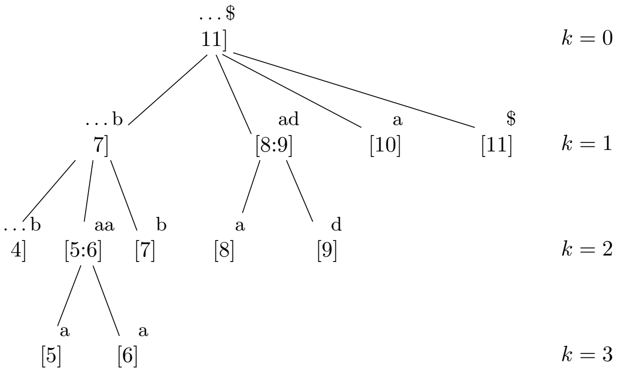
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



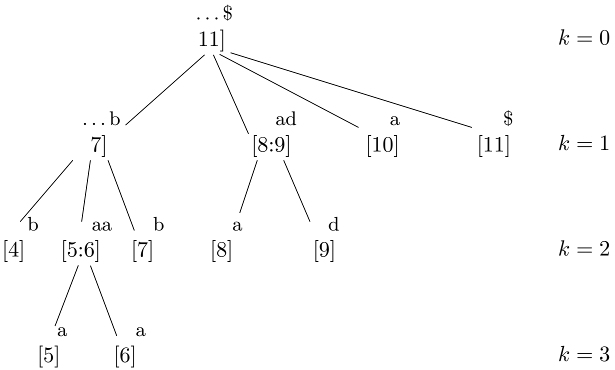
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



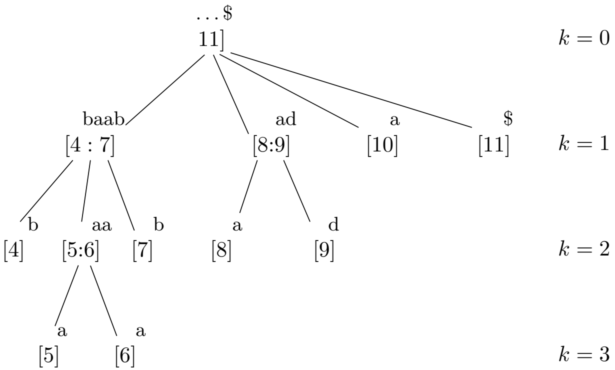
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	\$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



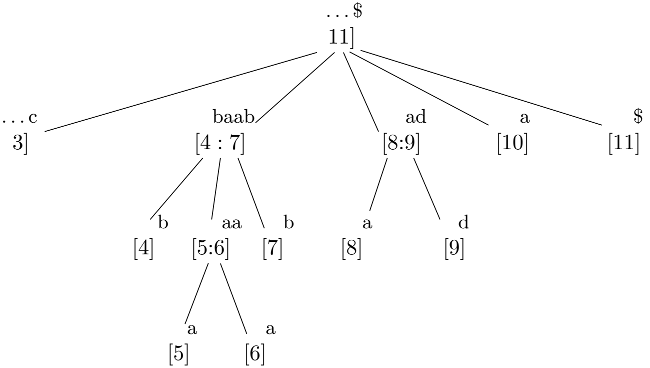
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	\$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



$k = 0$

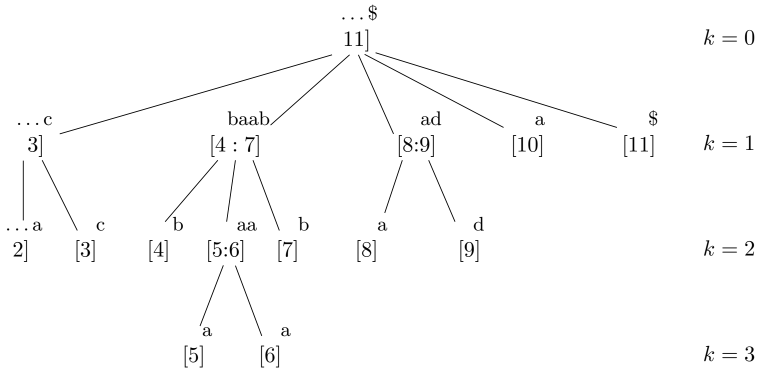
$k = 1$

$k = 2$

$k = 3$

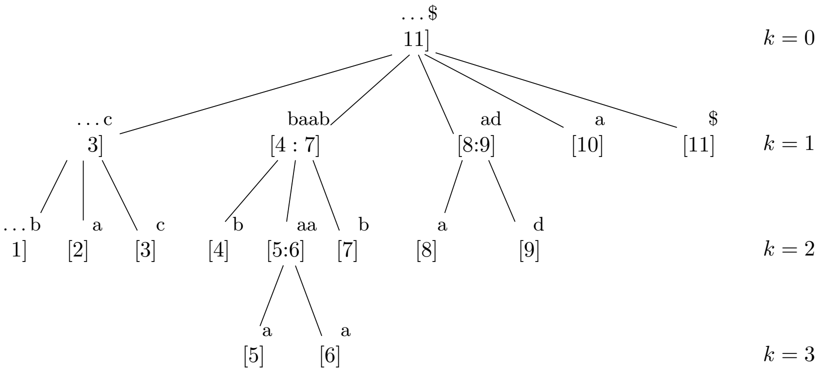
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	\$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



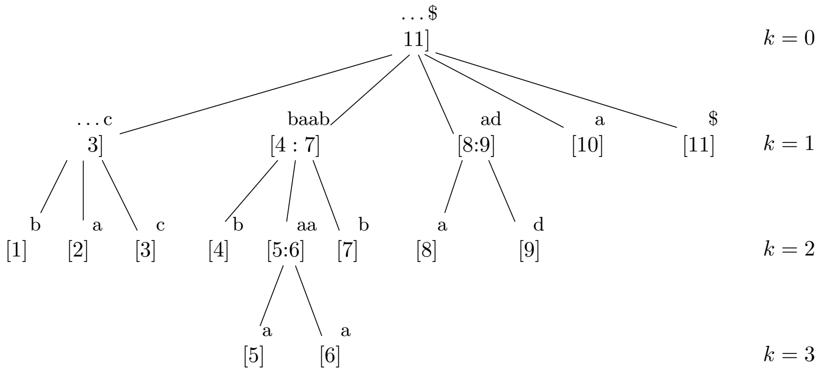
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



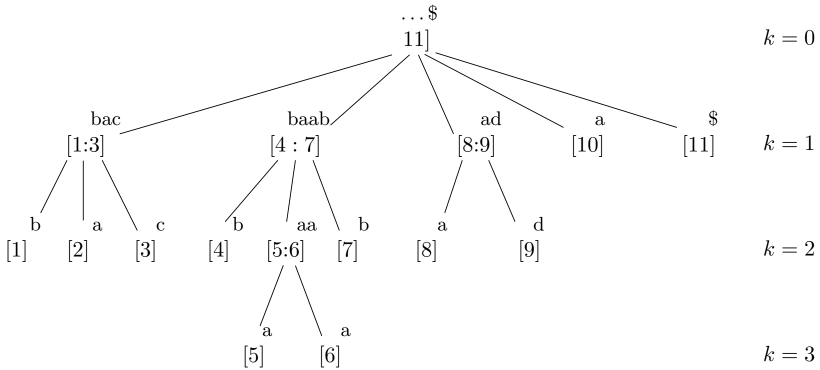
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	\$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



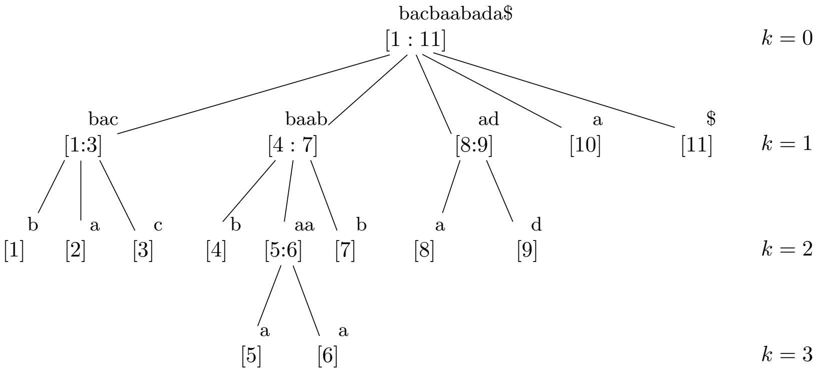
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



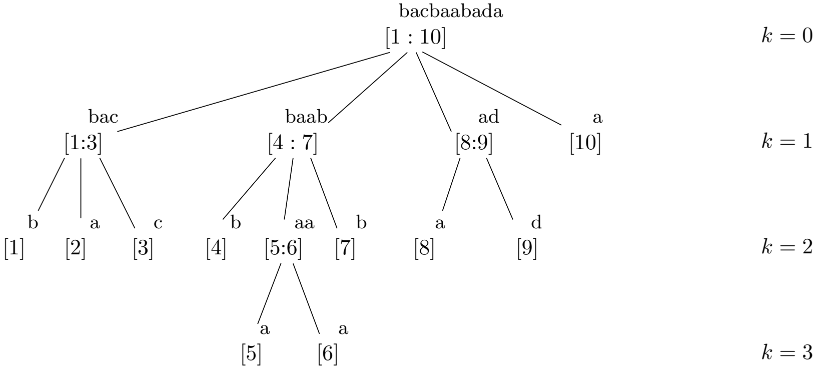
Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



Simon-tree Construction

position	1	2	3	4	5	6	7	8	9	10	11
w	b	a	c	b	a	a	b	a	d	a	\$
X	4	5	∞	7	6	8	∞	10	∞	∞	∞



Simon-tree Construction

- **Algorithm:** Build the Simon-tree right to left as the word is traversed right to left. Only the leftmost branch is edited during construction.
- **Complexity:** linear! All nodes appear only once on the leftmost path, until they are closed.

Short Recap

So far:
structure for one word representing the equivalence classes w.r.t. $\sim_k$

MaxSimK
Given two words s and t over an alphabet Σ, with $|s| = n$ and $|t| = n'$, with $n \geq n'$, find the maximum k for which $s \sim_k t$.

Now:
set two words in relation to each other by using their respective Simon-trees

Connecting Two Simon-trees

- Transform the words s and t into Simon-trees as shown.
- Use the tree structure to connect equivalent nodes of the two words.

Connecting Two Simon-trees

- Transform the words s and t into Simon-trees as shown.
- Use the tree structure to connect equivalent nodes of the two words.

S-Connection

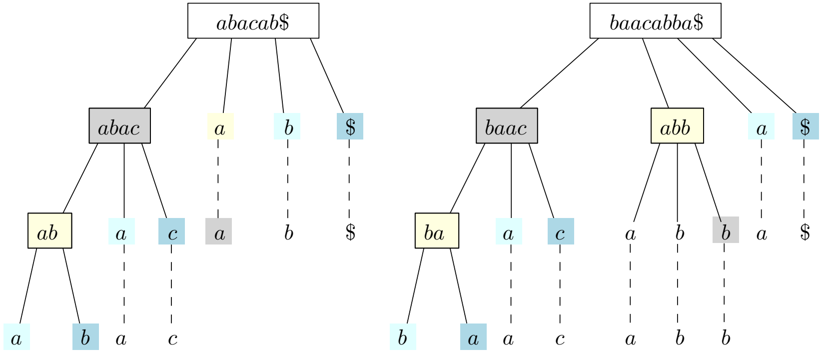
The k-node a of T_s and the k-node b of T_t are S-connected (i.e., the pair (a, b) is in the S-connection) if and only if $s[i:n] \sim_k t[j:n']$ for all positions i in block a and positions j in block b.

From P-Connection to S-Connection

Starting from a larger relation (P-Connection) which contains the S-Connection, and refine it.

- The 0-nodes of T_s and T_t are P-connected.
- For all levels k of T_s, if the explicit or implicit k-nodes a and b (from T_s and T_t, respectively) are P-connected, then the i^{th} child of a is P-connected to the i^{th} child of b, for all i.
- No other nodes are P-connected.

From P-Connection to S-Connection

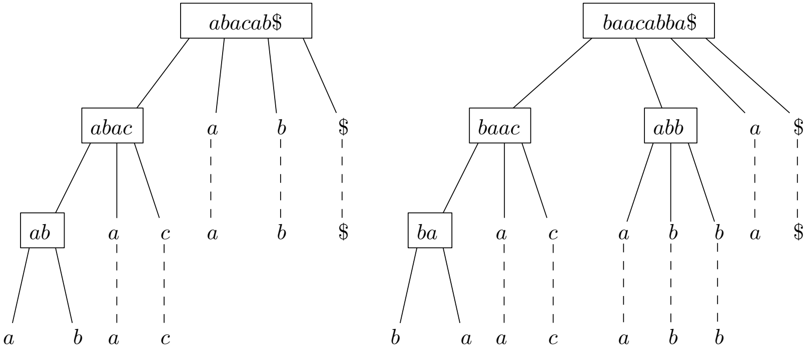


From P-Connection to S-Connection

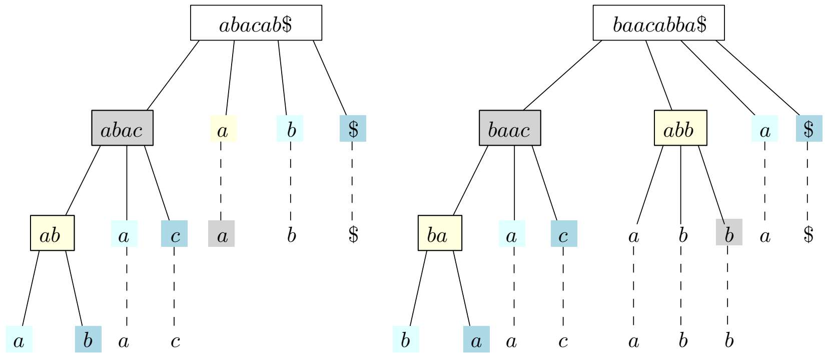
How to refine the P-Connection:

- Let $k \geq 1$ and a, b be k-blocks in the word t, resp. s, which are S-connected.
- Let a' be child of a, b' be child of b.
- $a' \sim_{k+1} b'$ if and only if there exists a letter x such that $s[\mathtt{next}(a', x) + 1 : n] \sim_k t[\mathtt{next}(b', x) + 1 : n']$.

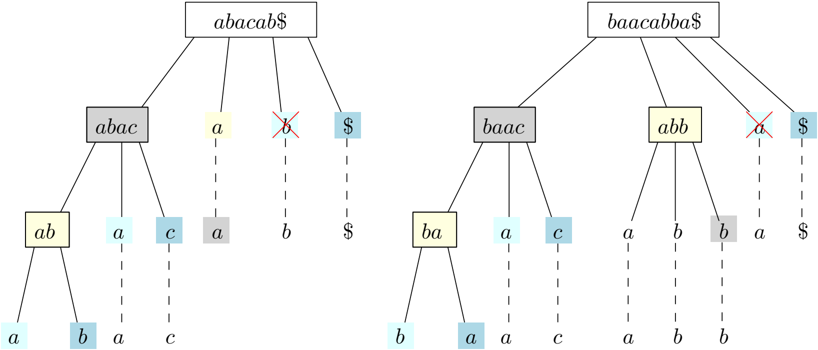
From P-Connection to S-Connection



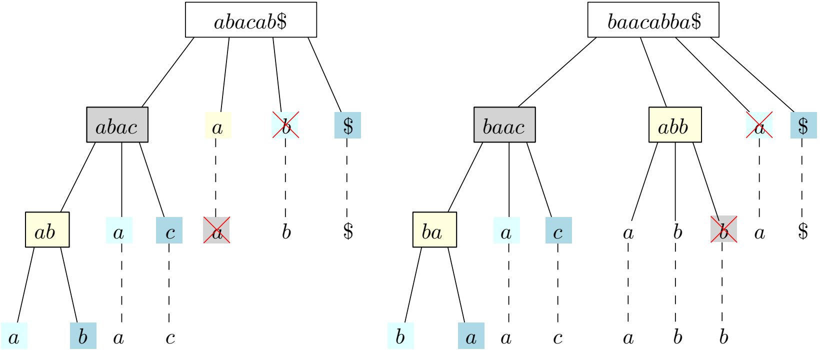
From P-Connection to S-Connection



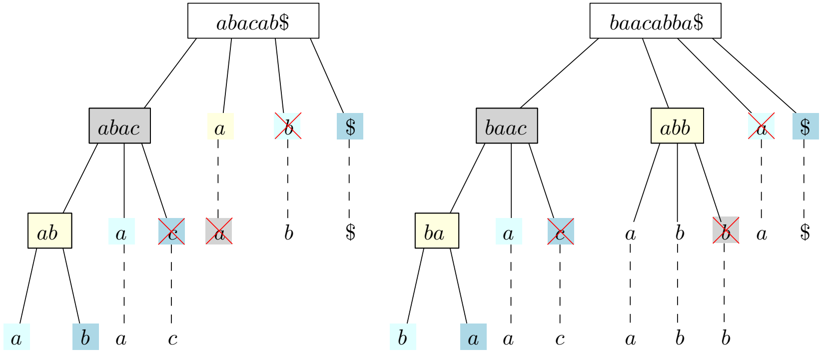
From P-Connection to S-Connection



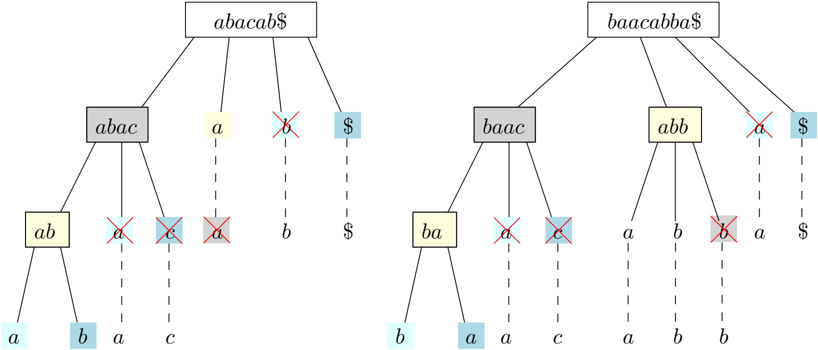
From P-Connection to S-Connection



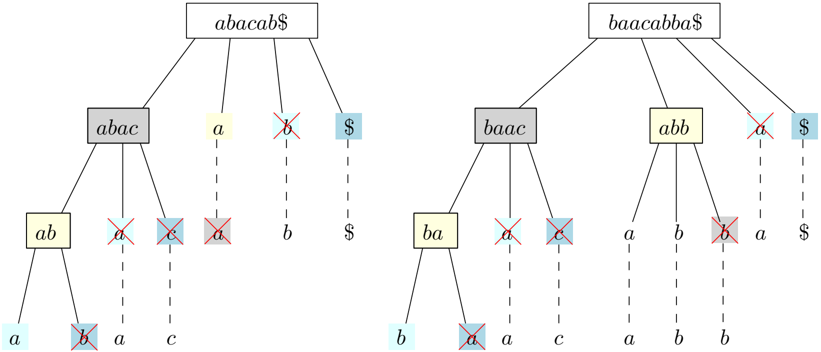
From P-Connection to S-Connection



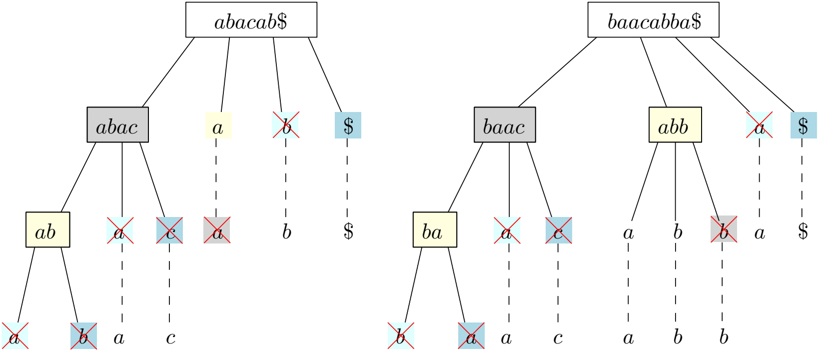
From P-Connection to S-Connection



From P-Connection to S-Connection



From P-Connection to S-Connection



Additional Notes and Analysis

- Solution of MaxSimK: last level k where the k-blocks containing position 1 of the input words are equivalent.
- Distinguishing word can be obtained.
- By efficiently using interval-union-find and -split-find data structures the algorithm achieves an optimal linear runtime.

Theorem

MaxSimK*can be solved in optimal linear time.*

The End

Future Work

- Edit/Hamming Distance to $\sim_k$-equivalence.
  First steps: [Day, Fleischmann, Kosche, Tore Koß, M., Siemer: The Edit Distance to k-Subsequence Universality, STACS'21]

The End

Future Work

- Edit/Hamming Distance to $\sim_k$-equivalence.
  First steps: [Day, Fleischmann, Kosche, Tore Koß, M., Siemer: The Edit Distance to k-Subsequence Universality, STACS'21]

Thank You!