Practical Performance of Space Efficient Data Structures for Longest Common Extensions

Patrick Dinklage   Johannes Fischer   Alexander Herlez
Tomasz Kociumaka   *Florian Kurpicz*

technische universität dortmund

Bar-Ilan University
אוניברסיטת בר אילן

Longest Common Extensions (LCEs)

Given: Text $T[1, n]$ over an alphabet of size σ

Wanted: Data Structure that answers

$$\mathrm{lce}_T(i, j) = \max\{\ell \geq 0 \colon T[i, i + \ell) = T[j, j + \ell)\}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	A	B	C	D	A	B	C	C	D	B	C	C	B	A	B	C	D	A	D	A

Longest Common Extensions (LCEs)

Given: Text $T[1, n]$ over an alphabet of size σ

Wanted: Data Structure that answers

$$\mathrm{lce}_T(i, j) = \max\{\ell \geq 0 \colon T[i, i+\ell) = T[j, j+\ell)\}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	A	B	C	D	A	B	C	C	D	B	C	C	B	A	B	C	D	A	D	A

$$\mathrm{lce}_T(1, 14) = 0\ 1\ 2\ 3\ 4\ 5$$

Longest Common Extensions (LCEs)

Given: Text $T[1,n]$ over an alphabet of size σ

Wanted: Data Structure that answers

$$\mathrm{lce}_T(i,j) = \max\{\ell \geq 0 \colon T[i, i+\ell) = T[j, j+\ell)\}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	A	B	C	D	A	B	C	C	D	B	C	C	B	A	B	C	D	A	D	A

$\mathrm{lce}_T(1,14) = 0\ 1\ 2\ 3\ 4\ 5$

Longest Common Extensions (LCEs)

Given: Text $T[1, n]$ over an alphabet of size σ

Wanted: Data Structure that answers

$$\mathrm{lce}_T(i, j) = \max\{\ell \geq 0 \colon T[i, i+\ell) = T[j, j+\ell)\}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	A	B	C	D	A	B	C	C	D	B	C	C	B	A	B	C	D	A	D	A

$\mathrm{lce}_T(1, 14) = 0\ 1\ 2\ 3\ 4\ 5$

Applications

- (sparse) suffix sorting
- approximate pattern matching
- ...

Longest Common Extensions (LCEs)

Given: Text $T[1, n]$ over an alphabet of size σ

Wanted: Data Structure that answers

$$\mathrm{lce}_T(i, j) = \max\{\ell \geq 0 \colon T[i, i+\ell) = T[j, j+\ell)\}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	A	B	C	D	A	B	C	C	D	B	C	C	B	A	B	C	D	A	D	A

$\mathrm{lce}_T(1, 14) = 0\ 1\ 2\ 3\ 4\ 5$

Applications

- (sparse) suffix sorting
- approximate pattern matching
- ...

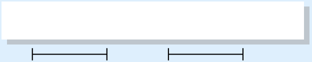
- we are interested in practical results
- for related theoretical work see paper

Practical Algorithms for LCEs [Ilie and Tinta, SPIRE'09]

Practical Algorithms for LCEs [Ilie and Tinta, SPIRE'09]

Ultra Naive Scan (UNS)

- compare character by character



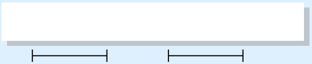
Query Time: $O(n)$

Space: no additional space

Practical Algorithms for LCEs [Ilie and Tinta, SPIRE'09]

Ultra Naive Scan (UNS)

- compare character by character

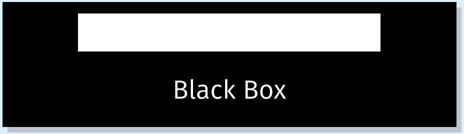


Query Time: $O(n)$

Space: no additional space

Sophisticated Black Box (BB)

- based on ISA, LCP, and RMQ



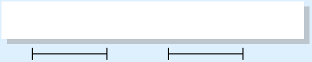
Query Time: $O(1)$

Space: $\approx 9n$ additional bytes

Practical Algorithms for LCEs [Ilie and Tinta, SPIRE'09]

Ultra Naive Scan (UNS)

- compare character by character

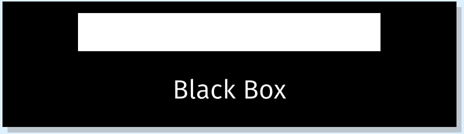


Query Time: $O(n)$

Space: no additional space

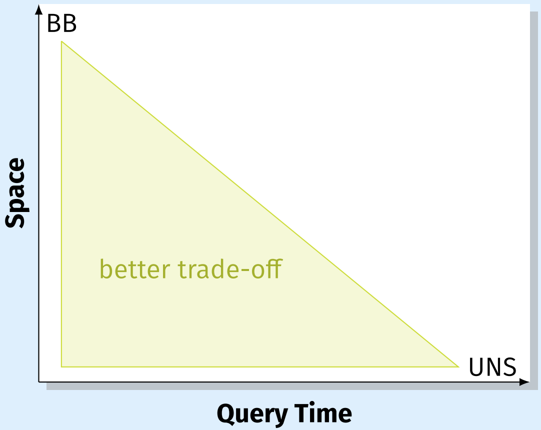
Sophisticated Black Box (BB)

- based on ISA, LCP, and RMQ



Query Time: $O(1)$

Space: $\approx 9n$ additional bytes



Space Efficient LCE Data Structures

Fingerprints [Prezza, SODA'18]

- Karp-Rabin fingerprints for random prime q
- $(i, j) = \left(\sum_{z=i}^{j} T[z] \cdot \sigma^{j-z}\right) \mod q$

Space Efficient LCE Data Structures

Fingerprints [Prezza, SODA'18]

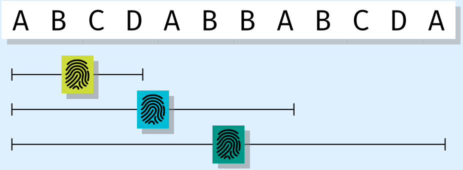
- Karp-Rabin fingerprints for random prime q
- 🖐$(i, j) = (\sum_{z=i}^{j} T[z] \cdot \sigma^{j-z}) \mod q$

A	B	C	D	A	B	B	A	B	C	D	A

Space Efficient LCE Data Structures

Fingerprints [Prezza, SODA'18]

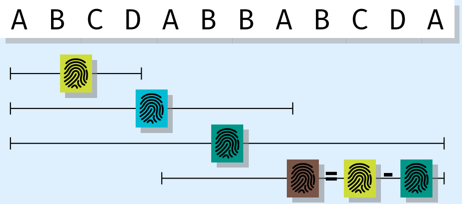
- Karp-Rabin fingerprints for random prime q
- 🫆$(i, j) = \left(\sum_{z=i}^{j} T[z] \cdot \sigma^{j-z}\right) \mod q$



Space Efficient LCE Data Structures

Fingerprints [Prezza, SODA'18]

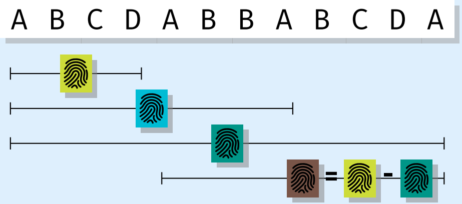
- Karp-Rabin fingerprints for random prime q
- $(i, j) = \left(\sum_{z=i}^{j} T[z] \cdot \sigma^{j-z}\right) \mod q$



Space Efficient LCE Data Structures

Fingerprints [Prezza, SODA'18]

- Karp-Rabin fingerprints for random prime q
- $(i, j) = \left(\sum_{z=i}^{j} T[z] \cdot \sigma^{j-z}\right) \mod q$

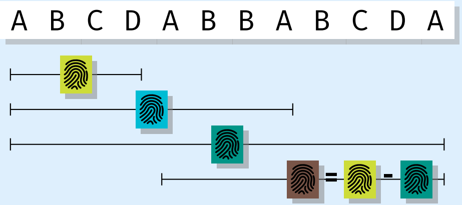


- overwrite text with fingerprints (in-place)

Space Efficient LCE Data Structures

Fingerprints [Prezza, SODA'18]

- Karp-Rabin fingerprints for random prime q
- $(i,j) = \left(\sum_{z=i}^{j} T[z] \cdot \sigma^{j-z}\right) \mod q$



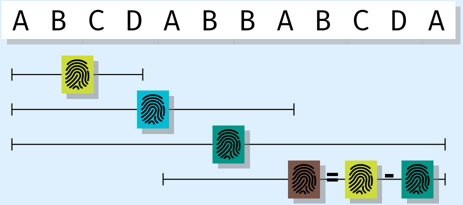
- overwrite text with fingerprints (in-place)

- all parts of text are restorable

Space Efficient LCE Data Structures

Fingerprints [Prezza, SODA'18]

- Karp-Rabin fingerprints for random prime q
- $(i,j) = \left(\sum_{z=i}^{j} T[z] \cdot \sigma^{j-z}\right) \mod q$



- overwrite text with fingerprints (in-place)

- all parts of text are restorable

Compute LCE with Fingerprints

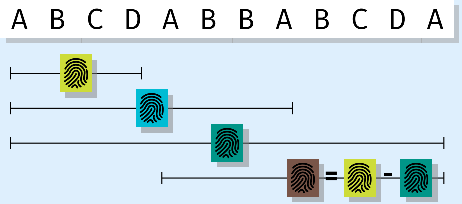
exponential search: fingerprints mismatch

binary search: identify block mismatch

Space Efficient LCE Data Structures

Fingerprints [Prezza, SODA'18]

- Karp-Rabin fingerprints for random prime q
- $(i,j) = \left(\sum_{z=i}^{j} T[z] \cdot \sigma^{j-z}\right) \mod q$



- overwrite text with fingerprints (in-place)

- all parts of text are restorable

Compute LCE with Fingerprints

exponential search: fingerprints mismatch

binary search: identify block mismatch

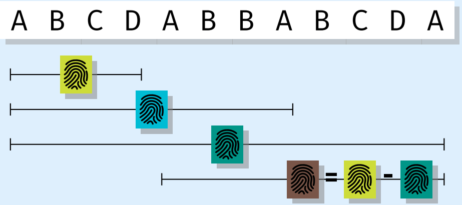
In Practice

- 8 characters per block (byte alphabet)
- use `uint_128` to compute fingerprints
- restore text for 256 characters before starting exponential search

Space Efficient LCE Data Structures

Fingerprints [Prezza, SODA'18]

- Karp-Rabin fingerprints for random prime q
- $(i, j) = \left(\sum_{z=i}^{j} T[z] \cdot \sigma^{j-z}\right) \mod q$



- overwrite text with fingerprints (in-place)

- all parts of text are restorable

Compute LCE with Fingerprints

exponential search: fingerprints mismatch

binary search: identify block mismatch

In Practice

- 8 characters per block (byte alphabet)
- use `uint_128` to compute fingerprints
- restore text for 256 characters before starting exponential search

String Synchronizing Sets [Kempa & Kociumaka, STOC'19]

1. string synchronizing sets in practice
2. solving LCE queries
3. practical improvements

String Synchronizing Sets (SSS) [Kempa and Kociumaka, STOC'19]

Simplified τ-Synchronizing Set

Given: Text $T[1, n]$ and $0 < \tau \leq n/2$

Wanted: τ-synchronizing set S of T

$$S = \{i \in [1, n{-}2\tau{+}1] : min\{\text{🫆}(j, j{+}\tau{-}1) : j \in [i, i{+}\tau]\} \in \{\text{🫆}(i, i{+}\tau{-}1), \text{🫆}(i{+}\tau, i{+}2\tau{-}1)\}\}$$

T 

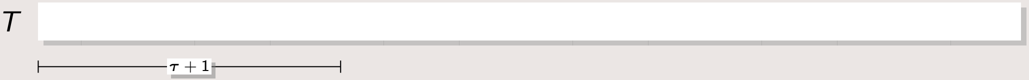
String Synchronizing Sets (SSS) [Kempa and Kociumaka, STOC'19]

Simplified τ-Synchronizing Set

Given: Text $T[1, n]$ and $0 < \tau \leq n/2$

Wanted: τ-synchronizing set S of T

$$S = \{i \in [1, n{-}2\tau{+}1] : \min\{\mathrm{fp}(j, j{+}\tau{-}1) : j \in [i, i{+}\tau]\} \in \{\mathrm{fp}(i, i{+}\tau{-}1), \mathrm{fp}(i{+}\tau, i{+}2\tau{-}1)\}\}$$



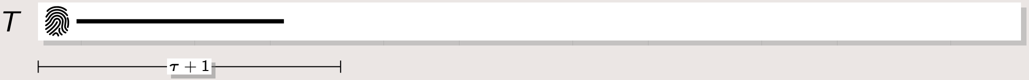
String Synchronizing Sets (SSS) [Kempa and Kociumaka, STOC'19]

Simplified τ-Synchronizing Set

Given: Text $T[1, n]$ and $0 < \tau \leq n/2$

Wanted: τ-synchronizing set S of T

$$S = \{i \in [1, n{-}2\tau{+}1] : min\{\text{🫆}(j, j{+}\tau{-}1) : j \in [i, i{+}\tau]\} \in \{\text{🫆}(i, i{+}\tau{-}1), \text{🫆}(i{+}\tau, i{+}2\tau{-}1)\}\}$$



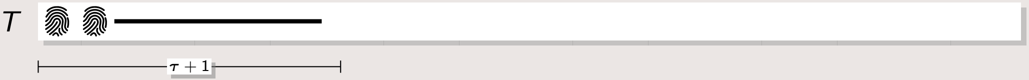
String Synchronizing Sets (SSS) [Kempa and Kociumaka, STOC'19]

Simplified τ-Synchronizing Set

Given: Text $T[1, n]$ and $0 < \tau \leq n/2$

Wanted: τ-synchronizing set S of T

$$S = \{i \in [1, n{-}2\tau{+}1] \colon \mathit{min}\{\text{🫆}(j, j{+}\tau{-}1) \colon j \in [i, i{+}\tau]\} \in \{\text{🫆}(i, i{+}\tau{-}1), \text{🫆}(i{+}\tau, i{+}2\tau{-}1)\}\}$$



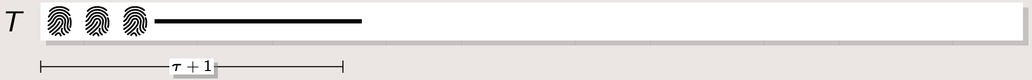
String Synchronizing Sets (SSS) [Kempa and Kociumaka, STOC'19]

Simplified τ-Synchronizing Set

Given: Text $T[1,n]$ and $0 < \tau \leq n/2$

Wanted: τ-synchronizing set S of T

$$S = \{i \in [1, n{-}2\tau{+}1] : min\{\text{🫆}(j, j{+}\tau{-}1) : j \in [i, i{+}\tau]\} \in \{\text{🫆}(i, i{+}\tau{-}1), \text{🫆}(i{+}\tau, i{+}2\tau{-}1)\}\}$$



String Synchronizing Sets (SSS) [Kempa and Kociumaka, STOC’19]

Simplified τ-Synchronizing Set

Given: Text $T[1, n]$ and $0 < \tau \leq n/2$

Wanted: τ-synchronizing set S of T

$$S = \{i \in [1, n{-}2\tau{+}1] \colon \mathit{min}\{🫆(j, j{+}\tau{-}1) \colon j \in [i, i{+}\tau]\} \in \{🫆(i, i{+}\tau{-}1), 🫆(i{+}\tau, i{+}2\tau{-}1)\}\}$$

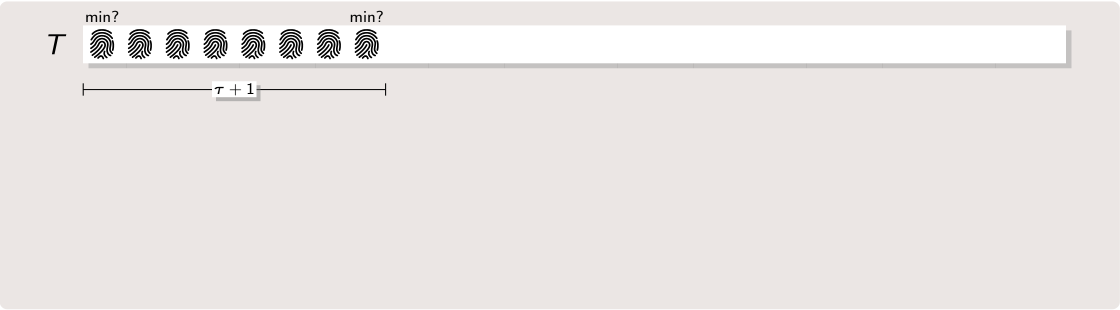
String Synchronizing Sets (SSS) [Kempa and Kociumaka, STOC'19]

Simplified τ-Synchronizing Set

Given: Text $T[1, n]$ and $0 < \tau \leq n/2$

Wanted: τ-synchronizing set S of T

$$S = \{i \in [1, n{-}2\tau{+}1] : \mathit{min}\{🫆(j, j{+}\tau{-}1) : j \in [i, i{+}\tau]\} \in \{🫆(i, i{+}\tau{-}1), 🫆(i{+}\tau, i{+}2\tau{-}1)\}\}$$



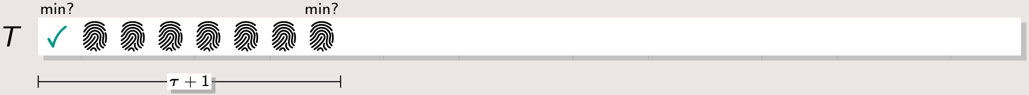
String Synchronizing Sets (SSS) [Kempa and Kociumaka, STOC'19]

Simplified τ-Synchronizing Set

Given: Text $T[1, n]$ and $0 < \tau \leq n/2$

Wanted: τ-synchronizing set S of T

$$S = \{i \in [1, n{-}2\tau{+}1] :\ min\{🫆(j, j{+}\tau{-}1) : j \in [i, i{+}\tau]\} \in \{🫆(i, i{+}\tau{-}1), 🫆(i{+}\tau, i{+}2\tau{-}1)\}\}$$



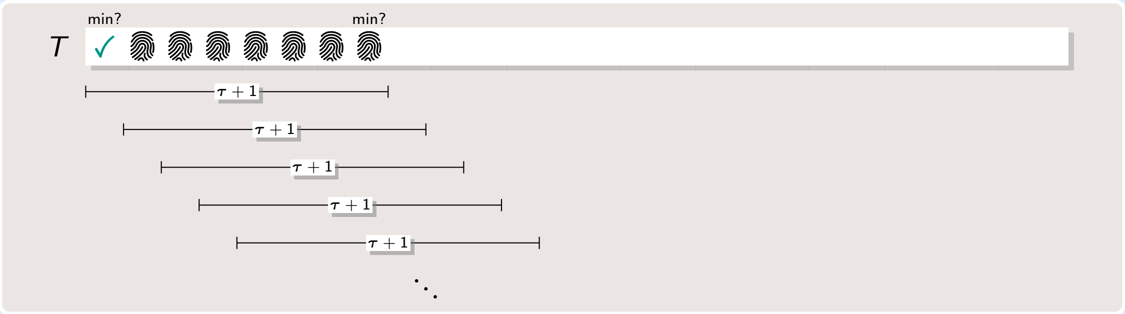
String Synchronizing Sets (SSS) [Kempa and Kociumaka, STOC'19]

Simplified τ-Synchronizing Set

Given: Text $T[1,n]$ and $0 < \tau \leq n/2$

Wanted: τ-synchronizing set S of T

$$S = \{i \in [1, n{-}2\tau{+}1] : \mathit{min}\{\text{🫆}(j, j{+}\tau{-}1) : j \in [i, i{+}\tau]\} \in \{\text{🫆}(i, i{+}\tau{-}1), \text{🫆}(i{+}\tau, i{+}2\tau{-}1)\}\}$$



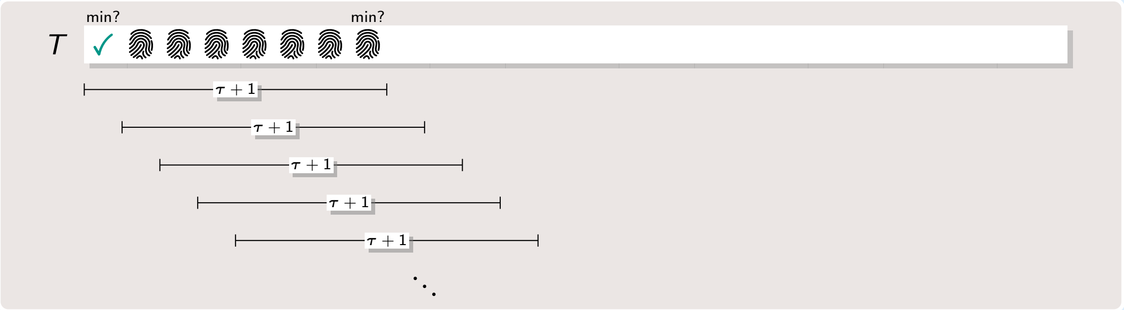
String Synchronizing Sets (SSS) [Kempa and Kociumaka, STOC'19]

Simplified τ-Synchronizing Set

Given: Text $T[1, n]$ and $0 < \tau \leq n/2$

Wanted: τ-synchronizing set S of T

$$S = \{i \in [1, n{-}2\tau{+}1] : \mathit{min}\{\text{🫆}(j, j{+}\tau{-}1) : j \in [i, i{+}\tau]\} \in \{\text{🫆}(i, i{+}\tau{-}1), \text{🫆}(i{+}\tau, i{+}2\tau{-}1)\}\}$$



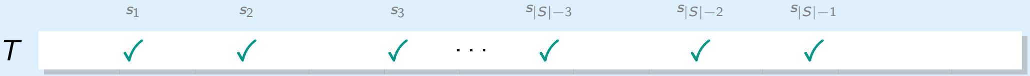
- $|S| = \Theta(n/\tau)$ in practice (on most data sets)
- more complex definition required to obtain this size

SSS for LCEs

Consistency & (Simplified) Density Property of S

- for all $i, j \in [1, n - 2\tau + 1]$ we have $T[i, i + 2\tau - 1] = T[j, j + 2\tau - 1] \Rightarrow i \in S \Leftrightarrow j \in S$
- for any τ consecutive positions there is at least one position in S

Text T' for Positions in S

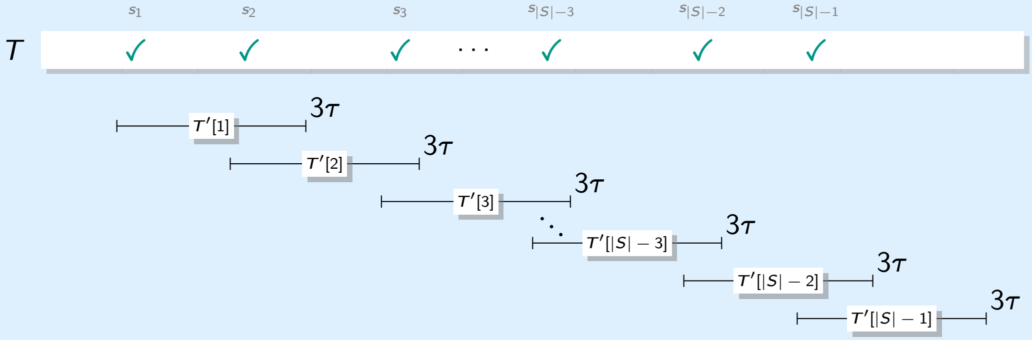


SSS for LCEs

Consistency & (Simplified) Density Property of S

- for all $i, j \in [1, n-2\tau+1]$ we have $T[i, i+2\tau-1] = T[j, j+2\tau-1] \Rightarrow i \in S \Leftrightarrow j \in S$
- for any τ consecutive positions there is at least one position in S

Text T' for Positions in S

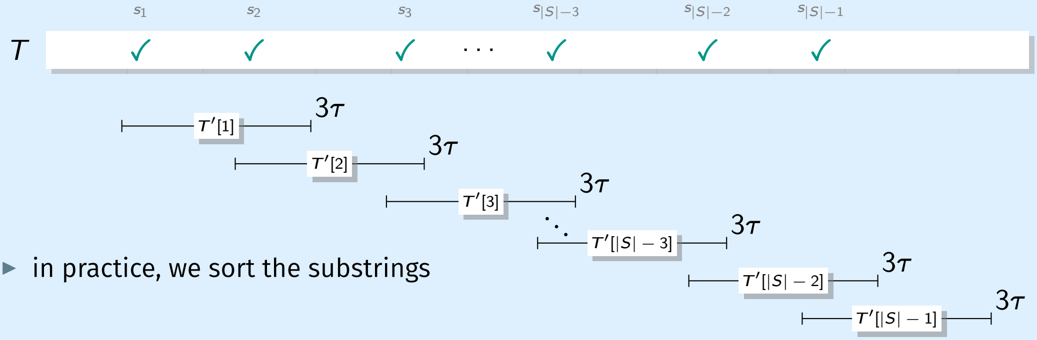


SSS for LCEs

Consistency & (Simplified) Density Property of S

- for all $i, j \in [1, n - 2\tau + 1]$ we have $T[i, i + 2\tau - 1] = T[j, j + 2\tau - 1] \Rightarrow i \in S \Leftrightarrow j \in S$
- for any τ consecutive positions there is at least one position in S

Text T' for Positions in S



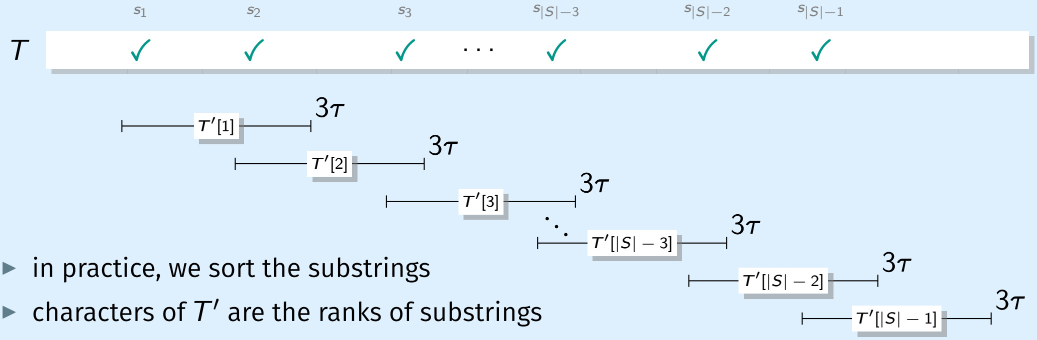
- in practice, we sort the substrings

SSS for LCEs

Consistency & (Simplified) Density Property of S

- for all $i, j \in [1, n - 2\tau + 1]$ we have $T[i, i + 2\tau - 1] = T[j, j + 2\tau - 1] \Rightarrow i \in S \Leftrightarrow j \in S$
- for any τ consecutive positions there is at least one position in S

Text T' for Positions in S



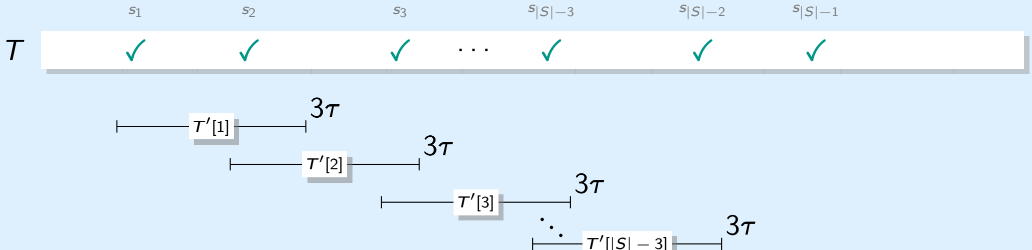
- in practice, we sort the substrings
- characters of T' are the ranks of substrings

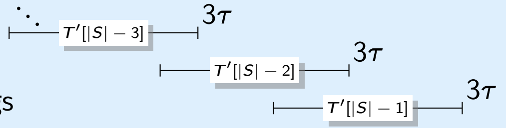
SSS for LCEs

Consistency & (Simplified) Density Property of S

- for all $i, j \in [1, n - 2\tau + 1]$ we have $T[i, i + 2\tau - 1] = T[j, j + 2\tau - 1] \Rightarrow i \in S \Leftrightarrow j \in S$
- for any τ consecutive positions there is at least one position in S

Text T' for Positions in S





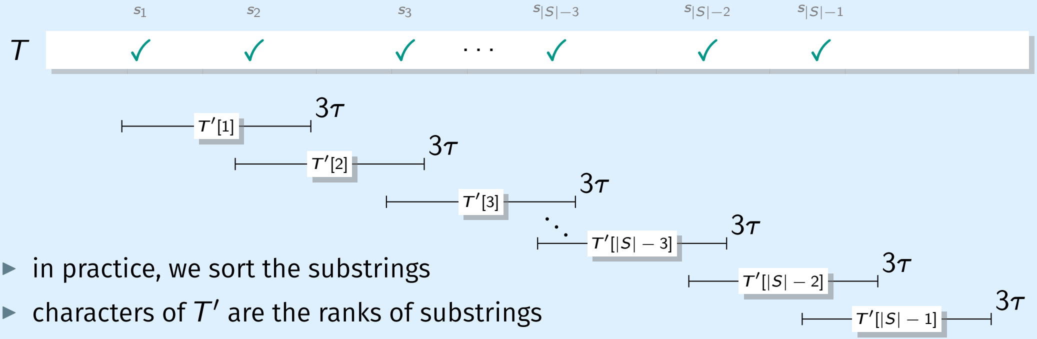
- in practice, we sort the substrings
- characters of T' are the ranks of substrings
- build black box LCE data structure for T' w.r.t. length in T

SSS for LCEs

Consistency & (Simplified) Density Property of S

- for all $i, j \in [1, n-2\tau+1]$ we have $T[i, i+2\tau-1] = T[j, j+2\tau-1] \Rightarrow i \in S \Leftrightarrow j \in S$
- for any τ consecutive positions there is at least one position in S

Text T' for Positions in S

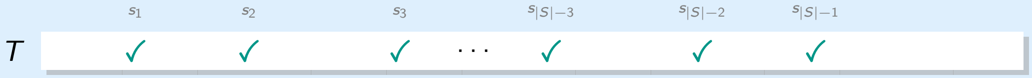


- in practice, we sort the substrings
- characters of T' are the ranks of substrings
- build black box LCE data structure for T' w.r.t. length in T
- ranks of $T[s_i, s_i + 3\tau]$ correspond to lexicographical order of $T[s_i, n]$

Answering LCE Queries using SSS and T'

General Idea for $\mathrm{lce}_T(i, j)$

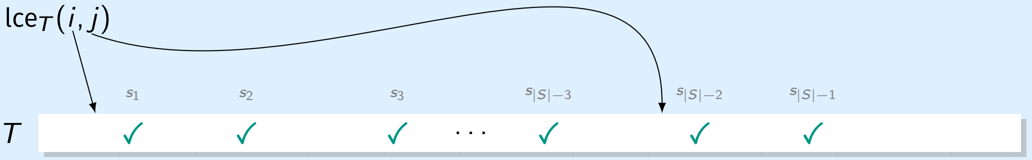
- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'



Answering LCE Queries using SSS and T'

General Idea for $\mathrm{lce}_T(i, j)$

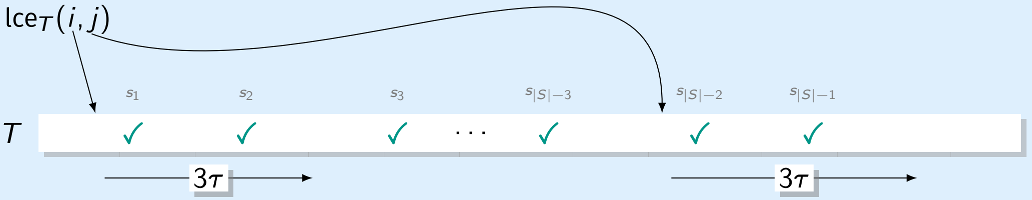
- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'



Answering LCE Queries using SSS and T'

General Idea for $\mathsf{lce}_T(i, j)$

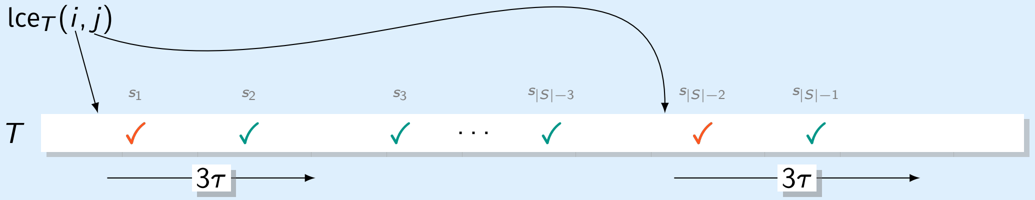
- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'



Answering LCE Queries using SSS and T'

General Idea for $\mathrm{lce}_T(i, j)$

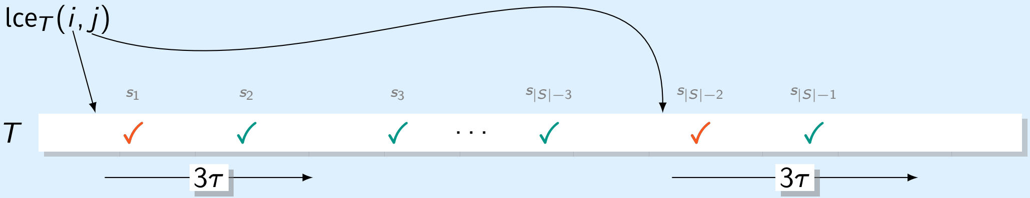
- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'



Answering LCE Queries using SSS and T'

General Idea for $\mathrm{lce}_T(i, j)$

- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'

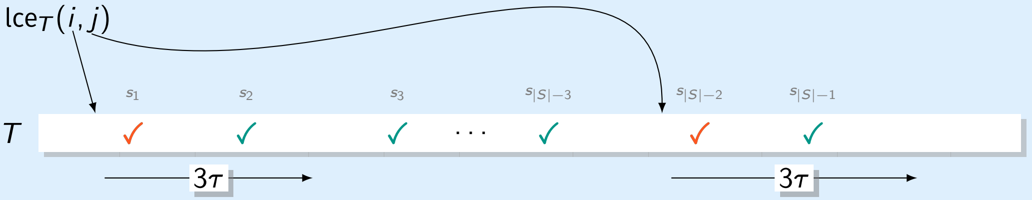


- in this example: $\mathrm{lce}_T(i,j) = s_1 - i + \mathrm{lce}_{T'}(1, |S| - 2)$

Answering LCE Queries using SSS and T'

General Idea for $\mathrm{lce}_T(i, j)$

- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'



- in this example: $\mathrm{lce}_T(i, j) = s_1 - i + \mathrm{lce}_{T'}(1, |S| - 2)$
- in practice: we have a very fast static successor data structure

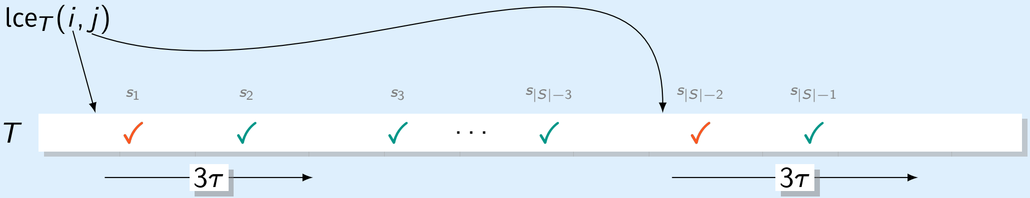
Answering LCE Queries using SSS and T'

General Idea for $\mathrm{lce}_T(i,j)$

- compare naively for 3τ characters
- if equal find successors of i and j in S
- compute LCE of successors in T'

Prefer Long LCEs for $\mathrm{lce}_T(i,j)$

- find successors i' and j' of i and j in S
- compare 2τ characters if $i' - i \neq j' - j$ and $i' - i$ characters otherwise
- compute LCE of successor in T'



- in this example: $\mathrm{lce}_T(i,j) = s_1 - i + \mathrm{lce}_{T'}(1, |S| - 2)$
- in practice: we have a very fast static successor data structure

Experimental Setup

Algorithms

- our algorithms and data structures
  - naive and ultra_naive
  - our-rk
  - sss_τ and $\text{sss}_\tau^{\text{pl}}$
- compared with
  - prezza-rk [Prezza, SODA'18]
  - sada and sct3 [part of SDSL]

Hardware

- two Intel Xeon E5-2640v4 with 2.4 GHz
- 64 GB RAM

Experimental Setup

Algorithms

- our algorithms and data structures
  - naive and ultra_naive
  - our-rk
  - sss_τ and sss_τ^{pl}
- compared with
  - prezza-rk [Prezza, SODA'18]
  - sada and sct3 [part of SDSL]

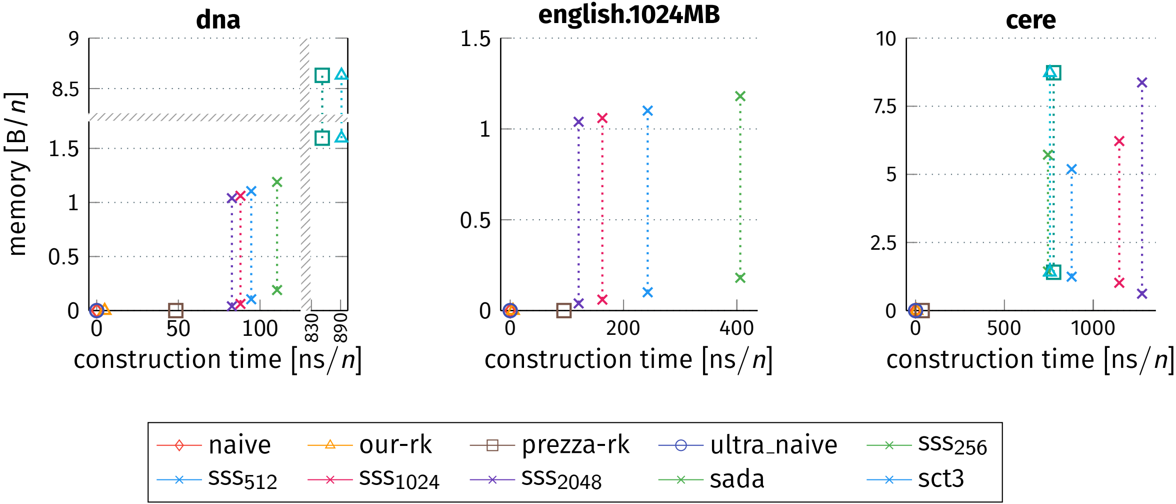
Hardware

- two Intel Xeon E5-2640v4 with 2.4 GHz
- 64 GB RAM

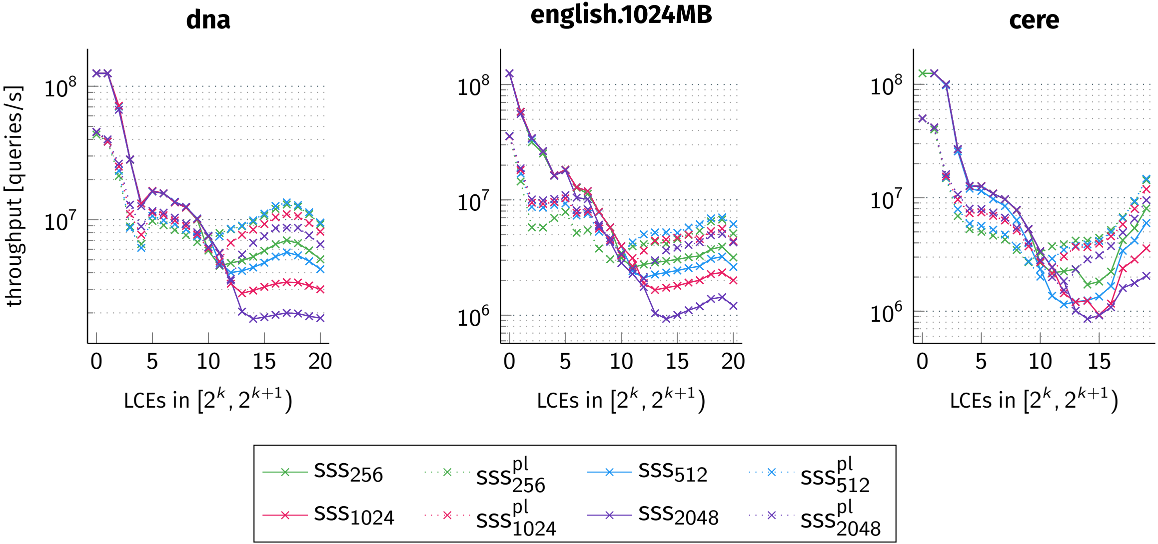
Texts

- Pizza & Chili corpus
- regular and repetitive
- now
  - dna ($\sigma = 16$)
  - english.1024MB ($\sigma = 239$)
  - cere ($\sigma = 6$)
- 9 more in the paper

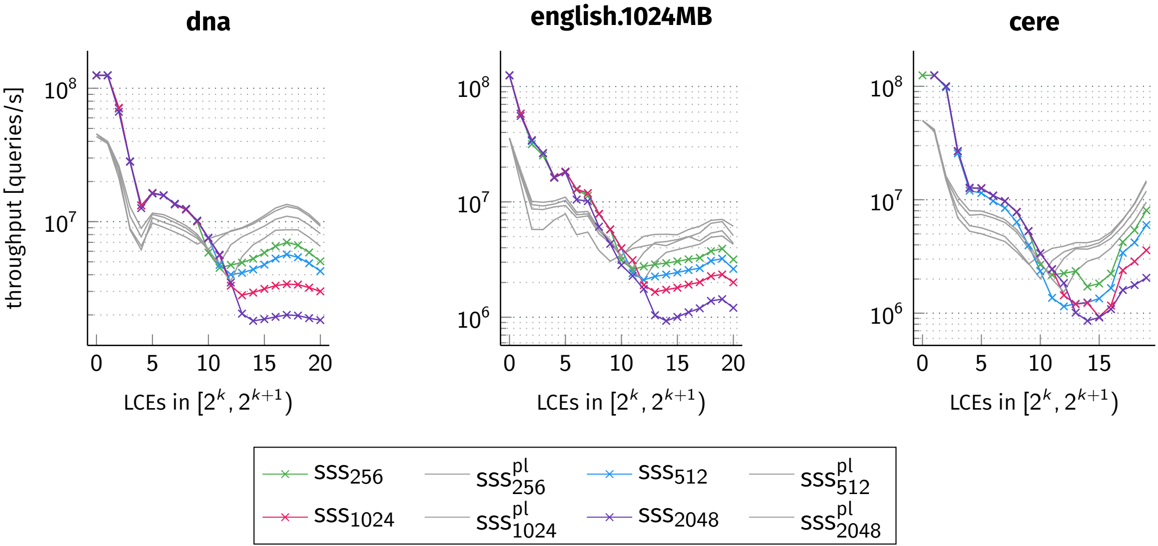
Evaluation: Construction Time and Memory Consumption



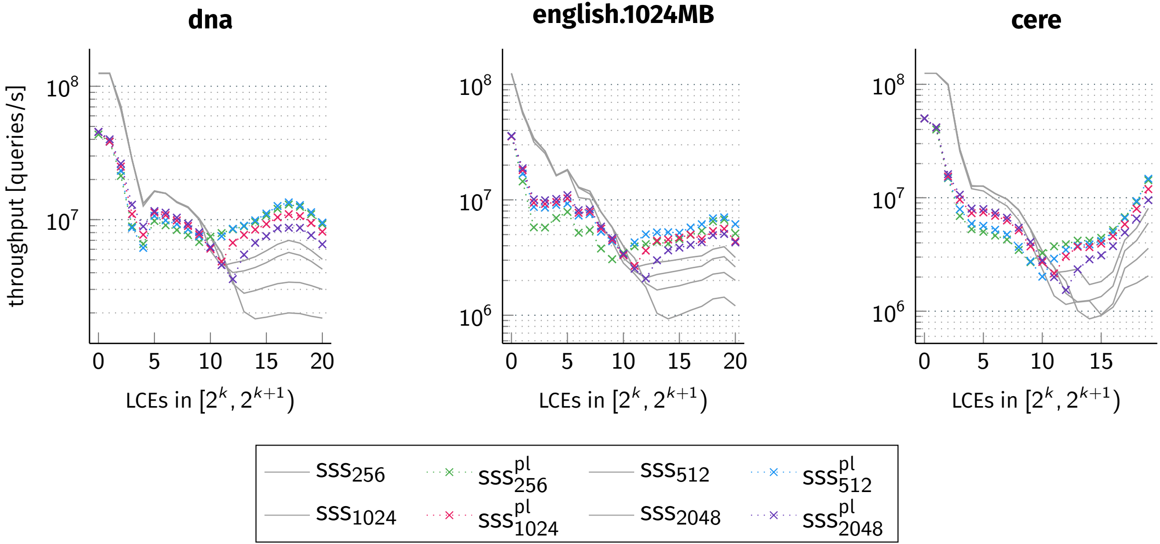
Evaluation: Choosing τ for SSS



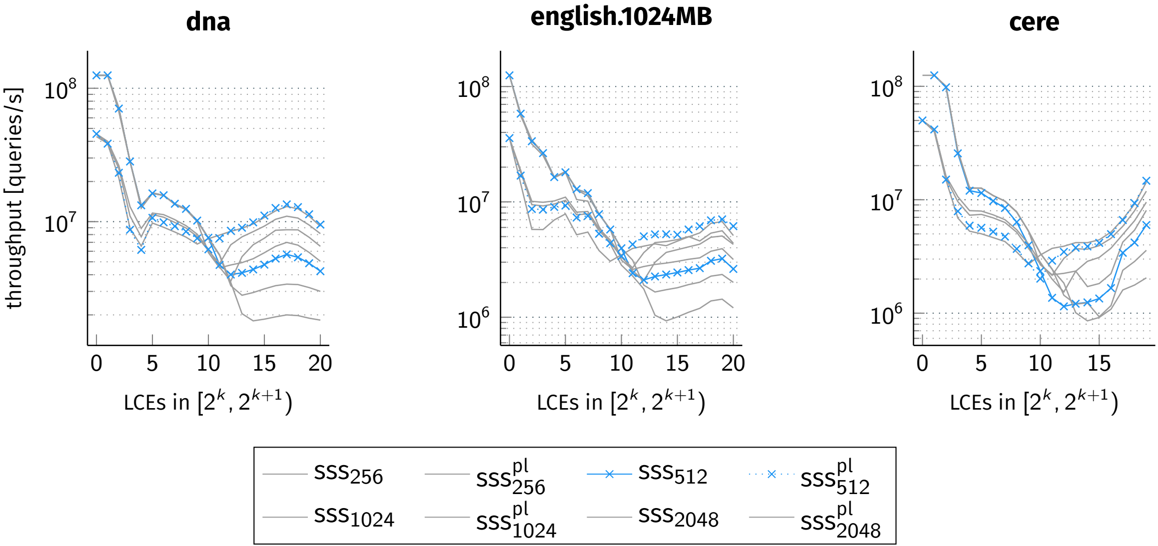
Evaluation: Choosing τ for SSS



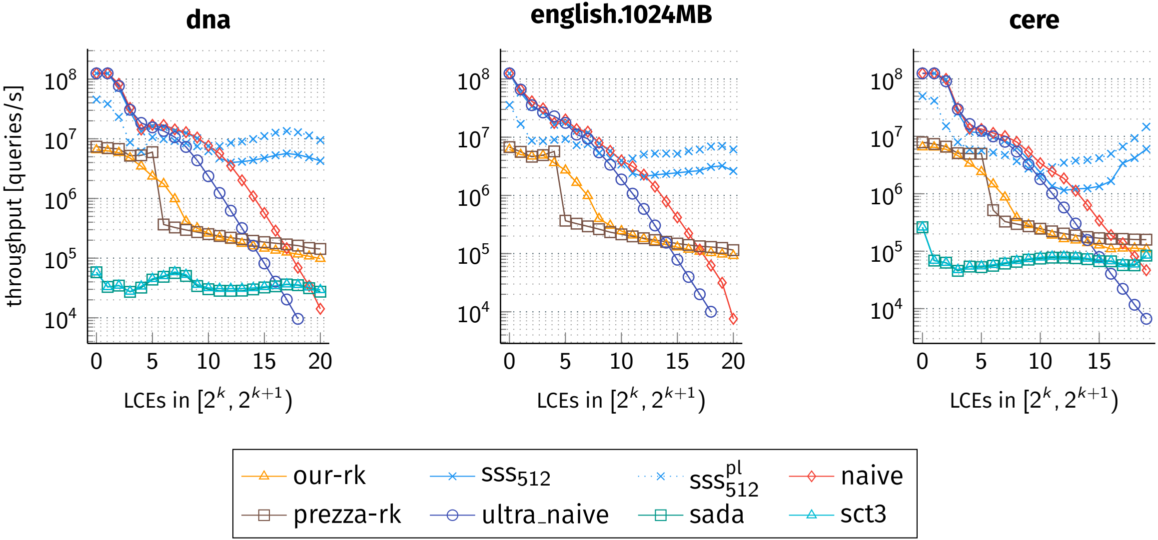
Evaluation: Choosing τ for SSS



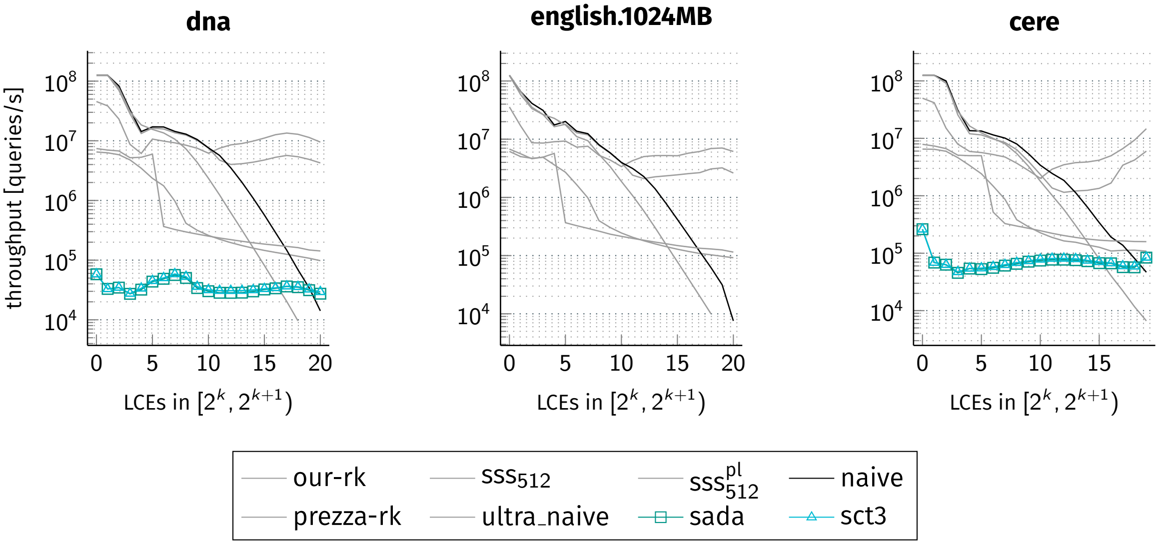
Evaluation: Choosing τ for SSS



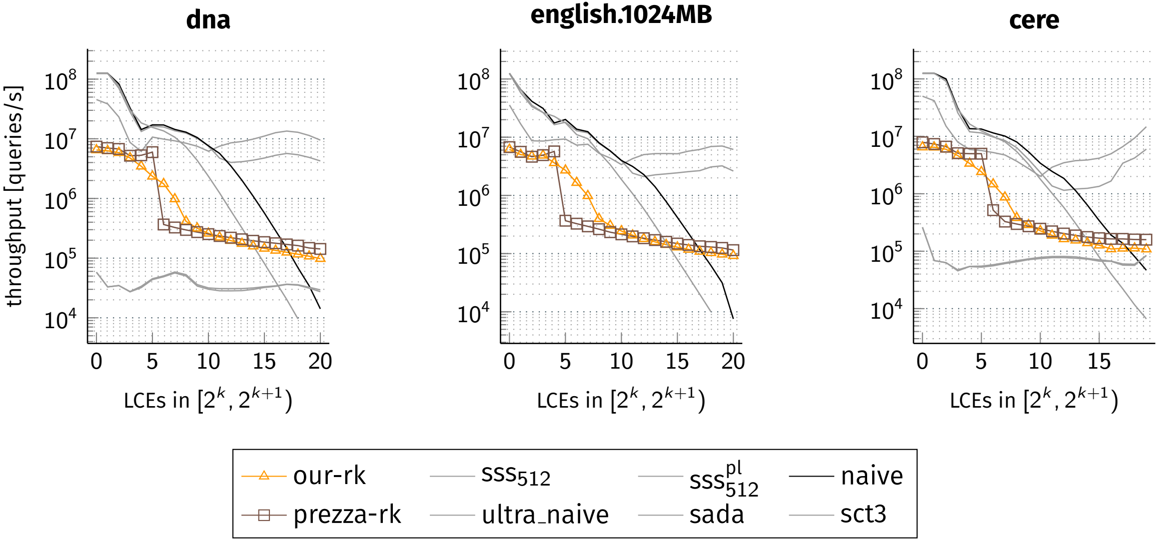
Evaluation: LCE Queries



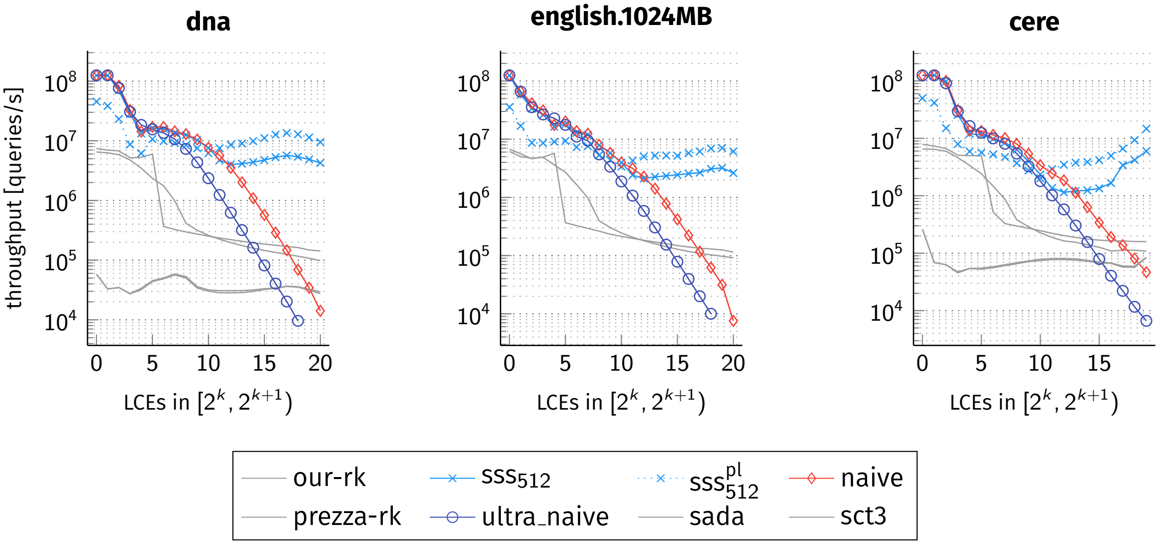
Evaluation: LCE Queries



Evaluation: LCE Queries



Evaluation: LCE Queries



Conclusion

- if slight memory overhead fits into RAM, SSS is the best LCE data structure
- else, in-place fingerprint data structures are best for longer queries
- otherwise, naive scan is the best for all other cases

Conclusion

- if slight memory overhead fits into RAM, SSS is the best LCE data structure
- else, in-place fingerprint data structures are best for longer queries
- otherwise, naive scan is the best for all other cases

- code available at `https://github.com/herlez/lce-test`

Conclusion

- if slight memory overhead fits into RAM, SSS is the best LCE data structure
- else, in-place fingerprint data structures are best for longer queries
- otherwise, naive scan is the best for all other cases

- code available at `https://github.com/herlez/lce-test`

Thank You