Fast Searching in Packed Strings

Philip Bille

String Matching

- Problem: Given strings P and Q of lengths m and n, resp., report all occurrences of P in Q.

$$Q = \mathtt{aababcabbababcababcaaabbab}$$

$$P = \mathtt{ababca}$$

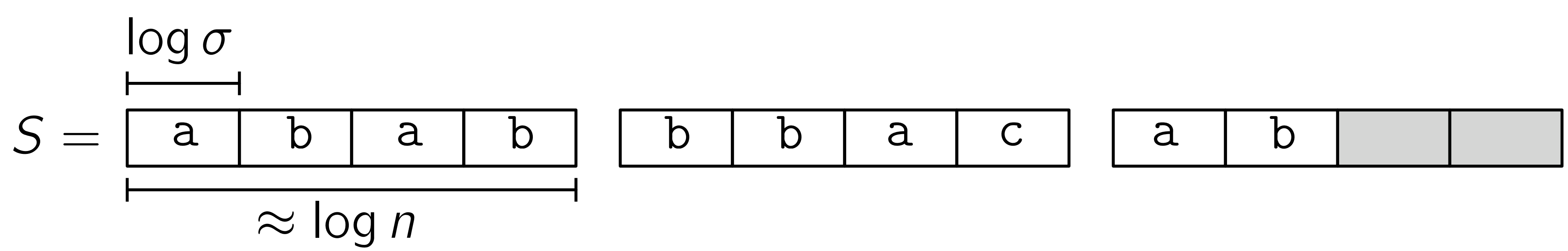
- KMP-algorithm [KMP1977] uses O(n) time (assume w.l.o.g. m ≤ n).

- Optimal if strings are stored with one char per memory word.

Packed Strings

- Real strings are *packed*:

$S = \texttt{ababbbacab}$



- With word-length log n a memory word holds ≈ log n / log σ characters.

- S uses $O(|S| \log \sigma / \log n) = O(|S| / \log_\sigma n)$ words.

Packed String Matching

- Problem: String matching with P and Q in packed representation.
- Lower bound: $\Omega\left(\frac{n+m}{\log_\sigma n} + \mathrm{occ}\right)$
- What is the best upper bound?
- Can we do better than O(n)?

A Simple Algorithm: Use a lots of space

$P =$ `ababca`

$Q =$

a	b	a	b	b	b	a	c	a	b		
			4				0		2		

A Simple Algorithm: Use a lots of space

$P = \texttt{ababca}$

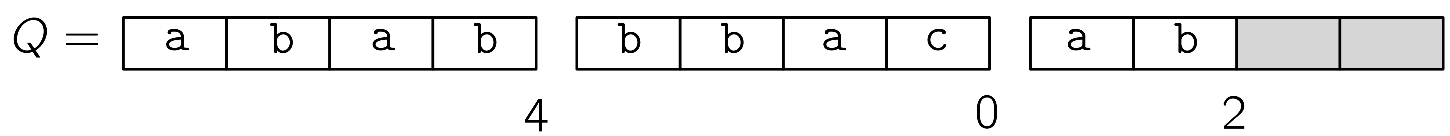
$Q =$

a	b	a	b	b	b	a	c	a	b		
			4				0		2		

- Idea: Traverse Q from left-to-right reading r ≈ #number of characters per word at a time.

A Simple Algorithm: Use a lots of space

$P =$ `ababca`

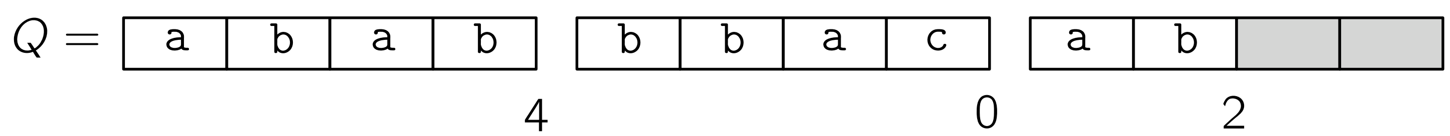


- Idea: Traverse Q from left-to-right reading r ≈ #number of characters per word at a time.

- At each step compute the *longest prefix* of P matching the current suffix of Q. (slightly more information needed to also report occurrences).

A Simple Algorithm: Use a lots of space

$P = \texttt{ababca}$



- Idea: Traverse Q from left-to-right reading r ≈ #number of characters per word at a time.
- At each step compute the *longest prefix* of P matching the current suffix of Q. (slightly more information needed to also report occurrences).
- To do step in constant time store for each prefix of P and each combination of r characters a pointer to the next prefix. (called a "super-alphabet technique" [Fre02]).

A Simple Algorithm: Use a lots of space

$P =$ `ababca`

$Q =$

a	b	a	b	b	b	a	c	a	b		
			4				0		2		

- Idea: Traverse Q from left-to-right reading r ≈ #number of characters per word at a time.

- At each step compute the *longest prefix* of P matching the current suffix of Q. (slightly more information needed to also report occurrences).

- To do step in constant time store for each prefix of P and each combination of r characters a pointer to the next prefix. (called a “super-alphabet technique” [Fre02]).

Space: $O(m\sigma^r)$

Time: $O(n/r + m\sigma^r + \mathrm{occ})$

A Simple Algorithm: Use a lots of space

$P = \texttt{ababca}$

$Q =$ | a | b | a | b | | b | b | a | c | | a | b | | |

$Q =$	a	b	a	b		b	b	a	c		a	b		
				4					0			2		

- Idea: Traverse Q from left-to-right reading r ≈ #number of characters per word at a time.
- At each step compute the *longest prefix* of P matching the current suffix of Q. (slightly more information needed to also report occurrences).
- To do step in constant time store for each prefix of P and each combination of r characters a pointer to the next prefix. (called a “super-alphabet technique” [Fre02]).

		$r = \epsilon \log_\sigma n$
Space:	$O(m\sigma^r)$	$O(mn^\epsilon)$
Time:	$O(n/r + m\sigma^r + \text{occ})$	$O(n/\log_\sigma n + mn^\epsilon + \text{occ})$

Complexities

Time	Space	
$O\left(\frac{n}{r} + m\sigma^r + \mathrm{occ}\right)$	$O(m\sigma^r)$	Simple
$O\left(\frac{n}{\log_\sigma n} + mn^\varepsilon + \mathrm{occ}\right)$	$O(mn^\varepsilon)$	
$O\left(\frac{n}{r} + m + \sigma^r + \mathrm{occ}\right)$	$O(m + \sigma^r)$	This paper
$O\left(\frac{n}{\log_\sigma n} + m + \mathrm{occ}\right)$	$O(m + n^\varepsilon)$	

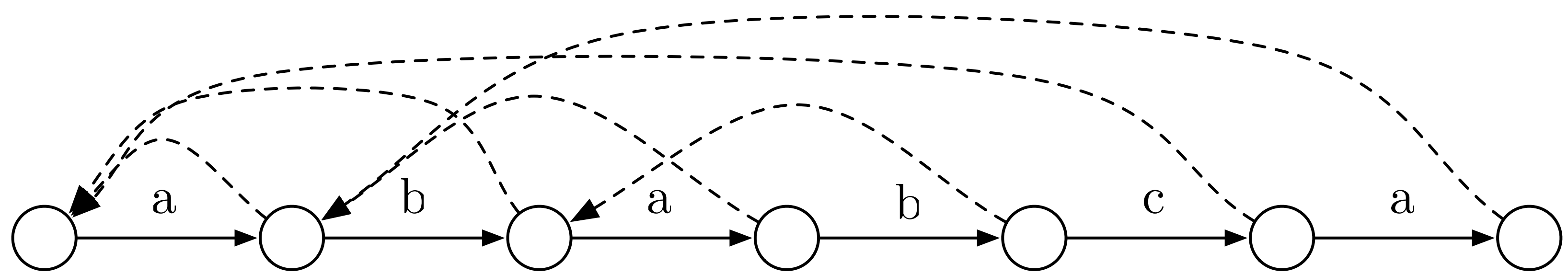
Algorithm Overview

- Based on the Knuth-Morris-Pratt automaton.

- The “Four-Russian Technique” (divide and tabulate) with new twists.

The Knuth-Morris-Pratt Automaton

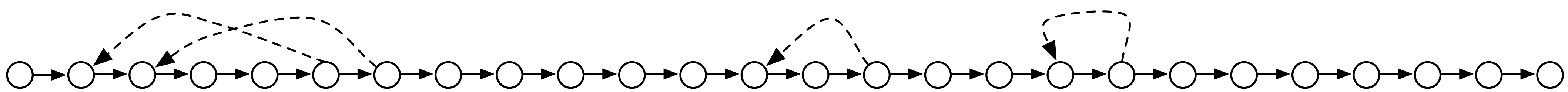
$P = \texttt{ababca}$



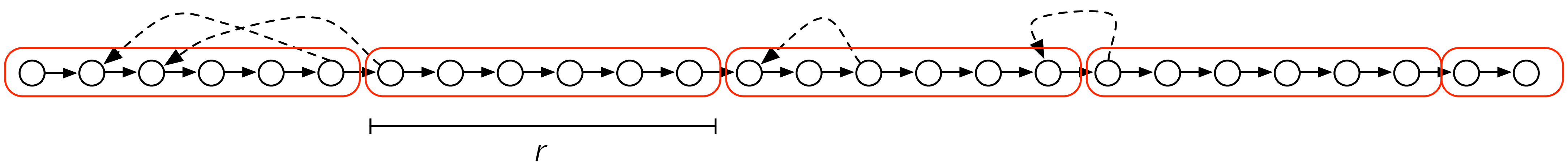
KMP(P)

A First Attempt: The Four-Russian Technique

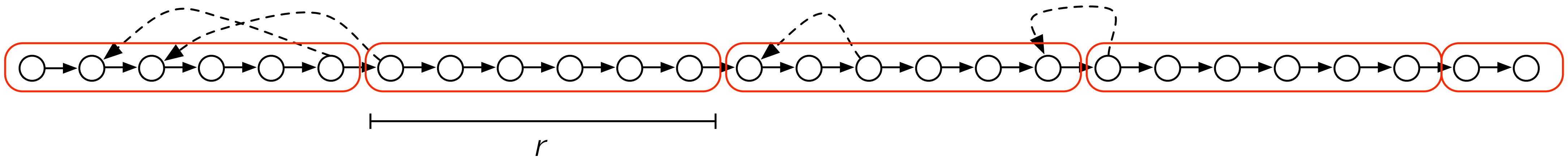
A First Attempt: The Four-Russian Technique



A First Attempt: The Four-Russian Technique

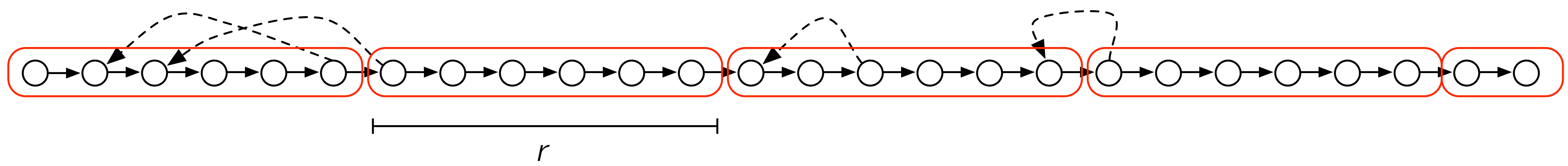


A First Attempt: The Four-Russian Technique



- Tabulate information for each subautomata to allow up to r *internal* transitions in constant time.

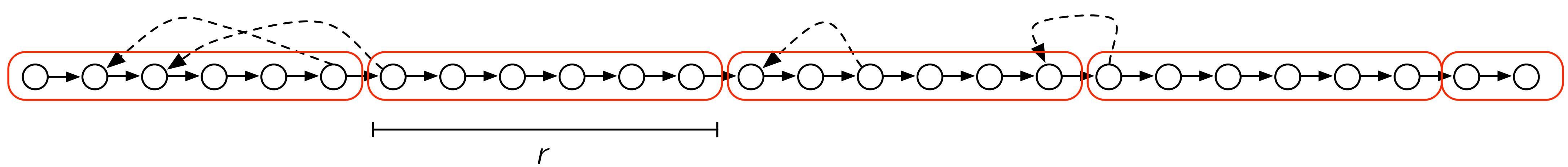
A First Attempt: The Four-Russian Technique



- Tabulate information for each subautomata to allow up to r *internal* transitions in constant time.

- Simulate by doing *external* transitions explicitly and internal transitions using the tabulated information.

A First Attempt: The Four-Russian Technique

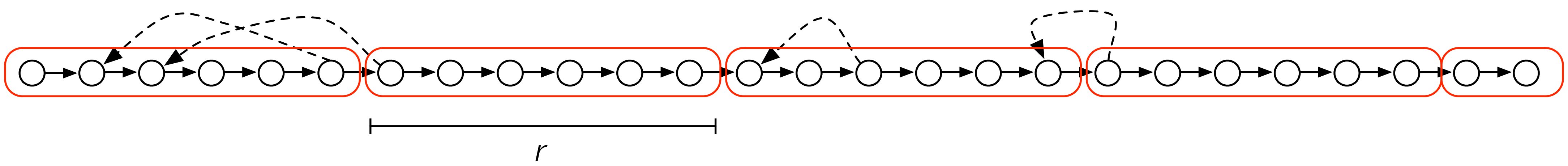


- Tabulate information for each subautomata to allow up to r *internal* transitions in constant time.

- Simulate by doing *external* transitions explicitly and internal transitions using the tabulated information.

- Issue 1: Too many external transitions.

A First Attempt: The Four-Russian Technique



- Tabulate information for each subautomata to allow up to r *internal* transitions in constant time.

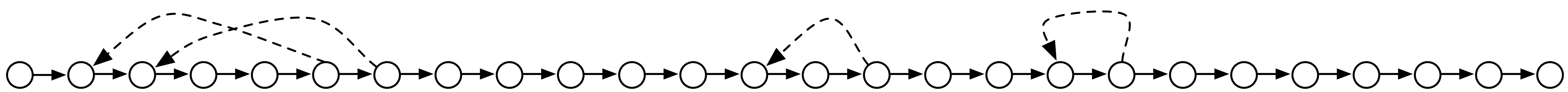
- Simulate by doing *external* transitions explicitly and internal transitions using the tabulated information.

- Issue 1: Too many external transitions.

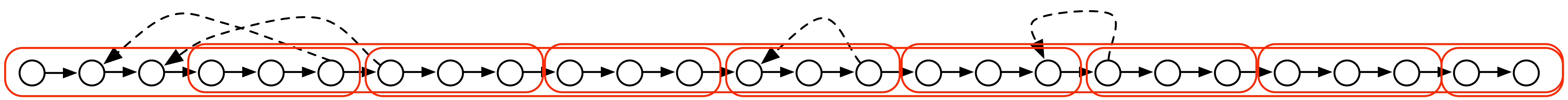
- Issue 2: Representing subautomata compactly.

Fixing 1: Too Many External Transitions

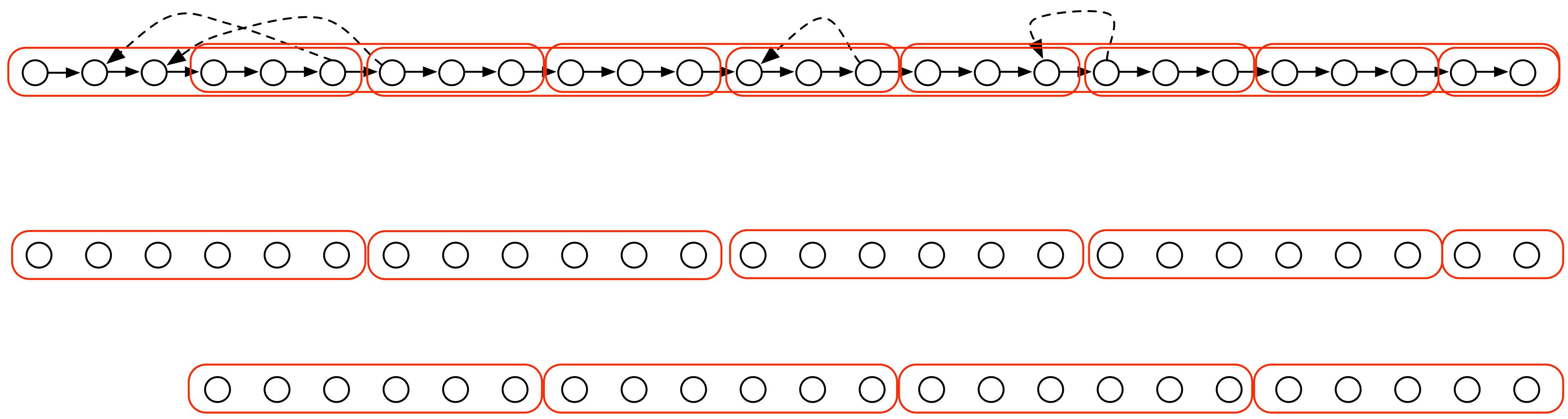
Fixing 1: Too Many External Transitions



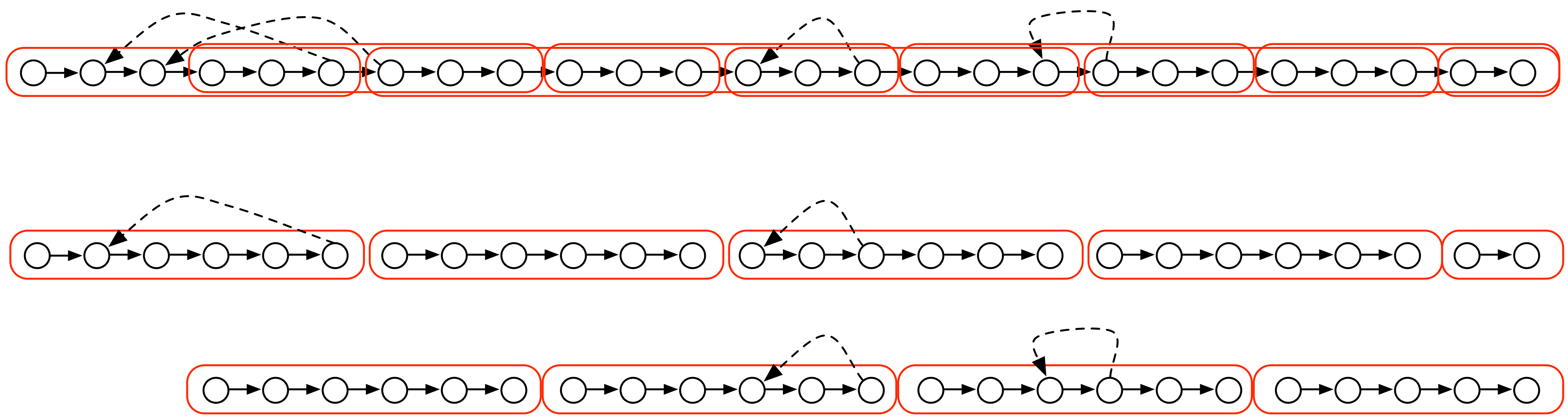
Fixing 1: Too Many External Transitions



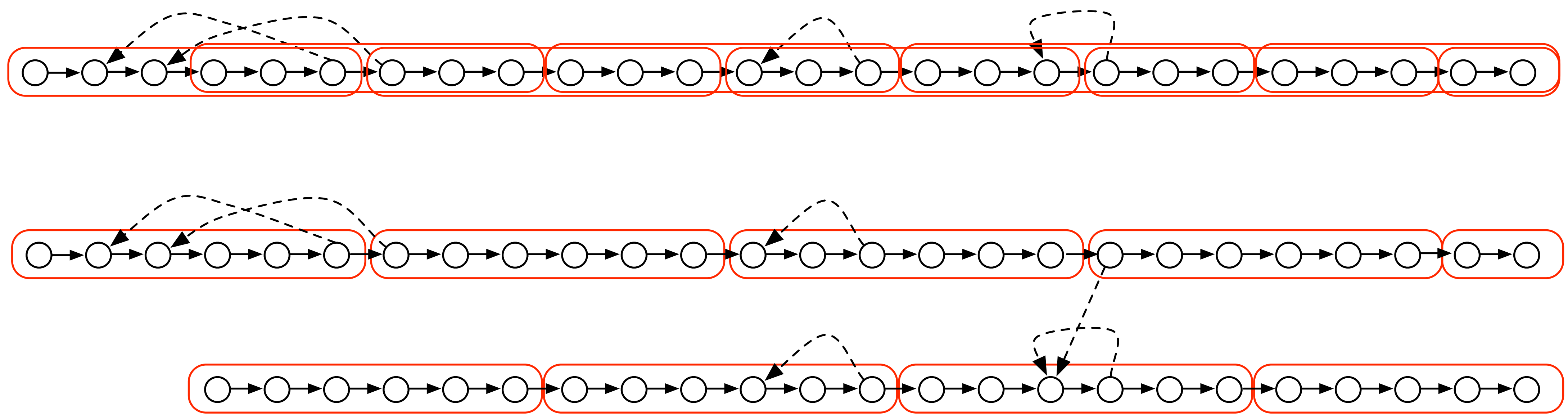
Fixing 1: Too Many External Transitions



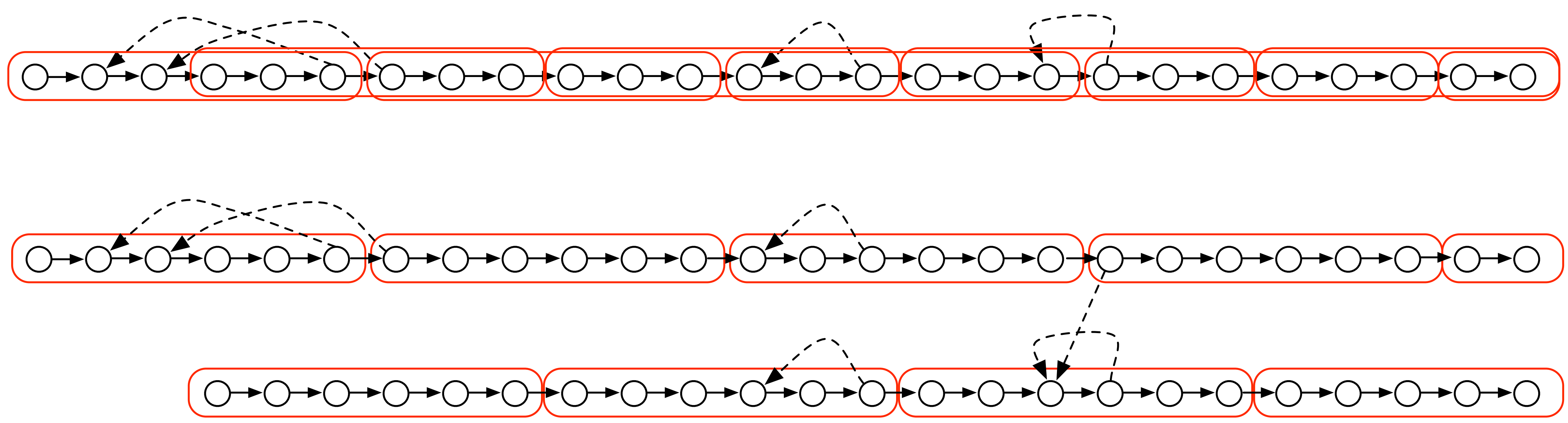
Fixing 1: Too Many External Transitions



Fixing 1: Too Many External Transitions

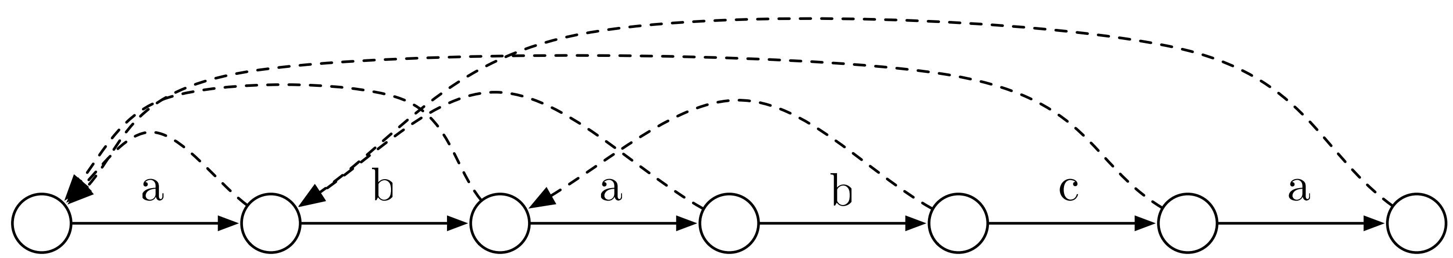


Fixing 1: Too Many External Transitions



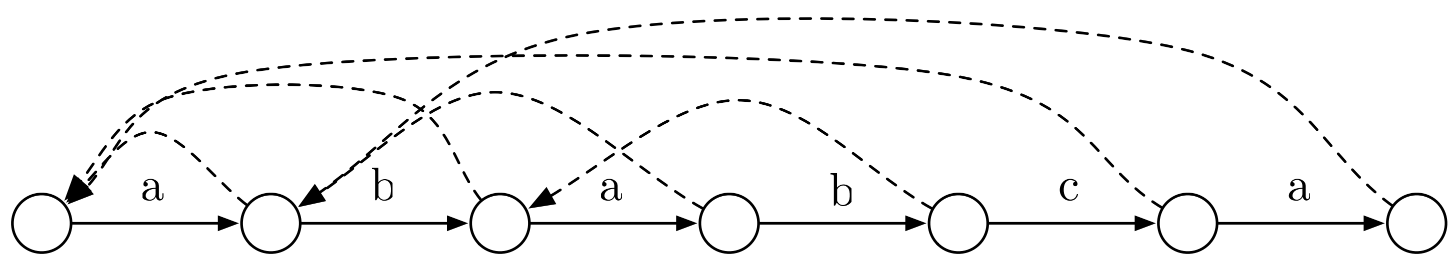
At most O(n/r) external transitions in simulation of Q

Fixing 2: Representing Subautomata Compactly

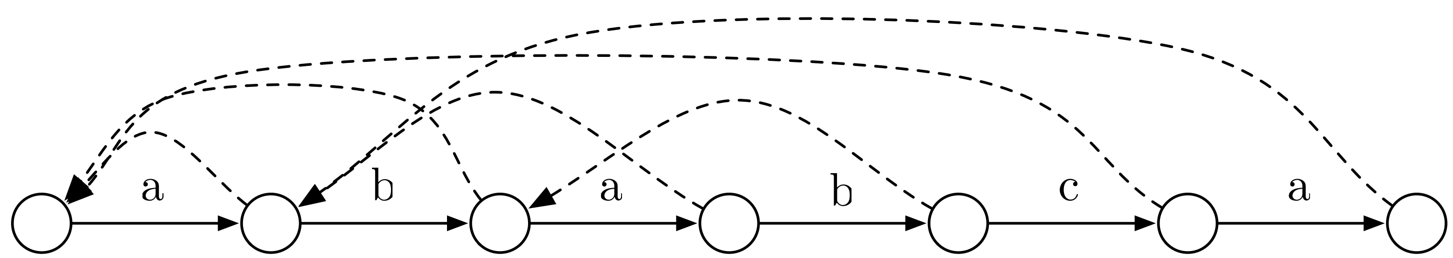


- We want to encode an arbitrary subautomaton of KMP(P) in O(r log σ) bits.
- Non-failure transitions encoded by the sequence of labels in O(r log σ) bits.
- How about the failure transitions in S?

Fixing 2: Representing Subautomata Compactly

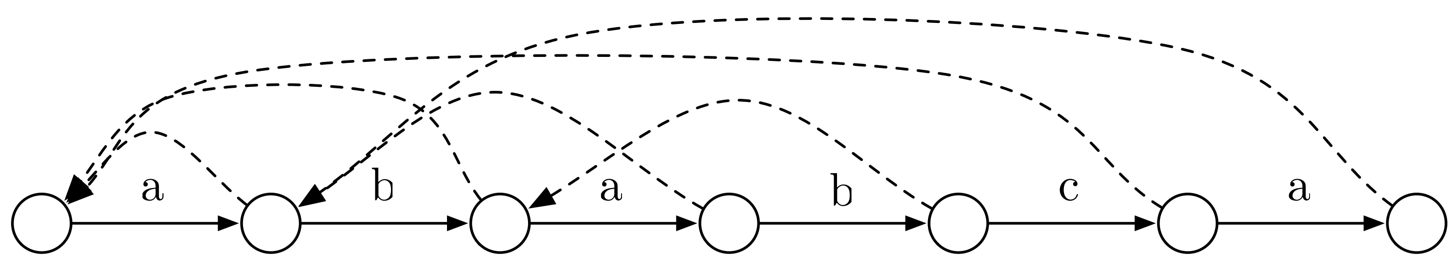


Fixing 2: Representing Subautomata Compactly



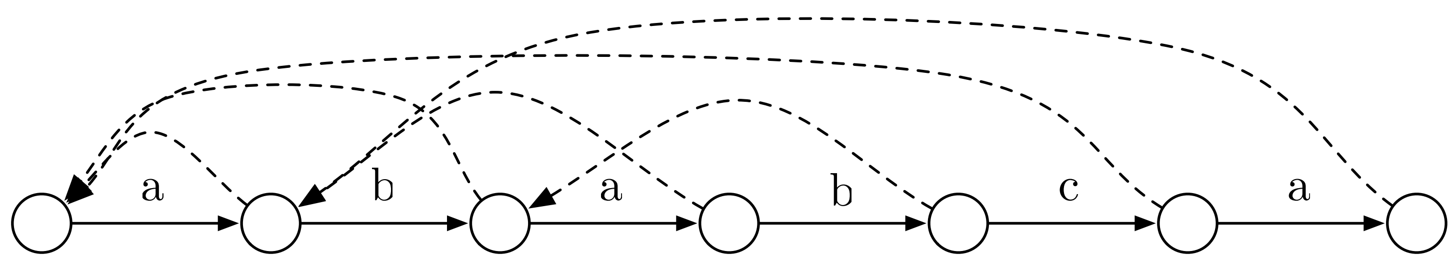
- Storing r explicit pointers uses Ω(r log r) bits.

Fixing 2: Representing Subautomata Compactly



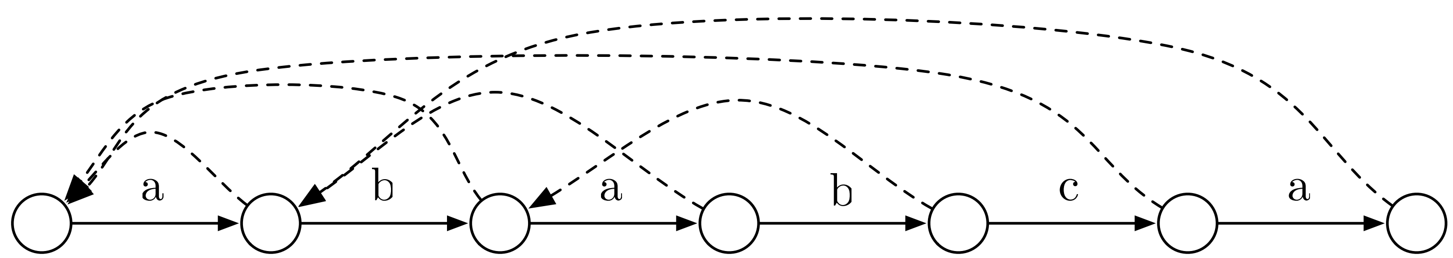
- Storing r explicit pointers uses Ω(r log r) bits.
- Instead we exploit a basic property of KMP-automata: In any subautomaton failure transition endpoints increase by at most 1 between consecutive states.

Fixing 2: Representing Subautomata Compactly



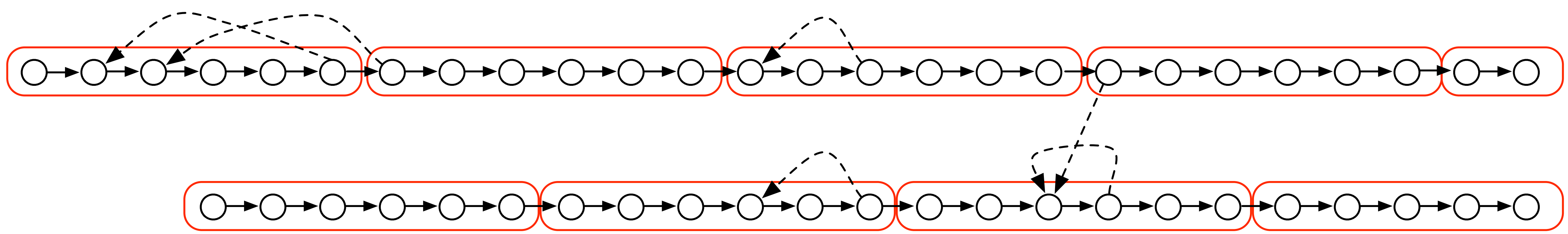
- Storing r explicit pointers uses Ω(r log r) bits.
- Instead we exploit a basic property of KMP-automata: In any subautomaton failure transition endpoints increase by at most 1 between consecutive states.
- => Total increase at most r => Total decrease at most O(r).

Fixing 2: Representing Subautomata Compactly



- Storing r explicit pointers uses Ω(r log r) bits.
- Instead we exploit a basic property of KMP-automata: In any subautomaton failure transition endpoints increase by at most 1 between consecutive states.
- => Total increase at most r => Total decrease at most O(r).
- => We can difference encode all failure transitions with O(r) bits.

Putting the Pieces together



- Construct segment automaton and tabulate transitions for subautomata using the compact encoding.

- Simulate the segment automaton. Each external transitions is done explicitly. Internal transitions are done using the tabulation.

- Complexity:

$$r = \epsilon \log_\sigma n$$

Space:	$O(m + \sigma^r)$	$O(m + n^\epsilon)$
Time:	$O(n/r + m + \sigma^r + \text{occ})$	$O(n/\log_\sigma n + m + \text{occ})$

Directions

Directions

- Packed string matching:

Directions

- Packed string matching:
  - Practical?

Directions

- Packed string matching:
  - Practical?
  - Long word lengths?

Directions

- Packed string matching:
    - Practical?
    - Long word lengths?
    - Multi-string matching?

Directions

- Packed string matching:
  - Practical?
  - Long word lengths?
  - Multi-string matching?
- Packed problems appear everywhere.

Directions

- Packed string matching:
    - Practical?
    - Long word lengths?
    - Multi-string matching?
- Packed problems appear everywhere.
    - Longer word lengths => more packing.

Directions

- Packed string matching:
  - Practical?
  - Long word lengths?
  - Multi-string matching?
- Packed problems appear everywhere.
  - Longer word lengths => more packing.
  - Most packed problems are not well-solved.