Reduction ratio of the IS-algorithm: worst and random cases

Vincent Jugé

Université Gustave Eiffel (LIGM)

28/06/2022
Contents

1. Induced-sorting algorithm for computing the suffix array of a word
2. Worst-case reduction ratio
3. Average reduction ratio: Letters generated by a nice Markov chain
4. Number of recursive calls: Letters generated by a finite Markov chain
Suffix arrays\cite{1}

**Suffix array**: permutation that orders lexicographically suffixes of a word

```
B   A   L   A   L   A   I   K   A
```
Suffix arrays

**Suffix array**: permutation that orders lexicographically suffixes of a word

```
B  A  L  A  L  A  I  K  A
A  K  A
I  K  A
A  I  K  A
L  A  I  K  A
A  L  A  I  K  A
L  A  L  A  I  K  A
A  L  A  L  A  I  K  A
B  A  L  A  L  A  I  K  A
```
Suffix arrays[1]

**Suffix array:** permutation that orders lexicographically suffixes of a word

```
B   A   L   A   L   A   I   K   A
A
A   I   K   A
A   L   A   I   K   A
A   L   A   L   A   I   K   A
B   A   L   A   L   A   I   K   A
I   K   A
K   A
L   A   I   K   A
L   A   L   A   I   K   A
```
**Suffix arrays**[1]

**Suffix array:** permutation that orders lexicographically suffixes of a word

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>A</th>
<th>L</th>
<th>A</th>
<th>L</th>
<th>A</th>
<th>I</th>
<th>K</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rank</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

A

A I K A

A L A I K A

A L A L A I K A

B A L A L A I K A

I K A

K A

L A I K A

L A L A I K A
Suffix arrays\cite{1}

**Suffix array**: permutation that orders lexicographically suffixes of a word

\begin{align*}
\text{B A L A L A I K A} \\
4 & 3 & 8 & 2 & 7 & 1 & 5 & 6 & 0
\end{align*}

\begin{align*}
A & \\
A & I & K & A \\
A & L & A & I & K & A \\
A & L & A & L & A & I & K & A \\
B & A & L & A & L & A & I & K & A \\
I & K & A \\
K & A \\
L & A & I & K & A \\
L & A & L & A & I & K & A
\end{align*}

Useful for longest common factors, Burrows-Wheeler transform\cite{2}, \ldots
Induced-sorting (SA-IS) algorithm\cite{[3]}

**Goal:** Computing the suffix array of a word $w$
with letters in $\{0, 1, \ldots, |w|\}$ or in a finite alphabet

\begin{center}
  B A L A L A I K A
\end{center}

\begin{itemize}
  \item If no symbol of $w$ occurs twice, just sort them
\end{itemize}
Induced-sorting (SA-IS) algorithm\[^3\]

**Goal:** Computing the suffix array of a word \( w \) with letters in \( \{0, 1, \ldots, |w|\} \) or in a finite alphabet

\[ B\ A\ L\ A\ L\ A\ I\ K\ A\ $ \]

1. If no symbol of \( w \) occurs twice, just sort them
2. Append a $ symbol (minimal symbol) to \( w \)
Induced-sorting (SA-IS) algorithm$^3$

**Goal:** Computing the suffix array of a word $w$

with letters in \{0, 1, \ldots, |w|\} or in a finite alphabet

\begin{align*}
B & A L A L A I K A \$
\end{align*}

\begin{align*}
A & L A \\
A & L A \\
A & I K A \$
\end{align*}

0. If no symbol of $w$ occurs twice, just sort them
1. Append a $\$ \text{ symbol (minimal symbol) to } w$
2. Subdivide $w \cdot \$ \text{ into unimodal (LMS) factors}$
**Induced-sorting (SA-IS) algorithm[^3]**

**Goal:** Computing the suffix array of a word $w$ with letters in \( \{0, 1, \ldots, |w|\} \) or in a finite alphabet $B A L A L A I K A$.

1. If no symbol of $w$ occurs twice, just sort them.
2. Append a $\$\$ symbol (minimal symbol) to $w$.
3. Subdivide $w \cdot \$\$ into **unimodal** (LMS) factors.
4. Sort these and relabel them in increasing order (gives you $w'$).

[^3]: Vincent Jugé

Reduction ratio of the IS-algorithm
Induced-sorting (SA-IS) algorithm

**Goal:** Computing the suffix array of a word $w$

with letters in $\{0, 1, \ldots, |w|\}$ or in a finite alphabet

```
BALALAIKA
2 1 0
```

1

```
A L A
1
```

1

```
A L A
```

0

```
A I K A $
```

0. If no symbol of $w$ occurs twice, just sort them
1. Append a $\$\$ symbol (minimal symbol) to $w$
2. Subdivide $w \cdot \$\$ into **unimodal** (LMS) factors
3. Sort these and relabel them in increasing order
4. Compute the suffix array of $w'$

(gives you $w'$)
Induced-sorting (SA-IS) algorithm\cite{3}

**Goal:** Computing the suffix array of a word $w$

with letters in $\{0, 1, \ldots, |w|\}$ or in a finite alphabet

```
B A L A L A I K A $
4 3 8 2 7 1 5 6 0
```

```
1 A L A
1 A L A
0 A I K A $
```

1. If no symbol of $w$ occurs twice, just sort them
2. Append a $\$\$ symbol (minimal symbol) to $w$
3. Subdivide $w \cdot \$\$ into unimodal (LMS) factors
4. Sort these and relabel them in increasing order
5. Compute the suffix array of $w'$
6. Finish computing the suffix array of $w$

(gives you $w'$)
Induced sorting (SA-IS) algorithm

Theorem

IS algorithm computes the suffix array of \(w\) in time linear in \(|w|\).

Proof elements:

- Steps 1 and 2 can be performed in time \(O(|w|)\)
- Unimodal words of total length \(\ell\) and their suffixes can be sorted in time \(O(\ell)\): Steps 3 and 5 can be performed in time \(O(|w|)\)
- Step 4 is performed on a word of length \(|w'| \leq (|w| - 1)/2\)

Suffix array computed in time \(O(|w| + |w|/2 + |w|/4 + \cdots) = O(|w|)\)

Further questions:
- Can we repeatedly have \(|w'| = (|w| - 1)/2\)?
- What is the reduction ratio \(|w'|/|w|\) in practice?
- How many recursive calls shall we expect?
Induced sorting (SA-IS) algorithm

Theorem

IS algorithm computes the suffix array of $w$ in time linear in $|w|$.

Proof elements:

- Steps 1 and 2 can be performed in time $O(|w|)$
- Unimodal words of total length $\ell$ and their suffixes can be sorted in time $O(\ell)$: Steps 3 and 5 can be performed in time $O(|w|)$
- Step 4 is performed on a word of length $|w'| \leq (|w| - 1)/2$

Suffix array computed in time $O(|w| + |w|/2 + |w|/4 + \cdots) = O(|w|)$

Further questions:

- Can we repeatedly have $|w'| = (|w| - 1)/2$?
- What is the reduction ratio $|w'|/|w|$ in practice?
- How many recursive calls shall we expect?
Reduction ratio: worst case

Worst-case scenario\(^{[5]}\)

We can keep having \(|w'| = (|w| - 1)/2\) for \(\log_2(|w|)\) recursive steps

Example:

\[
\begin{array}{ccccccccccccc}
2 & 1 & 2 & 0 & 4 & 1 & 4 & 0 & 2 & 1 & 4 & 0 & 4 & 1 & 3 & \$
\end{array}
\]
Reduction ratio: worst case

Worst-case scenario\cite{5}

We can keep having $|w'| = (|w| - 1)/2$ for $\log_2(|w|)$ recursive steps.

Example:

<table>
<thead>
<tr>
<th>2</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>4</th>
<th>1</th>
<th>4</th>
<th>0</th>
<th>2</th>
<th>1</th>
<th>4</th>
<th>0</th>
<th>4</th>
<th>1</th>
<th>3</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reduction ratio: worst case

Worst-case scenario\textsuperscript{[5]}:

We can keep having $|w'| = (|w| − 1)/2$ for $\log_2(|w|)$ recursive steps.

Example:

\[
\begin{array}{cccccccccccccc}
2 & 1 & 2 & 0 & 4 & 1 & 4 & 0 & 2 & 1 & 4 & 0 & 4 & 1 & 3 \\
2 & 1 & 4 & 0 & 4 & 1 & 3 \\
1 & 2 & 0 & 1 & 4 & 0 & 1 & 4 & 0 & 1 & 3 \\
0 & 4 & 1 & 0 & 2 & 1 & 0 & 4 & 1
\end{array}
\]
Reduction ratio: worst case

Worst-case scenario\textsuperscript{[5]}

We can keep having $|w'| = (|w| - 1)/2$ for $\log_2(|w|)$ recursive steps

Example:

\[
\begin{array}{cccccccccccc}
2 & 1 & 2 & 0 & 4 & 1 & 4 & 0 & 2 & 1 & 4 & 0 & 4 & 1 & 3 & $
\end{array}
\]

\[
\begin{array}{cccccccccccc}
2 & 1 & 4 & 0 & 4 & 1 & 3 \\
1 & 2 & 0 & 1 & 4 & 0 & 1 & 3 & $ \\
0 & 4 & 1 & 0 & 2 & 1 & 0 & 4 & 1
\end{array}
\]

Word obtained by applying the increasing morphism

\[
\begin{array}{cccccccc}
0 & \mapsto & 02 & & 1 & \mapsto & 04 & & 2 & \mapsto & 12 & & 3 & \mapsto & 13 & & 4 & \mapsto & 14
\end{array}
\]

$k$ times on the letter 3, and then deleting the first letter
Infinitely many independent letters

Sample the letters of \( w : \mathbb{Z} \mapsto \{0, 1\} \) independently uniformly at random:

Example:

\[
\ldots \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ \ldots
\]
Infinitely many independent letters

Sample the letters of $w : \mathbb{Z} \mapsto \{0, 1\}$ independently uniformly at random:

- Ends of unimodal factors are the subwords 10: $|w'| \sim |w|/4$

Example:

\[
\begin{array}{cccccccccccccccc}
\ldots & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & \ldots \\
\ldots & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & \ldots \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & \ldots \\
\end{array}
\]
Infinitely many independent letters

Sample the letters of \( w : \mathbb{Z} \mapsto \{0, 1\} \) independently uniformly at random:

- Ends of unimodal factors are the subwords 10: \( |w'| \sim |w|/4 \)
- Unimodal factors of \( w \) are independent words, with \( \mathbb{P}[0^a 1^b 0] = 2^{-a-b} \)

Example:

\[
\begin{array}{cccccccccccccccc}
\ldots & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & \ldots \\
\ldots & 1 & 0 & & & & & & & & & & & & & & \ldots \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \ldots \\
\end{array}
\]
Infinitely many independent letters

Sample the letters of $w : \mathbb{Z} \mapsto \{0, 1\}$ independently uniformly at random:

- Ends of unimodal factors are the subwords 10: $|w'| \sim |w|/4$
- Unimodal factors of $w$ are independent words, with $\mathbb{P}[0^a1^b0] = 2^{-a-b}$
  - **Infinite** alphabet! (countable, not isomorphic to $\mathbb{Z}$ or $\mathbb{N}$)

Example:

```
... 1 0 1 1 0 1 0 1 1 0 0 1 1 0 1 1 1 ...
... 1 0 0 1 0 0 0 1 1 0 ...
  0 1 1 0 0 1 1 0 0 1 1 1 ...
  ... 0^11^20 0^21^20 0^11^10 0^21^10 0^21^20 0^21^20 0^31^10 0^11^40 ...
```
Infinitely many independent letters

Sample the letters of \( w : \mathbb{Z} \mapsto \{0, 1\} \) independently uniformly at random:

- Ends of unimodal factors are the subwords 10: \( |w'| \sim |w|/4 \)
- Unimodal factors of \( w \) are independent words, with \( \mathbb{P}[0^a1^b0] = 2^{a-b} \)
  - **Infinite** alphabet! (countable, not isomorphic to \( \mathbb{Z} \) or \( \mathbb{N} \))
- Unimodal factors of \( w' \) are **not** independent, and \( |w''| \sim 0.353 \ldots |w'| \)
- Things keep getting more complicated after further recursive calls

Example:

\[
\begin{array}{cccccccccccccccc}
\ldots & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & \ldots \\
\ldots & 1 & 0 & & 0 & 1 & 0 & & 0 & 0 & 1 & 1 & 0 \\
& 0 & 1 & 1 & 0 & & 0 & 1 & 1 & 0 & & 0 & 1 & 1 & \ldots \\
\ldots & 0^11^20 & 0^21^20 & 0^11^10 & 0^21^10 & 0^21^20 & 0^21^20 & 0^31^10 & 0^11^40 & \ldots \\
\ldots & 0^11^20 & 0^21^20 & 0^21^10 & 0^21^20 & 0^21^20 & 0^21^20 & 0^31^10 & & & \ldots \\
& 0^21^20 & 0^11^10 & 0^21^10 & & & & 0^31^10 & 0^11^40 & \ldots \\
\end{array}
\]
Main challenges

Questions:

- What about relabelling (in step 2)?
- What about letters that are not independent?
- What when leftmost and rightmost letters are eventually reached?
Main challenges

Questions:
- What about relabelling (in step 2)?
- What about letters that are not independent?
- What when leftmost and rightmost letters are eventually reached?

Answers:
- Relabelling is useful for actual computations, not here
- Assume that letters are given (from left to right or right to left) by a nice Markov chain
- Truncate your Markov chain when you have enough symbols!
Main challenges

Questions:

- What about relabelling (in step 2)?
- What about letters that are not independent?
- What when leftmost and rightmost letters are eventually reached?

Answers:

- Relabelling is useful for actual computations, not here
- Assume that letters are given (from left to right or right to left) by a nice Markov chain
- Truncate your Markov chain when you have enough symbols!
Main challenges

Questions:

- What about relabelling (in step 2)?
- What about letters that are not independent?
- What when leftmost and rightmost letters are eventually reached?

Answers:

- Relabelling is useful for actual computations, not here
- Assume that letters are given (from left to right or right to left) by a nice Markov chain
- Truncate your Markov chain when you have enough symbols!

\[
\begin{array}{cccccccccccccc}
1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & $
\end{array}
\]

\[
\begin{array}{cccccccccc}
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & $
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}
\]
Nice Markov chains

Constraints to satisfy:

- i.i.d. Markov chains are nice
- Unimodular factors of a nice Markov chain are nice
- Ends of unimodular factors must have some density of occurrence

Example:
Nice Markov chains

Constraints to satisfy:
- i.i.d. Markov chains are nice
- Unimodular factors of a nice Markov chain are nice
- Ends of unimodular factors must have some density of occurrence

EPRI Markov chain[5]

A countable Markov chain \( M \) is almost surely eventually positive, recurrent and irreducible if it has a terminal component \( X \) that is almost surely reached, and on which \( M \) is positive recurrent.
Nice Markov chains

Constraints to satisfy:
- i.i.d. Markov chains are nice
- Unimodular factors of a nice Markov chain are nice
- Ends of unimodular factors must have some density of occurrence

EPRI Markov chain[5]

A countable Markov chain $M$ is **almost surely eventually positive, recurrent and irreducible** if it has a terminal component $X$ that is almost surely reached, and on which $M$ is positive recurrent.

Example:

$$\mathbb{E}[1 \rightarrow 3] = 2$$
Nice Markov chains

Constraints to satisfy:
- i.i.d. Markov chains are nice
- Unimodular factors of a nice Markov chain are nice
- Ends of unimodular factors must have some density of occurrence

EPRI Markov chain\[^{[5]}\]

A countable Markov chain $M$ is **almost surely eventually positive, recurrent and irreducible** if it has a terminal component $\mathcal{X}$ that is almost surely reached, and on which $M$ is positive recurrent.

Counter-example:

$\mathbb{E}[2 \to 1] = +\infty$
Nice Markov chains

Constraints to satisfy:
- i.i.d. Markov chains are nice
- Unimodular factors of a nice Markov chain are nice
- Ends of unimodular factors must have some density of occurrence

EPRI Markov chain\([^5]\)

A countable Markov chain \(M\) is almost surely eventually positive, recurrent and irreducible if it has a terminal component \(\mathcal{X}\) that is almost surely reached, and on which \(M\) is positive recurrent.

Counter-example: \[\mathbb{E}[1 \rightarrow 0] = +\infty\]
Nice Markov chains

Constraints to satisfy:
- i.i.d. Markov chains are nice
- Unimodular factors of a nice Markov chain are nice
- Ends of unimodular factors must have some density of occurrence

EPRI Markov chain[5]

A countable Markov chain $M$ is almost surely eventually positive, recurrent and irreducible if it has a terminal component $\mathcal{X}$ that is almost surely reached, and on which $M$ is positive recurrent.

Example:
\[
\mathbb{E}[1 \rightarrow 0] = 3
\]
Letters generated by a nice Markov chain

Theorem[5]

Let \( w \) be a word whose letters are generated by an EPRI Markov chain, and let \( w^{(k)} \) be the word obtained after \( k \) recursive calls. The ratios

\[
\frac{|w^{(k)}|}{|w|}
\]

converge, in probability, towards a constant \( \gamma^{(k)} \).
Letters generated by a nice Markov chain

**Theorem**\(^5\)

Let \( w \) be a word whose letters are generated by an EPRI Markov chain, and let \( w^{(k)} \) be the word obtained after \( k \) recursive calls. The ratios

\[
\frac{|w^{(k)}|}{|w|}
\]

converge, in probability, towards a constant \( \gamma^{(k)} \).

**Bonus result**\(^4,5\)

If the letters of \( w \) are i.i.d, \( \gamma^{(1)} < 1/3 \).
Number of recursive calls
Step 0 (direct letter sorting if possible) is very useful!

Theorem [5]
Let $w$ be a word whose letters are generated by a finite Markov chain. There exists a constant $k$ such that, for all $\ell \geq 0$, the SA-IS algorithm has a probability $P \leq k/|w|^{2\ell}$ of performing more than $2\log_2(\log_2(|w|)) + \ell$ recursive calls.

Proof elements:
Each letter of $w$ ($i$) represents at least $2^i$ letters of $w$.
Letters of $w$ reach a terminal component $X$ in expected time $O(1)$.
If $X$ is a cycle, end up with a one-letter word in $O(1)$ recursive calls.
Otherwise, factors of $w$ of length $2^{\ell}(\log_2(|w|))^2$ are likely to be distinct.

Vincent Jugé

Reduction ratio of the IS-algorithm
Number of recursive calls

Step 0 (direct letter sorting if possible) is very useful!

**Theorem**

Let $w$ be a word whose letters are generated by a finite Markov chain. There exists a constant $k$ such that, for all $\ell \geq 0$, the SA-IS algorithm has a probability

$$\mathbb{P} \leq \frac{k}{|w|^{2\ell}}$$

of performing more than $2\log_2(\log_2(|w|)) + \ell$ recursive calls.
Number of recursive calls

Step 0 (direct letter sorting if possible) is very useful!

Theorem[5]
Let $w$ be a word whose letters are generated by a finite Markov chain. There exists a constant $k$ such that, for all $\ell \geq 0$, the SA-IS algorithm has a probability
\[
P \leq k/|w|^{2\ell}
\]
of performing more than $2\log_2(\log_2(|w|)) + \ell$ recursive calls.

Proof elements:
- Each letter of $w^{(i)}$ represents at least $2^i$ letters of $w$
- Letters of $w$ reach a terminal component $\mathcal{X}$ in expected time $O(1)$
- If $\mathcal{X}$ is a cycle, end up with a one-letter word in $O(1)$ recursive calls
- Otherwise, factors of $w$ of length $2^\ell(\log_2(|w|))^2$ are likely to be distinct

Vincent Jugé
Reduction ratio of the IS-algorithm
Some references


M. Burrows & D. Wheeler (1994)

[3] Two efficient algorithms for linear time suffix array construction

C. Nicaud (2015)

[5] Reduction ratio of the IS-algorithm: worst and random cases
V. Jugé (2022)
THANK YOU FOR LISTENING!

DO YOU HAVE EASY QUESTIONS?