Compression by Contracting Straight-Line Programs

Moses Ganardi

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

Algorithms on Compressed Data

Two goals in data compression:

- Store data in a compact form (lossless).
- Support efficient queries directly on the compressed representation.
  $\rightarrow$ Avoid decompression!

compression — algorithmics

Kolmogorov complexity | LZ77 | grammar-based compression | RLE | uncompressed

Algorithms on Compressed Data

Two goals in data compression:

- Store data in a compact form (lossless).
- Support efficient queries directly on the compressed representation.
  $\rightarrow$ Avoid decompression!

Algorithms on Compressed Data

Two goals in data compression:

- Store data in a compact form (lossless).
- Support efficient queries directly on the compressed representation.
  $\rightarrow$ Avoid decompression!

compression — algorithmics

Kolmogorov complexity | LZ77 | grammar-based compression | RLE | uncompressed

Grammar-based compression

Grammar-based compression

A **straight-line program (SLP)** is a context-free grammar $\mathcal{G}$ which produces exactly one string.

Every variable occurs exactly once on the left-hand side of a rule and the variables are topologically ordered.

grammar

$$
\begin{aligned}
S_4 &\to S_3\, S_2 \\
S_3 &\to S_2\, S_1 \\
S_2 &\to S_1\, S_0 \\
S_1 &\to \mathtt{1} \\
S_0 &\to \mathtt{0}
\end{aligned}
$$

derivation tree

The string length is denoted by $N \leqslant 2^{\mathcal{O}(|\mathcal{G}|)}$.

Chomsky normal form: rules of the form $A \to BC$ or $A \to \mathtt{a}$.

Grammar-based compression

A **straight-line program (SLP)** is a context-free grammar $\mathcal{G}$ which produces exactly one string.

Every variable occurs exactly once on the left-hand side of a rule
and the variables are topologically ordered.

grammar

$$\begin{aligned} S_4 &\to S_3\, S_2 \\ S_3 &\to S_2\, S_1 \\ S_2 &\to S_1\, S_0 \\ S_1 &\to \mathtt{1} \\ S_0 &\to \mathtt{0} \end{aligned}$$

derivation tree

The string length is denoted by $N \leqslant 2^{\mathcal{O}(|\mathcal{G}|)}$.
Chomsky normal form: rules of the form $A \to BC$ or $A \to \mathtt{a}$.

Grammar-based compression

A **straight-line program (SLP)** is a context-free grammar $\mathcal{G}$ which produces exactly one string.

Every variable occurs exactly once on the left-hand side of a rule and the variables are topologically ordered.

grammar

$$
\begin{aligned}
S_4 &\to S_3\, S_2 \\
S_3 &\to S_2\, S_1 \\
S_2 &\to S_1\, S_0 \\
S_1 &\to \mathtt{1} \\
S_0 &\to \mathtt{0}
\end{aligned}
$$

derivation tree

The string length is denoted by $N \leqslant 2^{\mathcal{O}(|\mathcal{G}|)}$.

Chomsky normal form: rules of the form $A \to BC$ or $A \to \mathtt{a}$.

Grammar-based compression

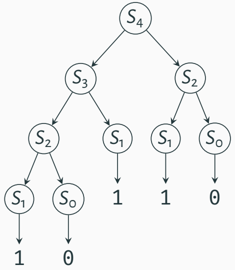
A **straight-line program (SLP)** is a context-free grammar $\mathcal{G}$ which produces exactly one string.

Every variable occurs exactly once on the left-hand side of a rule and the variables are topologically ordered.

grammar

$$
\begin{aligned}
S_4 &\to S_3\, S_2 \\
S_3 &\to S_2\, S_1 \\
S_2 &\to S_1\, S_0 \\
S_1 &\to \mathtt{1} \\
S_0 &\to \mathtt{0}
\end{aligned}
$$

derivation tree



The string length is denoted by $N \leqslant 2^{\mathcal{O}(|\mathcal{G}|)}$.

Chomsky normal form: rules of the form $A \to BC$ or $A \to \mathtt{a}$.

Grammar-based compression

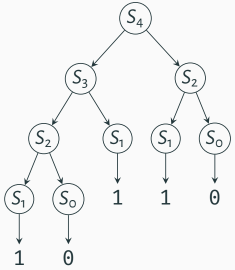
A **straight-line program (SLP)** is a context-free grammar $\mathcal{G}$ which produces exactly one string.

Every variable occurs exactly once on the left-hand side of a rule
and the variables are topologically ordered.

grammar

$$\begin{aligned} S_4 &\to S_3\, S_2 \\ S_3 &\to S_2\, S_1 \\ S_2 &\to S_1\, S_0 \\ S_1 &\to \mathtt{1} \\ S_0 &\to \mathtt{0} \end{aligned}$$

derivation tree



The string length is denoted by $N \leqslant 2^{\mathcal{O}(|\mathcal{G}|)}$.

Chomsky normal form: rules of the form $A \to BC$ or $A \to \mathtt{a}$.

Grammar-based compression

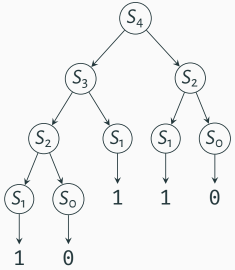
A **straight-line program (SLP)** is a context-free grammar $\mathcal{G}$ which produces exactly one string.

Every variable occurs exactly once on the left-hand side of a rule and the variables are topologically ordered.

grammar

$$\begin{aligned} S_4 &\to S_3\,S_2 \\ S_3 &\to S_2\,S_1 \\ S_2 &\to S_1\,S_0 \\ S_1 &\to \mathtt{1} \\ S_0 &\to \mathtt{0} \end{aligned}$$

derivation tree



The string length is denoted by $N \leqslant 2^{\mathcal{O}(|\mathcal{G}|)}$.
Chomsky normal form: rules of the form $A \to BC$ or $A \to \mathtt{a}$.

The random access problem

For algorithmic applications the two important parameters are **size** and **height**.

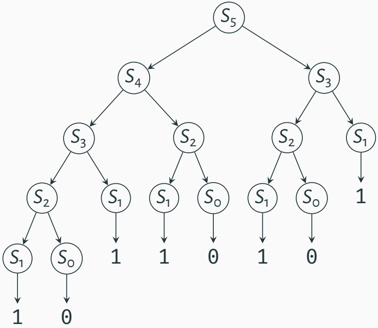
Example: Random access in time $\mathcal{O}(\text{height})$:

$\longrightarrow$ desirable: height $\mathcal{O}(\log N)$ (“balanced SLPs”)

The random access problem

For algorithmic applications the two important parameters are **size** and **height**.

Example: Random access in time $\mathcal{O}(\text{height})$:

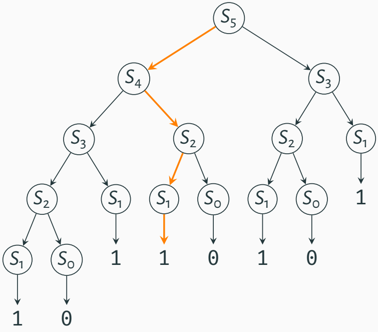


$\longrightarrow$ desirable: height $\mathcal{O}(\log N)$ ("balanced SLPs")

The random access problem

For algorithmic applications the two important parameters are **size** and **height**.

Example: Random access in time $\mathcal{O}(\text{height})$:

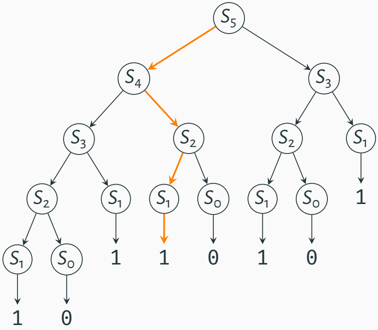


$\longrightarrow$ desirable: height $\mathcal{O}(\log N)$ (“balanced SLPs”)

The random access problem

For algorithmic applications the two important parameters are **size** and **height**.

Example: Random access in time $\mathcal{O}(\text{height})$:



$\longrightarrow$ desirable: height $\mathcal{O}(\log N)$ ("balanced SLPs")

Logarithmic height

Balancing Theorem (G, Jeż, Lohrey, FOCS 2019, JACM 2021)

Given an SLP $\mathcal{G}$ for a string of length N. One can compute in linear time an equivalent SLP of height $\mathcal{O}(\log N)$ and size $\mathcal{O}(|\mathcal{G}|)$.

$\rightarrow$ previously: $\mathcal{O}(|\mathcal{G}| \cdot \log N)$ [Rytter, 2002; Charikar et al., 2002]

$\rightarrow$ simple solution for random access in $\mathcal{O}(\log N)$ time and **linear** space

Other applications:

- rank and select queries, computing fingerprints, range minimum queries, subsequence matching
- spanner evaluation [Schmid, Schweikardt 2021]

Logarithmic height

Balancing Theorem (G, Jeż, Lohrey, FOCS 2019, JACM 2021)

Given an SLP $\mathcal{G}$ for a string of length N. One can compute in linear time an equivalent SLP of height $\mathcal{O}(\log N)$ and size $\mathcal{O}(|\mathcal{G}|)$.

$\rightarrow$ previously: $\mathcal{O}(|\mathcal{G}| \cdot \log N)$ [Rytter, 2002; Charikar et al., 2002]

$\rightarrow$ simple solution for random access in $\mathcal{O}(\log N)$ time and **linear** space

Other applications:

- rank and select queries, computing fingerprints, range minimum queries, subsequence matching
- spanner evaluation [Schmid, Schweikardt 2021]

Logarithmic height

Balancing Theorem (G, Jeż, Lohrey, FOCS 2019, JACM 2021)

Given an SLP $\mathcal{G}$ for a string of length N. One can compute in linear time an equivalent SLP of height $\mathcal{O}(\log N)$ and size $\mathcal{O}(|\mathcal{G}|)$.

$\rightarrow$ previously: $\mathcal{O}(|\mathcal{G}| \cdot \log N)$ [Rytter, 2002; Charikar et al., 2002]

$\rightarrow$ simple solution for random access in $\mathcal{O}(\log N)$ time and **linear** space

Other applications:

- rank and select queries, computing fingerprints, range minimum queries, subsequence matching
- spanner evaluation [Schmid, Schweikardt 2021]

Logarithmic height

Balancing Theorem (G, Jeż, Lohrey, FOCS 2019, JACM 2021)

Given an SLP $\mathcal{G}$ for a string of length N. One can compute in linear time an equivalent SLP of height $\mathcal{O}(\log N)$ and size $\mathcal{O}(|\mathcal{G}|)$.

$\rightarrow$ previously: $\mathcal{O}(|\mathcal{G}| \cdot \log N)$ [Rytter, 2002; Charikar et al., 2002]

$\rightarrow$ simple solution for random access in $\mathcal{O}(\log N)$ time and **linear** space

Other applications:

- rank and select queries, computing fingerprints, range minimum queries, subsequence matching
- spanner evaluation [Schmid, Schweikardt 2021]

This paper

1. Can we refine the balancing theorem, establishing stronger balancedness properties “for free”? (= $\mathcal{O}(1)$ factor increase)

2. Which algorithmic applications can be obtained using such balancing results?

Compressed pattern matching

Does balancing lead to improved algorithms for compressed pattern matching?

Given an uncompressed pattern P of length m, and a compressed text T of length N and compressed size n.

Question Does P occur in T?

Theorem (Gawrychowksi, 2011)

Compressed pattern matching can be solved in time

- $\mathcal{O}(m + n \cdot \log N)$ for LZ77-compression and for SLPs,
- $\mathcal{O}(m + n)$ for weight-balanced SLPs.

[Charikar et al. '02], [Gawrychowski '11]

Compressed pattern matching

Does balancing lead to improved algorithms for compressed pattern matching?

Given an uncompressed pattern P of length m, and
a compressed text T of length N and compressed size n.

Question Does P occur in T?

Theorem (Gawrychowksi, 2011)

Compressed pattern matching can be solved in time

- $\mathcal{O}(m + n \cdot \log N)$ for LZ77-compression and for SLPs,
- $\mathcal{O}(m + n)$ for weight-balanced SLPs.

[Charikar et al. '02], [Gawrychowski '11]

Compressed pattern matching

Does balancing lead to improved algorithms for compressed pattern matching?

Given an uncompressed pattern P of length m, and
a compressed text T of length N and compressed size n.

Question Does P occur in T?

Theorem (Gawrychowksi, 2011)

Compressed pattern matching can be solved in time

- $\mathcal{O}(m + n \cdot \log N)$ for LZ77-compression and for SLPs,
- $\mathcal{O}(m + n)$ for weight-balanced SLPs.

[Charikar et al. '02], [Gawrychowski '11]

Compressed pattern matching

Does balancing lead to improved algorithms for compressed pattern matching?

Given an uncompressed pattern P of length m, and
a compressed text T of length N and compressed size n.

Question Does P occur in T?

Theorem (Gawrychowksi, 2011)

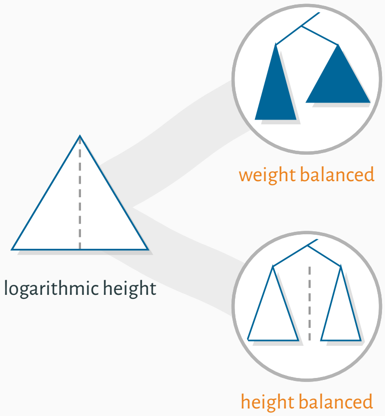
Compressed pattern matching can be solved in time

- $\mathcal{O}(m + n \cdot \log N)$ for LZ77-compression and for SLPs,
- $\mathcal{O}(m + n)$ for weight-balanced SLPs.

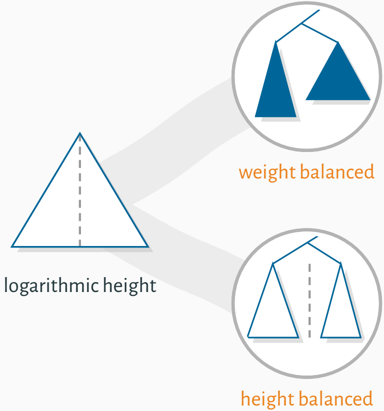
$$\begin{tikzcd} \boxed{\text{LZ77}} \arrow[dr, "\cdot \log N"] & \\ & \boxed{\text{weight-balanced SLPs}} \\ \boxed{\text{SLPs}} \arrow[ur, "\cdot \log N"'] & \end{tikzcd}$$

[Charikar et al. '02], [Gawrychowski '11]

Zoo of balanced SLPs

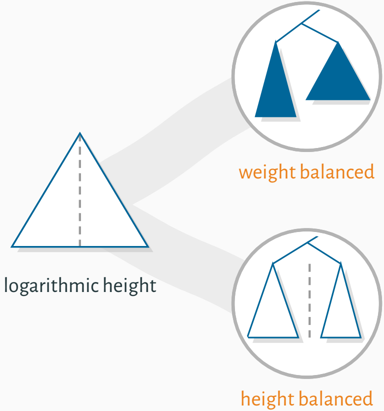


Question: Is the multiplicative cost of $\mathcal{O}(\log N)$ optimal?



For all $A \rightarrow BC$: $|B|/|C| = \Theta(1)$.

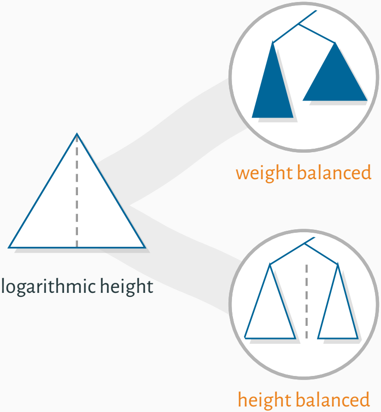
Question: Is the multiplicative cost of $\mathcal{O}(\log N)$ optimal?



For all $A \to BC$: $|B|/|C| = \Theta(1)$.

For all $A \to BC$: $|\text{height}(B) - \text{height}(C)| \leqslant 1$.

Question: Is the multiplicative cost of $\mathcal{O}(\log N)$ optimal?



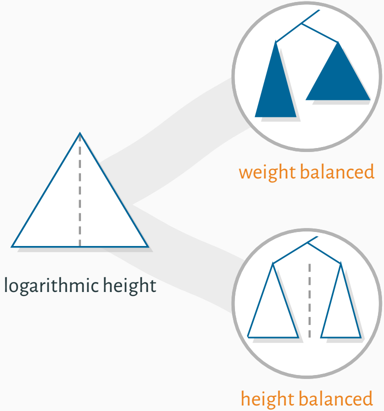
For all $A \to BC$: $|B|/|C| = \Theta(1)$.

multiplicative cost: $\mathcal{O}(\log N)$ [Charikar et al. '02]

For all $A \to BC$: $|\text{height}(B) - \text{height}(C)| \leqslant 1$.

multiplicative cost: $\mathcal{O}(\log N)$ [Rytter '02]

Question: Is the multiplicative cost of $\mathcal{O}(\log N)$ optimal?



For all $A \to BC$: $|B|/|C| = \Theta(1)$.

multiplicative cost: $\mathcal{O}(\log N)$ [Charikar et al. '02]

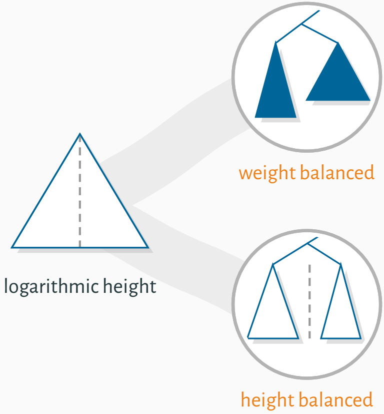
- grammar-based self-index [Gagie et al. '12]
- compr. pattern matching [Gawrychowski '11]

For all $A \to BC$: $|\text{height}(B) - \text{height}(C)| \leqslant 1$.

multiplicative cost: $\mathcal{O}(\log N)$ [Rytter '02]

- grammar-based self-index [Gagie et al. '12]
- fine-grained complexity [Abboud et al. '17]

Question: Is the multiplicative cost of $\mathcal{O}(\log N)$ optimal?



For all $A \to BC$: $|B|/|C| = \Theta(1)$.

multiplicative cost: $\mathcal{O}(\log N)$ [Charikar et al. '02]

- grammar-based self-index [Gagie et al. '12]
- compr. pattern matching [Gawrychowski '11]

For all $A \to BC$: $|\text{height}(B) - \text{height}(C)| \leqslant 1$.

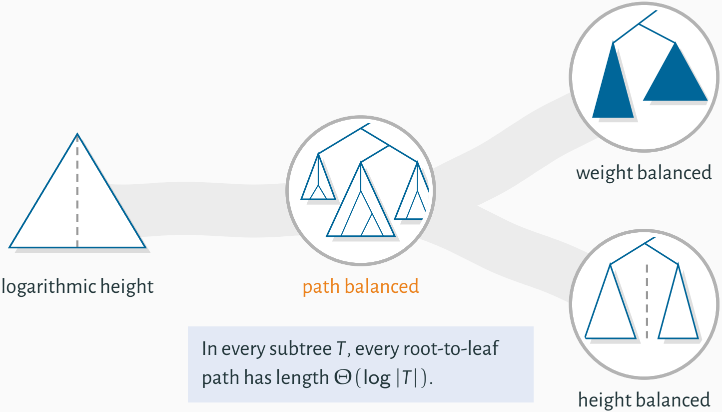
multiplicative cost: $\mathcal{O}(\log N)$ [Rytter '02]

- grammar-based self-index [Gagie et al. '12]
- fine-grained complexity [Abboud et al. '17]

Question: Is the multiplicative cost of $\mathcal{O}(\log N)$ optimal?

Theorem

There exist SLPs of size $\mathcal{O}(n)$ such that any equivalent path balanced SLP has size $\Omega(n \log N)$.



In every subtree T, every root-to-leaf path has length $\Theta(\log |T|)$.

Theorem

There exist SLPs of size $\mathcal{O}(n)$ such that any equivalent path balanced SLP has size $\Omega(n \log N)$.

Good News

1. Compressed Pattern Matching

Theorem (G, Gawrychowksi, 2022)

Compressed pattern matching for SLP-compressed texts can be solved in time $\mathcal{O}(m+n)$.

Relies only on logarithmic height SLPs (and new data structures).

2. A Refined Balancing Theorem

New algorithmic applications:

- finger search problem
- navigation on compressed trees

Good News

1. Compressed Pattern Matching

Theorem (G, Gawrychowksi, 2022)

Compressed pattern matching for SLP-compressed texts can be solved in time $\mathcal{O}(m+n)$.

Relies only on logarithmic height SLPs (and new data structures).

2. A Refined Balancing Theorem

New algorithmic applications:

- finger search problem
- navigation on compressed trees

Contracting SLPs

Definition

An SLP is **contracting** if for every rule $A \to \beta_1 \ldots \beta_k$ and every variable β_i we have $|\beta_i| \leqslant |A|/2$.

- Every variable A has height $\mathcal{O}(\log |A|)$ (locally balanced).
- Given a variable A, one can access $A[i]$ in time $\mathcal{O}(\log |A|)$.
- Useful when multiple strings $s_1, \ldots, s_m$ are compressed using a single SLP.

Theorem

One can convert an SLP $\mathcal{G}$ in linear time into an equivalent contracting SLP of size $\mathcal{O}(|\mathcal{G}|)$ with rules of constant length.

Contracting SLPs

Definition

An SLP is **contracting** if for every rule $A \to \beta_1 \ldots \beta_k$ and every variable β_i we have $|\beta_i| \leqslant |A|/2$.

- Every variable A has height $\mathcal{O}(\log |A|)$ (locally balanced).
- Given a variable A, one can access $A[i]$ in time $\mathcal{O}(\log |A|)$.
- Useful when multiple strings $s_1, \ldots, s_m$ are compressed using a single SLP.

Theorem

One can convert an SLP $\mathcal{G}$ in linear time into an equivalent contracting SLP of size $\mathcal{O}(|\mathcal{G}|)$ with rules of constant length.

Contracting SLPs

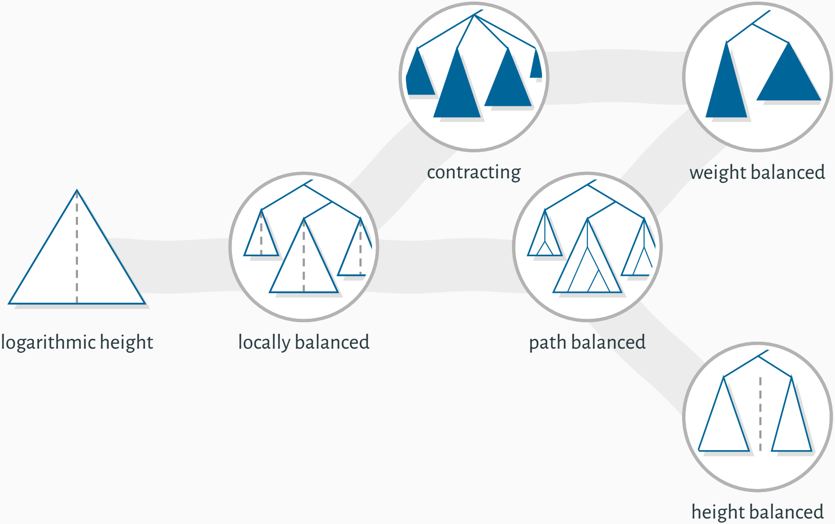
Definition

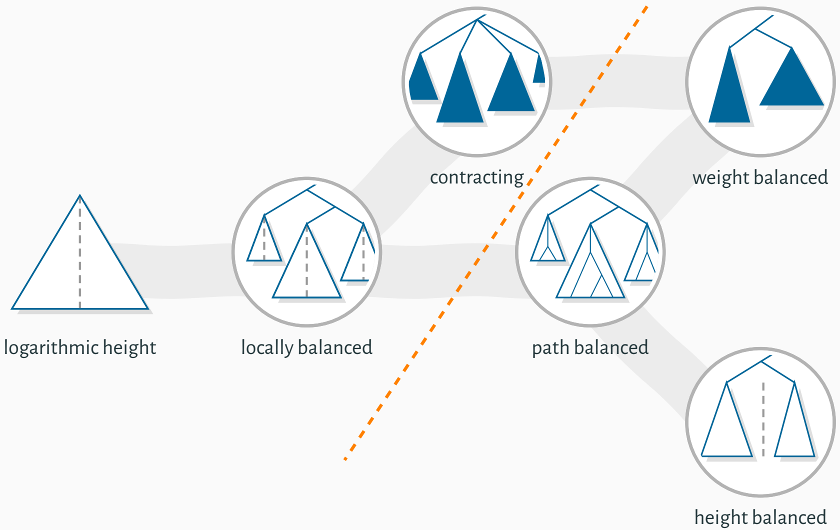
An SLP is **contracting** if for every rule $A \to \beta_1 \ldots \beta_k$ and every variable β_i we have $|\beta_i| \leqslant |A|/2$.

- Every variable A has height $\mathcal{O}(\log |A|)$ (locally balanced).
- Given a variable A, one can access $A[i]$ in time $\mathcal{O}(\log |A|)$.
- Useful when multiple strings $s_1, \ldots, s_m$ are compressed using a single SLP.

Theorem

One can convert an SLP $\mathcal{G}$ in linear time into an equivalent contracting SLP of size $\mathcal{O}(|\mathcal{G}|)$ with rules of constant length.


contracting
weight balanced
logarithmic height
locally balanced
path balanced
height balanced

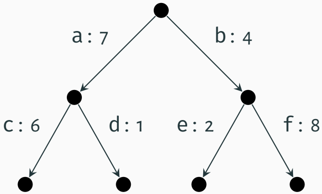

contracting
weight balanced
logarithmic height
locally balanced
path balanced
height balanced

Proof sketch

Theorem

One can convert an SLP $\mathcal{G}$ in linear time into an equivalent contracting SLP of size $\mathcal{O}(|\mathcal{G}|)$ with rules of constant length.

Given a trie T with edges labeled by weighted symbols,
define all prefixes by a contracting SLP.



Possible with a contracting SLP of size $O(|T|)$.

Applications

Finger search

In **finger search** on a (compressed) string we want to support the following operations: [Bille, Christiansen, Cording, Gørtz, 2018]

- `setFinger`(i)
- `access`(i)
- `moveFinger`(i)

$\boxed{\texttt{loremipsumdolorsitametconsetetursadipscing}}$

Finger search

In **finger search** on a (compressed) string we want to support the following operations: [Bille, Christiansen, Cording, Gørtz, 2018]

- `setFinger`(i)
- `access`(i)
- `moveFinger`(i)

`loremipsumdolorsitametconsetetursadipscing`

Finger search

In **finger search** on a (compressed) string we want to support the following operations: [Bille, Christiansen, Cording, Gørtz, 2018]

- `setFinger`(i)
- `access`(i)
- `moveFinger`(i)

`loremipsumdolorsitametconsetetursadipscing`

d

Finger search

In **finger search** on a (compressed) string we want to support the following operations: [Bille, Christiansen, Cording, Gørtz, 2018]

- `setFinger`(i)
- `access`(i)
- `moveFinger`(i)

`loremipsumdolorsitametconseteturs adipscing`

d

ideally in $\mathcal{O}(\log d)$ time

Finger search

In **finger search** on a (compressed) string we want to support the following operations: [Bille, Christiansen, Cording, Gørtz, 2018]

- `setFinger`(i)
- `access`(i)
- `moveFinger`(i)

`loremipsumdolorsitametconsetetursadipscing`

👆

Finger search

In **finger search** on a (compressed) string we want to support the following operations: [Bille, Christiansen, Cording, Gørtz, 2018]

- `setFinger`(i)
- `access`(i)
- `moveFinger`(i)

`loremipsumdolorsitametconsetetursadipscing`

d

Finger search

In **finger search** on a (compressed) string we want to support the following operations: [Bille, Christiansen, Cording, Gørtz, 2018]

- `setFinger`(i)
- `access`(i)
- `moveFinger`(i)

`loremipsumdolorsitametconsetetursadipscing`

$\xrightarrow[d]{}$ 👆

ideally in $\mathcal{O}(\log d)$ time

Finger search

Theorem (Bille, Christiansen, Cording, Gørtz, 2018)

Given an SLP $\mathcal{G}$ for a string of length N, one can support

- `setFinger`(i) in time $\mathcal{O}(\log N)$
- `access`(i) in time $\mathcal{O}(\log d + \log\log N)$
- `moveFinger`(i) in time $\mathcal{O}(\log d + \log\log N)$

where d is the distance between i and the finger position,
using $\mathcal{O}(|\mathcal{G}|)$ preprocessing time and space.

Choosing $t = \log^* N$ yields $\mathcal{O}(\log d)$ time for `access`(i) and `moveFinger`(i).

Finger search

Theorem (Bille, Christiansen, Cording, Gørtz, 2018; G, 2021)

Given an SLP $\mathcal{G}$ for a string of length N, one can support

- `setFinger`(i) in time $\mathcal{O}(\log N)$
- `access`(i) in time $\mathcal{O}(\log d + \log^{(t)} N)$
- `moveFinger`(i) in time $\mathcal{O}(\log d + \log^{(t)} N)$

where d is the distance between i and the finger position, using $\mathcal{O}(t \cdot |\mathcal{G}|)$ preprocessing time and space, for any $t \geqslant 1$.

Choosing $t = \log^* N$ yields $\mathcal{O}(\log d)$ time for `access`(i) and `moveFinger`(i).

Finger search

Theorem (Bille, Christiansen, Cording, Gørtz, 2018; G, 2021)

Given an SLP $\mathcal{G}$ for a string of length N, one can support

- `setFinger`(i) in time $\mathcal{O}(\log N)$
- `access`(i) in time $\mathcal{O}(\log d + \log^{(t)} N)$
- `moveFinger`(i) in time $\mathcal{O}(\log d + \log^{(t)} N)$

where d is the distance between i and the finger position, using $\mathcal{O}(t \cdot |\mathcal{G}|)$ preprocessing time and space, for any $t \geq 1$.

Choosing $t = \log^* N$ yields $\mathcal{O}(\log d)$ time for `access`(i) and `moveFinger`(i).

Navigation on compressed trees

Extend a navigation data structure on FSLP-compressed trees:

Theorem (Reh, Sieber, 2020)

Given a forest SLP for a tree T, one can support in linear space the following navigation steps on T in constant time:

- `parent()` in $\mathcal{O}(1)$ time
- `first_child()`, `last_child()` in $\mathcal{O}(1)$ time
- `next_sibling()`, `prev_sibling()` in $\mathcal{O}(1)$ time
- `get_symbol()` in $\mathcal{O}(1)$ time

Navigation on compressed trees

Extend a navigation data structure on FSLP-compressed trees:

Theorem (Reh, Sieber, 2020)

Given a forest SLP for a tree T, one can support in linear space the following navigation steps on T in constant time:

- `parent()` in $\mathcal{O}(1)$ time
- `first_child()`, `last_child()` in $\mathcal{O}(1)$ time
- `next_sibling()`, `prev_sibling()` in $\mathcal{O}(1)$ time
- `get_symbol()` in $\mathcal{O}(1)$ time

Navigation on compressed trees

Extend a navigation data structure on FSLP-compressed trees:

Theorem (Reh, Sieber, 2020; G, 2021)

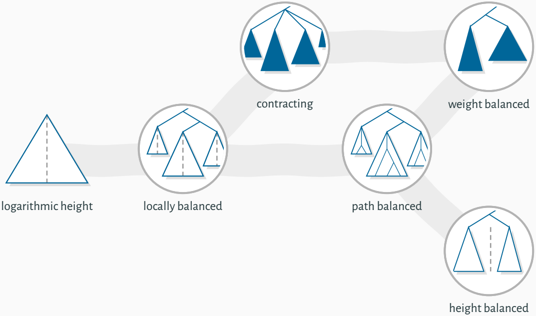
Given a forest SLP for a tree T, one can support in linear space the following navigation steps on T in constant time:

- `parent()` in $\mathcal{O}(1)$ time
- `first_child()`, `last_child()` in $\mathcal{O}(1)$ time
- `next_sibling()`, `prev_sibling()` in $\mathcal{O}(1)$ time
- `get_symbol()` in $\mathcal{O}(1)$ time
- `child(`i`)` in $\mathcal{O}(\log d)$ time

where d is the degree of the current node.

Conclusion

Balancing in grammar-based compression as a preprocessing step that enables fast queries on the compressed data.

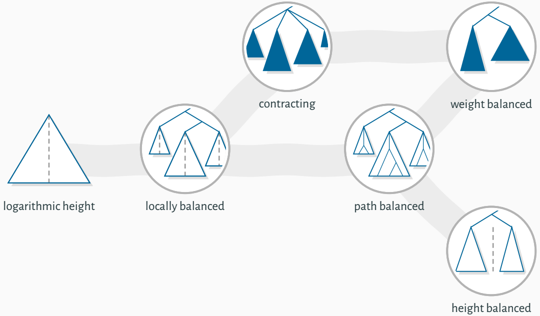


Open questions:

Finger search in $\mathcal{O}(\log d)$ time and $\mathcal{O}(|\mathcal{G}|)$ space?
random access for LZ77 in $\mathcal{O}(\log N)$ time and linear space?
Balancing for LZ77/collage systems?

Conclusion

Balancing in grammar-based compression as a preprocessing step that enables fast queries on the compressed data.



Open questions:

Finger search in $\mathcal{O}(\log d)$ time and $\mathcal{O}(|\mathcal{G}|)$ space?
random access for LZ77 in $\mathcal{O}(\log N)$ time and linear space?
Balancing for LZ77/collage systems?