Compression by Contracting Straight-Line Programs

Moses Ganardi

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS
Two goals in data compression:

- Store data in a **compact form** (lossless).
- Support **efficient queries** directly on the compressed representation.

→ Avoid decompression!
Two goals in data compression:

- Store data in a *compact form* (lossless).
- Support *efficient queries* directly on the compressed representation.
  → Avoid decompression!
Two goals in data compression:

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

<table>
<thead>
<tr>
<th>compression</th>
<th>algorithmics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kolmogorov complexity</td>
<td>LZ77</td>
</tr>
<tr>
<td>grammar-based compression</td>
<td>RLE</td>
</tr>
<tr>
<td>RLE</td>
<td>uncompressed</td>
</tr>
</tbody>
</table>
Grammar-based compression
Grammar-based compression

A **straight-line program (SLP)** is a context-free grammar $G$ 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**

$$
egin{align*}
S_4 & \rightarrow S_3 S_2 \\
S_3 & \rightarrow S_2 S_1 \\
S_2 & \rightarrow S_1 S_0 \\
S_1 & \rightarrow 1 \\
S_0 & \rightarrow \emptyset
\end{align*}
$$

**Derivation tree**

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

**Chomsky normal form**: rules of the form $A \rightarrow BC$ or $A \rightarrow a$. 
A **straight-line program (SLP)** is a context-free grammar $G$ which produces exactly one string.

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

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

**Chomsky normal form**: rules of the form $A \rightarrow BC$ or $A \rightarrow a$. 

\[
\begin{align*}
S_4 & \rightarrow S_3 S_2 \\
S_3 & \rightarrow S_2 S_1 \\
S_2 & \rightarrow S_1 S_0 \\
S_1 & \rightarrow 1 \\
S_0 & \rightarrow \emptyset
\end{align*}
\]
A **straight-line program (SLP)** is a context-free grammar $G$ which produces exactly one string.

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

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

**Chomsky normal form**: rules of the form $A \rightarrow BC$ or $A \rightarrow a$. 

---

**Grammar**

- $S_4 \rightarrow S_3 S_2$
- $S_3 \rightarrow S_2 S_1$
- $S_2 \rightarrow S_1 S_0$
- $S_1 \rightarrow 1$
- $S_0 \rightarrow 0$

**Derivation tree**
A **straight-line program (SLP)** is a context-free grammar $G$ which produces exactly one string.

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

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

**Chomsky normal form**: rules of the form $A \rightarrow BC$ or $A \rightarrow a$. 

**Grammar**

$$
\begin{align*}
S_4 & \rightarrow S_3 S_2 \\
S_3 & \rightarrow S_2 S_1 \\
S_2 & \rightarrow S_1 S_0 \\
S_1 & \rightarrow 1 \\
S_0 & \rightarrow 0
\end{align*}
$$

**Derivation tree**

```
                              S_4
                             /   \
                            S_3   S_2
                           /   \   /   \  
                          S_2   S_1 S_1 S_0
                         /     /     /      /
                         S_2   S_1   S_1   S_0
                        /   /    /    /  
                       S_1 S_0 1 1 0
                        |  |     |  |
                      1 0 1 0
```


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

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

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

**Chomsky normal form**: rules of the form $A \rightarrow BC$ or $A \rightarrow a$. 
A **straight-line program (SLP)** is a context-free grammar $G$ 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**

$$S_4 \rightarrow S_3 S_2$$

$$S_3 \rightarrow S_2 S_1$$

$$S_2 \rightarrow S_1 S_0$$

$$S_1 \rightarrow 1$$

$$S_0 \rightarrow 0$$

**derivation tree**

The string length is denoted by $N \leq 2^{\Theta(|G|)}$.

**Chomsky normal form**: rules of the form $A \rightarrow BC$ or $A \rightarrow a$. 
The random access problem

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

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

Desirable: height $O(\log N)$ (“balanced SLPs”)
The random access problem

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

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

$$
\begin{array}{c}
S_5 \\
S_4 \\
S_3 \\
S_2 \\
S_1 \\
S_0
\end{array}
$$

$\rightarrow$ 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 $\mathcal{O}(\text{height})$:

$\rightarrow$ desirable: height $\mathcal{O}(\log N)$ (“balanced SLPs”)
For algorithmic applications the two important parameters are **size** and **height**.

**Example:** Random access in time $\Theta(\text{height})$: 

\[
\begin{align*}
S_5 & \\
S_4 & \quad S_3 \\
S_2 & \quad S_1 \quad S_0 \\
S_1 & \quad S_0 \\
\end{align*}
\]

\[
\begin{align*}
1 & \quad 1 & \quad 0 & \quad 1 & \quad 0 \\
1 & \quad 0 \\
\end{align*}
\]

\[\longrightarrow \text{desirable: height } \Theta(\log N) \text{ ("balanced SLPs")}\]
Balancing Theorem (G, Jeż, Lohrey, FOCS 2019, JACM 2021)

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

→ previously: $\Theta(|G| \cdot \log N)$  
[Rytter, 2002; Charikar et al., 2002]

→ simple solution for random access in $\Theta(\log N)$ time and linear space

Other applications:

- rank and select queries, computing fingerprints, range minimum queries, subsequence matching
- spanner evaluation  
[Schmid, Schweikardt 2021]
Balancing Theorem (G, Jeż, Lohrey, FOCS 2019, JACM 2021)

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

→ previously: $O(|G| \cdot \log N)$  

→ simple solution for random access in $O(\log N)$ time and linear space

Other applications:

- rank and select queries, computing fingerprints, range minimum queries, subsequence matching
- spanner evaluation

[Rytter, 2002; Charikar et al., 2002]  

[Schmid, Schweikardt 2021]
Balancing Theorem (G, Jeż, Lohrey, FOCS 2019, JACM 2021)

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

$\rightarrow$ previously: $O(|G| \cdot \log N)$  

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

Other applications:

- rank and select queries, computing fingerprints, range minimum queries, subsequence matching
- spanner evaluation

[Rytter, 2002; Charikar et al., 2002]  
[Schmid, Schweikardt 2021]
Logarithmic height

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

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

$\rightarrow$ previously: $\Theta(|G| \cdot \log N)$

$\rightarrow$ simple solution for random access in $\Theta(\log N)$ time and linear space

Other applications:

- rank and select queries, computing fingerprints, range minimum queries, subsequence matching
- spanner evaluation

[Schmid, Schweikardt 2021]
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?
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 (Gawrychowsk, 2011)**

Compressed pattern matching can be solved in time

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

[Charikar et al. ’02], [Gawrychowski ’11]
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]
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 (Gawrychowski, 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 (Gawrychowski, 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]
Zoo of balanced SLPs
Question: Is the multiplicative cost of $\Theta(\log N)$ optimal?
For all $A \rightarrow BC$: $|B|/|C| = \Theta(1)$.

**Question:** Is the multiplicative cost of $\Theta(\log N)$ optimal?
Question: Is the multiplicative cost of $\Theta(\log N)$ optimal?

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

For all $A \rightarrow BC$: $|\text{height}(B) - \text{height}(C)| \leq 1$. 
Question: Is the multiplicative cost of $O(\log N)$ optimal?

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

Multiplicative cost: $O(\log N)$  
[Charikar et al. '02]

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

Multiplicative cost: $O(\log N)$  
[Rytter '02]
Question: Is the multiplicative cost of $O(\log N)$ optimal?

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

- multiplicative cost: $O(\log N)$  
  - grammar-based self-index  
  - compr. pattern matching

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

- multiplicative cost: $O(\log N)$
  - grammar-based self-index
  - fine-grained complexity
Question: Is the multiplicative cost of $\Theta(\log N)$ optimal?
Theorem

There exist SLPs of size $\Theta(n)$ such that any equivalent path balanced SLP has size $\Omega(n \log N)$. 
Theorem
There exist SLPs of size $\Theta(n)$ such that any equivalent path balanced SLP has size $\Omega(n \log N)$.

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

- logarithmic height
- path balanced
- weight balanced
- height balanced
1. Compressed Pattern Matching

**Theorem (G, Gawrychowskki, 2022)**

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

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
1. Compressed Pattern Matching

**Theorem (G, Gawrychowskii, 2022)**

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

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 $A \rightarrow \beta_1 \ldots \beta_k$ and every variable $\beta_i$ we have $|\beta_i| \leq |A|/2$.

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

**Theorem**
One can convert an SLP $G$ in linear time into an equivalent contracting SLP of size $\Theta(|G|)$ with rules of constant length.
Contracting SLPs

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

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

**Definition**

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

- 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.
logarithmic height

locally balanced

contracting

path balanced

weight balanced

height balanced
logarithmic height

locally balanced

contracting

path balanced

weight balanced

height balanced
Proof sketch

Theorem
One can convert an SLP $G$ in linear time into an equivalent contracting SLP of size $O(|G|)$ 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
In **finger search** on a (compressed) string we want to support the following operations:

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

**lorem ipsum dolor sit amet consetetur sadipscing**
In **finger search** on a (compressed) string we want to support the following operations:

- setFinger($i$)
- access($i$)
- moveFinger($i$)

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

- setFinger($i$)
- access($i$)
- moveFinger($i$)

---

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

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

ideally in $O(\log d)$ time
In **finger search** on a (compressed) string we want to support the following operations:

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

---

`loremipsumdolorsitametconsetetetursadipscing`
In finger search on a (compressed) string we want to support the following operations:

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

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

- setFinger($i$)
- access($i$)
- moveFinger($i$)

loremipsumdolorsitametconsetetursadipscing

ideally in $\Theta(\log d)$ time
Finger search

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

Given an SLP $G$ for a string of length $N$, one can support

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

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

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

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

Given an SLP $G$ for a string of length $N$, one can support

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

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

Choosing $t = \log^* N$ yields $\mathcal{O}(\log d)$ time for $\text{access}(i)$ and $\text{moveFinger}(i)$. 
Theorem (Bille, Christiansen, Cording, Gørtz, 2018; G, 2021)

Given an SLP $G$ for a string of length $N$, one can support

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

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

Choosing $t = \log^* N$ yields $O(\log d)$ time for $\text{access}(i)$ and $\text{moveFinger}(i)$. 
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 $O(1)$ time
- `first_child()`, `last_child()` in $O(1)$ time
- `next_sibling()`, `prev_sibling()` in $O(1)$ time
- `get_symbol()` in $O(1)$ time
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 $O(1)$ time
- `first_child()`, `last_child()` in $O(1)$ time
- `next_sibling()`, `prev_sibling()` in $O(1)$ time
- `get_symbol()` in $O(1)$ time
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 $O(1)$ time
- first_child(), last_child() in $O(1)$ time
- next_sibling(), prev_sibling() in $O(1)$ time
- get_symbol() in $O(1)$ time
- child($i$) in $O(\log d)$ time

where $d$ is the degree of the current node.
Balancing in grammar-based compression as a preprocessing step that enables fast queries on the compressed data.

**Open questions:**

Finger search in $O(\log d)$ time and $O(|S|)$ space?
random access for LZ77 in $O(\log N)$ time and linear space?
Balancing for LZ77/collage systems?
Balancing in grammar-based compression as a preprocessing step that enables fast queries on the compressed data.

**Open questions:**

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