Slowing Down Top Trees for Better Worst-Case Compression

Bartłomiej Dudek[1] Paweł Gawrychowski[1]

[1]University of Wrocław

February 8, 2019

Straight-line program (SLP)

A context-free grammar in Chomsky normal form with exactly one production for each nonterminal, hence generating exactly one string.

Fibonacci words

$$\begin{aligned} F_0 &= \mathtt{a} & \mathtt{a} \\ F_1 &= \mathtt{b} & \mathtt{b} \\ F_2 &= F_1F_0 & \mathtt{ba} \\ F_3 &= F_2F_1 & \mathtt{bab} \\ F_4 &= F_3F_2 & \mathtt{babba} \\ F_5 &= F_4F_3 & \mathtt{babbabab} \\ F_6 &= F_5F_4 & \mathtt{babbababbabba} \end{aligned}$$

Straight-line program (SLP)

A context-free grammar in Chomsky normal form with exactly one production for each nonterminal, hence generating exactly one string.

Fibonacci words

F_0	$=$	a	a
F_1	$=$	b	b
F_2	$=$	$F_1 F_0$	ba
F_3	$=$	$F_2 F_1$	bab
F_4	$=$	$F_3 F_2$	babba
F_5	$=$	$F_4 F_3$	babbabab
F_6	$=$	$F_5 F_4$	babbababbabba

Straight-line program (SLP)

A context-free grammar in Chomsky normal form with exactly one production for each nonterminal, hence generating exactly one string.

Fibonacci words

$F_0 = \mathtt{a}$		a
$F_1 = \mathtt{b}$		b
$F_2 = F_1 F_0$		ba
$F_3 = F_2 F_1$		bab
$F_4 = F_3 F_2$		babba
$F_5 = F_4 F_3$		babbabab
$F_6 = F_5 F_4$		babbababbabba

Straight-line program (SLP)

What is the size of the smallest SLP deriving a string $s[1..n]$ over an alphabet of size σ?

By a counting argument: $\Omega(\frac{n}{\log_\sigma n})$.

Constructing an SLP of size $\mathcal{O}(\frac{n}{\log_\sigma n})$

1. Let $b = \frac{1}{2}\log_\sigma n$.
2. For every string t s.t. $|t| \leq b$ prepare a nonterminal deriving t.
3. Cut s into blocks of length b and create a production $S \rightarrow B_1 B_2 \ldots B_{n/b}$, where B_i derives the i-th block.
4. Overall size is $\mathcal{O}(n/b + \sum_{i=0}^{b} \sigma^i) = \mathcal{O}(n/b + \sqrt{n})$.

Straight-line program (SLP)

What is the size of the smallest SLP deriving a string $s[1..n]$ over an alphabet of size σ?

By a counting argument: $\Omega(\frac{n}{\log_\sigma n})$.

Constructing an SLP of size $\mathcal{O}(\frac{n}{\log_\sigma n})$

1. Let $b = \frac{1}{2} \log_\sigma n$.
2. For every string t s.t. $|t| \le b$ prepare a nonterminal deriving t.
3. Cut s into blocks of length b and create a production $S \to B_1 B_2 \ldots B_{n/b}$, where B_i derives the i-th block.
4. Overall size is $\mathcal{O}(n/b + \sum_{i=0}^{b} \sigma^i) = \mathcal{O}(n/b + \sqrt{n})$.

Straight-line program (SLP)

What is the size of the smallest SLP deriving a string $s[1..n]$ over an alphabet of size σ?

By a counting argument: $\Omega(\frac{n}{\log_\sigma n})$.

Constructing an SLP of size $\mathcal{O}(\frac{n}{\log_\sigma n})$

1. Let $b = \frac{1}{2}\log_\sigma n$.
2. For every string t s.t. $|t| \leq b$ prepare a nonterminal deriving t.
3. Cut s into blocks of length b and create a production $S \rightarrow B_1 B_2 \ldots B_{n/b}$, where B_i derives the i-th block.
4. Overall size is $\mathcal{O}(n/b + \sum_{i=0}^{b} \sigma^i) = \mathcal{O}(n/b + \sqrt{n})$.

Straight-line program (SLP)

What is the size of the smallest SLP deriving a string $s[1..n]$ over an alphabet of size σ?

By a counting argument: $\Omega(\frac{n}{\log_\sigma n})$.

Constructing an SLP of size $\mathcal{O}(\frac{n}{\log_\sigma n})$

1. Let $b = \frac{1}{2}\log_\sigma n$.
2. For every string t s.t. $|t| \leq b$ prepare a nonterminal deriving t.
3. Cut s into blocks of length b and create a production $S \rightarrow B_1 B_2 \ldots B_{n/b}$, where B_i derives the i-th block.
4. Overall size is $\mathcal{O}(n/b + \sum_{i=0}^{b} \sigma^i) = \mathcal{O}(n/b + \sqrt{n})$.

Straight-line program (SLP)

What is the size of the smallest SLP deriving a string $s[1..n]$ over an alphabet of size σ?

By a counting argument: $\Omega(\frac{n}{\log_\sigma n})$.

Constructing an SLP of size $\mathcal{O}(\frac{n}{\log_\sigma n})$

1. Let $b = \frac{1}{2}\log_\sigma n$.
2. For every string t s.t. $|t| \leq b$ prepare a nonterminal deriving t.
3. Cut s into blocks of length b and create a production $S \rightarrow B_1 B_2 \dots B_{n/b}$, where B_i derives the i-th block.
4. Overall size is $\mathcal{O}(n/b + \sum_{i=0}^{b} \sigma^i) = \mathcal{O}(n/b + \sqrt{n})$.

Straight-line program (SLP)

What is the size of the smallest SLP deriving a string $s[1..n]$ over an alphabet of size σ?

By a counting argument: $\Omega(\frac{n}{\log_\sigma n})$.

Constructing an SLP of size $\mathcal{O}(\frac{n}{\log_\sigma n})$

1. Let $b = \frac{1}{2}\log_\sigma n$.
2. For every string t s.t. $|t| \leq b$ prepare a nonterminal deriving t.
3. Cut s into blocks of length b and create a production $S \rightarrow B_1 B_2 \ldots B_{n/b}$, where B_i derives the i-th block.
4. Overall size is $\mathcal{O}(n/b + \sum_{i=0}^{b} \sigma^i) = \mathcal{O}(n/b + \sqrt{n})$.

Top Tree Compression

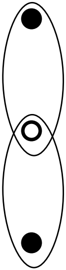
Aim: to represent a tree with clusters
Cluster: a single edge or two clusters merged
(has at most two “boundary“ nodes)

Five possible merges (Bille, Gørtz, Landau, Weimann [ICALP ’13]):

Top Tree Compression

Aim: to represent a tree with clusters
Cluster: a single edge or two clusters merged
(has at most two “boundary“ nodes)

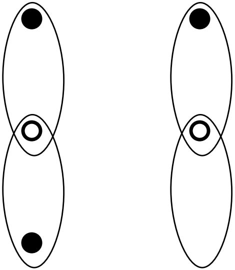
Five possible merges (Bille, Gørtz, Landau, Weimann [ICALP ’13]):



Top Tree Compression

Aim: to represent a tree with clusters
Cluster: a single edge or two clusters merged
(has at most two “boundary“ nodes)

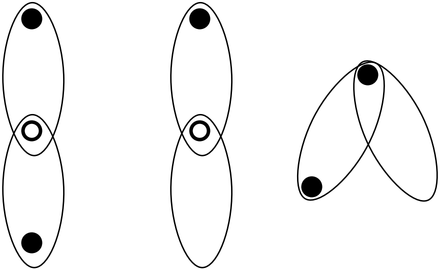
Five possible merges (Bille, Gørtz, Landau, Weimann [ICALP ’13]):



Top Tree Compression

Aim: to represent a tree with clusters
Cluster: a single edge or two clusters merged
(has at most two “boundary“ nodes)

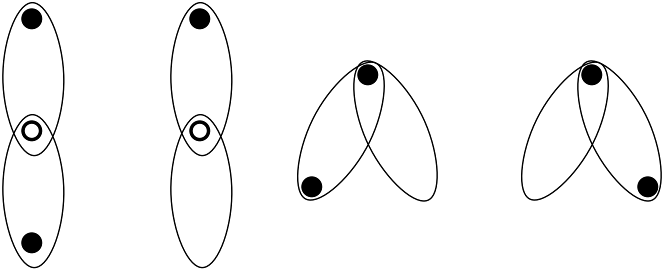
Five possible merges (Bille, Gørtz, Landau, Weimann [ICALP ’13]):



Top Tree Compression

Aim: to represent a tree with clusters
Cluster: a single edge or two clusters merged
(has at most two “boundary“ nodes)

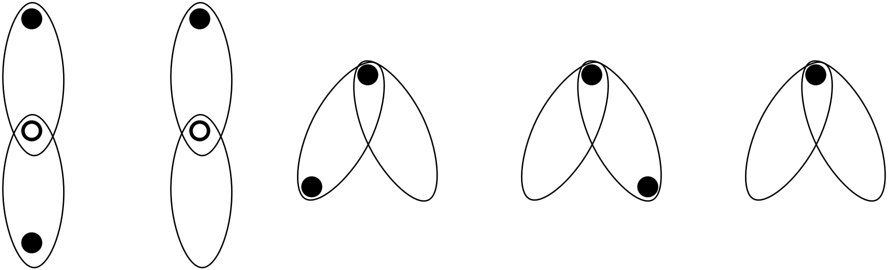
Five possible merges (Bille, Gørtz, Landau, Weimann [ICALP ’13]):



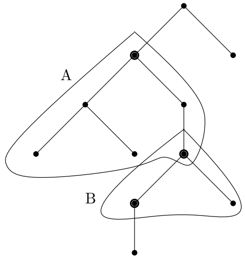
Top Tree Compression

Aim: to represent a tree with clusters
Cluster: a single edge or two clusters merged
(has at most two “boundary“ nodes)

Five possible merges (Bille, Gørtz, Landau, Weimann [ICALP ’13]):



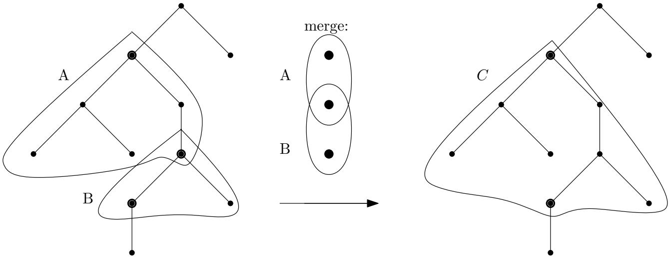
Top Tree Compression



Compression:

1. tree $T \rightarrow$ binary tree $\mathcal{T}$ of clusters
   goal: short
2. binary tree $\mathcal{T} \rightarrow$ top DAG $\mathcal{TD}$ without repeating subtrees
   goal: small

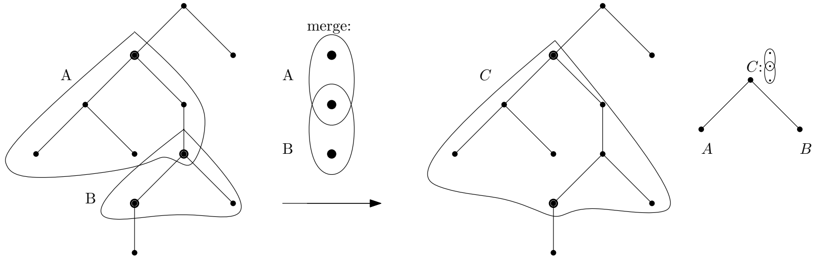
Top Tree Compression



Compression:

1. tree $T \rightarrow$ binary tree $\mathcal{T}$ of clusters
   goal: short
2. binary tree $\mathcal{T} \rightarrow$ top DAG $\mathcal{TD}$ without repeating subtrees
   goal: small

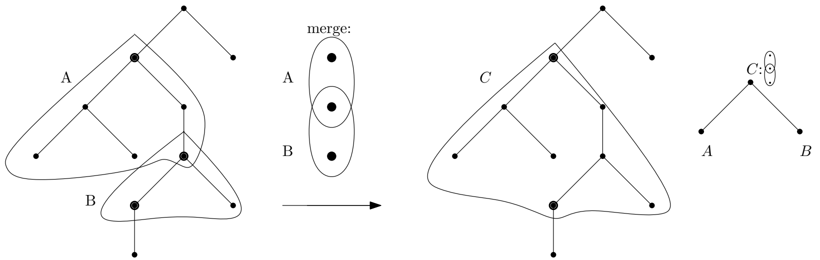
Top Tree Compression



Compression:

1. tree $T \rightarrow$ binary tree $\mathcal{T}$ of clusters
   goal: short
2. binary tree $\mathcal{T} \rightarrow$ top DAG $\mathcal{TD}$ without repeating subtrees
   goal: small

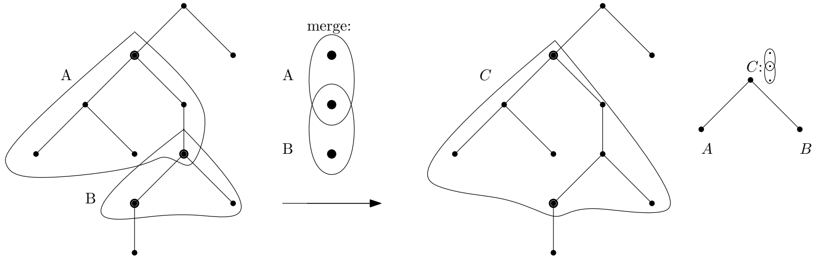
Top Tree Compression



Compression:

1. tree $T \rightarrow$ binary tree $\mathcal{T}$ of clusters
   goal: short
2. binary tree $\mathcal{T} \rightarrow$ top DAG $\mathcal{TD}$ without repeating subtrees
   goal: small

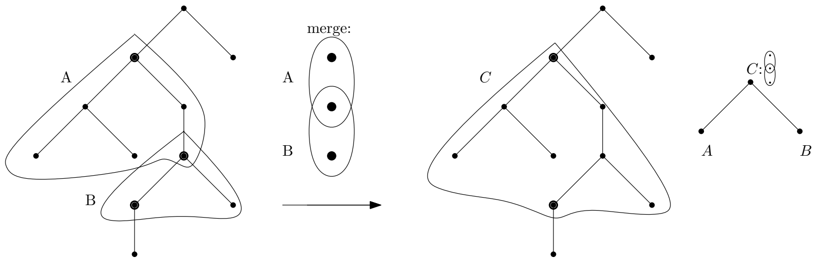
Top Tree Compression



Compression:

1. tree $T \rightarrow$ binary tree $\mathcal{T}$ of clusters
   goal: short
2. binary tree $\mathcal{T} \rightarrow$ top DAG $\mathcal{TD}$ without repeating subtrees
   goal: small

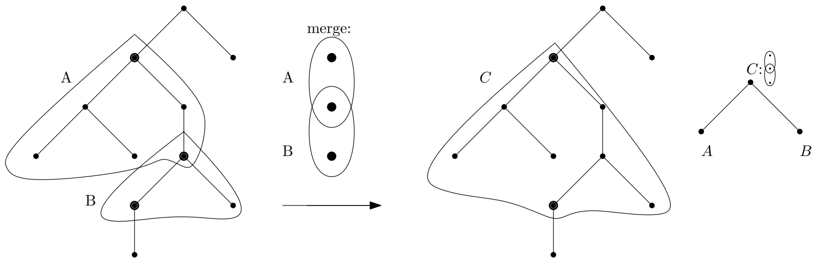
Top Tree Compression



Compression:

1. tree $T \to$ binary tree $\mathcal{T}$ of clusters
   goal: short
2. binary tree $\mathcal{T} \to$ top DAG $\mathcal{TD}$ without repeating subtrees
   goal: small

Top Tree Compression



Compression:

1. tree $T \rightarrow$ binary tree $\mathcal{T}$ of clusters
   goal: short
2. binary tree $\mathcal{T} \rightarrow$ top DAG $\mathcal{TD}$ without repeating subtrees
   goal: small

Top tree decomposition by Bille et al.

The decomposition proceeds in iterations. Each iteration decreases the size of the current tree by a constant factor.

Bille, Gørtz, Landau, Weimann [ICALP '13]

The size of the top DAG is $\mathcal{O}(\frac{n}{\log_\sigma^{0.19} n})$.

Hübschle-Schneider and Raman [SEA '15]

The size of the top DAG is $\mathcal{O}(\frac{n}{\log_\sigma n} \log\log_\sigma n)$.

Similarly as for SLP, the size of the top DAG is $\Omega(\frac{n}{\log_\sigma n})$.

Is the analysis of Hübschle-Schneider and Raman tight?

Top tree decomposition by Bille et al.

The decomposition proceeds in iterations. Each iteration decreases the size of the current tree by a constant factor.

Bille, Gørtz, Landau, Weimann [ICALP '13]

The size of the top DAG is $\mathcal{O}(\frac{n}{\log_\sigma^{0.19} n})$.

Hübschle-Schneider and Raman [SEA '15]

The size of the top DAG is $\mathcal{O}(\frac{n}{\log_\sigma n} \log \log_\sigma n)$.

Similarly as for SLP, the size of the top DAG is $\Omega(\frac{n}{\log_\sigma n})$.

Is the analysis of Hübschle-Schneider and Raman tight?

Top tree decomposition by Bille et al.

The decomposition proceeds in iterations. Each iteration decreases the size of the current tree by a constant factor.

Bille, Gørtz, Landau, Weimann [ICALP '13]

The size of the top DAG is $\mathcal{O}(\frac{n}{\log_\sigma^{0.19} n})$.

Hübschle-Schneider and Raman [SEA '15]

The size of the top DAG is $\mathcal{O}(\frac{n}{\log_\sigma n} \log \log_\sigma n)$.

Similarly as for SLP, the size of the top DAG is $\Omega(\frac{n}{\log_\sigma n})$.

Is the analysis of Hübschle-Schneider and Raman tight?

Top tree decomposition by Bille et al.

The decomposition proceeds in iterations. Each iteration decreases the size of the current tree by a constant factor.

Bille, Gørtz, Landau, Weimann [ICALP '13]

The size of the top DAG is $\mathcal{O}(\frac{n}{\log_\sigma^{0.19} n})$.

Hübschle-Schneider and Raman [SEA '15]

The size of the top DAG is $\mathcal{O}(\frac{n}{\log_\sigma n} \log\log_\sigma n)$.

Similarly as for SLP, the size of the top DAG is $\Omega(\frac{n}{\log_\sigma n})$.

Is the analysis of Hübschle-Schneider and Raman tight?

Top tree decomposition by Bille et al.

The decomposition proceeds in iterations. Each iteration decreases the size of the current tree by a constant factor.

Bille, Gørtz, Landau, Weimann [ICALP '13]

The size of the top DAG is $\mathcal{O}(\frac{n}{\log_\sigma^{0.19} n})$.

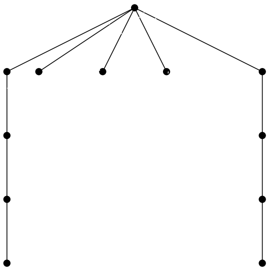
Hübschle-Schneider and Raman [SEA '15]

The size of the top DAG is $\mathcal{O}(\frac{n}{\log_\sigma n} \log\log_\sigma n)$.

Similarly as for SLP, the size of the top DAG is $\Omega(\frac{n}{\log_\sigma n})$.

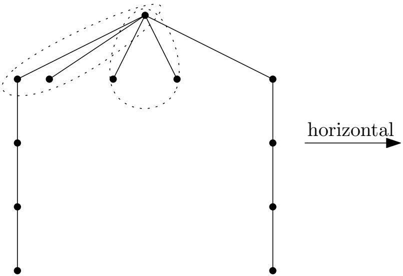
Is the analysis of Hübschle-Schneider and Raman tight?

Single iteration of the algorithm by Bille et al.



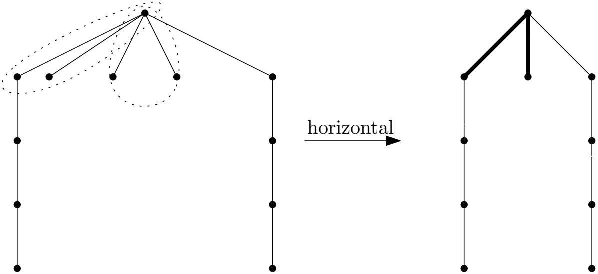
Such iteration decreases the size of the current tree by a constant factor.

Single iteration of the algorithm by Bille et al.



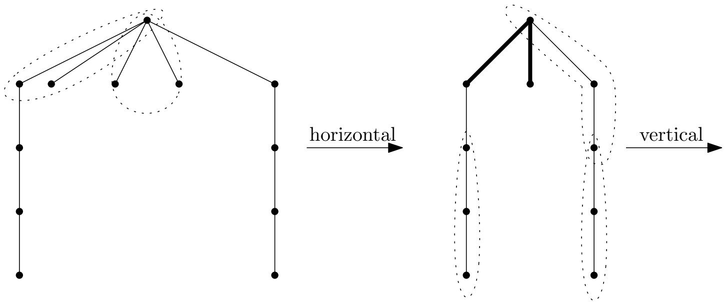
Such iteration decreases the size of the current tree by a constant factor.

Single iteration of the algorithm by Bille et al.



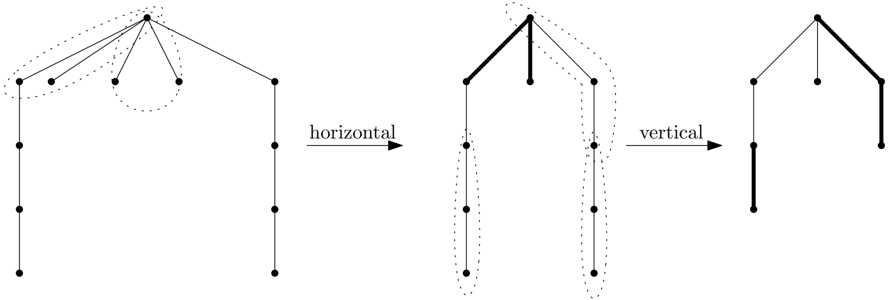
Such iteration decreases the size of the current tree by a constant factor.

Single iteration of the algorithm by Bille et al.



Such iteration decreases the size of the current tree by a constant factor.

Single iteration of the algorithm by Bille et al.



Such iteration decreases the size of the current tree by a constant factor.

A difficult instance

For every k we construct a tree on $n = \Theta(\sigma^{8^k})$ nodes for which the top DAG is of size $\Theta(\frac{n}{\log_\sigma n} \log \log_\sigma n)$.

Let $t = 8^k$. A gadget G_k contains $2^k - 1$ full ternary trees on 3^k leaves, and a path on 8^k nodes.

A difficult instance

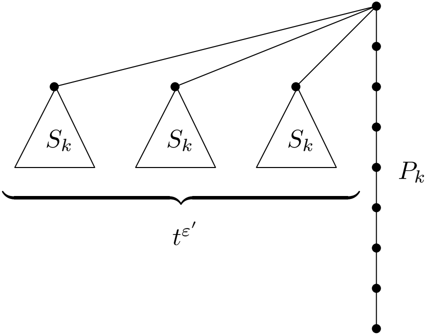
For every k we construct a tree on $n = \Theta(\sigma^{8^k})$ nodes for which the top DAG is of size $\Theta(\frac{n}{\log_\sigma n} \log \log_\sigma n)$.

Let $t = 8^k$. A gadget G_k contains $2^k - 1$ full ternary trees on 3^k leaves, and a path on 8^k nodes.

A difficult instance

For every k we construct a tree on $n = \Theta(\sigma^{8^k})$ nodes for which the top DAG is of size $\Theta(\frac{n}{\log_\sigma n} \log \log_\sigma n)$.

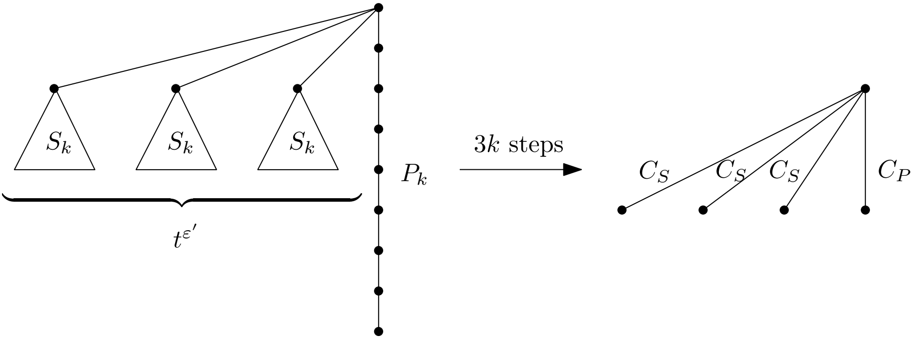
Let $t = 8^k$. A gadget G_k contains $2^k - 1$ full ternary trees on 3^k leaves, and a path on 8^k nodes.



A difficult instance

For every k we construct a tree on $n = \Theta(\sigma^{8^k})$ nodes for which the top DAG is of size $\Theta(\frac{n}{\log_\sigma n} \log\log_\sigma n)$.

Let $t = 8^k$. A gadget G_k contains $2^k - 1$ full ternary trees on 3^k leaves, and a path on 8^k nodes.



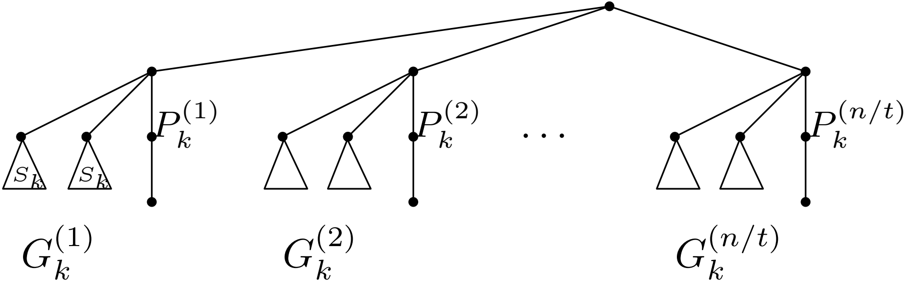
A difficult instance

The whole instance consists of $\Theta(n/t)$ copies of the gadget G_k with the labels of the paths chosen to spell out distinct words of length 8^k.

After the initial $3k$ iterations:

A difficult instance

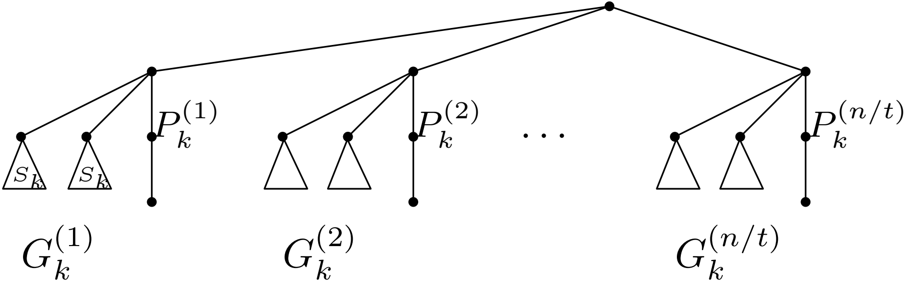
The whole instance consists of $\Theta(n/t)$ copies of the gadget G_k with the labels of the paths chosen to spell out distinct words of length 8^k.



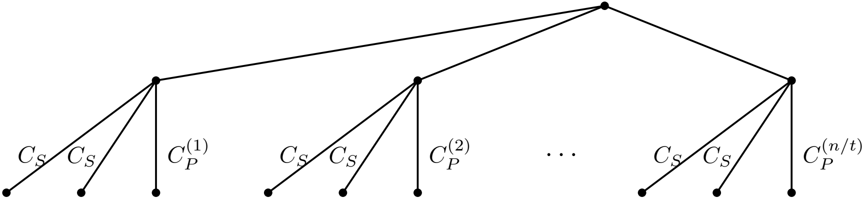
After the initial $3k$ iterations:

A difficult instance

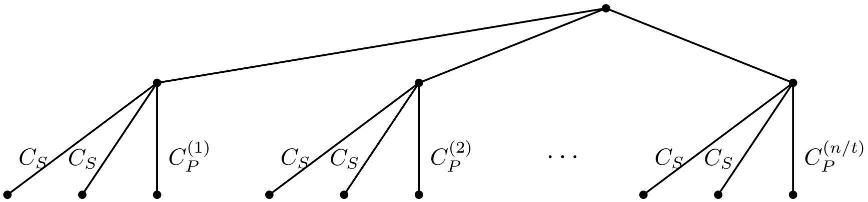
The whole instance consists of $\Theta(n/t)$ copies of the gadget G_k with the labels of the paths chosen to spell out distinct words of length 8^k.



After the initial $3k$ iterations:



A difficult instance

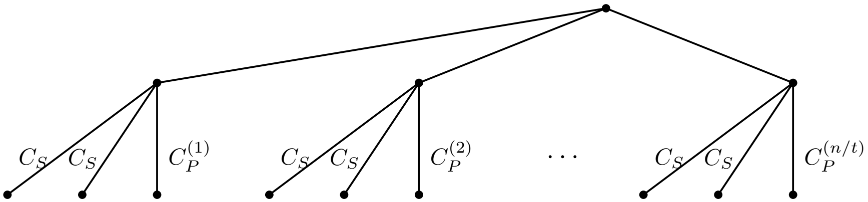


Now each of the n/t distinct clusters $C_P^{(i)}$ is merged with its $2^k - 1$ neighbors C_S in $k = \log t$ iterations and introduces new clusters.
As $t = \log_\sigma n$:

There are $\Omega(n/t \cdot \log t) = \Omega(\frac{n}{\log_\sigma n} \log \log_\sigma n)$ different clusters in the top DAG.

Slightly more complicated construction works for unlabeled trees.

A difficult instance

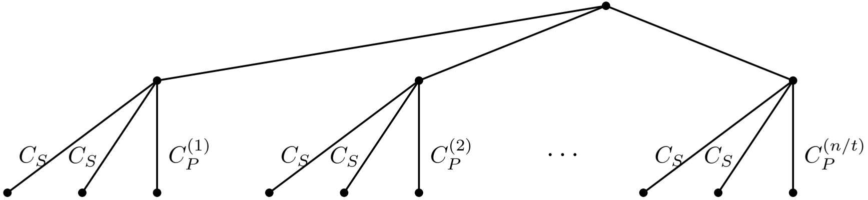


Now each of the n/t distinct clusters $C_P^{(i)}$ is merged with its $2^k - 1$ neighbors C_S in $k = \log t$ iterations and introduces new clusters. As $t = \log_\sigma n$:

There are $\Omega(n/t \cdot \log t) = \Omega(\frac{n}{\log_\sigma n} \log \log_\sigma n)$ different clusters in the top DAG.

Slightly more complicated construction works for unlabeled trees.

A difficult instance



Now each of the n/t distinct clusters $C_P^{(i)}$ is merged with its $2^k - 1$ neighbors C_S in $k = \log t$ iterations and introduces new clusters.
As $t = \log_\sigma n$:

There are $\Omega(n/t \cdot \log t) = \Omega(\frac{n}{\log_\sigma n} \log \log_\sigma n)$ different clusters in the top DAG.

Slightly more complicated construction works for unlabeled trees.

An optimal compression algorithm

Our instance exploits the fact that different parts of the tree are being shrunk with different speeds. Can this be avoided?

Yes!

Simply slow down the compression.

In t-th iteration merge only clusters smaller than α^t for some $\alpha < 2$.

An optimal compression algorithm

Our instance exploits the fact that different parts of the tree are being shrunk with different speeds. Can this be avoided?

Yes!

Simply slow down the compression.

In t-th iteration merge only clusters smaller than α^t for some $\alpha < 2$.

An optimal compression algorithm

Our instance exploits the fact that different parts of the tree are being shrunk with different speeds. Can this be avoided?

Yes!

Simply slow down the compression.

In t-th iteration merge only clusters smaller than α^t for some $\alpha < 2$.

An optimal compression algorithm

```
T̃ := T
initialize leaves of 𝒯 with edges of T
for t = 1, ..., Θ(log n), as long as T̃ is not a single edge do
    list merges that would have been made by one original iteration
    filter out the merges that use a cluster of size bigger than α^t
    modify T̃ and 𝒯 by applying the remaining merges
construct DAG 𝒯𝒟 of 𝒯
```

If we have $m = p + q$ clusters after $t - 1$ iterations, q larger than α^t, then the next iteration decreases this to $7/8m + q$.

After t iterations there are $\mathcal{O}(n/\alpha^{t+1})$ clusters in $\widetilde{T}$, for $\alpha = 10/9$.

By appropriately choosing t and a similar calculation as the one used for SLP we obtain that the size of the top DAG is $\mathcal{O}(n/\log_\sigma n)$.

An optimal compression algorithm

$\widetilde{T} := T$
initialize leaves of $\mathcal{T}$ with edges of T
for $t = 1, \ldots, \Theta(\log n)$, as long as $\widetilde{T}$ is not a single edge **do**
    list merges that would have been made by one original iteration
    filter out the merges that use a cluster of size bigger than α^t
    modify $\widetilde{T}$ and $\mathcal{T}$ by applying the remaining merges
construct DAG $\mathcal{TD}$ of $\mathcal{T}$

If we have $m = p + q$ clusters after $t - 1$ iterations, q larger than α^t, then the next iteration decreases this to $7/8m + q$.

After t iterations there are $\mathcal{O}(n/\alpha^{t+1})$ clusters in $\widetilde{T}$, for $\alpha = 10/9$.

By appropriately choosing t and a similar calculation as the one used for SLP we obtain that the size of the top DAG is $\mathcal{O}(n/\log_\sigma n)$.

An optimal compression algorithm

$\widetilde{T} := T$
initialize leaves of $\mathcal{T}$ with edges of T
for $t = 1, \ldots, \Theta(\log n)$, as long as $\widetilde{T}$ is not a single edge **do**
    list merges that would have been made by one original iteration
    filter out the merges that use a cluster of size bigger than α^t
    modify $\widetilde{T}$ and $\mathcal{T}$ by applying the remaining merges
construct DAG $\mathcal{TD}$ of $\mathcal{T}$

If we have $m = p + q$ clusters after $t - 1$ iterations, q larger than α^t, then the next iteration decreases this to $7/8m + q$.

After t iterations there are $\mathcal{O}(n/\alpha^{t+1})$ clusters in $\widetilde{T}$, for $\alpha = 10/9$.

By appropriately choosing t and a similar calculation as the one used for SLP we obtain that the size of the top DAG is $\mathcal{O}(n/\log_\sigma n)$.

An optimal compression algorithm

$\widetilde{T} := T$
initialize leaves of $\mathcal{T}$ with edges of T
for $t = 1, \ldots, \Theta(\log n)$, as long as $\widetilde{T}$ is not a single edge **do**
    list merges that would have been made by one original iteration
    filter out the merges that use a cluster of size bigger than α^t
    modify $\widetilde{T}$ and $\mathcal{T}$ by applying the remaining merges
construct DAG $\mathcal{TD}$ of $\mathcal{T}$

If we have $m = p + q$ clusters after $t - 1$ iterations, q larger than α^t, then the next iteration decreases this to $7/8m + q$.

After t iterations there are $\mathcal{O}(n/\alpha^{t+1})$ clusters in $\widetilde{T}$, for $\alpha = 10/9$.

By appropriately choosing t and a similar calculation as the one used for SLP we obtain that the size of the top DAG is $\mathcal{O}(n/\log_\sigma n)$.

An optimal compression algorithm

Lohrey, Reh, and Sieber arXiv 2017

Another construction algorithm guaranteeing that the size of the top DAG is $\mathcal{O}(n/\log_\sigma n)$.

(but less “uniform”)

Questions?

An optimal compression algorithm

Lohrey, Reh, and Sieber arXiv 2017

Another construction algorithm guaranteeing that the size of the top DAG is $\mathcal{O}(n/\log_\sigma n)$.

(but less "uniform")

Questions?

An optimal compression algorithm

Lohrey, Reh, and Sieber arXiv 2017

Another construction algorithm guaranteeing that the size of the top DAG is $\mathcal{O}(n/\log_\sigma n)$.

(but less “uniform”)

Questions?