On the Compressibility of Highly Repetitive Sequences

Gonzalo Navarro

CeBiB — Center for Biotechnology and Bioengineering
IMFD — Millennium Institute for Foundational Research on Data
Department of Computer Science, University of Chile

The world is drowning in data! (Jeff Vitter, 2008)

1

Introduction

The world is drowning in data! In recent years, we have been deluged by a torrent of data from a variety of increasingly data-intensive applications, including databases, scientific computations, graphics, entertainment, multimedia, sensors, web applications, and email. NASA's Earth Observing System project, the core part of the Earth Science Enterprise (formerly Mission to Planet Earth), produces petabytes (10^{15} bytes) of raster data per year [148]. A petabyte corresponds roughly to the amount of information in one billion graphically formatted books. The online databases of satellite images used by Microsoft TerraServer (part of MSN Virtual Earth) [325] and Google Earth [180] are multiple terabytes (10^{12} bytes) in size. Wal-Mart's sales data warehouse contains over a half petabyte (500 terabytes) of data. A major challenge is to develop mechanisms for processing the data, or else much of the data will be useless.

The world is drowning in data! (Jeff Vitter, 2008)

1

Introduction

The world is drowning in data! In recent years, we have been deluged by a torrent of data from a variety of increasingly data-intensive applications, including databases, scientific computations, graphics, entertainment, multimedia, sensors, web applications, and email. NASA's Earth Observing System project, the core part of the Earth Science Enterprise (formerly Mission to Planet Earth), produces petabytes (10^{15} bytes) of raster data per year [148]. A petabyte corresponds roughly to the amount of information in one billion graphically formatted books. The online databases of satellite images used by Microsoft TerraServer (part of MSN Virtual Earth) [325] and Google Earth [180] are multiple terabytes (10^{12} bytes) in size. Wal-Mart's sales data warehouse contains over a half petabyte (500 terabytes) of data. A major challenge is to develop mechanisms for processing the data, or else much of the data will be useless.

- We are still drowning in data, but...

The world is drowning in data! (Jeff Vitter, 2008)

1

Introduction

The world is drowning in data! In recent years, we have been deluged by a torrent of data from a variety of increasingly data-intensive applications, including databases, scientific computations, graphics, entertainment, multimedia, sensors, web applications, and email. NASA's Earth Observing System project, the core part of the Earth Science Enterprise (formerly Mission to Planet Earth), produces petabytes (10^{15} bytes) of raster data per year [148]. A petabyte corresponds roughly to the amount of information in one billion graphically formatted books. The online databases of satellite images used by Microsoft TerraServer (part of MSN Virtual Earth) [325] and Google Earth [180] are multiple terabytes (10^{12} bytes) in size. Wal-Mart's sales data warehouse contains over a half petabyte (500 terabytes) of data. A major challenge is to develop mechanisms for processing the data, or else much of the data will be useless.

- We are still drowning in data, but...
- ... are we drowning in information?

Are we drowning in information?

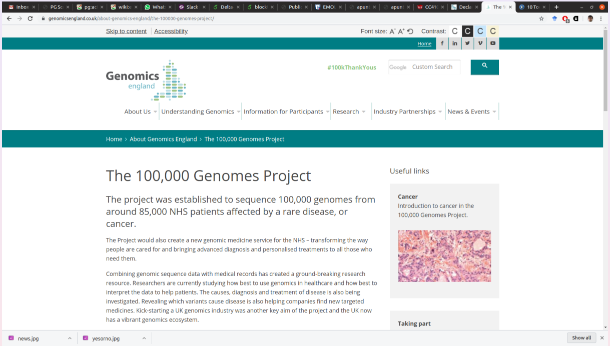
- Much of the fastest-growing data is highly redundant.

Are we drowning in information?

- Much of the fastest-growing data is highly redundant.
- It carries much less information than data.

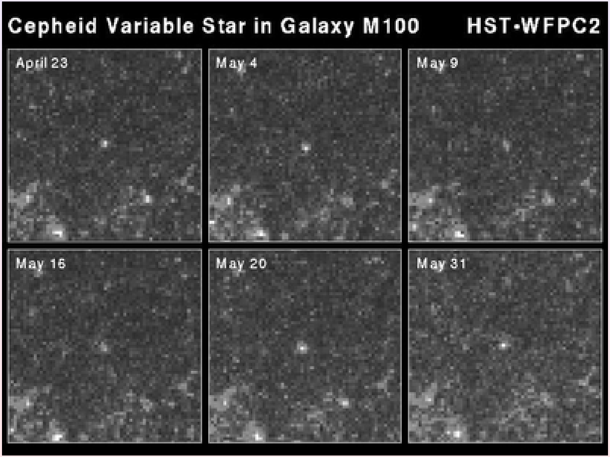
Are we drowning in information?

- Much of the fastest-growing data is highly redundant.
- It carries much less information than data.
- Myriad genomes of the same species.



Are we drowning in information?

- Much of the fastest-growing data is highly redundant.
- It carries much less information than data.
- Periodic sky surveys.

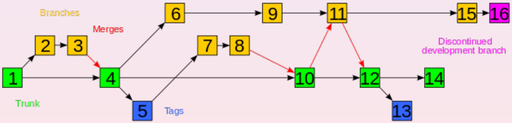


Are we drowning in information?

- Much of the fastest-growing data is highly redundant.
- It carries much less information than data.
- Time-versioned collections.

Are we drowning in information?

- Much of the fastest-growing data is highly redundant.
- It carries much less information than data.
- Tree-versioned collections.



Some numbers

- Two human genomes differ by about 0.1%.

Some numbers

- Two human genomes differ by about 0.1%.
  - Typically SNPs, more rarely block edits.

Some numbers

- Two human genomes differ by about 0.1%.
  - Typically SNPs, more rarely block edits.
- There are about 20 versions per major release in GitHub.

Some numbers

- Two human genomes differ by about 0.1%.
  - Typically SNPs, more rarely block edits.
- There are about 20 versions per major release in GitHub.
  - Ratio of “commit” over “create”.

Some numbers

- Two human genomes differ by about 0.1%.
  - Typically SNPs, more rarely block edits.
- There are about 20 versions per major release in GitHub.
  - Ratio of “commit” over “create”.
- There are about 20 versions per article in Wikipedia.

Some numbers

- Two human genomes differ by about 0.1%.
  - Typically SNPs, more rarely block edits.
- There are about 20 versions per major release in GitHub.
  - Ratio of “commit” over “create”.
- There are about 20 versions per article in Wikipedia.
  - And versions grow faster than new articles.

Some numbers

- Two human genomes differ by about 0.1%.
  - Typically SNPs, more rarely block edits.
- There are about 20 versions per major release in GitHub.
  - Ratio of “commit” over “create”.
- There are about 20 versions per article in Wikipedia.
  - And versions grow faster than new articles.
- 100-to-1 compression in Wikipedia and 1000-Genomes

Some numbers

- Two human genomes differ by about 0.1%.
  - Typically SNPs, more rarely block edits.
- There are about 20 versions per major release in GitHub.
  - Ratio of “commit” over “create”.
- There are about 20 versions per article in Wikipedia.
  - And versions grow faster than new articles.
- 100-to-1 compression in Wikipedia and 1000-Genomes
  - Using Lempel-Ziv compression.

Our focus

We will focus on sequence data, and on the following questions:

- How to best measure the entropy, or amount of information, of an individual text $T[1..n]$?

Our focus

We will focus on sequence data, and on the following questions:

- How to best measure the entropy, or amount of information, of an individual text $T[1..n]$?
- Can a text $T[1..n]$ be stored in space close to its amount of information?

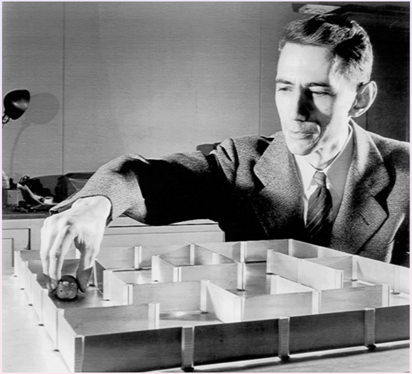
Our focus

We will focus on sequence data, and on the following questions:

- How to best measure the entropy, or amount of information, of an individual text $T[1..n]$?
- Can a text $T[1..n]$ be stored in space close to its amount of information?
- Can we access the text efficiently within that space?

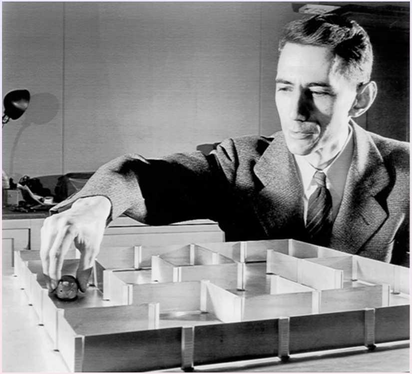
Shannon's entropy?

- Shannon's entropy has been immensely successful to measure amount of information depending on frequencies.



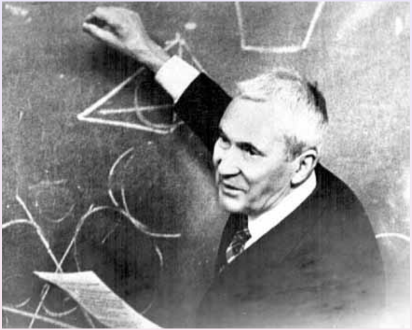
Shannon's entropy?

- Shannon's entropy has been immensely successful to measure amount of information depending on frequencies.
- But it is useless to capture repetitiveness, $H(T \cdot T) \approx H(T)$.



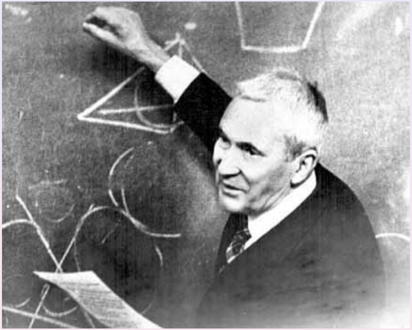
Kolmogorov's entropy?

- Kolmogorov's entropy is the size of the smallest program outputting the text.



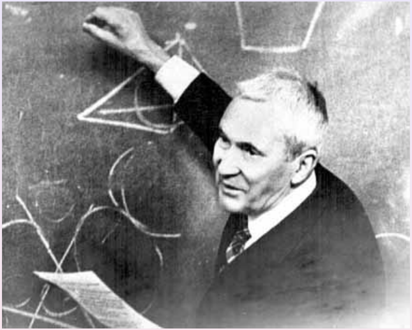
Kolmogorov's entropy?

- Kolmogorov's entropy is the size of the smallest program outputting the text.
- It would be adequate, but... it is uncomputable.



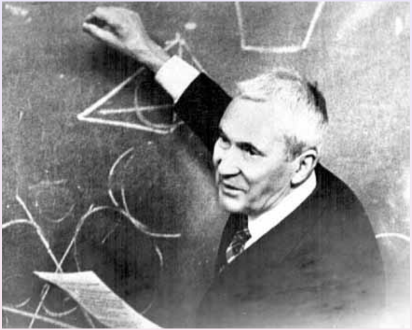
Kolmogorov's entropy?

- Kolmogorov's entropy is the size of the smallest program outputting the text.
- It would be adequate, but... it is uncomputable.
- It is also too general, not just about repetitiveness.



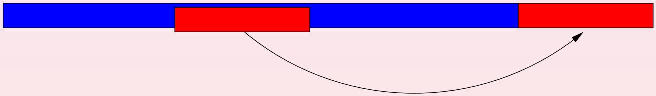
Kolmogorov's entropy?

- Kolmogorov's entropy is the size of the smallest program outputting the text.
- It would be adequate, but... it is uncomputable.
- It is also too general, not just about repetitiveness.
- Ad-hoc measures from dictionary compression are used as gold standards.



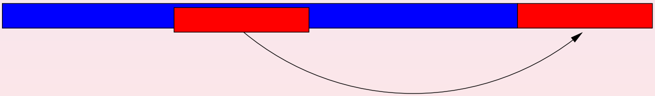
Lempel-Ziv complexity

- In 1976, Lempel and Ziv proposed the following measure.



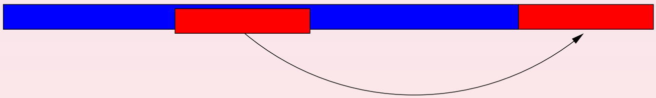
Lempel-Ziv complexity

- In 1976, Lempel and Ziv proposed the following measure.
  - We start at the beginning of the text $T[1..n]$, $i = 0$.



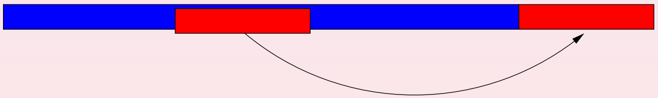
Lempel-Ziv complexity

- In 1976, Lempel and Ziv proposed the following measure.
  - We start at the beginning of the text $T[1..n]$, $i = 0$.
  - We advance as much as possible, $T[i+1..]$, as long as there is a previous occurrence of $T[i+1..]$.



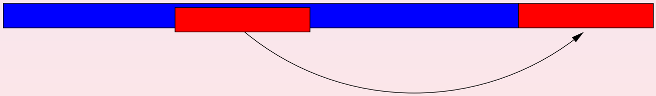
Lempel-Ziv complexity

- In 1976, Lempel and Ziv proposed the following measure.
  - We start at the beginning of the text $T[1..n]$, $i = 0$.
  - We advance as much as possible, $T[i+1..]$, as long as there is a previous occurrence of $T[i+1..]$.
  - If we can advance until $T[i+1..j]$ (which occurs in $T[s..r]$) and fail with $T[j+1]$ then $T[i+1..j+1]$ is a phrase.



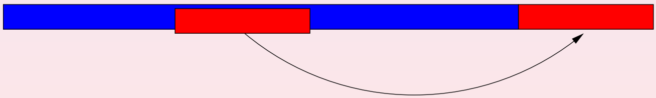
Lempel-Ziv complexity

- In 1976, Lempel and Ziv proposed the following measure.
  - We start at the beginning of the text $T[1..n]$, $i = 0$.
  - We advance as much as possible, $T[i+1..]$, as long as there is a previous occurrence of $T[i+1..]$.
  - If we can advance until $T[i+1..j]$ (which occurs in $T[s..r]$) and fail with $T[j+1]$ then $T[i+1..j+1]$ is a phrase.
  - We encode the phrase as $(r, j-i, T[j+1])$ and continue from $i = j+2$.



Lempel-Ziv complexity

- In 1976, Lempel and Ziv proposed the following measure.
  - We start at the beginning of the text $T[1..n]$, $i = 0$.
  - We advance as much as possible, $T[i+1..]$, as long as there is a previous occurrence of $T[i+1..]$.
  - If we can advance until $T[i+1..j]$ (which occurs in $T[s..r]$) and fail with $T[j+1]$ then $T[i+1..j+1]$ is a phrase.
  - We encode the phrase as $(r, j-i, T[j+1])$ and continue from $i = j+2$.
- The number z of phrases is the Lempel-Ziv complexity.



Lempel-Ziv complexity

a l a b a r a l a l a b a r d a $

Output: $(0, 0, a)$

Lempel-Ziv complexity

a l a b a r a l a l a b a r d a $

Output: $(0, 0, a)$ $(0, 0, l)$

Lempel-Ziv complexity

a l a b a r a l a l a b a r d a $

Output: $(0, 0, a)$ $(0, 0, l)$ $(1, 1, b)$

Lempel-Ziv complexity

a l a b a r a l a l a b a r d a $

Output: $(0, 0, a)$ $(0, 0, l)$ $(1, 1, b)$ $(1, 1, r)$

Lempel-Ziv complexity

a l a b a r a l a l a b a r d a $

Output: $(0, 0, a)$ $(0, 0, l)$ $(1, 1, b)$ $(1, 1, r)$ $(1, 3, l)$

Lempel-Ziv complexity

a l a b a r a l a l a b a r d a $

Output: $(0, 0, a)$ $(0, 0, l)$ $(1, 1, b)$ $(1, 1, r)$ $(1, 3, l)$ $(3, 4, d)$

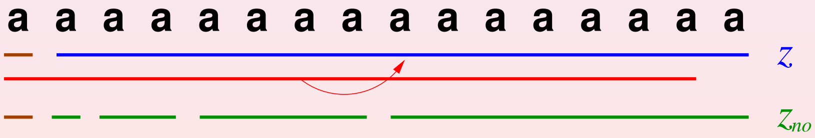
Lempel-Ziv complexity

a l a b a r a l a l a b a r d a $

Output: $(0, 0, a)$ $(0, 0, l)$ $(1, 1, b)$ $(1, 1, r)$ $(1, 3, l)$ $(3, 4, d)$ $(1, 1, \$)$

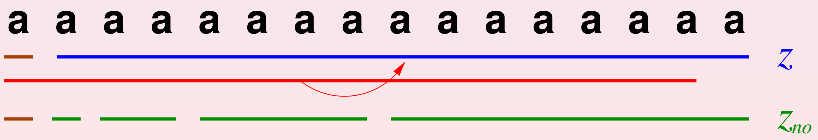
Lempel-Ziv complexity

- Two flavors, actually:



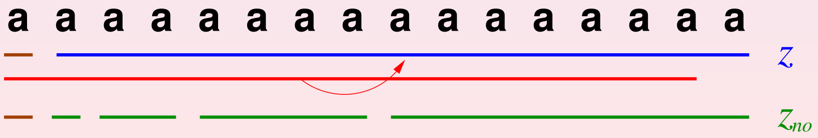
Lempel-Ziv complexity

- Two flavors, actually:
  - If the source must finish before the target starts, z_{no} [Farach & Thorup 1995].



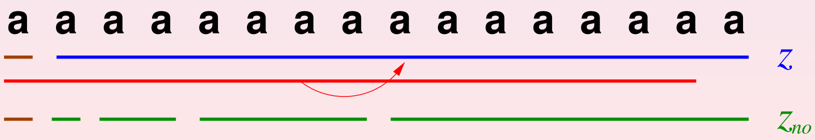
Lempel-Ziv complexity

- Two flavors, actually:
  - If the source must finish before the target starts, z_{no} [Farach & Thorup 1995].
  - If the source only must start before the source, z (the original).



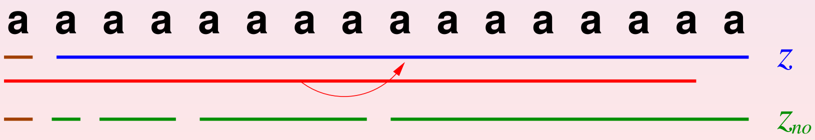
Lempel-Ziv complexity

- Two flavors, actually:
  - If the source must finish before the target starts, z_{no} [Farach & Thorup 1995].
  - If the source only must start before the source, z (the original).
  - The latter permits self-reference.



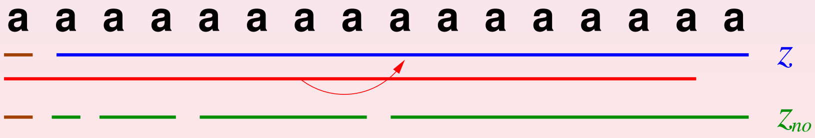
Lempel-Ziv complexity

- Two flavors, actually:
  - If the source must finish before the target starts, z_{no} [Farach & Thorup 1995].
  - If the source only must start before the source, z (the original).
  - The latter permits self-reference.
  - Both left-to-right greedy parsings are optimal, so $z \leq z_{no}$.



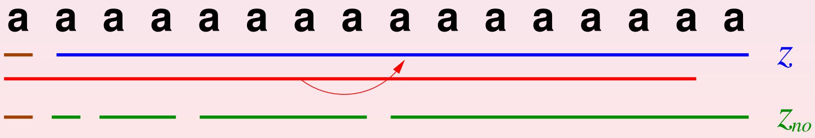
Lempel-Ziv complexity

- Two flavors, actually:
  - If the source must finish before the target starts, z_{no} [Farach & Thorup 1995].
  - If the source only must start before the source, z (the original).
  - The latter permits self-reference.
  - Both left-to-right greedy parsings are optimal, so $z \leq z_{no}$.
  - In some families, $z_{no} = \Omega(z \log n)$, e.g., $T = a^n$.



Lempel-Ziv complexity

- Two flavors, actually:
  - If the source must finish before the target starts, z_{no} [Farach & Thorup 1995].
  - If the source only must start before the source, z (the original).
  - The latter permits self-reference.
  - Both left-to-right greedy parsings are optimal, so $z \leq z_{no}$.
  - In some families, $z_{no} = \Omega(z \log n)$, e.g., $T = a^n$.
- The base of practical compressors like LZ77 and LZ78, with immense success.



Lempel-Ziv complexity

Good properties

- It converges to Shannon's entropy on ergodic sources.

Lempel-Ziv complexity

Good properties

- It converges to Shannon's entropy on ergodic sources.
- It holds $z_{no} = O(n/\log_\sigma n)$, so space is $O(n \log \sigma)$ bits.

Lempel-Ziv complexity

Good properties

- It converges to Shannon's entropy on ergodic sources.
- It holds $z_{no} = O(n/\log_\sigma n)$, so space is $O(n \log \sigma)$ bits.
- It can be computed in $O(n)$ time for both z_{no} [Rodeh, Pratt, Even 1981] and z [Crochemore et al. 2012].

Lempel-Ziv complexity

Good properties

- It converges to Shannon's entropy on ergodic sources.
- It holds $z_{no} = O(n/\log_\sigma n)$, so space is $O(n \log \sigma)$ bits.
- It can be computed in $O(n)$ time for both z_{no} [Rodeh, Pratt, Even 1981] and z [Crochemore et al. 2012].
- It performs very well on repetitive sequences.

Lempel-Ziv complexity

Good properties

- It converges to Shannon's entropy on ergodic sources.
- It holds $z_{no} = O(n/\log_\sigma n)$, so space is $O(n \log \sigma)$ bits.
- It can be computed in $O(n)$ time for both z_{no} [Rodeh, Pratt, Even 1981] and z [Crochemore et al. 2012].
- It performs very well on repetitive sequences.
- Measure z is taken as a gold standard.

Lempel-Ziv complexity

Good properties

- It converges to Shannon's entropy on ergodic sources.
- It holds $z_{no} = O(n/\log_\sigma n)$, so space is $O(n \log \sigma)$ bits.
- It can be computed in $O(n)$ time for both z_{no} [Rodeh, Pratt, Even 1981] and z [Crochemore et al. 2012].
- It performs very well on repetitive sequences.
- Measure z is taken as a gold standard.

Lempel-Ziv complexity

Good properties

- It converges to Shannon's entropy on ergodic sources.
- It holds $z_{no} = O(n/\log_\sigma n)$, so space is $O(n \log \sigma)$ bits.
- It can be computed in $O(n)$ time for both z_{no} [Rodeh, Pratt, Even 1981] and z [Crochemore et al. 2012].
- It performs very well on repetitive sequences.
- Measure z is taken as a gold standard.

Bad properties

- It is not robust, e.g., it changes if we reverse T and may decrease upon appends on T.

Lempel-Ziv complexity

Good properties

- It converges to Shannon's entropy on ergodic sources.
- It holds $z_{no} = O(n/\log_\sigma n)$, so space is $O(n \log \sigma)$ bits.
- It can be computed in $O(n)$ time for both z_{no} [Rodeh, Pratt, Even 1981] and z [Crochemore et al. 2012].
- It performs very well on repetitive sequences.
- Measure z is taken as a gold standard.

Bad properties

- It is not robust, e.g., it changes if we reverse T and may decrease upon appends on T.
- It is not known how to access $T[i]$ within $O(z_{no})$ space.

Lempel-Ziv complexity

Good properties

- It converges to Shannon's entropy on ergodic sources.
- It holds $z_{no} = O(n/\log_\sigma n)$, so space is $O(n \log \sigma)$ bits.
- It can be computed in $O(n)$ time for both z_{no} [Rodeh, Pratt, Even 1981] and z [Crochemore et al. 2012].
- It performs very well on repetitive sequences.
- Measure z is taken as a gold standard.

Bad properties

- It is not robust, e.g., it changes if we reverse T and may decrease upon appends on T.
- It is not known how to access $T[i]$ within $O(z_{no})$ space.
- It may double upon a single character edit on T [Akagi, Funakoshi, Inenaga 2022].

Enabling access: LZ-End [Kreft & N. 2013]

- It requires that the source ends at a phrase boundary.

Enabling access: LZ-End [Kreft & N. 2013]

- It requires that the source ends at a phrase boundary.
- This time a greedy parsing does not yield the smallest parse.

Enabling access: LZ-End [Kreft & N. 2013]

- It requires that the source ends at a phrase boundary.
- This time a greedy parsing does not yield the smallest parse.
- The optimal parsing produces $z_{end} \geq z_{no}$ phrases.

Enabling access: LZ-End [Kreft & N. 2013]

- It requires that the source ends at a phrase boundary.
- This time a greedy parsing does not yield the smallest parse.
- The optimal parsing produces $z_{end} \geq z_{no}$ phrases.
- The greedy parsing produces $z_e \geq z_{end}$ phrases

Enabling access: LZ-End [Kreft & N. 2013]

- It requires that the source ends at a phrase boundary.
- This time a greedy parsing does not yield the smallest parse.
- The optimal parsing produces $z_{end} \geq z_{no}$ phrases.
- The greedy parsing produces $z_e \geq z_{end}$ phrases
  - $z_e = O(n/\log_\sigma n)$

Enabling access: LZ-End [Kreft & N. 2013]

- It requires that the source ends at a phrase boundary.
- This time a greedy parsing does not yield the smallest parse.
- The optimal parsing produces $z_{end} \geq z_{no}$ phrases.
- The greedy parsing produces $z_e \geq z_{end}$ phrases
  - $z_e = O(n/\log_\sigma n)$
  - $z_e = O(z \log^2(n/z))$ [Kempa & Saha 2022]

Enabling access: LZ-End [Kreft & N. 2013]

- It requires that the source ends at a phrase boundary.
- This time a greedy parsing does not yield the smallest parse.
- The optimal parsing produces $z_{end} \geq z_{no}$ phrases.
- The greedy parsing produces $z_e \geq z_{end}$ phrases
  - $z_e = O(n/\log_\sigma n)$
  - $z_e = O(z \log^2(n/z))$ [Kempa & Saha 2022]
- The greedy parsing can be computed in $O(n)$ time [Kempa & Kosolobov 2017].

Enabling access: LZ-End [Kreft & N. 2013]

- It requires that the source ends at a phrase boundary.
- This time a greedy parsing does not yield the smallest parse.
- The optimal parsing produces $z_{end} \geq z_{no}$ phrases.
- The greedy parsing produces $z_e \geq z_{end}$ phrases
  - $z_e = O(n/\log_\sigma n)$
  - $z_e = O(z \log^2(n/z))$ [Kempa & Saha 2022]
- The greedy parsing can be computed in $O(n)$ time [Kempa & Kosolobov 2017].
- Any LZ-End parse enables accessing an individual symbol in time $O(\log^5 n)$ [Kempa & Saha 2022].

Enabling access: LZ-End [Kreft & N. 2013]

- It requires that the source ends at a phrase boundary.
- This time a greedy parsing does not yield the smallest parse.
- The optimal parsing produces $z_{end} \geq z_{no}$ phrases.
- The greedy parsing produces $z_e \geq z_{end}$ phrases
  - $z_e = O(n/\log_\sigma n)$
  - $z_e = O(z \log^2(n/z))$ [Kempa & Saha 2022]
- The greedy parsing can be computed in $O(n)$ time [Kempa & Kosolobov 2017].
- Any LZ-End parse enables accessing an individual symbol in time $O(\log^5 n)$ [Kempa & Saha 2022].
- z_e is reasonably close to z in practice.

Enabling access: LZ-End [Kreft & N. 2013]

a l a b a r a l a l a b a r d a $

$z = 7$

$z_{no} = 7$

a l a b a r a l a l a b a r d a $

$z_e = 7$

$z_{end} = 7$

Bidirectional macro schemes

- In 1982, Storer and Szymanski proposed a more principled measure:

Bidirectional macro schemes

- In 1982, Storer and Szymanski proposed a more principled measure:
  - The text is parsed into phrases as in Lempel-Ziv.

Bidirectional macro schemes

- In 1982, Storer and Szymanski proposed a more principled measure:
  - The text is parsed into phrases as in Lempel-Ziv.
  - But their sources can be forwards or backwards in T.

Bidirectional macro schemes

- In 1982, Storer and Szymanski proposed a more principled measure:
  - The text is parsed into phrases as in Lempel-Ziv.
  - But their sources can be forwards or backwards in T.
  - As long as no cycles are introduced for individual positions.

Bidirectional macro schemes

- In 1982, Storer and Szymanski proposed a more principled measure:
  - The text is parsed into phrases as in Lempel-Ziv.
  - But their sources can be forwards or backwards in T.
  - As long as no cycles are introduced for individual positions.
  - Explicit symbols are also permitted.

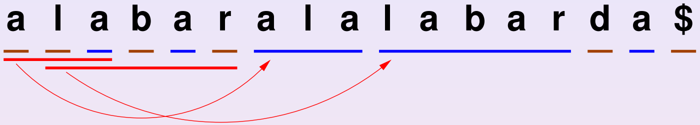
Bidirectional macro schemes

- In 1982, Storer and Szymanski proposed a more principled measure:
  - The text is parsed into phrases as in Lempel-Ziv.
  - But their sources can be forwards or backwards in T.
  - As long as no cycles are introduced for individual positions.
  - Explicit symbols are also permitted.
- The associated measure is b, the least number of phrases one can achieve.

Bidirectional macro schemes

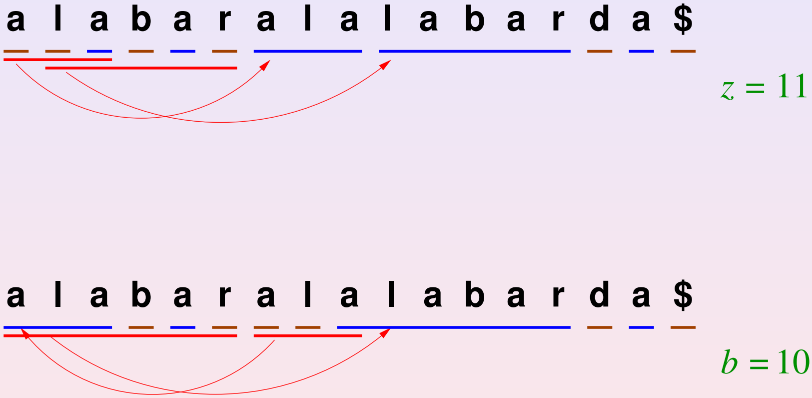
- In 1982, Storer and Szymanski proposed a more principled measure:
  - The text is parsed into phrases as in Lempel-Ziv.
  - But their sources can be forwards or backwards in T.
  - As long as no cycles are introduced for individual positions.
  - Explicit symbols are also permitted.
- The associated measure is b, the least number of phrases one can achieve.
- Never reached popularity, though.

Bidirectional macro schemes



$z = 11$

Bidirectional macro schemes



Bidirectional macro schemes

Relation with z

- Obviously it holds $b \leq z$ for every text.

Bidirectional macro schemes

Relation with z

- Obviously it holds $b \leq z$ for every text.
- It holds $z = O(b \log(n/b))$ [N., Ochoa, Prezza 2021].

Bidirectional macro schemes

Relation with z

- Obviously it holds $b \leq z$ for every text.
- It holds $z = O(b \log(n/b))$ [N., Ochoa, Prezza 2021].
  - By using locally consistent parsing on top of the macro scheme.

Bidirectional macro schemes

Relation with z

- Obviously it holds $b \leq z$ for every text.
- It holds $z = O(b \log(n/b))$ [N., Ochoa, Prezza 2021].
  - By using locally consistent parsing on top of the macro scheme.
  - Such a parsing cuts the text so that identical substrings are largely parsed in the same way.

Bidirectional macro schemes

Relation with z

- Obviously it holds $b \leq z$ for every text.
- It holds $z = O(b \log(n/b))$ [N., Ochoa, Prezza 2021].
  - By using locally consistent parsing on top of the macro scheme.
  - Such a parsing cuts the text so that identical substrings are largely parsed in the same way.
  - The resulting chunks are processed in successive rounds.

Bidirectional macro schemes

Relation with z

- Obviously it holds $b \leq z$ for every text.
- It holds $z = O(b \log(n/b))$ [N., Ochoa, Prezza 2021].
  - By using locally consistent parsing on top of the macro scheme.
  - Such a parsing cuts the text so that identical substrings are largely parsed in the same way.
  - The resulting chunks are processed in successive rounds.
  - More details later.

Bidirectional macro schemes

Relation with z

- Obviously it holds $b \leq z$ for every text.
- It holds $z = O(b \log(n/b))$ [N., Ochoa, Prezza 2021].
  - By using locally consistent parsing on top of the macro scheme.
  - Such a parsing cuts the text so that identical substrings are largely parsed in the same way.
  - The resulting chunks are processed in successive rounds.
  - More details later.
- For some families, $z = \Omega(b \log n)$ [Gagie, N., Prezza 2018].

Bidirectional macro schemes

Relation with z

- Obviously it holds $b \leq z$ for every text.
- It holds $z = O(b \log(n/b))$ [N., Ochoa, Prezza 2021].
  - By using locally consistent parsing on top of the macro scheme.
  - Such a parsing cuts the text so that identical substrings are largely parsed in the same way.
  - The resulting chunks are processed in successive rounds.
  - More details later.
- For some families, $z = \Omega(b \log n)$ [Gagie, N., Prezza 2018].
  - E.g., Fibonacci words, $F_1 = b$, $F_2 = a$, $F_k = F_{k-1}F_{k-2}$.

Bidirectional macro schemes

Relation with z

- Obviously it holds $b \leq z$ for every text.
- It holds $z = O(b \log(n/b))$ [N., Ochoa, Prezza 2021].
  - By using locally consistent parsing on top of the macro scheme.
  - Such a parsing cuts the text so that identical substrings are largely parsed in the same way.
  - The resulting chunks are processed in successive rounds.
  - More details later.
- For some families, $z = \Omega(b \log n)$ [Gagie, N., Prezza 2018].
  - E.g., Fibonacci words, $F_1 = b$, $F_2 = a$, $F_k = F_{k-1} F_{k-2}$.
  - b, a, ab, aba, $abaab$, $abaababa$, etc.

Bidirectional macro schemes

Relation with z

- Obviously it holds $b \leq z$ for every text.
- It holds $z = O(b \log(n/b))$ [N., Ochoa, Prezza 2021].
  - By using locally consistent parsing on top of the macro scheme.
  - Such a parsing cuts the text so that identical substrings are largely parsed in the same way.
  - The resulting chunks are processed in successive rounds.
  - More details later.
- For some families, $z = \Omega(b \log n)$ [Gagie, N., Prezza 2018].
  - E.g., Fibonacci words, $F_1 = b$, $F_2 = a$, $F_k = F_{k-1} F_{k-2}$.
  - b, a, ab, aba, $abaab$, $abaababa$, etc.
  - The family has $b = O(1)$ and $z = \Theta(\log n)$.

Bidirectional macro schemes

Good properties

- It is never worse than z.

Bidirectional macro schemes

Good properties

- It is never worse than z.
- The text can still be encoded within $O(b \log n)$ bits.

Bidirectional macro schemes

Good properties

- It is never worse than z.
- The text can still be encoded within $O(b \log n)$ bits.
- It is more robust, e.g., b is the same if we reverse T.

Bidirectional macro schemes

Good properties

- It is never worse than z.
- The text can still be encoded within $O(b \log n)$ bits.
- It is more robust, e.g., b is the same if we reverse T.

Bidirectional macro schemes

Good properties

- It is never worse than z.
- The text can still be encoded within $O(b \log n)$ bits.
- It is more robust, e.g., b is the same if we reverse T.

Bad properties

- Still non-monotonic upon symbol appends [Ferragina & Tosoni 2021].

Bidirectional macro schemes

Good properties

- It is never worse than z.
- The text can still be encoded within $O(b \log n)$ bits.
- It is more robust, e.g., b is the same if we reverse T.

Bad properties

- Still non-monotonic upon symbol appends [Ferragina & Tosoni 2021].
- It may grow by 50% upon a single character edit on T [Akagi, Funakoshi, Inenaga 2022].

Bidirectional macro schemes

Good properties

- It is never worse than z.
- The text can still be encoded within $O(b \log n)$ bits.
- It is more robust, e.g., b is the same if we reverse T.

Bad properties

- Still non-monotonic upon symbol appends [Ferragina & Tosoni 2021].
- It may grow by 50% upon a single character edit on T [Akagi, Funakoshi, Inenaga 2022].
- It is NP-hard to compute b [Gallant 1982].

Bidirectional macro schemes

Good properties

- It is never worse than z.
- The text can still be encoded within $O(b \log n)$ bits.
- It is more robust, e.g., b is the same if we reverse T.

Bad properties

- Still non-monotonic upon symbol appends [Ferragina & Tosoni 2021].
- It may grow by 50% upon a single character edit on T [Akagi, Funakoshi, Inenaga 2022].
- It is NP-hard to compute b [Gallant 1982].
- No interesting approximations except for z.

Grammar compression

- In 2000, Kieffer and Yang proposed to find a small grammar that generates T and only T.

Grammar compression

- In 2000, Kieffer and Yang proposed to find a small grammar that generates T and only T.
- The size g of the smallest such grammar is then a new measure of compressibility.

Grammar compression

- In 2000, Kieffer and Yang proposed to find a small grammar that generates T and only T.
- The size g of the smallest such grammar is then a new measure of compressibility.
- The smallest grammar size converges to Shannon's entropy too.

Grammar compression

- In 2000, Kieffer and Yang proposed to find a small grammar that generates T and only T.
- The size g of the smallest such grammar is then a new measure of compressibility.
- The smallest grammar size converges to Shannon's entropy too.
- Finding the smallest grammar is NP-hard, however [Charikar et al. 2005].

Grammar compression

- In 2000, Kieffer and Yang proposed to find a small grammar that generates T and only T.
- The size g of the smallest such grammar is then a new measure of compressibility.
- The smallest grammar size converges to Shannon's entropy too.
- Finding the smallest grammar is NP-hard, however [Charikar et al. 2005].
- Not so unpopular because decent heuristics exist, in fact preceding the formalization:

Grammar compression

- In 2000, Kieffer and Yang proposed to find a small grammar that generates T and only T.
- The size g of the smallest such grammar is then a new measure of compressibility.
- The smallest grammar size converges to Shannon's entropy too.
- Finding the smallest grammar is NP-hard, however [Charikar et al. 2005].
- Not so unpopular because decent heuristics exist, in fact preceding the formalization:
  - [Nakamura & Murashima 1996].

Grammar compression

- In 2000, Kieffer and Yang proposed to find a small grammar that generates T and only T.
- The size g of the smallest such grammar is then a new measure of compressibility.
- The smallest grammar size converges to Shannon's entropy too.
- Finding the smallest grammar is NP-hard, however [Charikar et al. 2005].
- Not so unpopular because decent heuristics exist, in fact preceding the formalization:
  - [Nakamura & Murashima 1996].
  - Byte-Pair Encoding [Manber 1997]

Grammar compression

- In 2000, Kieffer and Yang proposed to find a small grammar that generates T and only T.
- The size g of the smallest such grammar is then a new measure of compressibility.
- The smallest grammar size converges to Shannon's entropy too.
- Finding the smallest grammar is NP-hard, however [Charikar et al. 2005].
- Not so unpopular because decent heuristics exist, in fact preceding the formalization:
  - [Nakamura & Murashima 1996].
  - Byte-Pair Encoding [Manber 1997]
  - Sequitur [Nevill-Manning & Witten 1997].

Grammar compression

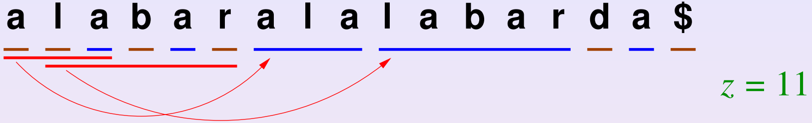
- In 2000, Kieffer and Yang proposed to find a small grammar that generates T and only T.
- The size g of the smallest such grammar is then a new measure of compressibility.
- The smallest grammar size converges to Shannon's entropy too.
- Finding the smallest grammar is NP-hard, however [Charikar et al. 2005].
- Not so unpopular because decent heuristics exist, in fact preceding the formalization:
  - [Nakamura & Murashima 1996].
  - Byte-Pair Encoding [Manber 1997]
  - Sequitur [Nevill-Manning & Witten 1997].
  - RePair [Larsson & Moffat 1999].

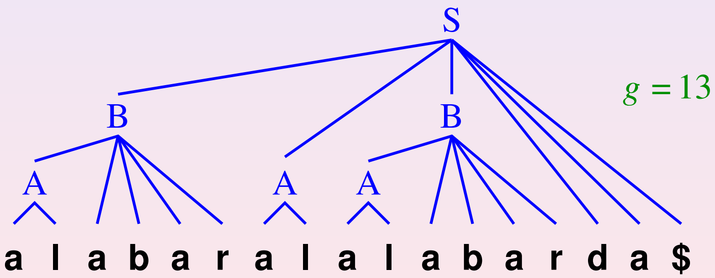
Grammar compression

a l a b a r a l a l a b a r d a $

$z = 11$

Grammar compression



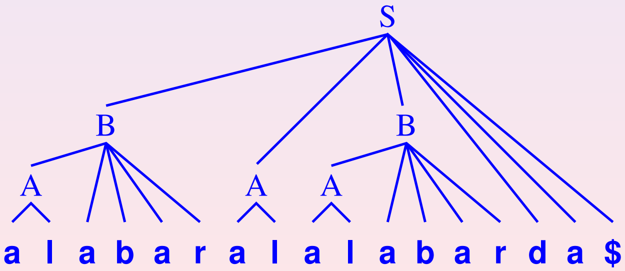


A → a l   B → A a b a r   S → B A B d a $

The relation between grammars and Lempel-Ziv

- Rytter [2003] and Charikar et al. [2005].

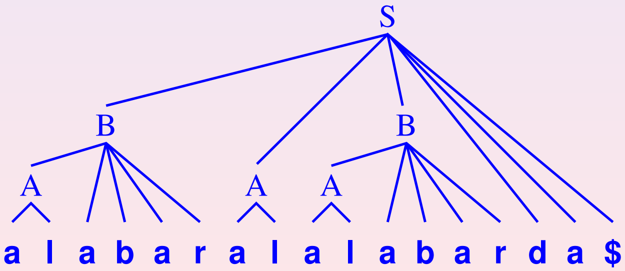
$A \rightarrow$ **a l** $\quad B \rightarrow A$ **a b a r** $\quad S \rightarrow B\ A\ B$ **d a $**



The relation between grammars and Lempel-Ziv

- Rytter [2003] and Charikar et al. [2005].
- They show that $z_{no} = O(g)$.

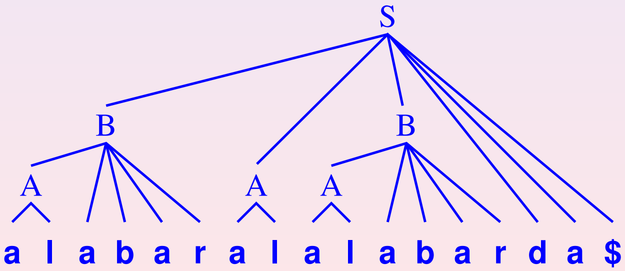
$A \rightarrow \textbf{a l} \quad B \rightarrow A\ \textbf{a b a r} \quad S \rightarrow B\ A\ B\ \textbf{d a \$}$



The relation between grammars and Lempel-Ziv

- Rytter [2003] and Charikar et al. [2005].
- They show that $z_{no} = O(g)$.
  - By creating a left-to-right parse from the grammar tree.

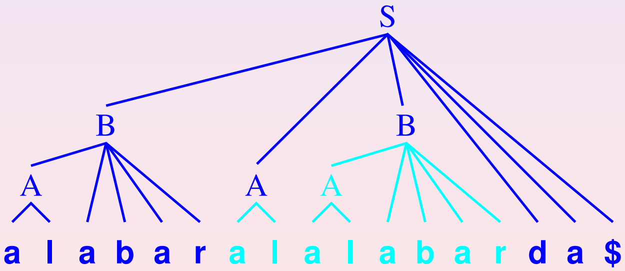
$A \rightarrow$ **a l** $\quad B \rightarrow A$ **a b a r** $\quad S \rightarrow B\ A\ B$ **d a $**



The relation between grammars and Lempel-Ziv

- Rytter [2003] and Charikar et al. [2005].
- They show that $z_{no} = O(g)$.
  - By creating a left-to-right parse from the grammar tree.
- The same proof shows that $z_{end} \leq O(g)$.

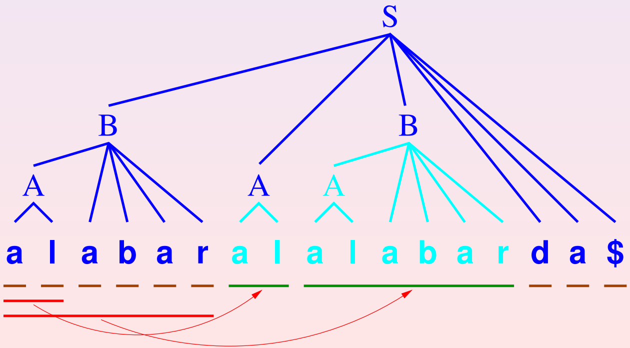
$$A \rightarrow \mathbf{a}\ \mathbf{l} \quad B \rightarrow A\ \mathbf{a}\ \mathbf{b}\ \mathbf{a}\ \mathbf{r} \quad S \rightarrow B\ A\ B\ \mathbf{d}\ \mathbf{a}\ \mathbf{\$}$$



The relation between grammars and Lempel-Ziv

- Rytter [2003] and Charikar et al. [2005].
- They show that $z_{no} = O(g)$.
  - By creating a left-to-right parse from the grammar tree.
- The same proof shows that $z_{end} \leq O(g)$.

$A \rightarrow$ **a l** $B \rightarrow A$ **a b a r** $S \rightarrow B\ A\ B$ **d a $**



The relation between grammars and Lempel-Ziv

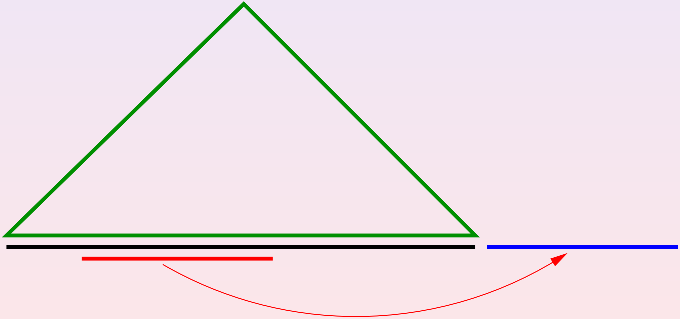
- They also show that $g = O(z_{no} \log(n/z_{no}))$.

The relation between grammars and Lempel-Ziv

- They also show that $g = O(z_{no} \log(n/z_{no}))$.
  - By creating a grammar from a Lempel-Ziv parse.

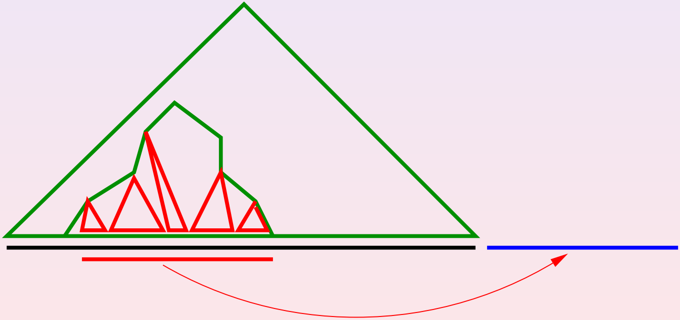
The relation between grammars and Lempel-Ziv

- They also show that $g = O(z_{no} \log(n/z_{no}))$.
  - By creating a grammar from a Lempel-Ziv parse.



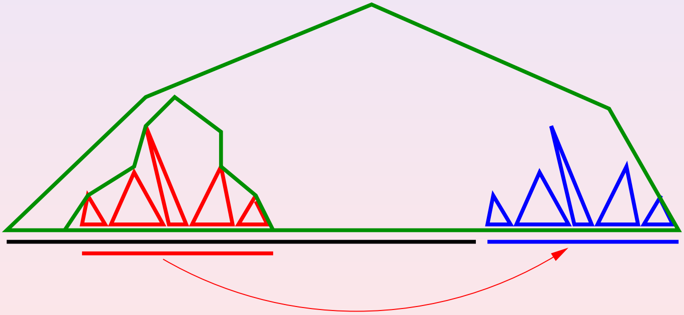
The relation between grammars and Lempel-Ziv

- They also show that $g = O(z_{no} \log(n/z_{no}))$.
  - By creating a grammar from a Lempel-Ziv parse.



The relation between grammars and Lempel-Ziv

- They also show that $g = O(z_{no} \log(n/z_{no}))$.
  - By creating a grammar from a Lempel-Ziv parse.



The relation between grammars and Lempel-Ziv

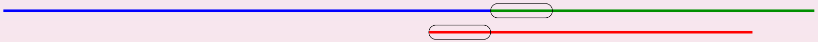
- Gawrychowski [2011] proved that $g = O(z \log(n/z))$ also holds.

The relation between grammars and Lempel-Ziv

- Gawrychowski [2011] proved that $g = O(z \log(n/z))$ also holds.
  - By extending the previous proof to the self-referential case.

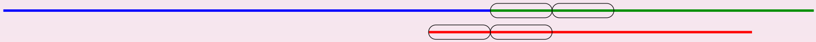
The relation between grammars and Lempel-Ziv

- Gawrychowski [2011] proved that $g = O(z \log(n/z))$ also holds.
  - By extending the previous proof to the self-referential case.
  - These imply that the phrase is periodic.



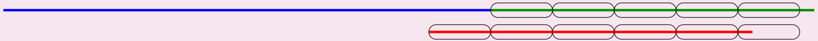
The relation between grammars and Lempel-Ziv

- Gawrychowski [2011] proved that $g = O(z \log(n/z))$ also holds.
  - By extending the previous proof to the self-referential case.
  - These imply that the phrase is periodic.



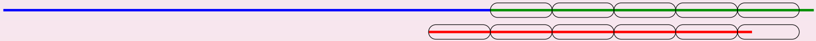
The relation between grammars and Lempel-Ziv

- Gawrychowski [2011] proved that $g = O(z \log(n/z))$ also holds.
  - By extending the previous proof to the self-referential case.
  - These imply that the phrase is periodic.



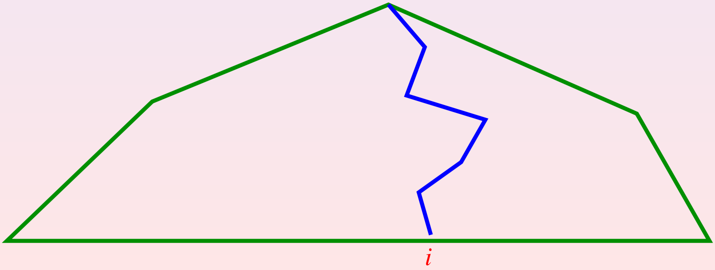
The relation between grammars and Lempel-Ziv

- Gawrychowski [2011] proved that $g = O(z \log(n/z))$ also holds.
  - By extending the previous proof to the self-referential case.
  - These imply that the phrase is periodic.
  - The periodic string is covered by a grammar of logarithmic size.



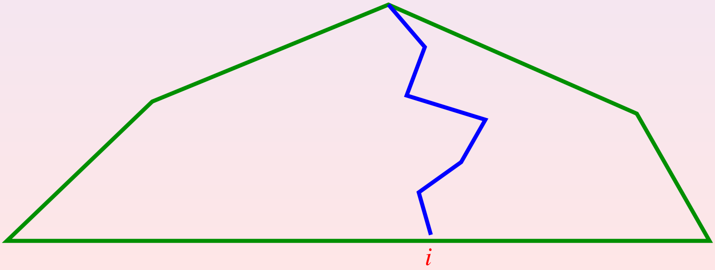
Accessing T in compressed space

- The grammar, in addition, is binary and balanced.



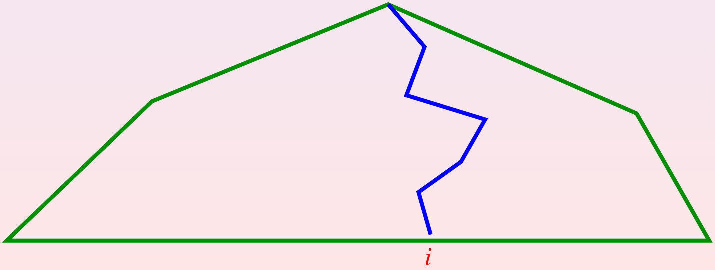
Accessing T in compressed space

- The grammar, in addition, is binary and balanced.
- That is, its height is $O(\log n)$.



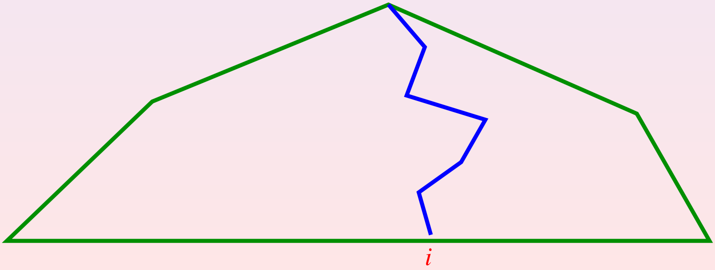
Accessing T in compressed space

- The grammar, in addition, is binary and balanced.
- That is, its height is $O(\log n)$.
- This yields the first access method within space $O(z_{no} \log(n/z_{no}))$.



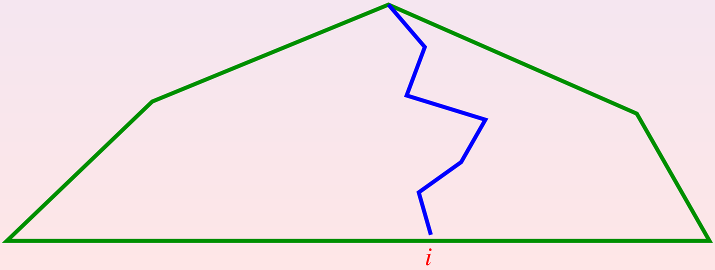
Accessing T in compressed space

- The grammar, in addition, is binary and balanced.
- That is, its height is $O(\log n)$.
- This yields the first access method within space $O(z_{no} \log(n/z_{no}))$.
  - Store the lengths to which nonterminals expand.



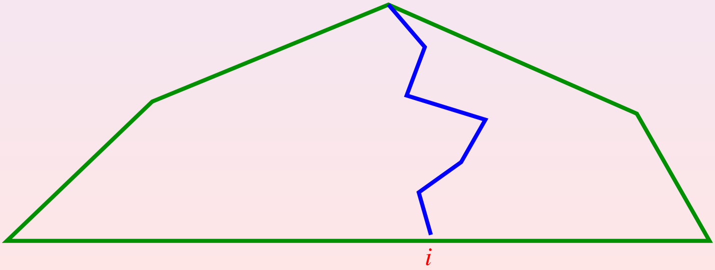
Accessing T in compressed space

- The grammar, in addition, is binary and balanced.
- That is, its height is $O(\log n)$.
- This yields the first access method within space $O(z_{no} \log(n/z_{no}))$.
  - Store the lengths to which nonterminals expand.
  - Walk down the grammar tree from the initial symbol towards the desired position i.



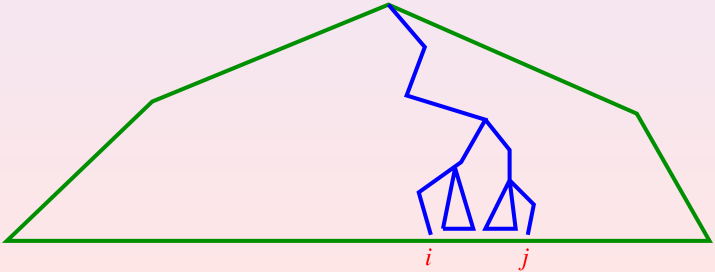
Accessing T in compressed space

- The grammar, in addition, is binary and balanced.
- That is, its height is $O(\log n)$.
- This yields the first access method within space $O(z_{no} \log(n/z_{no}))$.
  - Store the lengths to which nonterminals expand.
  - Walk down the grammar tree from the initial symbol towards the desired position i.
  - At the leaf we reach the terminal $T[i]$.



Accessing T in compressed space

- The grammar, in addition, is binary and balanced.
- That is, its height is $O(\log n)$.
- This yields the first access method within space $O(z_{no} \log(n/z_{no}))$.
  - Store the lengths to which nonterminals expand.
  - Walk down the grammar tree from the initial symbol towards the desired position i.
  - At the leaf we reach the terminal $T[i]$.
  - We can extract $T[i..i+\ell]$ in time $O(\ell + \log n)$.



Accessing T in compressed space

- This holds for $O(g)$ space in general.

Accessing T in compressed space

- This holds for $O(g)$ space in general.
- Because every grammar can be made binary and balanced within the same asymptotic size [Ganardi, Jez, Lohrey 2020].

The relation between grammars and Lempel-Ziv

- For some families, $g = \Omega(z_{no} \log n / \log \log n)$ [Charikar et al. 2005].

The relation between grammars and Lempel-Ziv

- For some families, $g = \Omega(z_{no} \log n / \log \log n)$ [Charikar et al. 2005].
- For example, $a^{k_1} b a^{k_2} b a^{k_3} b \cdots b a^{k_q}$, with $k_1 \geq k_i$ for all i, and $q = \Theta(\log k_1)$.

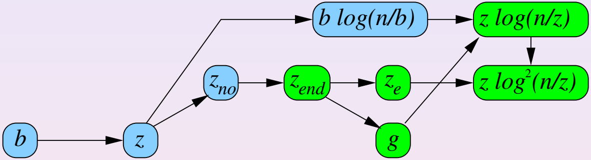
The relation between grammars and Lempel-Ziv

- For some families, $g = \Omega(z_{no} \log n / \log\log n)$ [Charikar et al. 2005].
- For example, $a^{k_1} b a^{k_2} b a^{k_3} b \cdots b a^{k_q}$, with $k_1 \geq k_i$ for all i, and $q = \Theta(\log k_1)$.
- It is parsed into $z_{no} = O(q + \log k_1) = O(\log k_1)$ phrases.

The relation between grammars and Lempel-Ziv

- For some families, $g = \Omega(z_{no} \log n / \log\log n)$ [Charikar et al. 2005].
- For example, $a^{k_1} b a^{k_2} b a^{k_3} b \cdots b a^{k_q}$, with $k_1 \geq k_i$ for all i, and $q = \Theta(\log k_1)$.
- It is parsed into $z_{no} = O(q + \log k_1) = O(\log k_1)$ phrases.
- Its grammar requires size $\Omega(\log^2 k_1 / \log\log k_1)$.

Recap



Run-length grammars

- They expand grammars by allowing rules $A \rightarrow B^k$.

Run-length grammars

- They expand grammars by allowing rules $A \rightarrow B^k$.
- The size g_{rl} of the smallest such grammar is another measure.

Run-length grammars

- They expand grammars by allowing rules $A \rightarrow B^k$.
- The size g_{rl} of the smallest such grammar is another measure.
- It obviously holds $g_{rl} \leq g$.

Run-length grammars

- They expand grammars by allowing rules $A \rightarrow B^k$.
- The size g_{rl} of the smallest such grammar is another measure.
- It obviously holds $g_{rl} \leq g$.
- In $T = a^n$ it holds $g, z_{no} = \Omega(g_{rl} \log n)$.

Run-length grammars

- They expand grammars by allowing rules $A \rightarrow B^k$.
- The size g_{rl} of the smallest such grammar is another measure.
- It obviously holds $g_{rl} \leq g$.
- In $T = a^n$ it holds $g, z_{no} = \Omega(g_{rl} \log n)$.
- In some texts, $g_{rl} = \Omega(z_{no} \log n / \log \log n)$ [Bille et al. 2017].

Run-length grammars

- They expand grammars by allowing rules $A \rightarrow B^k$.
- The size g_{rl} of the smallest such grammar is another measure.
- It obviously holds $g_{rl} \leq g$.
- In $T = a^n$ it holds $g, z_{no} = \Omega(g_{rl} \log n)$.
- In some texts, $g_{rl} = \Omega(z_{no} \log n / \log \log n)$ [Bille et al. 2017].
  - Thue-Morse sequences, 01, 01 10, 0110 1001, ...

Run-length grammars

- They expand grammars by allowing rules $A \rightarrow B^k$.
- The size g_{rl} of the smallest such grammar is another measure.
- It obviously holds $g_{rl} \leq g$.
- In $T = a^n$ it holds $g, z_{no} = \Omega(g_{rl} \log n)$.
- In some texts, $g_{rl} = \Omega(z_{no} \log n / \log \log n)$ [Bille et al. 2017].
  - Thue-Morse sequences, 01, 01 10, 0110 1001, ...
  - Differential Thue-Morse, $+$, $+=-$, $+=-+-=+$, ...

Run-length grammars

- They expand grammars by allowing rules $A \rightarrow B^k$.
- The size g_{rl} of the smallest such grammar is another measure.
- It obviously holds $g_{rl} \leq g$.
- In $T = a^n$ it holds $g, z_{no} = \Omega(g_{rl} \log n)$.
- In some texts, $g_{rl} = \Omega(z_{no} \log n / \log \log n)$ [Bille et al. 2017].
  - Thue-Morse sequences, 01, 01 10, 0110 1001, ...
  - Differential Thue-Morse, $+$, $+=-$, $+=-+-=+$, ...
  - In such family, $g = g_{rl}$.

Run-length grammars

- They expand grammars by allowing rules $A \rightarrow B^k$.
- The size g_{rl} of the smallest such grammar is another measure.
- It obviously holds $g_{rl} \leq g$.
- In $T = a^n$ it holds $g, z_{no} = \Omega(g_{rl} \log n)$.
- In some texts, $g_{rl} = \Omega(z_{no} \log n / \log \log n)$ [Bille et al. 2017].
  - Thue-Morse sequences, 01, 01 10, 0110 1001, ...
  - Differential Thue-Morse, $+$, $+=-$, $+=-+-=+$, ...
  - In such family, $g = g_{rl}$.
  - They combine it with the result of Charikar et al.

Run-length grammars

- They expand grammars by allowing rules $A \rightarrow B^k$.
- The size g_{rl} of the smallest such grammar is another measure.
- It obviously holds $g_{rl} \leq g$.
- In $T = a^n$ it holds $g, z_{no} = \Omega(g_{rl} \log n)$.
- In some texts, $g_{rl} = \Omega(z_{no} \log n / \log \log n)$ [Bille et al. 2017].
  - Thue-Morse sequences, 01, $01\ 10$, $0110\ 1001$, ...
  - Differential Thue-Morse, $+$, $+=-$, $+=-+-=+$, ...
  - In such family, $g = g_{rl}$.
  - They combine it with the result of Charikar et al.
  - They obtain $g_{rl} = \Omega(\log^2 n / \log \log n)$ and $z_{no} = O(\log n)$.

Run-length grammars

- They expand grammars by allowing rules $A \rightarrow B^k$.
- The size g_{rl} of the smallest such grammar is another measure.
- It obviously holds $g_{rl} \leq g$.
- In $T = a^n$ it holds $g, z_{no} = \Omega(g_{rl} \log n)$.
- In some texts, $g_{rl} = \Omega(z_{no} \log n / \log \log n)$ [Bille et al. 2017].
  - Thue-Morse sequences, 01, $01\ 10$, $0110\ 1001$, ...
  - Differential Thue-Morse, $+$, $+=-$, $+=-+-=+$, ...
  - In such family, $g = g_{rl}$.
  - They combine it with the result of Charikar et al.
  - They obtain $g_{rl} = \Omega(\log^2 n / \log \log n)$ and $z_{no} = O(\log n)$.
- The same proof shows that $g_{rl} = \Omega(z_e \log n / \log \log n)$.

More relations with g_{rl}

- Gagie, N., and Prezza [2018], corrected in N., Ochoa, and Prezza [2021].

More relations with g_{rl}

- Gagie, N., and Prezza [2018], corrected in N., Ochoa, and Prezza [2021].
- They define a locally consistent run-length grammar to prove $g_{rl} = O(b \log(n/b))$.

More relations with g_{rl}

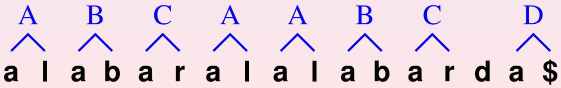
- Gagie, N., and Prezza [2018], corrected in N., Ochoa, and Prezza [2021].
- They define a locally consistent run-length grammar to prove $g_{rl} = O(b \log(n/b))$.
  - It is analyzed by considering an underlying bidirectional macro sheme.

More relations with g_{rl}

- Gagie, N., and Prezza [2018], corrected in N., Ochoa, and Prezza [2021].
- They define a locally consistent run-length grammar to prove $g_{rl} = O(b \log(n/b))$.
  - It is analyzed by considering an underlying bidirectional macro sheme.
  - They show that new nonterminals are formed near block boundaries only.

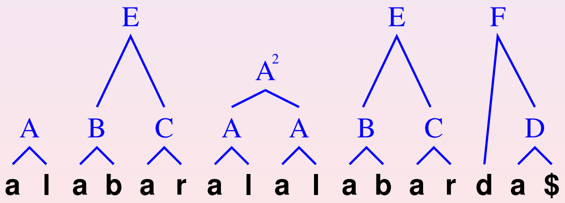
More relations with g_{rl}

- Gagie, N., and Prezza [2018], corrected in N., Ochoa, and Prezza [2021].
- They define a locally consistent run-length grammar to prove $g_{rl} = O(b \log(n/b))$.
  - It is analyzed by considering an underlying bidirectional macro sheme.
  - They show that new nonterminals are formed near block boundaries only.



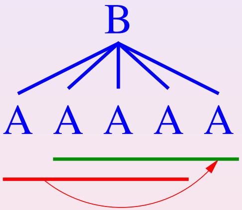
More relations with g_{rl}

- Gagie, N., and Prezza [2018], corrected in N., Ochoa, and Prezza [2021].
- They define a locally consistent run-length grammar to prove $g_{rl} = O(b \log(n/b))$.
  - It is analyzed by considering an underlying bidirectional macro sheme.
  - They show that new nonterminals are formed near block boundaries only.



More relations with g_{rl}

- They (easily) prove $z = O(g_{rl})$, thus the bound we already saw, $z = O(b \log(n/b))$.



Accessing run-length grammars

- It is unknown if one can balance run-length grammars.

Accessing run-length grammars

- It is unknown if one can balance run-length grammars.
- But Bille et al. [2011] had shown how to provide logarithmic access time on arbitrary grammars.

Accessing run-length grammars

- It is unknown if one can balance run-length grammars.
- But Bille et al. [2011] had shown how to provide logarithmic access time on arbitrary grammars.
- Christiansen et al. [2020] generalized the result to run-length grammars.

Accessing run-length grammars

- It is unknown if one can balance run-length grammars.
- But Bille et al. [2011] had shown how to provide logarithmic access time on arbitrary grammars.
- Christiansen et al. [2020] generalized the result to run-length grammars.
- So one can access any $T[i..i+\ell]$ within $O(g_{rl})$ space in time $O(\log n+\ell)$.

Collage systems

- In 2003, Kida et al. invented collage systems.

Collage systems

- In 2003, Kida et al. invented collage systems.
- Collage systems combine run-length grammars with composition systems.

Collage systems

- In 2003, Kida et al. invented collage systems.
- Collage systems combine run-length grammars with composition systems.
  - Permit rules of the form $A \to B^k$ (repetitions).

Collage systems

- In 2003, Kida et al. invented collage systems.
- Collage systems combine run-length grammars with composition systems.
  - Permit rules of the form $A \to B^k$ (repetitions).
  - Permit rules of the form $A \to B^{[t]}$ and $A \to {}^{[t]}B$ (prefix/suffix truncation).

Collage systems

- In 2003, Kida et al. invented collage systems.
- Collage systems combine run-length grammars with composition systems.
  - Permit rules of the form $A \rightarrow B^k$ (repetitions).
  - Permit rules of the form $A \rightarrow B^{[t]}$ and $A \rightarrow {}^{[t]}B$ (prefix/suffix truncation).
- The size $c \leq g_{rl}$ of the smallest collage system is another measure.

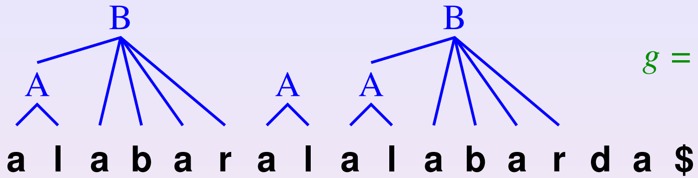
Collage systems

- In 2003, Kida et al. invented collage systems.
- Collage systems combine run-length grammars with composition systems.
  - Permit rules of the form $A \rightarrow B^k$ (repetitions).
  - Permit rules of the form $A \rightarrow B^{[t]}$ and $A \rightarrow {}^{[t]}B$ (prefix/suffix truncation).
- The size $c \leq g_{rl}$ of the smallest collage system is another measure.
- Not known how to find c nor how to approximate it.

Collage systems

- In 2003, Kida et al. invented collage systems.
- Collage systems combine run-length grammars with composition systems.
  - Permit rules of the form $A \rightarrow B^k$ (repetitions).
  - Permit rules of the form $A \rightarrow B^{[t]}$ and $A \rightarrow {}^{[t]}B$ (prefix/suffix truncation).
- The size $c \leq g_{rl}$ of the smallest collage system is another measure.
- Not known how to find c nor how to approximate it.
- Not much popular.

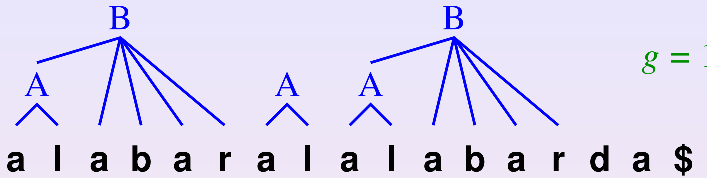
Collage systems



$g = 13$

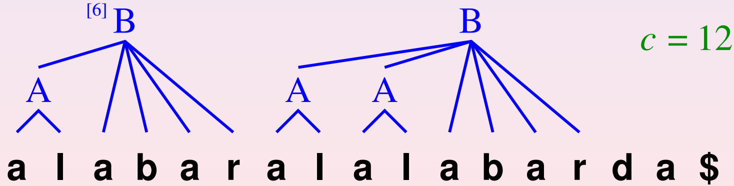
A → a l B → A a b a r S → B A B d a $

Collage systems



$g = 13$

$A \rightarrow \mathbf{a\ l} \quad B \rightarrow A\ \mathbf{a\ b\ a\ r} \quad S \rightarrow B\ A\ B\ \mathbf{d\ a\ \$}$



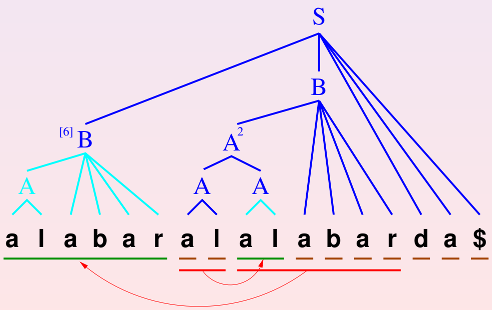
$c = 12$

$A \rightarrow \mathbf{a\ l} \quad B \rightarrow A^2\ \mathbf{a\ b\ a\ r} \quad S \rightarrow {}^{[6]}B\ B\ \mathbf{d\ a\ \$}$

Relations with collage systems

N., Ochoa, Prezza [2021] proved $b = O(c)$

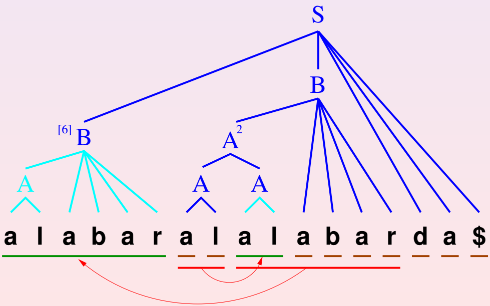
- From run-length grammar trees to collage systems.



Relations with collage systems

N., Ochoa, Prezza [2021] proved $b = O(c)$

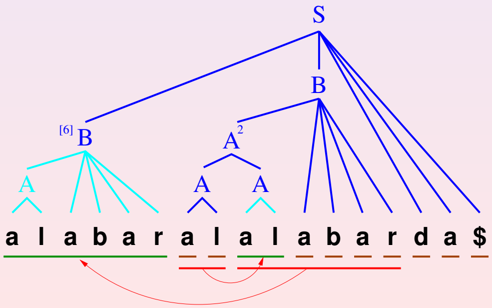
- From run-length grammar trees to collage systems.
- It chooses leftmost (non-trimmed) node A to be internal.



Relations with collage systems

N., Ochoa, Prezza [2021] proved $b = O(c)$

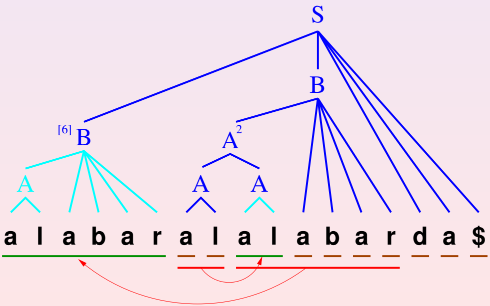
- From run-length grammar trees to collage systems.
- It chooses leftmost (non-trimmed) node A to be internal.
- All other trimmed/non-trimmed occurrences are leaves.



Relations with collage systems

N., Ochoa, Prezza [2021] proved $b = O(c)$

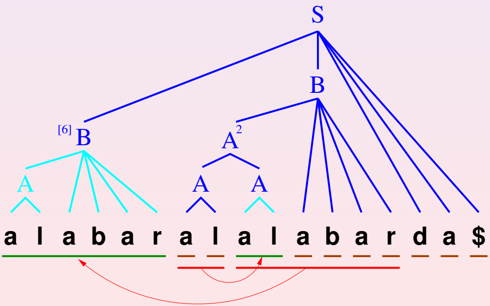
- From run-length grammar trees to collage systems.
- It chooses leftmost (non-trimmed) node A to be internal.
- All other trimmed/non-trimmed occurrences are leaves.
- Text positions are ordered by their leaf creation time.



Relations with collage systems

N., Ochoa, Prezza [2021] proved $b = O(c)$

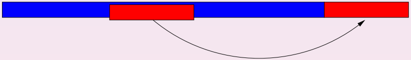
- From run-length grammar trees to collage systems.
- It chooses leftmost (non-trimmed) node A to be internal.
- All other trimmed/non-trimmed occurrences are leaves.
- Text positions are ordered by their leaf creation time.
- Then the blocks always point earlier in time.

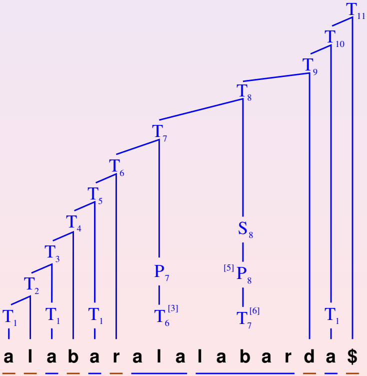


Relations with collage systems

N., Ochoa, Prezza [2021] proved $c = O(z)$

- Let T_i be the initial symbol up to phrase i.

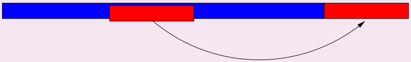


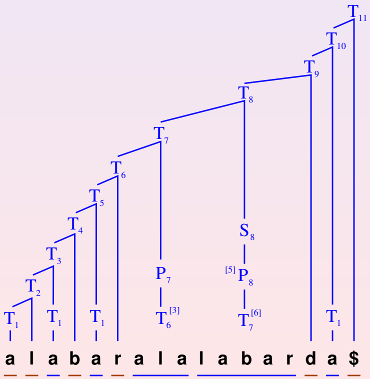


Relations with collage systems

N., Ochoa, Prezza [2021] proved $c = O(z)$

- Let T_i be the initial symbol up to phrase i.
- If the next phrase is a substring of T_i, we extract it with a prefix and a suffix rule.

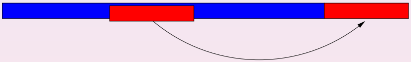


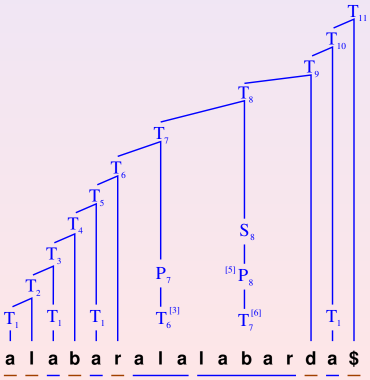


Relations with collage systems

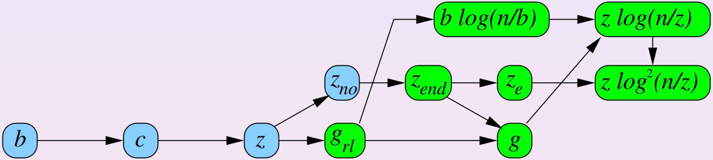
N., Ochoa, Prezza [2021] proved $c = O(z)$

- Let T_i be the initial symbol up to phrase i.
- If the next phrase is a substring of T_i, we extract it with a prefix and a suffix rule.
- Self-referential phrases are handled with run-length rules.





Recap



Runs in the Burrows-Wheeler Transform (BWT)

- Burrows and Wheeler had invented the BWT for text compression in 1994.

Runs in the Burrows-Wheeler Transform (BWT)

- Burrows and Wheeler had invented the BWT for text compression in 1994.
- It is a permutation of T where we pick the characters preceding the suffixes of the suffix array.

Runs in the Burrows-Wheeler Transform (BWT)

- Burrows and Wheeler had invented the BWT for text compression in 1994.
- It is a permutation of T where we pick the characters preceding the suffixes of the suffix array.
- It reaches high-order statistical entropy.

Runs in the Burrows-Wheeler Transform (BWT)

- Burrows and Wheeler had invented the BWT for text compression in 1994.
- It is a permutation of T where we pick the characters preceding the suffixes of the suffix array.
- It reaches high-order statistical entropy.
- It naturally groups symbols with similar contexts, and tends to form long runs of equal symbols.

Runs in the Burrows-Wheeler Transform (BWT)

- Burrows and Wheeler had invented the BWT for text compression in 1994.
- It is a permutation of T where we pick the characters preceding the suffixes of the suffix array.
- It reaches high-order statistical entropy.
- It naturally groups symbols with similar contexts, and tends to form long runs of equal symbols.
- The number r of those runs is another measure of compressibility.

Runs in the Burrows-Wheeler Transform (BWT)

- Burrows and Wheeler had invented the BWT for text compression in 1994.
- It is a permutation of T where we pick the characters preceding the suffixes of the suffix array.
- It reaches high-order statistical entropy.
- It naturally groups symbols with similar contexts, and tends to form long runs of equal symbols.
- The number r of those runs is another measure of compressibility.
  - Since the transformation is reversible, we can compress T in $O(r \log n)$ bits.

Runs in the Burrows-Wheeler Transform (BWT)

- Burrows and Wheeler had invented the BWT for text compression in 1994.
- It is a permutation of T where we pick the characters preceding the suffixes of the suffix array.
- It reaches high-order statistical entropy.
- It naturally groups symbols with similar contexts, and tends to form long runs of equal symbols.
- The number r of those runs is another measure of compressibility.
  - Since the transformation is reversible, we can compress T in $O(r \log n)$ bits.
  - For repetitive texts, r is small [Mäkinen et al. 2008].

Runs in the Burrows-Wheeler Transform (BWT)

```
$ a l a b a r a l a l a b a r d a
a $ a l a b a r a l a l a b a r d
a b a r a l a l a b a r d a $ a l
a b a r d a $ a l a b a r a l a l
a l a b a r a l a l a b a r d a $
a l a b a r d a $ a l a b a r a l
a l a l a b a r d a $ a l a b a r
a r a l a l a b a r d a $ a l a b
a r d a $ a l a b a r a l a l a b
b a r a l a l a b a r d a $ a l a
b a r d a $ a l a b a r a l a l a
d a $ a l a b a r a l a l a b a r
l a b a r a l a l a b a r d a $ a
l a b a r d a $ a l a b a r a l a
l a l a b a r d a $ a l a b a r a
r a l a l a b a r d a $ a l a b a
r d a $ a l a b a r a l a l a b a
```

$r = 10$

Runs in the Burrows-Wheeler Transform (BWT)

Good properties

- r is related to Shannon's entropy [Mäkinen and N. 2005].

Bad properties

Runs in the Burrows-Wheeler Transform (BWT)

Good properties

- r is related to Shannon's entropy [Mäkinen and N. 2005].
- It can be computed in $O(n)$ time.

Bad properties

Runs in the Burrows-Wheeler Transform (BWT)

Good properties

- r is related to Shannon's entropy [Mäkinen and N. 2005].
- It can be computed in $O(n)$ time.
- Closely related with suffix arrays: efficient pattern matching is enabled [Ferragina & Manzini 2000; Mäkinen & N. 2005; Gagie, N., Prezza 2018].

Bad properties

Runs in the Burrows-Wheeler Transform (BWT)

Good properties

- r is related to Shannon's entropy [Mäkinen and N. 2005].
- It can be computed in $O(n)$ time.
- Closely related with suffix arrays: efficient pattern matching is enabled [Ferragina & Manzini 2000; Mäkinen & N. 2005; Gagie, N., Prezza 2018].

Bad properties

- Larger in practice than most other measures.

Runs in the Burrows-Wheeler Transform (BWT)

Good properties

- r is related to Shannon's entropy [Mäkinen and N. 2005].
- It can be computed in $O(n)$ time.
- Closely related with suffix arrays: efficient pattern matching is enabled [Ferragina & Manzini 2000; Mäkinen & N. 2005; Gagie, N., Prezza 2018].

Bad properties

- Larger in practice than most other measures.
- The value of r depends on the permutation of the alphabet.

Runs in the Burrows-Wheeler Transform (BWT)

Good properties

- r is related to Shannon's entropy [Mäkinen and N. 2005].
- It can be computed in $O(n)$ time.
- Closely related with suffix arrays: efficient pattern matching is enabled [Ferragina & Manzini 2000; Mäkinen & N. 2005; Gagie, N., Prezza 2018].

Bad properties

- Larger in practice than most other measures.
- The value of r depends on the permutation of the alphabet.
  - Gap factors of $\Omega(\log n)$ can be obtained.

Runs in the Burrows-Wheeler Transform (BWT)

Good properties

- r is related to Shannon's entropy [Mäkinen and N. 2005].
- It can be computed in $O(n)$ time.
- Closely related with suffix arrays: efficient pattern matching is enabled [Ferragina & Manzini 2000; Mäkinen & N. 2005; Gagie, N., Prezza 2018].

Bad properties

- Larger in practice than most other measures.
- The value of r depends on the permutation of the alphabet.
  - Gap factors of $\Omega(\log n)$ can be obtained.
  - NP-hard to find the best permutation [Bentley, Gibney, Thankachan 2019].

Runs in the Burrows-Wheeler Transform (BWT)

Good properties

- r is related to Shannon's entropy [Mäkinen and N. 2005].
- It can be computed in $O(n)$ time.
- Closely related with suffix arrays: efficient pattern matching is enabled [Ferragina & Manzini 2000; Mäkinen & N. 2005; Gagie, N., Prezza 2018].

Bad properties

- Larger in practice than most other measures.
- The value of r depends on the permutation of the alphabet.
  - Gap factors of $\Omega(\log n)$ can be obtained.
  - NP-hard to find the best permutation [Bentley, Gibney, Thankachan 2019].
- It is not known how to access $T[i]$ within $O(r)$ space.

Relating the BWT runs

- Loosely upper bounded, $r = O(z \log z \log(n/z))$ [Kempa & Kociumaka 2019].

Relating the BWT runs

- Loosely upper bounded, $r = O(z \log z \log(n/z))$ [Kempa & Kociumaka 2019].
- It can reach $r = \Theta(n)$, where $r = \Omega(g \log n)$ [Belazzougui et al. 2015; N. & Prezza 2019].

Relating the BWT runs

- Loosely upper bounded, $r = O(z \log z \log(n/z))$ [Kempa & Kociumaka 2019].
- It can reach $r = \Theta(n)$, where $r = \Omega(g \log n)$ [Belazzougui et al. 2015; N. & Prezza 2019].
  - On de Bruijn sequences, $r = \Theta(n)$ and $z = O(n/\log n)$.

Relating the BWT runs

- Loosely upper bounded, $r = O(z \log z \log(n/z))$ [Kempa & Kociumaka 2019].
- It can reach $r = \Theta(n)$, where $r = \Omega(g \log n)$ [Belazzougui et al. 2015; N. & Prezza 2019].
  - On de Bruijn sequences, $r = \Theta(n)$ and $z = O(n/\log n)$.
  - The smallest grammar size is also $g = O(n/\log n)$.

Relating the BWT runs

- Loosely upper bounded, $r = O(z \log z \log(n/z))$ [Kempa & Kociumaka 2019].
- It can reach $r = \Theta(n)$, where $r = \Omega(g \log n)$ [Belazzougui et al. 2015; N. & Prezza 2019].
  - On de Bruijn sequences, $r = \Theta(n)$ and $z = O(n/\log n)$.
  - The smallest grammar size is also $g = O(n/\log n)$.
- There are families where $z = \Omega(r \log n)$ [Prezza 2016].

Relating the BWT runs

- Loosely upper bounded, $r = O(z \log z \log(n/z))$ [Kempa & Kociumaka 2019].
- It can reach $r = \Theta(n)$, where $r = \Omega(g \log n)$ [Belazzougui et al. 2015; N. & Prezza 2019].
  - On de Bruijn sequences, $r = \Theta(n)$ and $z = O(n/\log n)$.
  - The smallest grammar size is also $g = O(n/\log n)$.
- There are families where $z = \Omega(r \log n)$ [Prezza 2016].
  - On Fibonacci strings, $z = \Theta(\log n)$ and $r = O(1)$.

Relating the BWT runs

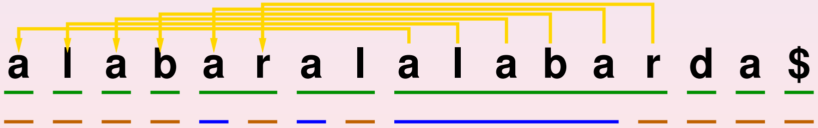
- Loosely upper bounded, $r = O(z \log z \log(n/z))$ [Kempa & Kociumaka 2019].
- It can reach $r = \Theta(n)$, where $r = \Omega(g \log n)$ [Belazzougui et al. 2015; N. & Prezza 2019].
  - On de Bruijn sequences, $r = \Theta(n)$ and $z = O(n/\log n)$.
  - The smallest grammar size is also $g = O(n/\log n)$.
- There are families where $z = \Omega(r \log n)$ [Prezza 2016].
  - On Fibonacci strings, $z = \Theta(\log n)$ and $r = O(1)$.
- So r and z are incomparable.

Relating the BWT runs

- Loosely upper bounded, $r = O(z \log z \log(n/z))$ [Kempa & Kociumaka 2019].
- It can reach $r = \Theta(n)$, where $r = \Omega(g \log n)$ [Belazzougui et al. 2015; N. & Prezza 2019].
  - On de Bruijn sequences, $r = \Theta(n)$ and $z = O(n/\log n)$.
  - The smallest grammar size is also $g = O(n/\log n)$.
- There are families where $z = \Omega(r \log n)$ [Prezza 2016].
  - On Fibonacci strings, $z = \Theta(\log n)$ and $r = O(1)$.
- So r and z are incomparable.
- Yet, it can be proved that $b = O(r)$ [Gagie, N., Prezza 2018].

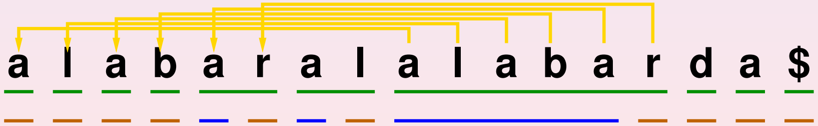
Relating the BWT runs

- They use locally consistent parsing again.



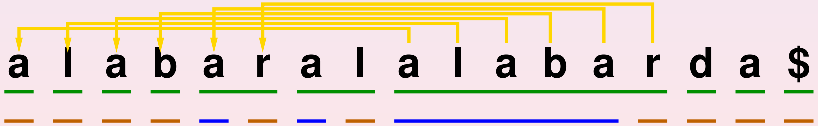
Relating the BWT runs

- They use locally consistent parsing again.
- The BWT runs induces a parsing of r phrases on T.



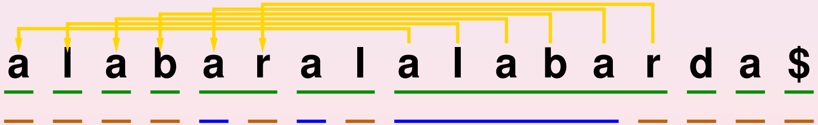
Relating the BWT runs

- They use locally consistent parsing again.
- The BWT runs induces a parsing of r phrases on T.
- They use it to build a bidirectional macro scheme of size $b = O(r)$.



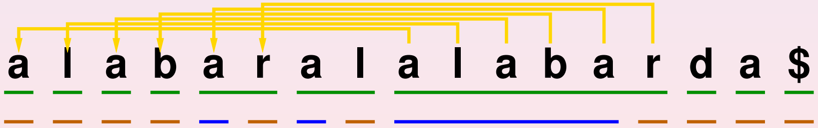
Relating the BWT runs

- They use locally consistent parsing again.
- The BWT runs induces a parsing of r phrases on T.
- They use it to build a bidirectional macro scheme of size $b = O(r)$.
- It has no cycles because the parsing is lexicographic:



Relating the BWT runs

- They use locally consistent parsing again.
- The BWT runs induces a parsing of r phrases on T.
- They use it to build a bidirectional macro scheme of size $b = O(r)$.
- It has no cycles because the parsing is lexicographic:
  - The source is lexicographically smaller than the target.



Lexicographic parsing

- N., Ochoa, Prezza [2021] define a new measure $v \leq r$:

Good properties (wrt z)

Bad properties (wrt z)

Lexicographic parsing

- N., Ochoa, Prezza [2021] define a new measure $v \leq r$:
  - This is the size of the smallest lexicographic parse.

Good properties (wrt z)

Bad properties (wrt z)

Lexicographic parsing

- N., Ochoa, Prezza [2021] define a new measure $v \leq r$:
  - This is the size of the smallest lexicographic parse.
  - They show how to compute v in linear time.

Good properties (wrt z)

Bad properties (wrt z)

Lexicographic parsing

- N., Ochoa, Prezza [2021] define a new measure $v \leq r$:
  - This is the size of the smallest lexicographic parse.
  - They show how to compute v in linear time.
- While z preserves text order, v preserves suffix array order

Good properties (wrt z)

Bad properties (wrt z)

Lexicographic parsing

- N., Ochoa, Prezza [2021] define a new measure $v \leq r$:
  - This is the size of the smallest lexicographic parse.
  - They show how to compute v in linear time.
- While z preserves text order, v preserves suffix array order

Good properties (wrt z)

- Similar compression ratios in many cases.

Bad properties (wrt z)

Lexicographic parsing

- N., Ochoa, Prezza [2021] define a new measure $v \leq r$:
  - This is the size of the smallest lexicographic parse.
  - They show how to compute v in linear time.
- While z preserves text order, v preserves suffix array order

Good properties (wrt z)

- Similar compression ratios in many cases.
- Better formal guarantees, e.g. $v \leq r$.

Bad properties (wrt z)

Lexicographic parsing

- N., Ochoa, Prezza [2021] define a new measure $v \leq r$:
  - This is the size of the smallest lexicographic parse.
  - They show how to compute v in linear time.
- While z preserves text order, v preserves suffix array order

Good properties (wrt z)

- Similar compression ratios in many cases.
- Better formal guarantees, e.g. $v \leq r$.
- Similarly efficient compression/decompression.

Bad properties (wrt z)

Lexicographic parsing

- N., Ochoa, Prezza [2021] define a new measure $v \leq r$:
  - This is the size of the smallest lexicographic parse.
  - They show how to compute v in linear time.
- While z preserves text order, v preserves suffix array order

Good properties (wrt z)

- Similar compression ratios in many cases.
- Better formal guarantees, e.g. $v \leq r$.
- Similarly efficient compression/decompression.

Bad properties (wrt z)

- Only clearly better compression on the Fibonacci words.

Lexicographic parsing

- N., Ochoa, Prezza [2021] define a new measure $v \leq r$:
  - This is the size of the smallest lexicographic parse.
  - They show how to compute v in linear time.
- While z preserves text order, v preserves suffix array order

Good properties (wrt z)

- Similar compression ratios in many cases.
- Better formal guarantees, e.g. $v \leq r$.
- Similarly efficient compression/decompression.

Bad properties (wrt z)

- Only clearly better compression on the Fibonacci words.
- Worse compression on versions with cumulative edits.

Lexicographic parsing

- N., Ochoa, Prezza [2021] define a new measure $v \leq r$:
  - This is the size of the smallest lexicographic parse.
  - They show how to compute v in linear time.
- While z preserves text order, v preserves suffix array order

Good properties (wrt z)

- Similar compression ratios in many cases.
- Better formal guarantees, e.g. $v \leq r$.
- Similarly efficient compression/decompression.

Bad properties (wrt z)

- Only clearly better compression on the Fibonacci words.
- Worse compression on versions with cumulative edits.
- It also varies upon symbol remappings.

Relations with lexicographic parsing

- They show $v = O(n/\log_\sigma n)$, thus $r = \Omega(v \log n)$ on de Bruijn sequences.

Relations with lexicographic parsing

- They show $v = O(n/\log_\sigma n)$, thus $r = \Omega(v \log n)$ on de Bruijn sequences.
- It also induces a bidirectional parse, so $b = O(v)$.

Relations with lexicographic parsing

- They show $v = O(n/\log_\sigma n)$, thus $r = \Omega(v \log n)$ on de Bruijn sequences.
- It also induces a bidirectional parse, so $b = O(v)$.
- On odd Fibonacci words, it holds $b = O(1)$ and $v = \Theta(\log n)$.

Relations with lexicographic parsing

- They show $v = O(n/\log_\sigma n)$, thus $r = \Omega(v \log n)$ on de Bruijn sequences.
- It also induces a bidirectional parse, so $b = O(v)$.
- On odd Fibonacci words, it holds $b = O(1)$ and $v = \Theta(\log n)$.
- Since in some texts $z = \Omega(r \log n)$, also $z = \Omega(v \log n)$.

Relations with lexicographic parsing

- They show $v = O(n/\log_\sigma n)$, thus $r = \Omega(v \log n)$ on de Bruijn sequences.
- It also induces a bidirectional parse, so $b = O(v)$.
- On odd Fibonacci words, it holds $b = O(1)$ and $v = \Theta(\log n)$.
- Since in some texts $z = \Omega(r \log n)$, also $z = \Omega(v \log n)$.
- Further, $v = O(g_{rl})$

Relations with lexicographic parsing

- They show $v = O(n/\log_\sigma n)$, thus $r = \Omega(v \log n)$ on de Bruijn sequences.
- It also induces a bidirectional parse, so $b = O(v)$.
- On odd Fibonacci words, it holds $b = O(1)$ and $v = \Theta(\log n)$.
- Since in some texts $z = \Omega(r \log n)$, also $z = \Omega(v \log n)$.
- Further, $v = O(g_{rl})$
  - Like the proof of z, choosing the internal grammar tree nodes as the lexicographically smallest.

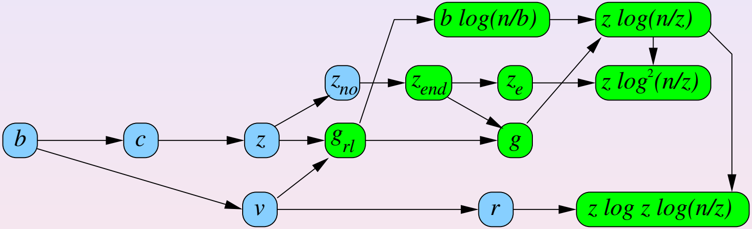
Relations with lexicographic parsing

- They show $v = O(n/\log_\sigma n)$, thus $r = \Omega(v \log n)$ on de Bruijn sequences.
- It also induces a bidirectional parse, so $b = O(v)$.
- On odd Fibonacci words, it holds $b = O(1)$ and $v = \Theta(\log n)$.
- Since in some texts $z = \Omega(r \log n)$, also $z = \Omega(v \log n)$.
- Further, $v = O(g_{rl})$
  - Like the proof of z, choosing the internal grammar tree nodes as the lexicographically smallest.
  - For run-length rules, one chooses $A \to B \cdot B^{t-1}$ or $A \to B^{t-1} \cdot B$.

Relations with lexicographic parsing

- They show $v = O(n/\log_\sigma n)$, thus $r = \Omega(v \log n)$ on de Bruijn sequences.
- It also induces a bidirectional parse, so $b = O(v)$.
- On odd Fibonacci words, it holds $b = O(1)$ and $v = \Theta(\log n)$.
- Since in some texts $z = \Omega(r \log n)$, also $z = \Omega(v \log n)$.
- Further, $v = O(g_{rl})$
  - Like the proof of z, choosing the internal grammar tree nodes as the lexicographically smallest.
  - For run-length rules, one chooses $A \rightarrow B \cdot B^{t-1}$ or $A \rightarrow B^{t-1} \cdot B$.
- Unknown if $z = o(v)$ for some text family.

Recap



Attractors

- In 2018, Kempa and Prezza invent the concept of attractor.

Relations

a l a b a r a l a l a b a r d a $

$\gamma = 6$

Attractors

- In 2018, Kempa and Prezza invent the concept of attractor.
  - The first measure designed as a lower bound, not ad-hoc.

Relations

a l a b a r a l a l a b a r d a $

$\gamma = 6$

Attractors

- In 2018, Kempa and Prezza invent the concept of attractor.
  - The first measure designed as a lower bound, not ad-hoc.
  - A set Γ of positions in T such that any substring of T has a copy including an element of Γ.

Relations

a l a b a r a l a l a b a r d a $

$\gamma = 6$

Attractors

- In 2018, Kempa and Prezza invent the concept of attractor.
  - The first measure designed as a lower bound, not ad-hoc.
  - A set Γ of positions in T such that any substring of T has a copy including an element of Γ.
  - The size γ of the smallest attractor is the measure.

Relations

a l a $\underline{\text{b}}$ $\underline{\text{a}}$ $\underline{\text{r}}$ a $\underline{\text{l}}$ a l a b a r $\underline{\text{d}}$ a $\underline{\text{\$}}$

$\gamma = 6$

Attractors

- In 2018, Kempa and Prezza invent the concept of attractor.
  - The first measure designed as a lower bound, not ad-hoc.
  - A set Γ of positions in T such that any substring of T has a copy including an element of Γ.
  - The size γ of the smallest attractor is the measure.
- They show that $\gamma = O(\min(b, c, v, z, z_{no}, r, g_{rl}, g))$.

Relations

a l a b a r a l a l a b a r d a $

$\gamma = 6$

Attractors

- In 2018, Kempa and Prezza invent the concept of attractor.
  - The first measure designed as a lower bound, not ad-hoc.
  - A set Γ of positions in T such that any substring of T has a copy including an element of Γ.
  - The size γ of the smallest attractor is the measure.
- They show that $\gamma = O(\min(b, c, v, z, z_{no}, r, g_{rl}, g))$.

Relations

- There is a string family where $\gamma = O(1)$ and $b = \Omega(\log n)$ [Kutsukake et al. 2020, Bannai et al. 2021]

a l a b a r a l a l a b a r d a $

$\gamma = 6$

Attractors

- In 2018, Kempa and Prezza invent the concept of attractor.
  - The first measure designed as a lower bound, not ad-hoc.
  - A set Γ of positions in T such that any substring of T has a copy including an element of Γ.
  - The size γ of the smallest attractor is the measure.
- They show that $\gamma = O(\min(b, c, v, z, z_{no}, r, g_{rl}, g))$.

Relations

- There is a string family where $\gamma = O(1)$ and $b = \Omega(\log n)$ [Kutsukake et al. 2020, Bannai et al. 2021]
  - Thue-Morse sequences, 01, 01 10, 0110 1001, ...

a l a <u>b</u> <u>a</u> <u>r</u> a <u>l</u> a l a b a r <u>d</u> a <u>$</u>

$\gamma = 6$

Attractors

- In 2018, Kempa and Prezza invent the concept of attractor.
  - The first measure designed as a lower bound, not ad-hoc.
  - A set Γ of positions in T such that any substring of T has a copy including an element of Γ.
  - The size γ of the smallest attractor is the measure.
- They show that $\gamma = O(\min(b, c, v, z, z_{no}, r, g_{rl}, g))$.

Relations

- There is a string family where $\gamma = O(1)$ and $b = \Omega(\log n)$ [Kutsukake et al. 2020, Bannai et al. 2021]
  - Thue-Morse sequences, 01, 01 10, 0110 1001, ...
  - So γ is not reachable via copy-paste methods.

a l a $\underline{\textbf{b}}$ $\underline{\textbf{a}}$ $\underline{\textbf{r}}$ a $\underline{\textbf{l}}$ a l a b a r $\underline{\textbf{d}}$ a $\underline{\textbf{\$}}$

$\gamma = 6$

Attractors

- In 2018, Kempa and Prezza invent the concept of attractor.
  - The first measure designed as a lower bound, not ad-hoc.
  - A set Γ of positions in T such that any substring of T has a copy including an element of Γ.
  - The size γ of the smallest attractor is the measure.
- They show that $\gamma = O(\min(b, c, v, z, z_{no}, r, g_{rl}, g))$.

Relations

- There is a string family where $\gamma = O(1)$ and $b = \Omega(\log n)$ [Kutsukake et al. 2020, Bannai et al. 2021]
  - Thue-Morse sequences, 01, 01 10, 0110 1001, ...
  - So γ is not reachable via copy-paste methods.
- Spoiler: Kociumaka thinks he can prove $g = O(\gamma \log(n/\gamma))$.

a l a b a r a l a l a b a r d a $

$\gamma = 6$

Attractors

Good properties

- Elegant definition, no ad-hoc measure.

Bad properties

Attractors

Good properties

- Elegant definition, no ad-hoc measure.
- Lower-bounds all the known reachable measures.

Bad properties

Attractors

Good properties

- Elegant definition, no ad-hoc measure.
- Lower-bounds all the known reachable measures.
- Invariant upon string reversal and symbol mappings.

Bad properties

Attractors

Good properties

- Elegant definition, no ad-hoc measure.
- Lower-bounds all the known reachable measures.
- Invariant upon string reversal and symbol mappings.
- In $O(\gamma \log(n/\gamma))$ space one can provide access.

Bad properties

Attractors

Good properties

- Elegant definition, no ad-hoc measure.
- Lower-bounds all the known reachable measures.
- Invariant upon string reversal and symbol mappings.
- In $O(\gamma \log(n/\gamma))$ space one can provide access.

Bad properties

- Non-monotonic upon appends [Mantaci et al. 2021].

Attractors

Good properties

- Elegant definition, no ad-hoc measure.
- Lower-bounds all the known reachable measures.
- Invariant upon string reversal and symbol mappings.
- In $O(\gamma \log(n/\gamma))$ space one can provide access.

Bad properties

- Non-monotonic upon appends [Mantaci et al. 2021].
- It may double upon a single character edit on T [Akagi, Funakoshi, Inenaga 2022].

Attractors

Good properties

- Elegant definition, no ad-hoc measure.
- Lower-bounds all the known reachable measures.
- Invariant upon string reversal and symbol mappings.
- In $O(\gamma \log(n/\gamma))$ space one can provide access.

Bad properties

- Non-monotonic upon appends [Mantaci et al. 2021].
- It may double upon a single character edit on T [Akagi, Funakoshi, Inenaga 2022].
- NP-hard to compute.

Attractors

Good properties

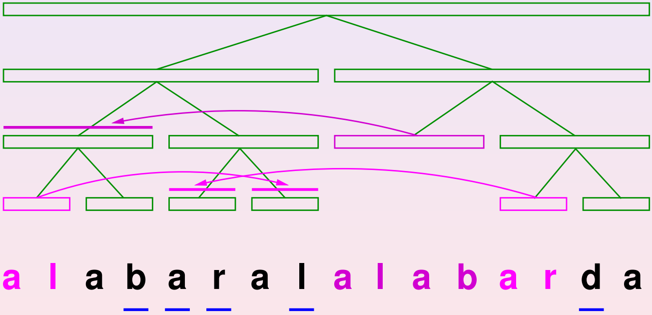
- Elegant definition, no ad-hoc measure.
- Lower-bounds all the known reachable measures.
- Invariant upon string reversal and symbol mappings.
- In $O(\gamma \log(n/\gamma))$ space one can provide access.

Bad properties

- Non-monotonic upon appends [Mantaci et al. 2021].
- It may double upon a single character edit on T [Akagi, Funakoshi, Inenaga 2022].
- NP-hard to compute.
- Unreachable? Can one represent T in $O(\gamma)$ space?

Accessing with attractors

Block-tree-like structure



String complexity δ: the holy grail?

- A stricter lower bound [Raskhodnikova et al. 2013]:

$$\delta \;=\; \max\{T(k)/k, k \geq 1\}$$

where $T(k)$ is the number of distinct k-length contexts in T.

a l a b a r a l a l a b a r d a $

T(1)/1 = 6

T(2)/2 = 4.5

T(3)/3 = 3.3

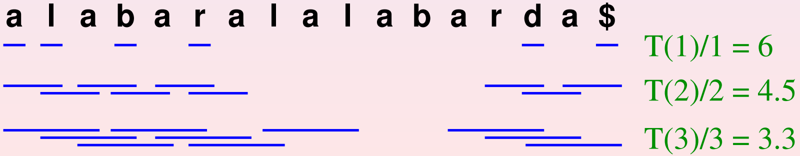
String complexity δ: the holy grail?

- A stricter lower bound [Raskhodnikova et al. 2013]:

$$\delta \;=\; \max\{T(k)/k, k \geq 1\}$$

where $T(k)$ is the number of distinct k-length contexts in T.

- It holds $\delta \leq \gamma$ because $T(k) \leq \gamma k$ for every k.

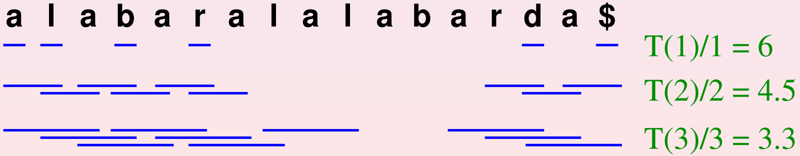


String complexity δ: the holy grail?

- A stricter lower bound [Raskhodnikova et al. 2013]:

$$\delta \quad = \quad \max\{T(k)/k, k \geq 1\}$$

  where $T(k)$ is the number of distinct k-length contexts in T.
- It holds $\delta \leq \gamma$ because $T(k) \leq \gamma k$ for every k.
- It can be computed in $O(n)$ time [Christiansen et al. 2020].

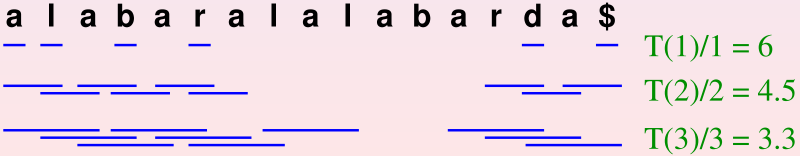


String complexity δ: the holy grail?

- A stricter lower bound [Raskhodnikova et al. 2013]:

$$\delta \quad = \quad \max\{T(k)/k, k \geq 1\}$$

  where $T(k)$ is the number of distinct k-length contexts in T.
- It holds $\delta \leq \gamma$ because $T(k) \leq \gamma k$ for every k.
- It can be computed in $O(n)$ time [Christiansen et al. 2020].
- Invariant upon reversals and symbol remappings.



String complexity δ: the holy grail?

- A stricter lower bound [Raskhodnikova et al. 2013]:

$$\delta \quad = \quad \max\{T(k)/k, k \geq 1\}$$

where $T(k)$ is the number of distinct k-length contexts in T.
- It holds $\delta \leq \gamma$ because $T(k) \leq \gamma k$ for every k.
- It can be computed in $O(n)$ time [Christiansen et al. 2020].
- Invariant upon reversals and symbol remappings.
- Monotonic upon symbol appends.

a l a b a r a l a l a b a r d a $

T(1)/1 = 6

T(2)/2 = 4.5

T(3)/3 = 3.3

String complexity δ: the holy grail?

- A stricter lower bound [Raskhodnikova et al. 2013]:

$$\delta \;=\; \max\{T(k)/k, k \geq 1\}$$

  where $T(k)$ is the number of distinct k-length contexts in T.
- It holds $\delta \leq \gamma$ because $T(k) \leq \gamma k$ for every k.
- It can be computed in $O(n)$ time [Christiansen et al. 2020].
- Invariant upon reversals and symbol remappings.
- Monotonic upon symbol appends.
- It grows only by 1 upon a single character edit on T [Akagi, Funakoshi, Inenaga 2022].

a l a b a r a l a l a b a r d a $

T(1)/1 = 6

T(2)/2 = 4.5

T(3)/3 = 3.3

Relations on δ

Some upper bounds in terms of δ

- $z = O(\delta \log(n/\delta))$ [Raskhodnikova et al. 2013], so this is reachable.

Relations on δ

Some upper bounds in terms of δ

- $z = O(\delta \log(n/\delta))$ [Raskhodnikova et al. 2013], so this is reachable.
- Block trees can be made of size $O(\delta \log(n/\delta))$, so direct access within that size [Kociumaka, N., Prezza 2020].

Relations on δ

Some upper bounds in terms of δ

- $z = O(\delta \log(n/\delta))$ [Raskhodnikova et al. 2013], so this is reachable.
- Block trees can be made of size $O(\delta \log(n/\delta))$, so direct access within that size [Kociumaka, N., Prezza 2020].
- Further, $g_{rl} = O(\delta \log(n/\delta))$ [Kociumaka, N., Prezza 2021].

Relations on δ

Some upper bounds in terms of δ

- $z = O(\delta \log(n/\delta))$ [Raskhodnikova et al. 2013], so this is reachable.
- Block trees can be made of size $O(\delta \log(n/\delta))$, so direct access within that size [Kociumaka, N., Prezza 2020].
- Further, $g_{rl} = O(\delta \log(n/\delta))$ [Kociumaka, N., Prezza 2021].
  - Similar to the proof for $g = O(b \log(n/b))$.

Relations on δ

Some upper bounds in terms of δ

- $z = O(\delta \log(n/\delta))$ [Raskhodnikova et al. 2013], so this is reachable.
- Block trees can be made of size $O(\delta \log(n/\delta))$, so direct access within that size [Kociumaka, N., Prezza 2020].
- Further, $g_{rl} = O(\delta \log(n/\delta))$ [Kociumaka, N., Prezza 2021].
  - Similar to the proof for $g = O(b \log(n/b))$.
  - “Pausing” long symbols to avoid them growing too fast.

Relations on δ

Some upper bounds in terms of δ

- $z = O(\delta \log(n/\delta))$ [Raskhodnikova et al. 2013], so this is reachable.
- Block trees can be made of size $O(\delta \log(n/\delta))$, so direct access within that size [Kociumaka, N., Prezza 2020].
- Further, $g_{rl} = O(\delta \log(n/\delta))$ [Kociumaka, N., Prezza 2021].
  - Similar to the proof for $g = O(b \log(n/b))$.
  - "Pausing" long symbols to avoid them growing too fast.
  - This cannot be achieved with g.

Relations on δ

Some upper bounds in terms of δ

- $z = O(\delta \log(n/\delta))$ [Raskhodnikova et al. 2013], so this is reachable.
- Block trees can be made of size $O(\delta \log(n/\delta))$, so direct access within that size [Kociumaka, N., Prezza 2020].
- Further, $g_{rl} = O(\delta \log(n/\delta))$ [Kociumaka, N., Prezza 2021].
  - Similar to the proof for $g = O(b \log(n/b))$.
  - "Pausing" long symbols to avoid them growing too fast.
  - This cannot be achieved with g.
- It also holds $r = O(\delta \log \delta \log(n/\delta))$ [Kociumaka & Kempa 2019] and $z_e = O(\delta \log^2(n/\delta))$ [Kempa & Saha 2022].

More relations on δ

Lower bounds in terms of δ [Kociumaka, N., Prezza 2020]

- There are string families where $\gamma = \Omega(\delta \log n)$.

More relations on δ

Lower bounds in terms of δ [Kociumaka, N., Prezza 2020]

- There are string families where $\gamma = \Omega(\delta \log n)$.
  - On $T = a^n$ where we put bs at positions 2^i.

More relations on δ

Lower bounds in terms of δ [Kociumaka, N., Prezza 2020]

- There are string families where $\gamma = \Omega(\delta \log n)$.
  - On $T = a^n$ where we put bs at positions 2^i.
  - *b b ab aaab aaaaaaab aaaaaaaaaaaaaaab aa...*

More relations on δ

Lower bounds in terms of δ [Kociumaka, N., Prezza 2020]

- There are string families where $\gamma = \Omega(\delta \log n)$.
  - On $T = a^n$ where we put bs at positions 2^i.
  - *b b ab aaab aaaaaaab aaaaaaaaaaaaaaab aa...*
  - It holds $\delta = O(1)$ and $\gamma = \Omega(\log n)$.

More relations on δ

Lower bounds in terms of δ [Kociumaka, N., Prezza 2020]

- There are string families where $\gamma = \Omega(\delta \log n)$.
  - On $T = a^n$ where we put bs at positions 2^i.
  - *b b ab aaab aaaaaaab aaaaaaaaaaaaaaab aa...*
  - It holds $\delta = O(1)$ and $\gamma = \Omega(\log n)$.
- It is impossible to always represent T in $O(\delta)$ space.

More relations on δ

Lower bounds in terms of δ [Kociumaka, N., Prezza 2020]

- There are string families where $\gamma = \Omega(\delta \log n)$.
  - On $T = a^n$ where we put bs at positions 2^i.
  - *b b ab aaab aaaaaaab aaaaaaaaaaaaaaab aa...*
  - It holds $\delta = O(1)$ and $\gamma = \Omega(\log n)$.
- It is impossible to always represent T in $O(\delta)$ space.
  - With a wider family where the positions of bs are perturbed.

More relations on δ

Lower bounds in terms of δ [Kociumaka, N., Prezza 2020]

- There are string families where $\gamma = \Omega(\delta \log n)$.
  - On $T = a^n$ where we put bs at positions 2^i.
  - *b b ab aaab aaaaaaab aaaaaaaaaaaaaaab aa...*
  - It holds $\delta = O(1)$ and $\gamma = \Omega(\log n)$.
- It is impossible to always represent T in $O(\delta)$ space.
  - With a wider family where the positions of bs are perturbed.
  - The family has $2^{\Omega(\log^2 n)}$ elements.

More relations on δ

Lower bounds in terms of δ [Kociumaka, N., Prezza 2020]

- There are string families where $\gamma = \Omega(\delta \log n)$.
  - On $T = a^n$ where we put bs at positions 2^i.
  - *b b ab aaab aaaaaaab aaaaaaaaaaaaaaab aa...*
  - It holds $\delta = O(1)$ and $\gamma = \Omega(\log n)$.
- It is impossible to always represent T in $O(\delta)$ space.
  - With a wider family where the positions of bs are perturbed.
  - The family has $2^{\Omega(\log^2 n)}$ elements.
  - Still $\delta = O(1)$, thus one needs $\Omega(\delta \log n)$ space.

More relations on δ

Lower bounds in terms of δ [Kociumaka, N., Prezza 2020]

- There are string families where $\gamma = \Omega(\delta \log n)$.
  - On $T = a^n$ where we put bs at positions 2^i.
  - *b b ab aaab aaaaaaab aaaaaaaaaaaaaaab aa...*
  - It holds $\delta = O(1)$ and $\gamma = \Omega(\log n)$.
- It is impossible to always represent T in $O(\delta)$ space.
  - With a wider family where the positions of bs are perturbed.
  - The family has $2^{\Omega(\log^2 n)}$ elements.
  - Still $\delta = O(1)$, thus one needs $\Omega(\delta \log n)$ space.
- Furthermore, space $O(\delta \log(n/\delta))$ is tight.

More relations on δ

Lower bounds in terms of δ [Kociumaka, N., Prezza 2020]

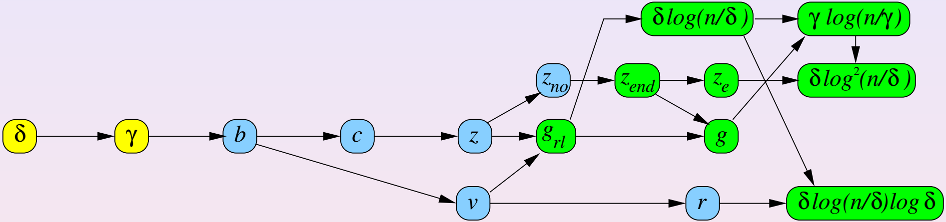
- There are string families where $\gamma = \Omega(\delta \log n)$.
  - On $T = a^n$ where we put bs at positions 2^i.
  - *b b ab aaab aaaaaaab aaaaaaaaaaaaaaab aa...*
  - It holds $\delta = O(1)$ and $\gamma = \Omega(\log n)$.
- It is impossible to always represent T in $O(\delta)$ space.
  - With a wider family where the positions of bs are perturbed.
  - The family has $2^{\Omega(\log^2 n)}$ elements.
  - Still $\delta = O(1)$, thus one needs $\Omega(\delta \log n)$ space.
- Furthermore, space $O(\delta \log(n/\delta))$ is tight.
  - For every n and δ, there are string families requiring that many bits.

More relations on δ

Lower bounds in terms of δ [Kociumaka, N., Prezza 2020]

- There are string families where $\gamma = \Omega(\delta \log n)$.
  - On $T = a^n$ where we put bs at positions 2^i.
  - *b b ab aaab aaaaaaab aaaaaaaaaaaaaaab aa...*
  - It holds $\delta = O(1)$ and $\gamma = \Omega(\log n)$.
- It is impossible to always represent T in $O(\delta)$ space.
  - With a wider family where the positions of bs are perturbed.
  - The family has $2^{\Omega(\log^2 n)}$ elements.
  - Still $\delta = O(1)$, thus one needs $\Omega(\delta \log n)$ space.
- Furthermore, space $O(\delta \log(n/\delta))$ is tight.
  - For every n and δ, there are string families requiring that many bits.
  - Using a generalized variant of the above family.

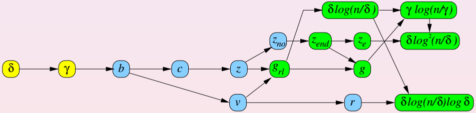
Recap



- $z \approx v \approx$ 1.5–2.5 $\cdot\, \delta$
- $g \approx$ 3–6 $\cdot\, \delta$
- $r \approx$ 7–11 $\cdot\, \delta$

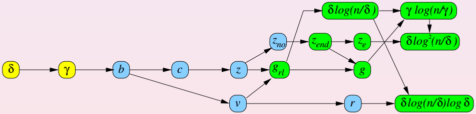
Where we are?

- *b* is the limit of copy-paste representations.



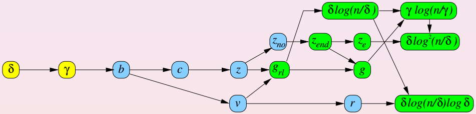
Where we are?

- b is the limit of copy-paste representations.
- g_{rl} and z_{end} are the (known) limits of efficiently accessible representations.



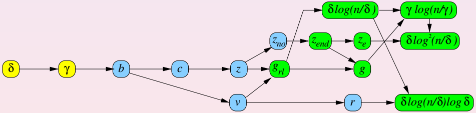
Where we are?

- b is the limit of copy-paste representations.
- g_{rl} and z_{end} are the (known) limits of efficiently accessible representations.
- γ is unreachable via copy-paste, perhaps with other methods?



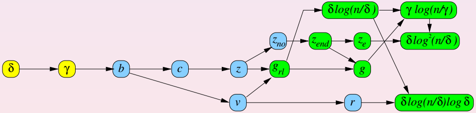
Where we are?

- b is the limit of copy-paste representations.
- g_{rl} and z_{end} are the (known) limits of efficiently accessible representations.
- γ is unreachable via copy-paste, perhaps with other methods?
- δ is unreachable, $\delta \log(n/\delta)$ is accesible.



Where we are?

- b is the limit of copy-paste representations.
- g_{rl} and z_{end} are the (known) limits of efficiently accessible representations.
- γ is unreachable via copy-paste, perhaps with other methods?
- δ is unreachable, $\delta \log(n/\delta)$ is accesible.
- Most measures within δ and $\delta \log(n/\delta)$.



Where we are?

- Can we represent T in $O(\gamma)$ space?

Where we are?

- Can we represent T in $O(\gamma)$ space?
- Can we access T in $O(z_{no})$ or $O(r)$ space?

Where we are?

- Can we represent T in $O(\gamma)$ space?
- Can we access T in $O(z_{no})$ or $O(r)$ space?
- Do we have the right repetitiveness measures?

Where we are?

- Can we represent T in $O(\gamma)$ space?
- Can we access T in $O(z_{no})$ or $O(r)$ space?
- Do we have the right repetitiveness measures?
  - Is at least δ a final lower bound?

Maybe not...

- Iteration of morphisms as a mechanism to capture repetitions.

Maybe not...

- Iteration of morphisms as a mechanism to capture repetitions.
- $0 \to 01$, $1 \to 10$ generates Thue-Morse words

Maybe not...

- Iteration of morphisms as a mechanism to capture repetitions.
- $0 \rightarrow 01$, $1 \rightarrow 10$ generates Thue-Morse words
- 0, 01, 0110, 01101001, ...

Maybe not...

- Iteration of morphisms as a mechanism to capture repetitions.
- $0 \to 01$, $1 \to 10$ generates Thue-Morse words
- 0, 01, 0110, 01101001, ...
- Deterministic Lindenmayer systems are like grammars without terminals.

Maybe not...

- Iteration of morphisms as a mechanism to capture repetitions.
- $0 \to 01$, $1 \to 10$ generates Thue-Morse words
- 0, 01, 0110, 01101001, ...
- Deterministic Lindenmayer systems are like grammars without terminals.
- The production is simply terminated at some level of the derivation tree.

Maybe not...

- Iteration of morphisms as a mechanism to capture repetitions.
- $0 \rightarrow 01$, $1 \rightarrow 10$ generates Thue-Morse words
- 0, 01, 0110, 01101001, ...
- Deterministic Lindenmayer systems are like grammars without terminals.
- The production is simply terminated at some level of the derivation tree.
- They are related with repetitiveness

Maybe not...

- Iteration of morphisms as a mechanism to capture repetitions.
- $0 \rightarrow 01$, $1 \rightarrow 10$ generates Thue-Morse words
- 0, 01, 0110, 01101001, ...
- Deterministic Lindenmayer systems are like grammars without terminals.
- The production is simply terminated at some level of the derivation tree.
- They are related with repetitiveness
  - If all the rules have right-hand sides of the same length...

Maybe not...

- Iteration of morphisms as a mechanism to capture repetitions.
- $0 \rightarrow 01$, $1 \rightarrow 10$ generates Thue-Morse words
- 0, 01, 0110, 01101001, ...
- Deterministic Lindenmayer systems are like grammars without terminals.
- The production is simply terminated at some level of the derivation tree.
- They are related with repetitiveness
  - If all the rules have right-hand sides of the same length...
  - Then the string has $\gamma = O(\log n)$ [Shallit 2020].

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.
- Plus a desired pruning depth and string length.

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.
- Plus a desired pruning depth and string length.
- Let ℓ be the size of the smallest L-system generating T.

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.
- Plus a desired pruning depth and string length.
- Let ℓ be the size of the smallest L-system generating T.
- We only have the upper bound $\ell = O(g)$.

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.
- Plus a desired pruning depth and string length.
- Let ℓ be the size of the smallest L-system generating T.
- We only have the upper bound $\ell = O(g)$.
- On Thue-Morse words:

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.
- Plus a desired pruning depth and string length.
- Let ℓ be the size of the smallest L-system generating T.
- We only have the upper bound $\ell = O(g)$.
- On Thue-Morse words:
  - $b = \Theta(\log n)$, $\gamma = O(1)$,

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.
- Plus a desired pruning depth and string length.
- Let ℓ be the size of the smallest L-system generating T.
- We only have the upper bound $\ell = O(g)$.
- On Thue-Morse words:
  - $b = \Theta(\log n)$, $\gamma = O(1)$,
  - $\ell = O(1)$ beats any cut-and-paste method

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.
- Plus a desired pruning depth and string length.
- Let ℓ be the size of the smallest L-system generating T.
- We only have the upper bound $\ell = O(g)$.
- On Thue-Morse words:
  - $b = \Theta(\log n)$, $\gamma = O(1)$,
  - $\ell = O(1)$ beats any cut-and-paste method
- In some cases, $\ell = \Omega(\delta \log n)$ because it is reachable.

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.
- Plus a desired pruning depth and string length.
- Let ℓ be the size of the smallest L-system generating T.
- We only have the upper bound $\ell = O(g)$.
- On Thue-Morse words:
  - $b = \Theta(\log n)$, $\gamma = O(1)$,
  - $\ell = O(1)$ beats any cut-and-paste method
- In some cases, $\ell = \Omega(\delta \log n)$ because it is reachable.
- But it might also be that $\delta = \Omega(\ell \log n)$:

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.
- Plus a desired pruning depth and string length.
- Let ℓ be the size of the smallest L-system generating T.
- We only have the upper bound $\ell = O(g)$.
- On Thue-Morse words:
  - $b = \Theta(\log n)$, $\gamma = O(1)$,
  - $\ell = O(1)$ beats any cut-and-paste method
- In some cases, $\ell = \Omega(\delta \log n)$ because it is reachable.
- But it might also be that $\delta = \Omega(\ell \log n)$:
  - Initial symbol 0, rules $0 \rightarrow 001$ and $1 \rightarrow 1$.

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.
- Plus a desired pruning depth and string length.
- Let ℓ be the size of the smallest L-system generating T.
- We only have the upper bound $\ell = O(g)$.
- On Thue-Morse words:
  - $b = \Theta(\log n)$, $\gamma = O(1)$,
  - $\ell = O(1)$ beats any cut-and-paste method
- In some cases, $\ell = \Omega(\delta \log n)$ because it is reachable.
- But it might also be that $\delta = \Omega(\ell \log n)$:
  - Initial symbol 0, rules $0 \rightarrow 001$ and $1 \rightarrow 1$.
  - 0, 001, 0010011, 001001100100111, ...

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.
- Plus a desired pruning depth and string length.
- Let ℓ be the size of the smallest L-system generating T.
- We only have the upper bound $\ell = O(g)$.
- On Thue-Morse words:
  - $b = \Theta(\log n)$, $\gamma = O(1)$,
  - $\ell = O(1)$ beats any cut-and-paste method
- In some cases, $\ell = \Omega(\delta \log n)$ because it is reachable.
- But it might also be that $\delta = \Omega(\ell \log n)$:
  - Initial symbol 0, rules $0 \rightarrow 001$ and $1 \rightarrow 1$.
  - 0, 001, 0010011, 001001100100111, ...
  - It is shown to have $\delta = \Omega(\log n)$, with $\ell = O(1)$.

L-systems [N. & Urbina, 2021]

- A deterministic Lindenmayer system.
- Plus a desired pruning depth and string length.
- Let ℓ be the size of the smallest L-system generating T.
- We only have the upper bound $\ell = O(g)$.
- On Thue-Morse words:
  - $b = \Theta(\log n)$, $\gamma = O(1)$,
  - $\ell = O(1)$ beats any cut-and-paste method
- In some cases, $\ell = \Omega(\delta \log n)$ because it is reachable.
- But it might also be that $\delta = \Omega(\ell \log n)$:
  - Initial symbol 0, rules $0 \rightarrow 001$ and $1 \rightarrow 1$.
  - 0, 001, 0010011, 001001100100111, ...
  - It is shown to have $\delta = \Omega(\log n)$, with $\ell = O(1)$.
  - Maybe ℓ captures better some repetitive structured texts?

Thanks for your attention!