Avoiding Additive Powers in Words

Lucas Mol

THOMPSON RIVERS UNIVERSITY

Joint work with James Currie, Narad Rampersad, and Jeffrey Shallit

CanaDAM 2023
Winnipeg, Manitoba

Plan

Power Avoidance

Decision Algorithms

Outlook

Alphabets and words

- An alphabet is a finite set of letters, e.g.,
  - $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ (the English alphabet)
  - $\{\mathtt{0}, \mathtt{1}\}$ (the binary alphabet)
  - $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ (the alphabet of DNA strings)

ALPHABETS AND WORDS

- An alphabet is a finite set of letters, e.g.,
  - $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ (the English alphabet)
  - $\{\mathtt{0}, \mathtt{1}\}$ (the binary alphabet)
  - $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ (the alphabet of DNA strings)
- A word is a sequence of letters taken from some alphabet, e.g.,
  - `apple`, `banana`, `clementine` (English words)
  - `0110100110010110` (a binary word)
  - `AAGATGCCGT` (a DNA string)

ALPHABETS AND WORDS

- An alphabet is a finite set of letters, e.g.,
  - $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ (the English alphabet)
  - $\{\mathtt{0}, \mathtt{1}\}$ (the binary alphabet)
  - $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ (the alphabet of DNA strings)
- A word is a sequence of letters taken from some alphabet, e.g.,
  - `apple`, `banana`, `clementine` (English words)
  - `0110100110010110` (a binary word)
  - `AAGATGCCGT` (a DNA string)
- We are mostly interested in long words over small alphabets.

ALPHABETS AND WORDS

- An alphabet is a finite set of letters, e.g.,
  - $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$ (the English alphabet)
  - $\{\mathtt{0}, \mathtt{1}\}$ (the binary alphabet)
  - $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ (the alphabet of DNA strings)
- A word is a sequence of letters taken from some alphabet, e.g.,
  - `apple`, `banana`, `clementine` (English words)
  - `0110100110010110` (a binary word)
  - `AAGATGCCGT` (a DNA string)
- We are mostly interested in long words over small alphabets.
- Which patterns can be avoided, and which patterns must inevitably occur?

SQUARES

- A square is a word of the form xx, e.g.,

SQUARES

- A square is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`

SQUARES

- A square is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`

SQUARES

- A square is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- A word is square-free if it contains no squares as factors.

SQUARES

- A square is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- A word is square-free if it contains no squares as factors.
  - `apple`

SQUARES

- A square is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- A word is square-free if it contains no squares as factors.
  - `apple` – not square-free

SQUARES

- A square is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- A word is square-free if it contains no squares as factors.
  - `apple` – not square-free
  - `banana`

SQUARES

- A square is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- A word is square-free if it contains no squares as factors.
  - `apple` – not square-free
  - `banana` – not square-free

Squares

- A square is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- A word is square-free if it contains no squares as factors.
  - `apple` – not square-free
  - `banana` – not square-free
  - `clementine`

SQUARES

- A square is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- A word is square-free if it contains no squares as factors.
  - `apple` – not square-free
  - `banana` – not square-free
  - `clementine` – square-free

SQUARES

- A square is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- A word is square-free if it contains no squares as factors.
  - `apple` – not square-free
  - `banana` – not square-free
  - `clementine` – square-free
- Question: Are squares avoidable over some finite alphabet?
  - That is, are there arbitrarily long square-free words over some finite alphabet?

A CONSTRUCTION

A CONSTRUCTION

- Define a map h by
  - $h(\texttt{0}) = \texttt{012}$,
  - $h(\texttt{1}) = \texttt{02}$, and
  - $h(\texttt{2}) = \texttt{1}$.

A CONSTRUCTION

- Define a map *h* by
  - $h(\mathtt{0}) = \mathtt{012}$,
  - $h(\mathtt{1}) = \mathtt{02}$, and
  - $h(\mathtt{2}) = \mathtt{1}$.
- Extend *h* to all words over $\{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$ in the obvious way:

$$h(\mathtt{0120}) = h(\mathtt{0})h(\mathtt{1})h(\mathtt{2})h(\mathtt{0}) = \mathtt{012021012}$$

A CONSTRUCTION

- Define a map h by
  - $h(\mathtt{0}) = \mathtt{012}$,
  - $h(\mathtt{1}) = \mathtt{02}$, and
  - $h(\mathtt{2}) = \mathtt{1}$.
- Extend h to all words over $\{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$ in the obvious way:

$$h(\mathtt{0120}) = h(\mathtt{0})h(\mathtt{1})h(\mathtt{2})h(\mathtt{0}) = \mathtt{012021012}$$

- We start with $\mathtt{0}$, and repeatedly apply h.

A CONSTRUCTION

- Define a map h by
  - $h(\mathtt{0}) = \mathtt{012}$,
  - $h(\mathtt{1}) = \mathtt{02}$, and
  - $h(\mathtt{2}) = \mathtt{1}$.
- Extend h to all words over $\{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$ in the obvious way:

$$h(\mathtt{0120}) = h(\mathtt{0})h(\mathtt{1})h(\mathtt{2})h(\mathtt{0}) = \mathtt{012021012}$$

- We start with $\mathtt{0}$, and repeatedly apply h.

$$h(\mathtt{0}) = \mathtt{012}$$

A CONSTRUCTION

- Define a map h by
  - $h(\mathtt{0}) = \mathtt{012}$,
  - $h(\mathtt{1}) = \mathtt{02}$, and
  - $h(\mathtt{2}) = \mathtt{1}$.
- Extend h to all words over $\{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$ in the obvious way:

$$h(\mathtt{0120}) = h(\mathtt{0})h(\mathtt{1})h(\mathtt{2})h(\mathtt{0}) = \mathtt{012021012}$$

- We start with $\mathtt{0}$, and repeatedly apply h.

$$h(\mathtt{0}) = \mathtt{012}$$
$$h^2(\mathtt{0}) = \mathtt{012021}$$

A CONSTRUCTION

- Define a map h by
  - $h(\mathtt{0}) = \mathtt{012}$,
  - $h(\mathtt{1}) = \mathtt{02}$, and
  - $h(\mathtt{2}) = \mathtt{1}$.
- Extend h to all words over $\{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$ in the obvious way:

$$h(\mathtt{0120}) = h(\mathtt{0})h(\mathtt{1})h(\mathtt{2})h(\mathtt{0}) = \mathtt{012021012}$$

- We start with $\mathtt{0}$, and repeatedly apply h.

$$h(\mathtt{0}) = \mathtt{012}$$
$$h^2(\mathtt{0}) = \mathtt{012021}$$
$$h^3(\mathtt{0}) = \mathtt{012021012102}$$

A CONSTRUCTION

- Define a map h by
  - $h(\mathtt{0}) = \mathtt{012}$,
  - $h(\mathtt{1}) = \mathtt{02}$, and
  - $h(\mathtt{2}) = \mathtt{1}$.
- Extend h to all words over $\{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$ in the obvious way:

$$h(\mathtt{0120}) = h(\mathtt{0})h(\mathtt{1})h(\mathtt{2})h(\mathtt{0}) = \mathtt{012021012}$$

- We start with $\mathtt{0}$, and repeatedly apply h.

$$\begin{aligned} h(\mathtt{0}) &= \mathtt{012} \\ h^2(\mathtt{0}) &= \mathtt{012021} \\ h^3(\mathtt{0}) &= \mathtt{012021012102} \\ h^4(\mathtt{0}) &= \mathtt{012021012102012021020121} \end{aligned}$$

A CONSTRUCTION

- Define a map h by
  - $h(\mathtt{0}) = \mathtt{012}$,
  - $h(\mathtt{1}) = \mathtt{02}$, and
  - $h(\mathtt{2}) = \mathtt{1}$.
- Extend h to all words over $\{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$ in the obvious way:

$$h(\mathtt{0120}) = h(\mathtt{0})h(\mathtt{1})h(\mathtt{2})h(\mathtt{0}) = \mathtt{012021012}$$

- We start with $\mathtt{0}$, and repeatedly apply h.

$$
\begin{aligned}
h(\mathtt{0}) &= \mathtt{012} \\
h^2(\mathtt{0}) &= \mathtt{012021} \\
h^3(\mathtt{0}) &= \mathtt{012021012102} \\
h^4(\mathtt{0}) &= \mathtt{012021012102012021020121} \\
&\vdots \\
h^\omega(\mathtt{0}) &= \mathtt{012021012102012021020121}\cdots
\end{aligned}
$$

The origin of combinatorics on words

Theorem: $h^{\omega}(0) = \mathtt{012021012102012021020121}\ldots$ is square-free.

Axel Thue (1863-1922)

Abelian and Additive Squares

An abelian square is a word of the form $x\tilde{x}$, where $\tilde{x}$ is an *anagram* of x.

- Examples: `mesosome`, `reappear`, `intestines`

Abelian and Additive Squares

An abelian square is a word of the form $x\tilde{x}$, where $\tilde{x}$ is an *anagram* of x.

- Examples: `mesosome`, `reappear`, `intestines`
- Question (Erdős 1961): Are abelian squares avoidable over some finite alphabet?

Abelian and Additive Squares

An abelian square is a word of the form $x\tilde{x}$, where $\tilde{x}$ is an *anagram* of x.

- Examples: `mesosome`, `reappear`, `intestines`
- Question (Erdős 1961): Are abelian squares avoidable over some finite alphabet?
- Theorem (Keränen 1992): $\sigma^\omega(\mathtt{0})$ avoids abelian squares, where

$$\sigma(0) = \mathtt{0120232123203231301020103101213121021232021013010203212320231210212320232132303132120}$$
$$\sigma(1) = \mathtt{1231303230310302012131210212320232132303132120121310323031302321323031303203010203231}$$
$$\sigma(2) = \mathtt{2302010301021013123202321323031303203010203231232021030102013032030102010310121310302}$$
$$\sigma(3) = \mathtt{3013121012132120230313032030102010310121310302303132101213120103101213121021232021013}$$

Abelian and Additive Squares

An abelian square is a word of the form $x\tilde{x}$, where $\tilde{x}$ is an *anagram* of x.

- Examples: `mesosome`, `reappear`, `intestines`
- Question (Erdős 1961): Are abelian squares avoidable over some finite alphabet?
- Theorem (Keränen 1992): $\sigma^{\omega}(\mathtt{0})$ avoids abelian squares, where

$\sigma(0) = 0120232123203231301020103101213121021232021013010203212320231210212320232132303132120$
$\sigma(1) = 1231303230310302012131210212320232132303132120121310323031302321323031303203010203231$
$\sigma(2) = 2302010301021013123202321323031303203010203231232021030102013032030102010310121310302$
$\sigma(3) = 3013121012132120230313032030102010310121310302303132101213120103101213121021232021013$

An additive square is a word of the form $x\tilde{x}$, where x and $\tilde{x}$ have the same length and the same sum.

- Examples: `012012`, `012021`, `013202`

ABELIAN AND ADDITIVE SQUARES

An abelian square is a word of the form $x\tilde{x}$, where $\tilde{x}$ is an *anagram* of x.

- Examples: `mesosome`, `reappear`, `intestines`
- Question (Erdős 1961): Are abelian squares avoidable over some finite alphabet?
- Theorem (Keränen 1992): $\sigma^\omega(\mathtt{0})$ avoids abelian squares, where

$$\sigma(0) = \mathtt{0120232123203231301020103101213121021232021013010203212320231210212320232132303132120}$$
$$\sigma(1) = \mathtt{1231303230310302012131210212320232132303132120121310323031302321323031303203010203231}$$
$$\sigma(2) = \mathtt{2302010301021013123202321323031303203010203231232021030102013032030102010310121310302}$$
$$\sigma(3) = \mathtt{3013121012132120230313032030102010310121310302303132101213120103101213121021232021013}$$

An additive square is a word of the form $x\tilde{x}$, where x and $\tilde{x}$ have the same length and the same sum.

- Examples: `012012`, `012021`, `013202`
- Question (Justin 1972): Are additive squares avoidable over some finite subset of $\mathbb{Z}$?

Abelian and Additive Squares

An abelian square is a word of the form $x\tilde{x}$, where $\tilde{x}$ is an *anagram* of x.

- Examples: `mesosome`, `reappear`, `intestines`
- Question (Erdős 1961): Are abelian squares avoidable over some finite alphabet?
- Theorem (Keränen 1992): $\sigma^\omega(\mathtt{0})$ avoids abelian squares, where

$$\sigma(0) = \mathtt{0120232123203231301020103101213121021232021013010203212320231210212320232132303132120}$$
$$\sigma(1) = \mathtt{1231303230310302012131210212320232132303132120121310323031302321323031303203010203231}$$
$$\sigma(2) = \mathtt{2302010301021013123202321323031303203010203231232021030102013032030102010310121310302}$$
$$\sigma(3) = \mathtt{3013121012132120230313032030102010310121310302303132101213120103101213121021232021013}$$

An additive square is a word of the form $x\tilde{x}$, where x and $\tilde{x}$ have the same length and the same sum.

- Examples: `012012`, `012021`, `013202`
- Question (Justin 1972): Are additive squares avoidable over some finite subset of $\mathbb{Z}$?
  - We don't know!

Some Progress

Theorem (Cassaigne, Currie, Schaeffer and Shallit 2014):
Additive cubes are avoidable over $\{0, 1, 3, 4\}$.

Some Progress

Theorem (Cassaigne, Currie, Schaeffer and Shallit 2014):
Additive cubes are avoidable over $\{0, 1, 3, 4\}$.

- The word

$$h^{\omega}(\texttt{0}) = \texttt{031430110343430310110110314 3} \cdots$$

avoids additive cubes, where

$$h(\texttt{0}) = \texttt{03}$$
$$h(\texttt{1}) = \texttt{43}$$
$$h(\texttt{3}) = \texttt{1}$$
$$h(\texttt{4}) = \texttt{01}$$

Some Progress

Theorem (Cassaigne, Currie, Schaeffer and Shallit 2014): Additive cubes are avoidable over $\{0, 1, 3, 4\}$.

- The word

$$h^{\omega}(\mathtt{0}) = \mathtt{031430110343430310110110314 3}\cdots$$

avoids additive cubes, where

$$\begin{aligned} h(\mathtt{0}) &= \mathtt{03} \\ h(\mathtt{1}) &= \mathtt{43} \\ h(\mathtt{3}) &= \mathtt{1} \\ h(\mathtt{4}) &= \mathtt{01} \end{aligned}$$

Theorem (Rao and Rosenfeld 2018): Additive squares are avoidable over a finite subset of $\mathbb{Z}^2$.

PLAN

POWER AVOIDANCE

DECISION ALGORITHMS

OUTLOOK

Decision Algorithms

Theorem: There is an algorithm which decides, under certain conditions on h, whether $h^{\omega}(\mathtt{0})$ contains

Decision Algorithms

Theorem: There is an algorithm which decides, under certain conditions on h, whether $h^\omega(\mathtt{0})$ contains

- regular powers (Cassaigne 1993)

DECISION ALGORITHMS

Theorem: There is an algorithm which decides, under certain conditions on h, whether $h^\omega(\texttt{0})$ contains

- regular powers (Cassaigne 1993)
- abelian powers (Currie and Rampersad 2012)

Decision Algorithms

Theorem: There is an algorithm which decides, under certain conditions on h, whether $h^\omega(\mathtt{0})$ contains

- regular powers (Cassaigne 1993)
- abelian powers (Currie and Rampersad 2012)
- additive powers (Rao and Rosenfeld 2018)

Decision Algorithms

Theorem: There is an algorithm which decides, under certain conditions on h, whether $h^{\omega}(\mathtt{0})$ contains

- regular powers (Cassaigne 1993)
- abelian powers (Currie and Rampersad 2012)
- additive powers (Rao and Rosenfeld 2018)

Theorem (Currie, Mol, Rampersad, and Shallit 2021+): A more efficient algorithm for additive powers (with stronger conditions on h).

DECISION ALGORITHMS

Theorem: There is an algorithm which decides, under certain conditions on h, whether $h^\omega(\mathtt{0})$ contains

- regular powers (Cassaigne 1993)
- abelian powers (Currie and Rampersad 2012)
- additive powers (Rao and Rosenfeld 2018)

Theorem (Currie, Mol, Rampersad, and Shallit 2021+): A more efficient algorithm for additive powers (with stronger conditions on h).

- The algorithm is easy to implement.

DECISION ALGORITHMS

Theorem: There is an algorithm which decides, under certain conditions on h, whether $h^{\omega}(\mathtt{0})$ contains

- regular powers (Cassaigne 1993)
- abelian powers (Currie and Rampersad 2012)
- additive powers (Rao and Rosenfeld 2018)

Theorem (Currie, Mol, Rampersad, and Shallit 2021+): A more efficient algorithm for additive powers (with stronger conditions on h).

- The algorithm is easy to implement.
- It is efficient enough to work in practice, even for “long” substitutions!

Example

- Define f by

$$\begin{aligned} f(\mathtt{0}) &= \mathtt{001} \\ f(\mathtt{1}) &= \mathtt{012} \\ f(\mathtt{2}) &= \mathtt{212} \end{aligned}$$

Example

- Define f by

$$f(\mathtt{0}) = \mathtt{001}$$
$$f(\mathtt{1}) = \mathtt{012}$$
$$f(\mathtt{2}) = \mathtt{212}$$

- Then

$$f^{\omega}(\mathtt{0}) = \mathtt{001001012001001012001012212}\cdots$$

is additive 4th-power-free.

EXAMPLE

- Define f by

$$f(\mathtt{0}) = \mathtt{001}$$
$$f(\mathtt{1}) = \mathtt{012}$$
$$f(\mathtt{2}) = \mathtt{212}$$

- Then

$$f^{\omega}(\mathtt{0}) = \mathtt{001001012001001012001012212}\cdots$$

  is additive 4th-power-free.
- We can establish this fact by simply running our algorithm.

Example

- Define f by

$$f(\texttt{0}) = \texttt{001}$$
$$f(\texttt{1}) = \texttt{012}$$
$$f(\texttt{2}) = \texttt{212}$$

- Then

$$f^{\omega}(\texttt{0}) = \texttt{001001012001001012001012212}\cdots$$

  is additive 4th-power-free.
- We can establish this fact by simply running our algorithm.
- In fact, $g(f^{\omega}(0))$ is additive 4th-power-free, where

$$g(\texttt{0}) = \texttt{0001001110010001100011}$$
$$g(\texttt{1}) = \texttt{0001001110011101100011}$$
$$g(\texttt{2}) = \texttt{0111001110011101100011}$$

The Main Idea of the Algorithm

THE MAIN IDEA OF THE ALGORITHM

- Any long additive power in $h^\omega(\mathtt{0})$ must have arisen by applying h repeatedly to some short "seed word".

The Main Idea of the Algorithm

- Any long additive power in $h^{\omega}(\mathtt{0})$ must have arisen by applying h repeatedly to some short "seed word".
- Show that these seed words cannot look "too different" from additive powers – there are only finitely many possible *templates* for these seed words.

THE MAIN IDEA OF THE ALGORITHM

- Any long additive power in $h^\omega(\texttt{0})$ must have arisen by applying h repeatedly to some short "seed word".
- Show that these seed words cannot look "too different" from additive powers – there are only finitely many possible *templates* for these seed words.
- Enumerate all short words in $h^\omega(\texttt{0})$, and check to see if they match any of the templates.

The Main Idea of the Algorithm

- Any long additive power in $h^\omega(\texttt{0})$ must have arisen by applying h repeatedly to some short "seed word".
- Show that these seed words cannot look "too different" from additive powers – there are only finitely many possible *templates* for these seed words.
- Enumerate all short words in $h^\omega(\texttt{0})$, and check to see if they match any of the templates.
- Our stronger conditions on h allow us to greatly reduce the number of templates that need to be checked.

Plan

Power Avoidance

Decision Algorithms

Outlook

OUTLOOK

We still don't have a construction of additive square-free words over a finite subset of $\mathbb{Z}$.

Outlook

We still don't have a construction of additive square-free words over a finite subset of $\mathbb{Z}$.

- If we do find a candidate construction $h^{\omega}(\texttt{0})$, then we just need to run our algorithm to prove it!

Outlook

We still don't have a construction of additive square-free words over a finite subset of $\mathbb{Z}$.

- If we do find a candidate construction $h^\omega(\mathtt{0})$, then we just need to run our algorithm to prove it!
- The conditions of our theorem are fairly restrictive.

Outlook

We still don't have a construction of additive square-free words over a finite subset of $\mathbb{Z}$.

- If we do find a candidate construction $h^{\omega}(\texttt{0})$, then we just need to run our algorithm to prove it!
- The conditions of our theorem are fairly restrictive.
- Our algorithm is (much) more efficient than the earlier one.

Outlook

We still don't have a construction of additive square-free words over a finite subset of $\mathbb{Z}$.

- If we do find a candidate construction $h^{\omega}(\texttt{0})$, then we just need to run our algorithm to prove it!
- The conditions of our theorem are fairly restrictive.
- Our algorithm is (much) more efficient than the earlier one.

Hopefully this method will prove useful!


Thank you!

Proof Sketch

Theorem (Currie, Mol, Rampersad, and Shallit 2021+): There is an algorithm which decides, under certain conditions on h, whether $h^\omega(\mathtt{0})$ contains additive squares.

Proof Sketch

Theorem (Currie, Mol, Rampersad, and Shallit 2021+): There is an algorithm which decides, under certain conditions on h, whether $h^{\omega}(\texttt{0})$ contains additive squares.

Some questions:

PROOF SKETCH

Theorem (Currie, Mol, Rampersad, and Shallit 2021+): There is an algorithm which decides, under certain conditions on h, whether $h^\omega(\texttt{0})$ contains additive squares.

Some questions:

- What are these conditions?

Proof Sketch

Theorem (Currie, Mol, Rampersad, and Shallit 2021+): There is an algorithm which decides, under certain conditions on h, whether $h^\omega(\mathtt{0})$ contains additive squares.

Some questions:

- What are these conditions?
- How do we describe the "seed words" for additive squares?

PROOF SKETCH

Theorem (Currie, Mol, Rampersad, and Shallit 2021+): There is an algorithm which decides, under certain conditions on h, whether $h^\omega(\mathtt{0})$ contains additive squares.

Some questions:

- What are these conditions?
- How do we describe the "seed words" for additive squares?

Let's find out by sketching the proof.

TEMPLATES

TEMPLATES

- Let $\vec{\sigma}(w)$ denote the vector $\begin{bmatrix} \text{length of } w \\ \text{sum of } w \end{bmatrix}$.

Templates

- Let $\vec{\sigma}(w)$ denote the vector $\begin{bmatrix}\text{length of } w \\ \text{sum of } w\end{bmatrix}$.
- A template is a 4-tuple

$$[a_0, a_1, a_2, \vec{d}]$$

letters or ε vector in $\mathbb{Z}^2$

Templates

- Let $\vec{\sigma}(w)$ denote the vector $\begin{bmatrix} \text{length of } w \\ \text{sum of } w \end{bmatrix}$.
- A template is a 4-tuple

$$[a_0, a_1, a_2, \vec{d}]$$

letters or ε vector in $\mathbb{Z}^2$

- A word w is an instance of this template if

$$w = a_0 w_0 a_1 w_1 a_2 \quad \text{and} \quad \vec{\sigma}(w_1) - \vec{\sigma}(w_0) = \vec{d}.$$

Templates

- Let $\vec{\sigma}(w)$ denote the vector $\begin{bmatrix} \text{length of } w \\ \text{sum of } w \end{bmatrix}$.
- A template is a 4-tuple

$$[a_0, a_1, a_2, \vec{d}]$$

letters or ε vector in $\mathbb{Z}^2$

- A word w is an instance of this template if

$$w = a_0 w_0 a_1 w_1 a_2 \quad \text{and} \quad \vec{\sigma}(w_1) - \vec{\sigma}(w_0) = \vec{d}.$$

- An instance of $[\varepsilon, \varepsilon, \varepsilon, \vec{0}]$ is an additive square!

Templates

- Let $\vec{\sigma}(w)$ denote the vector $\begin{bmatrix} \text{length of } w \\ \text{sum of } w \end{bmatrix}$.
- A template is a 4-tuple

$$[a_0, a_1, a_2, \vec{d}]$$

letters or ε vector in $\mathbb{Z}^2$

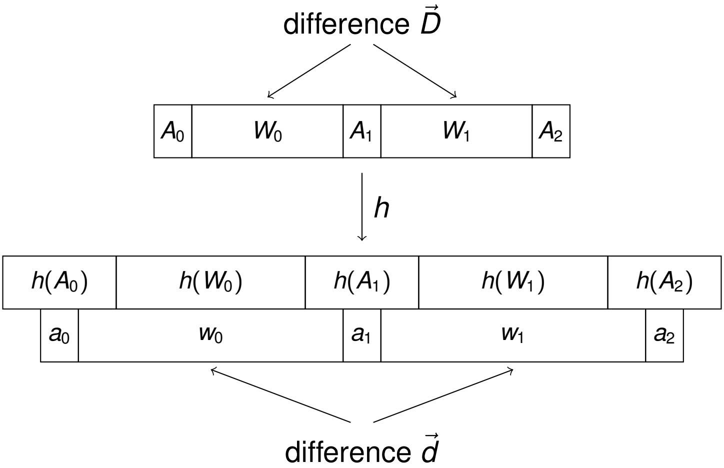
- A word w is an instance of this template if

$$w = a_0 w_0 a_1 w_1 a_2 \quad \text{and} \quad \vec{\sigma}(w_1) - \vec{\sigma}(w_0) = \vec{d}.$$

- An instance of $[\varepsilon, \varepsilon, \varepsilon, \vec{0}]$ is an additive square!
- An instance of $[0, 1, 0, [1, 3]^T]$ is "not too far" from an additive square.

PARENTS

Every long-enough instance of a template must have come from an instance of another template – a parent.



The First Two Conditions

- Condition 1: For all letters x,
  - the length of $h(x)$ is given by $ax + b$ for some $a, b \in \mathbb{Z}$, and
  - the sum of $h(x)$ is given by $cx + d$ for some $c, d \in \mathbb{Z}$.

The First Two Conditions

- Condition 1: For all letters x,
  - the length of $h(x)$ is given by $ax + b$ for some $a, b \in \mathbb{Z}$, and
  - the sum of $h(x)$ is given by $cx + d$ for some $c, d \in \mathbb{Z}$.
- Record this in the matrix $M_h = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$.

The First Two Conditions

- Condition 1: For all letters x,
  - the length of $h(x)$ is given by $ax + b$ for some $a, b \in \mathbb{Z}$, and
  - the sum of $h(x)$ is given by $cx + d$ for some $c, d \in \mathbb{Z}$.
- Record this in the matrix $M_h = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$.
- Then $\vec{\sigma}(h(W)) = M_h \vec{\sigma}(W)$.

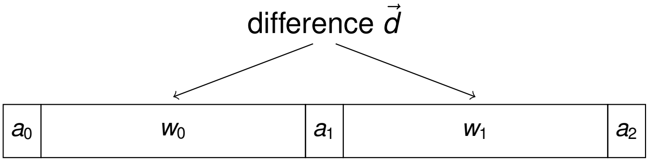
THE FIRST TWO CONDITIONS

- Condition 1: For all letters x,
  - the length of $h(x)$ is given by $ax + b$ for some $a, b \in \mathbb{Z}$, and
  - the sum of $h(x)$ is given by $cx + d$ for some $c, d \in \mathbb{Z}$.
- Record this in the matrix $M_h = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$.
- Then $\vec{\sigma}(h(W)) = M_h \vec{\sigma}(W)$.
- Condition 2: M_h is invertible, so that

$$\vec{\sigma}(W) = M_h^{-1} \vec{\sigma}(h(W)).$$

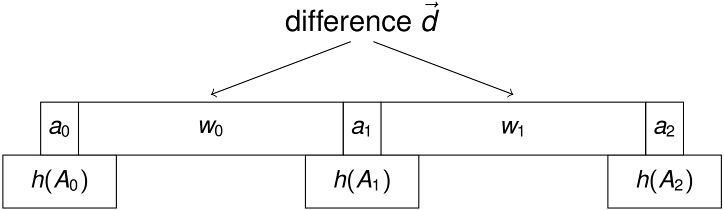
Finding Parents

These first two conditions allow us to find all possible parents of a given template.



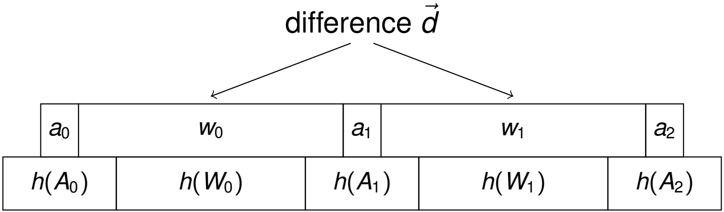
Finding Parents

These first two conditions allow us to find all possible parents of a given template.



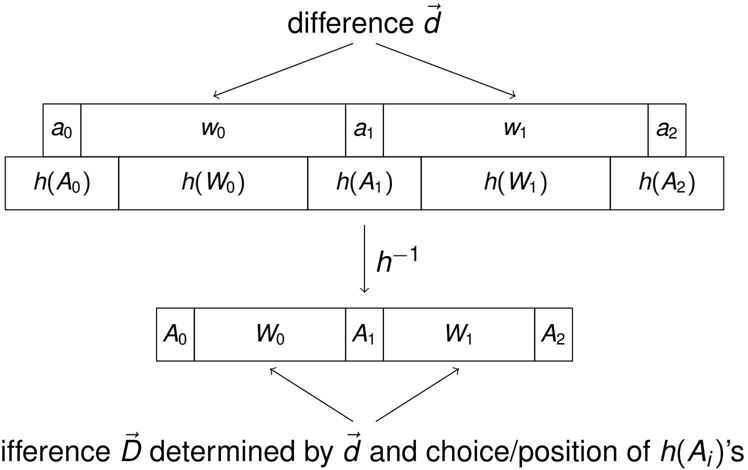
Finding Parents

These first two conditions allow us to find all possible parents of a given template.



Finding Parents

These first two conditions allow us to find all possible parents of a given template.



The Third Condition

- Now that we can compute the parents of a given template t, we want to compute the set of all ancestors of t (parents, grandparents, great-grandparents, etc.)

The Third Condition

- Now that we can compute the parents of a given template t, we want to compute the set of all ancestors of t (parents, grandparents, great-grandparents, etc.)
- How do we know that this set is finite?

THE THIRD CONDITION

- Now that we can compute the parents of a given template t, we want to compute the set of all ancestors of t (parents, grandparents, great-grandparents, etc.)
- How do we know that this set is finite?
- We need a condition on h which guarantees that for any ancestor $T = [A_0, A_1, A_2, \vec{D}]$ of t, the difference $\vec{D}$ is not too large.

The Third Condition

- Now that we can compute the parents of a given template t, we want to compute the set of all ancestors of t (parents, grandparents, great-grandparents, etc.)
- How do we know that this set is finite?
- We need a condition on h which guarantees that for any ancestor $T = [A_0, A_1, A_2, \vec{D}]$ of t, the difference $\vec{D}$ is not too large.
- Condition 3: All eigenvalues of M_h are larger than 1 in absolute value.

The Last Condition

- Condition 4: For all letters x, the length of $h(x)$ is at least 2.

The Last Condition

- Condition 4: For all letters x, the length of $h(x)$ is at least 2.
- So taking preimages makes words shorter!

THE LAST CONDITION

- Condition 4: For all letters x, the length of $h(x)$ is at least 2.
- So taking preimages makes words shorter!
- So if $h^\omega(0)$ contains an instance of a template t, then $h^\omega(0)$ contains a *short* instance of some ancestor of t.

Description of the Algorithm

Suppose that h satisfies these four conditions.

DESCRIPTION OF THE ALGORITHM

Suppose that *h* satisfies these four conditions.

- Consider the template $t = [\varepsilon, \varepsilon, \varepsilon, \vec{0}]$.

Description of the Algorithm

Suppose that h satisfies these four conditions.

- Consider the template $t = [\varepsilon, \varepsilon, \varepsilon, \vec{0}]$.
  - An instance of t is an additive square.

DESCRIPTION OF THE ALGORITHM

Suppose that *h* satisfies these four conditions.

- Consider the template $t = [\varepsilon, \varepsilon, \varepsilon, \vec{0}]$.
    - An instance of *t* is an additive square.
- We enumerate all ancestors of *t*.

Description of the Algorithm

Suppose that *h* satisfies these four conditions.

- Consider the template $t = [\varepsilon, \varepsilon, \varepsilon, \vec{0}]$.
  - An instance of t is an additive square.
- We enumerate all ancestors of t.
  - This set is finite!

Description of the Algorithm

Suppose that *h* satisfies these four conditions.

- Consider the template $t = [\varepsilon, \varepsilon, \varepsilon, \vec{0}]$.
  - An instance of *t* is an additive square.
- We enumerate all ancestors of *t*.
  - This set is finite!
- If $h^\omega(0)$ contains an additive square, then it must contain a *short* instance of one of these ancestors.

Description of the Algorithm

Suppose that h satisfies these four conditions.

- Consider the template $t = [\varepsilon, \varepsilon, \varepsilon, \vec{0}]$.
  - An instance of t is an additive square.
- We enumerate all ancestors of t.
  - This set is finite!
- If $h^\omega(0)$ contains an additive square, then it must contain a *short* instance of one of these ancestors.
  - These are our potential “seed words”.

DESCRIPTION OF THE ALGORITHM

Suppose that h satisfies these four conditions.

- Consider the template $t = [\varepsilon, \varepsilon, \varepsilon, \vec{0}]$.
  - An instance of t is an additive square.
- We enumerate all ancestors of t.
  - This set is finite!
- If $h^\omega(0)$ contains an additive square, then it must contain a *short* instance of one of these ancestors.
  - These are our potential "seed words".
- We enumerate all short factors of $h^\omega(0)$, and check to see if any of them is an instance of an ancestor of t.

DESCRIPTION OF THE ALGORITHM

Suppose that h satisfies these four conditions.

- Consider the template $t = [\varepsilon, \varepsilon, \varepsilon, \vec{0}]$.
  - An instance of t is an additive square.
- We enumerate all ancestors of t.
  - This set is finite!
- If $h^\omega(0)$ contains an additive square, then it must contain a *short* instance of one of these ancestors.
  - These are our potential "seed words".
- We enumerate all short factors of $h^\omega(0)$, and check to see if any of them is an instance of an ancestor of t.
  - If so, then $h^\omega(0)$ contains an additive square.

Description of the Algorithm

Suppose that h satisfies these four conditions.

- Consider the template $t = [\varepsilon, \varepsilon, \varepsilon, \vec{0}]$.
  - An instance of t is an additive square.
- We enumerate all ancestors of t.
  - This set is finite!
- If $h^\omega(0)$ contains an additive square, then it must contain a *short* instance of one of these ancestors.
  - These are our potential “seed words”.
- We enumerate all short factors of $h^\omega(0)$, and check to see if any of them is an instance of an ancestor of t.
  - If so, then $h^\omega(0)$ contains an additive square.
  - If not, then $h^\omega(0)$ is additive square-free!