Avoiding Additive Powers in Words

Lucas Mol

THOMPSON RIVERS UNIVERSITY

Atlantic Graph Theory Seminar
March 8, 2023

“The three spheres continued to dance in my dream, a patternless, never-repeating dance. Yet, in the depths of my mind, the dance did possess a rhythm.”

–from *The Three-Body Problem* by Cixin Liu

PLAN

SQUARES AND SQUARE-FREE WORDS

ABELIAN AND ADDITIVE SQUARES

OUTLOOK

A PROOF SKETCH

Alphabets and words

- An *alphabet* is a finite set of letters, treated simply as symbols, 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, treated simply as symbols, e.g.,
  - $\{\texttt{a}, \texttt{b}, \texttt{c}, \ldots, \texttt{z}\}$ (the English alphabet)
  - $\{\texttt{0}, \texttt{1}\}$ (the binary alphabet)
  - $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$ (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, treated simply as symbols, 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, treated simply as symbols, 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 AND SQUARE-FREE WORDS

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

Squares and square-free words

- A *square* is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.

SQUARES AND SQUARE-FREE WORDS

- A *square* is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`,

Squares and square-free words

- A *square* is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`, `1`,

Squares and square-free words

- A *square* is a word of the form xx, e.g.,
  - murmur, hotshots, caracara
  - 00, 010212010212
- The *factors* of a word are its contiguous subwords.
  - e.g. The word 0110 has factors:

    0, 1, 01,

SQUARES AND SQUARE-FREE WORDS

- A *square* is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`, `1`, `01`, `11`,

Squares and square-free words

- A *square* is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`, `1`, `01`, `11`, `10`,

Squares and square-free words

- A *square* is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`, `1`, `01`, `11`, `10`, `011`,

SQUARES AND SQUARE-FREE WORDS

- A *square* is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`, `1`, `01`, `11`, `10`, `011`, `110`,

SQUARES AND SQUARE-FREE WORDS

- A *square* is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`

SQUARES AND SQUARE-FREE WORDS

- A *square* is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:
    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.

SQUARES AND SQUARE-FREE WORDS

- A *square* is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:
    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.

SQUARES AND SQUARE-FREE WORDS

- A *square* is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:
    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.
- A word is *square-free* if it contains no squares as factors.
  - `apple`

SQUARES AND SQUARE-FREE WORDS

- A *square* is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:
    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.
- A word is *square-free* if it contains no squares as factors.
  - `apple` – not square-free

SQUARES AND SQUARE-FREE WORDS

- A *square* is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:
    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.
- A word is *square-free* if it contains no squares as factors.
  - `apple` – not square-free
  - `banana`

Squares and square-free words

- A *square* is a word of the form xx, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.
- A word is *square-free* if it contains no squares as factors.
  - `apple` – not square-free
  - `banana` – not square-free

Squares and square-free words

- A *square* is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:

    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.
- A word is *square-free* if it contains no squares as factors.
  - `apple` – not square-free
  - `banana` – not square-free
  - `clementine`

SQUARES AND SQUARE-FREE WORDS

- A *square* is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:
    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.
- A word is *square-free* if it contains no squares as factors.
  - `apple` – not square-free
  - `banana` – not square-free
  - `clementine` – square-free

SQUARES AND SQUARE-FREE WORDS

- A *square* is a word of the form *xx*, e.g.,
  - `murmur`, `hotshots`, `caracara`
  - `00`, `010212010212`
- The *factors* of a word are its contiguous subwords.
  - e.g. The word `0110` has factors:
    `0`, `1`, `01`, `11`, `10`, `011`, `110`, and `0110`, but NOT `00`.
- A word is *square-free* if it contains no squares as factors.
  - `apple` – not square-free
  - `banana` – not square-free
  - `clementine` – square-free
- One can define cubes, 4th powers, etc. in a similar manner.

ARE SQUARES AVOIDABLE?

Q: Are there arbitrarily long square-free words over a finite alphabet?

Are squares avoidable?

Q: Are there arbitrarily long square-free words over a finite alphabet?
A: Hmmmmm. . .

ARE SQUARES AVOIDABLE?

Q: Are there arbitrarily long square-free words over a finite alphabet?
A: Hmmmmm...

- Over an alphabet of size one, say $\{0\}$?

Are squares avoidable?

Q: Are there arbitrarily long square-free words over a finite alphabet?
A: Hmmmmm...

- Over an alphabet of size one, say $\{0\}$? No.

ARE SQUARES AVOIDABLE?

Q: Are there arbitrarily long square-free words over a finite alphabet?
A: Hmmmmm...

- Over an alphabet of size one, say $\{0\}$? No.
- Over an alphabet of size two, say $\{0, 1\}$?

Are squares avoidable?

Q: Are there arbitrarily long square-free words over a finite alphabet?
A: Hmmmmm…

- Over an alphabet of size one, say $\{0\}$? No.
- Over an alphabet of size two, say $\{0, 1\}$? No.

Are squares avoidable?

Q: Are there arbitrarily long square-free words over a finite alphabet?
A: Hmmmmm. . .

- Over an alphabet of size one, say $\{0\}$? No.
- Over an alphabet of size two, say $\{0, 1\}$? No.
- Over an alphabet of size three, say $\{0, 1, 2\}$?

ARE SQUARES AVOIDABLE?

Q: Are there arbitrarily long square-free words over a finite alphabet?
A: Hmmmmm...

- Over an alphabet of size one, say $\{0\}$? No.
- Over an alphabet of size two, say $\{0, 1\}$? No.
- Over an alphabet of size three, say $\{0, 1, 2\}$?

letters

"The three ~~spheres~~ continued to dance in my dream, a patternless, never-repeating dance. Yet, in the depths of my mind, the dance did possess a rhythm."

–from *The Three-Body Problem* by Cixin Liu

A CONSTRUCTION

A CONSTRUCTION

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

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

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

- We start with $\texttt{0}$, 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.

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

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.

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

THE ORIGIN OF COMBINATORICS ON WORDS

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

Axel Thue (1863-1922)

Plan

Squares and Square-Free Words

Abelian and Additive Squares

Outlook

A Proof Sketch

ABELIAN 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 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 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?
- Unlike regular squares, they are NOT avoidable over three letters.

Abelian 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?
- Unlike regular squares, they are NOT avoidable over three letters.
- Theorem (Keränen 1992): Abelian squares are avoidable over four letters.

ABELIAN 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?
- Unlike regular squares, they are NOT avoidable over three letters.
- Theorem (Keränen 1992): Abelian squares are avoidable over four letters.
- The word $\sigma^{\omega}(\mathtt{0})$ avoids abelian squares, where

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

ADDITIVE SQUARES

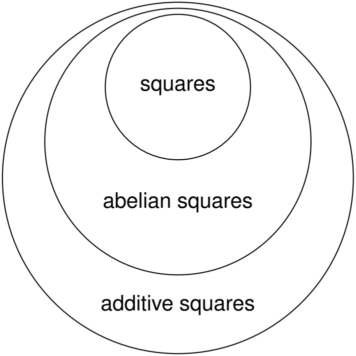
An *additive square* is a word of the form $x\tilde{x}$, where x and $\tilde{x}$ have the same length and the same sum.

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

ADDITIVE SQUARES

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.

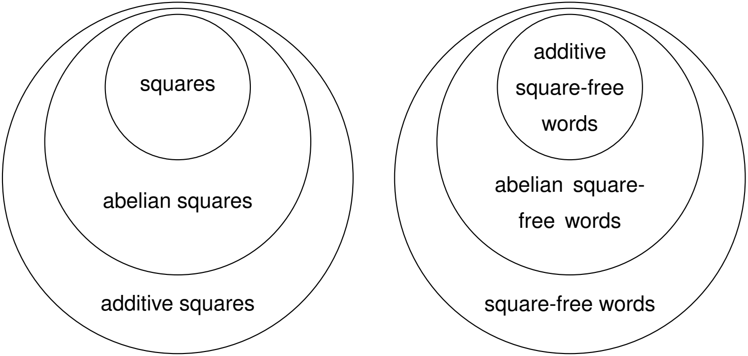
- Examples: `012012`, `012021`, `013202`



ADDITIVE SQUARES

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`



Additive Squares

- Question (Justin 1972): Are additive squares avoidable over some finite subset of $\mathbb{Z}$?

ADDITIVE SQUARES

- Question (Justin 1972): Are additive squares avoidable over some finite subset of $\mathbb{Z}$?
  - We. Don't. Know.

Additive Squares

- Question (Justin 1972): Are additive squares avoidable over some finite subset of $\mathbb{Z}$?
  - We. Don't. Know.
- Theorem (Cassaigne, Currie, Schaeffer and Shallit 2014): Additive *cubes* are avoidable over $\{0, 1, 3, 4\}$.

ADDITIVE SQUARES

- Question (Justin 1972): Are additive squares avoidable over some finite subset of $\mathbb{Z}$?
  - We. Don't. Know.
- Theorem (Cassaigne, Currie, Schaeffer and Shallit 2014): Additive *cubes* are avoidable over $\{0, 1, 3, 4\}$.
- The word $h^{\omega}(\mathtt{0})$ avoids additive cubes, where

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

Decision Algorithms

- Theorem (Currie and Rampersad 2012): There is an algorithm which decides, under certain conditions on h, whether $h^\omega(\mathtt{0})$ contains abelian squares (cubes, 4th powers, etc.)

Decision Algorithms

- Theorem (Currie and Rampersad 2012): There is an algorithm which decides, under certain conditions on h, whether $h^\omega(\mathtt{0})$ contains abelian squares (cubes, 4th powers, etc.)
- Theorem (Rao and Rosenfeld 2018): Weaker conditions on h, less efficient algorithm. Can be tweaked to handle abelian or additive powers.

Decision Algorithms

- Theorem (Currie and Rampersad 2012): There is an algorithm which decides, under certain conditions on h, whether $h^{\omega}(\mathtt{0})$ contains abelian squares (cubes, 4th powers, etc.)
- Theorem (Rao and Rosenfeld 2018): Weaker conditions on h, less efficient algorithm. Can be tweaked to handle abelian or additive powers.
- Theorem (Currie, Mol, Rampersad, and Shallit 2021+): Stronger conditions on h, more efficient algorithm for additive powers.

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 (cubes, 4th powers, etc.)

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 (cubes, 4th powers, etc.)

- In fact, the algorithm is easy to implement.

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 (cubes, 4th powers, etc.)

- In fact, the algorithm is easy to implement.

Here is the main idea:

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 (cubes, 4th powers, etc.)

- In fact, the algorithm is easy to implement.

Here is the main idea:

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

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 (cubes, 4th powers, etc.)

- In fact, the algorithm is easy to implement.

Here is the main idea:

- 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.

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 (cubes, 4th powers, etc.)

- In fact, the algorithm is easy to implement.

Here is the main idea:

- 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.

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.

- Our theorem shows that this fact can be established by a finite computer check.
- This word is also (regular) cube-free.

Plan

- Squares and Square-Free Words
- Abelian and Additive Squares
- Outlook
- A Proof Sketch

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 $h^{\omega}(\mathtt{0})$, 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 $h^{\omega}(\mathtt{0})$, 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 others.

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 others.

Hopefully this method will prove useful!

Plan

Squares and Square-Free Words

Abelian and Additive Squares

Outlook

A Proof Sketch

The theorem statement again

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.

The theorem statement again

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:

The theorem statement again

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?

THE THEOREM STATEMENT AGAIN

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?
- How do we describe the “seed words” for additive squares?

The theorem statement again

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

$$[\underbrace{a_0, a_1, a_2}_{\text{letters or } \varepsilon}, \underbrace{\vec{d}}_{\text{vector in } \mathbb{Z}^2}]$$

- 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 $\mathbb{Z}^2$

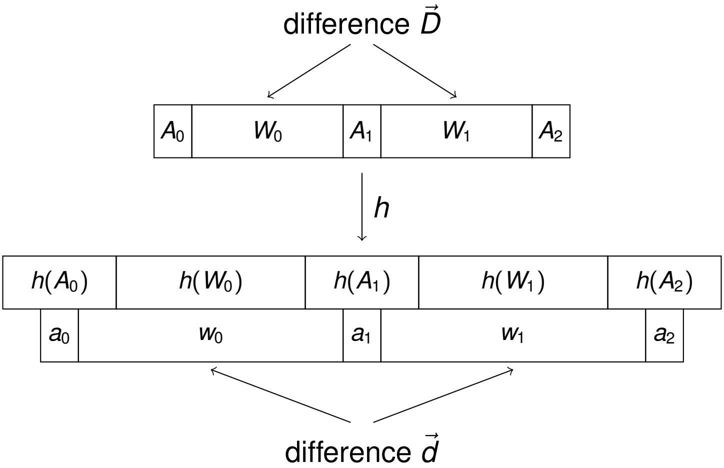
- A word w is an *instance* of this template if

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

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

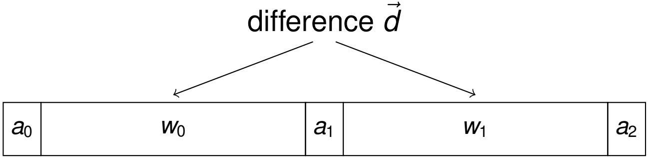
THE FIRST TWO CONDITIONS

- 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

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

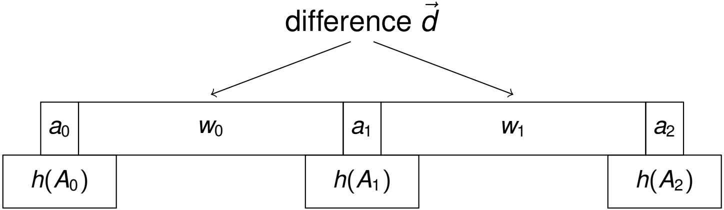
Finding Parents

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



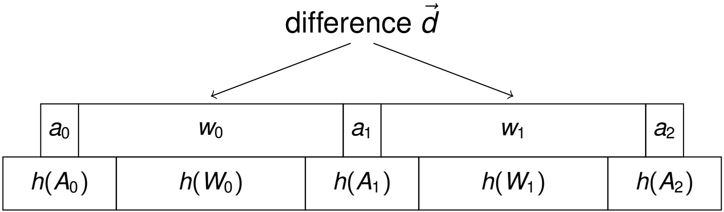
Finding Parents

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



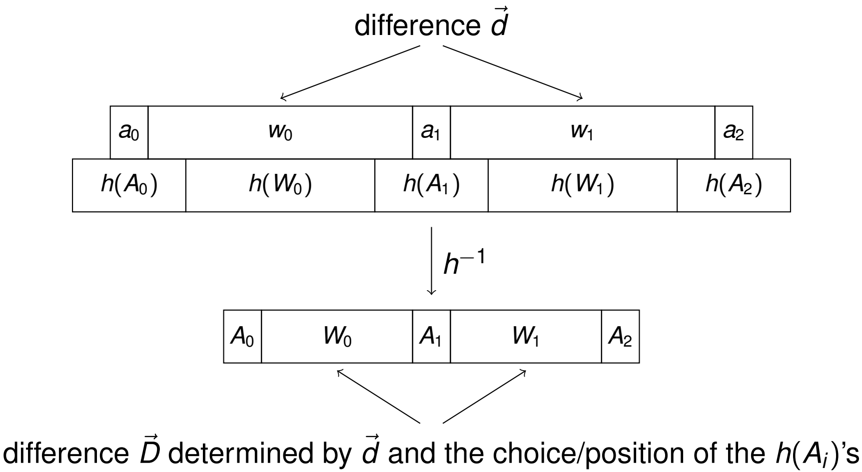
Finding Parents

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!

Thank you!