# Efficient Indexes for Jumbled Pattern Matching with Constant-Sized Alphabet

**Tomasz Kociumaka**, Jakub Radoszewski,
Wojciech Rytter

University of Warsaw, Poland

Sophia Antipolis, September 2, 2013

# Commutative Equivalence and Parikh Vectors

**Definition**

Let $w$ be a word over $\Sigma$. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in $w$.

$$\Sigma = \{a, b, c\} \quad w = \mathtt{a\,b\,b\,a\,c} \qquad \mathcal{P}(w) = (2, 2, 1)$$

# Commutative Equivalence and Parikh Vectors

**Definition**

Let $w$ be a word over $\Sigma$. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in $w$.

$$\Sigma = \{a, b, c\} \quad w = \mathtt{a\,b\,b\,a\,c} \qquad \mathcal{P}(w) = (2, 2, 1)$$

# Commutative Equivalence and Parikh Vectors

**Definition**

Let $w$ be a word over $\Sigma$. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in $w$.

$$\Sigma = \{a, b, c\} \quad w = \mathtt{a\,b\,b\,a\,c} \qquad \mathcal{P}(w) = (2, 2, 1)$$

# Commutative Equivalence and Parikh Vectors

**Definition**

Let $w$ be a word over $\Sigma$. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in $w$.

$$\Sigma = \{a, b, c\} \quad w = \mathtt{a\,b\,b\,a\,c} \qquad \mathcal{P}(w) = (2, 2, 1)$$

# Commutative Equivalence and Parikh Vectors

**Definition**

Let $w$ be a word over $\Sigma$. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in $w$.

$$\Sigma = \{a, b, c\} \quad w = \mathtt{a\,b\,b\,a\,c} \qquad \mathcal{P}(w) = (2, 2, 1)$$

**Definition**

Words $u, w$ are *commutatively equivalent* if $\mathcal{P}(u) = \mathcal{P}(w)$.

$$\mathtt{a\,b\,b\,a\,c} \approx \mathtt{a\,c\,b\,a\,b} \qquad \mathtt{b\,a\,b} \not\approx \mathtt{a\,b\,a}$$

# Commutative Equivalence and Parikh Vectors

**Definition**

Let $w$ be a word over $\Sigma$. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in $w$.

$$\Sigma = \{a, b, c\} \quad w = \mathtt{a\,b\,b\,a\,c} \qquad \mathcal{P}(w) = (2, 2, 1)$$

**Definition**

Words $u, w$ are *commutatively equivalent* if $\mathcal{P}(u) = \mathcal{P}(w)$.

$$\mathtt{a\,b\,b\,a\,c} \approx \mathtt{a\,c\,b\,a\,b} \qquad \mathtt{b\,a\,b} \not\approx \mathtt{a\,b\,a}$$

**Definition**

The *norm* $|p|$ of a Parikh vector $p$ is the sum of its entries.

# Jumbled Pattern Matching, Index Definition

**Definition**

Let $p$ be a Parikh vector of norm $d$. We say that $p$ *occurs* at position $i$ of a word $w$ if $p = \mathcal{P}(w[i, i+d-1])$.

The occurrences of $(1, 2, 2, 1)$ in `abcabcdcbacdabbcacdc` are

`a b c a b c d c b a c d a b b c a c d c`

# Jumbled Pattern Matching, Index Definition

**Definition**

Let $p$ be a Parikh vector of norm $d$. We say that $p$ *occurs* at position $i$ of a word $w$ if $p = \mathcal{P}(w[i, i+d-1])$.

The occurrences of $(1, 2, 2, 1)$ in `abcabcdcbacdabbcacdc` are 2

$$\texttt{a}\ \boxed{\texttt{b c a b c d}}\ \texttt{c b a c d a b b c a c d c}$$

# Jumbled Pattern Matching, Index Definition

**Definition**

Let $p$ be a Parikh vector of norm $d$. We say that $p$ *occurs* at position $i$ of a word $w$ if $p = \mathcal{P}(w[i, i+d-1])$.

The occurrences of $(1, 2, 2, 1)$ in `abcabcdcbacdabbcacdc` are 2, 4

a b c $\boxed{\text{a b c d c b}}$ a c d a b b c a c d c

# Jumbled Pattern Matching, Index Definition

**Definition**

Let $p$ be a Parikh vector of norm $d$. We say that $p$ *occurs* at position $i$ of a word $w$ if $p = \mathcal{P}(w[i, i+d-1])$.

The occurrences of $(1, 2, 2, 1)$ in `abcabcdcbacdabbcacdc` are 2, 4, 5

$$\mathtt{a\ b\ c\ a}\ \boxed{\mathtt{b\ c\ d\ c\ b\ a}}\ \mathtt{c\ d\ a\ b\ b\ c\ a\ c\ d\ c}$$

# Jumbled Pattern Matching, Index Definition

**Definition**

Let $p$ be a Parikh vector of norm $d$. We say that $p$ *occurs* at position $i$ of a word $w$ if $p = \mathcal{P}(w[i, i + d - 1])$.

The occurrences of $(1, 2, 2, 1)$ in `abcabcdcbacdabbcacdc` are 2, 4, 5, 11

$$\mathtt{a\ b\ c\ a\ b\ c\ d\ c\ b\ a}\ \boxed{\mathtt{c\ d\ a\ b\ b\ c}}\ \mathtt{a\ c\ d\ c}$$

# Jumbled Pattern Matching, Index Definition

**Definition**

Let $p$ be a Parikh vector of norm $d$. We say that $p$ *occurs* at position $i$ of a word $w$ if $p = \mathcal{P}(w[i, i+d-1])$.

The occurrences of $(1, 2, 2, 1)$ in `abcabcdcbacdabbcacdc` are 2, 4, 5, 11, 14.

$$\mathtt{a\ b\ c\ a\ b\ c\ d\ c\ b\ a\ c\ d\ a}\ \boxed{\mathtt{b\ b\ c\ a\ c\ d}}\ \mathtt{c}$$

# Jumbled Pattern Matching, Index Definition

**Definition**

Let $p$ be a Parikh vector of norm $d$. We say that $p$ *occurs* at position $i$ of a word $w$ if $p = \mathcal{P}(w[i, i+d-1])$.

The occurrences of $(1, 2, 2, 1)$ in `abcabcdcbacdabbcacdc` are 2, 4, 5, 11, 14.

`a b c a b c d c b a c d a b b c a c d c`

**Problem (Abelian index)**

*For a word $w$ build a data structure which given a Parikh vector $p$ efficiently decides whether $p$ occurs in $w$.*

E.g. $(1, 2, 2, 1)$ occurs in `abcabcdcbacdabbcacdc`, while $(1, 2, 1, 2)$ does not.

# Previous Results for Abelian Index

Binary alphabet:

- $\mathcal{O}(n)$ size, $\mathcal{O}(1)$ query time, $\mathcal{O}(n^2)$ construction time (Cicalese et al., 2009)
- $\mathcal{O}(\frac{n^2}{\log n})$ construction time (Burcsi et al., 2010; Moosa & Rahman, 2010)
- $\mathcal{O}(\frac{n^2}{\log^2 n})$ construction time (Moosa & Rahman, 2012)

# Previous Results for Abelian Index

Binary alphabet:

- $\mathcal{O}(n)$ size, $\mathcal{O}(1)$ query time, $\mathcal{O}(n^2)$ construction time (Cicalese et al., 2009)
- $\mathcal{O}(\frac{n^2}{\log n})$ construction time (Burcsi et al., 2010; Moosa & Rahman, 2010)
- $\mathcal{O}(\frac{n^2}{\log^2 n})$ construction time (Moosa & Rahman, 2012)

For larger alphabets naive solutions only:

- $\mathcal{O}(n)$ query time (run a jumbled pattern matching algorithm),
- $\mathcal{O}(n^2)$ size (memorize all answers in a hash table).

# Our Results

- Alphabet with $\sigma = \mathcal{O}(1)$ letters.
- Word-RAM, randomized (construction only).

# Our Results

- Alphabet with $\sigma = \mathcal{O}(1)$ letters.
- Word-RAM, randomized (construction only).

**Theorem (in this presentation & in the paper)**

*For any $\delta \in (0,1)$ there exists an index with $\mathcal{O}(n^{2-\delta})$ size, $\mathcal{O}(m^{\delta(2\sigma-1)})$ query time, where $m$ is the norm of the pattern, and $\mathcal{O}(n^2)$ construction time.*

# Our Results

- Alphabet with $\sigma = \mathcal{O}(1)$ letters.
- Word-RAM, randomized (construction only).

**Theorem (in this presentation & in the paper)**

*For any $\delta \in (0,1)$ there exists an index with $\mathcal{O}(n^{2-\delta})$ size, $\mathcal{O}(m^{\delta(2\sigma-1)})$ query time, where $m$ is the norm of the pattern, and $\mathcal{O}(n^2)$ construction time.*

**Theorem (in the paper)**

*There exists an index with $\mathcal{O}(\frac{n^2 \log\log n}{\log n})$ size, $\mathcal{O}((\frac{\log n}{\log\log n})^{(2\sigma-1)})$ query time and $\mathcal{O}(\frac{n^2 \log^2 \log n}{\log n})$ construction time.*

# Our Results

- Alphabet with $\sigma = \mathcal{O}(1)$ letters.
- Word-RAM, randomized (construction only).

**Theorem (in this presentation & in the paper)**

*For any $\delta \in (0, 1)$ there exists an index with $\mathcal{O}(n^{2-\delta})$ size, $\mathcal{O}(m^{\delta(2\sigma-1)})$ query time, where $m$ is the norm of the pattern, and $\mathcal{O}(n^2)$ construction time.*

**Theorem (in the paper)**

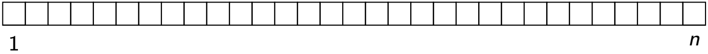
*There exists an index with $\mathcal{O}(\frac{n^2 \log\log n}{\log n})$ size, $\mathcal{O}((\frac{\log n}{\log\log n})^{(2\sigma-1)})$ query time and $\mathcal{O}(\frac{n^2 \log^2\log n}{\log n})$ construction time.*

**Theorem (unpublished)**

*There exists an index with $\mathcal{O}(n^{2-\delta})$ size, $\mathcal{O}((\frac{m^\delta \log m}{\log\log m})^{(2\sigma-1)})$ query time and $\mathcal{O}(\frac{n^2 \log^2\log n}{\log n})$ construction time.*

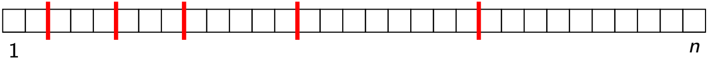
# Layers

- Every (reasonable) query $q$ has its norm $|q| \in \{1, \ldots, n\}$.



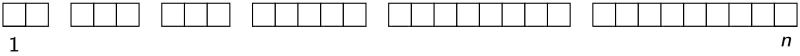
# Layers

- Every (reasonable) query $q$ has its norm $|q| \in \{1, \ldots, n\}$.
- We divide $\{1, \ldots, n\}$ into several ranges. For each range, queries with norm in that range are answered by a separate data structure.



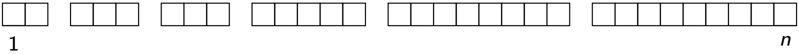
# Layers

- Every (reasonable) query $q$ has its norm $|q| \in \{1, \ldots, n\}$.
- We divide $\{1, \ldots, n\}$ into several ranges. For each range, queries with norm in that range are answered by a separate data structure.
- For each possible query norm we need to remember the responsible layer.



# Layers

- Every (reasonable) query $q$ has its norm $|q| \in \{1, \ldots, n\}$.
- We divide $\{1, \ldots, n\}$ into several ranges. For each range, queries with norm in that range are answered by a separate data structure.
- For each possible query norm we need to remember the responsible layer.
- The layer responsible for $\{d, \ldots, d + L\}$ is called the $(d, L)$-*layer*.



# Layers

- Every (reasonable) query $q$ has its norm $|q| \in \{1, \dots, n\}$.
- We divide $\{1, \dots, n\}$ into several ranges. For each range, queries with norm in that range are answered by a separate data structure.
- For each possible query norm we need to remember the responsible layer.
- The layer responsible for $\{d, \dots, d + L\}$ is called the $(d, L)$-*layer*.

| | |
|---|---|
| 1 | +0 +1 |
| 3 | +0 +1 +2 |
| 6 | +0 +1 +2 |
| 9 | +0 +1 +2 +3 +4 |
| 14 | +0 +1 +2 +3 +4 +5 +6 +7 |
| 22 | +0 +1 +2 +3 +4 +5 +6 +7 +8 |

# Layers

- Every (reasonable) query $q$ has its norm $|q| \in \{1, \ldots, n\}$.
- We divide $\{1, \ldots, n\}$ into several ranges. For each range, queries with norm in that range are answered by a separate data structure.
- For each possible query norm we need to remember the responsible layer.
- The layer responsible for $\{d, \ldots, d+L\}$ is called the $(d, L)$-*layer*.

| +0 | +1 |
|---|---|
| 1 | |

| +0 | +1 | +2 |
|---|---|---|
| 3 | | |

| +0 | +1 | +2 |
|---|---|---|
| 6 | | |

| +0 | +1 | +2 | +3 | +4 |
|---|---|---|---|---|
| 9 | | | | |

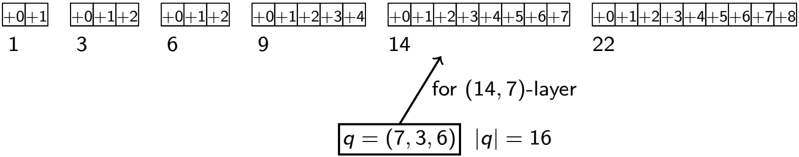
| +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 |
|---|---|---|---|---|---|---|---|
| 14 | | | | | | | |

| +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 | +8 |
|---|---|---|---|---|---|---|---|---|
| 22 | | | | | | | | |

$\boxed{q = (7, 3, 6)}$ $|q| = 16$

# Layers

- Every (reasonable) query $q$ has its norm $|q| \in \{1, \ldots, n\}$.
- We divide $\{1, \ldots, n\}$ into several ranges. For each range, queries with norm in that range are answered by a separate data structure.
- For each possible query norm we need to remember the responsible layer.
- The layer responsible for $\{d, \ldots, d + L\}$ is called the $(d, L)$-*layer*.



# The $(d, L)$-Layer

- Fix a word $w$ and positive integers $(d, L)$ such that $d + L \leq |w|$.
- We develop the $(d, L)$-layer for $w$.

# The $(d, L)$-Layer

- Fix a word $w$ and positive integers $(d, L)$ such that $d + L \leq |w|$.
- We develop the $(d, L)$-layer for $w$.

**Definition**

Parikh vetors of norm within $\{d, \ldots, d + L\}$ are called *relevant vectors* and vectors of norm $d$ – *ground vectors*.

# The $(d, L)$-Layer

- Fix a word $w$ and positive integers $(d, L)$ such that $d + L \leq |w|$.
- We develop the $(d, L)$-layer for $w$.

**Definition**

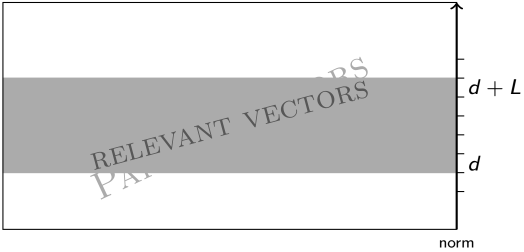
Parikh vetors of norm within $\{d, \ldots, d + L\}$ are called *relevant vectors* and vectors of norm $d$ – *ground vectors*.

# The $(d, L)$-Layer

- Fix a word $w$ and positive integers $(d, L)$ such that $d + L \leq |w|$.
- We develop the $(d, L)$-layer for $w$.

**Definition**

Parikh vetors of norm within $\{d, \ldots, d + L\}$ are called *relevant vectors* and vectors of norm $d$ – *ground vectors*.

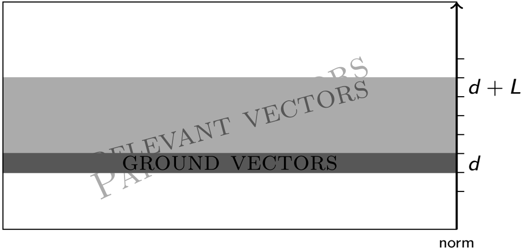


# The $(d, L)$-Layer

- Fix a word $w$ and positive integers $(d, L)$ such that $d + L \leq |w|$.
- We develop the $(d, L)$-layer for $w$.

## Definition

Parikh vetors of norm within $\{d, \dots, d + L\}$ are called *relevant vectors* and vectors of norm $d$ – *ground vectors*.

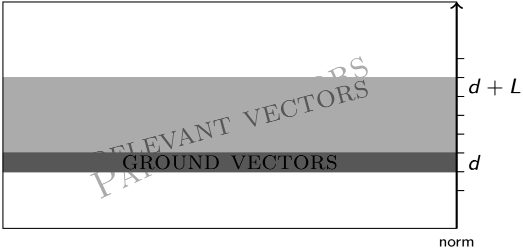


# The $(d, L)$-Layer

- Fix a word $w$ and positive integers $(d, L)$ such that $d + L \leq |w|$.
- We develop the $(d, L)$-layer for $w$.

**Definition**

Parikh vetors of norm within $\{d, \dots, d + L\}$ are called *relevant vectors* and vectors of norm $d$ – *ground vectors*.

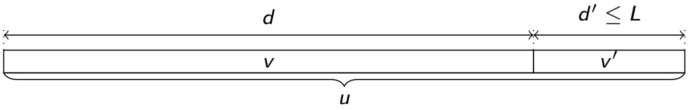



For words we use an analogous terminology.

# Extensions

**Definition**

Let $v$ be a ground word. We say that a relevant word $u$ is *an extension* of $v$ if $u = vv'$ for some word $v'$.

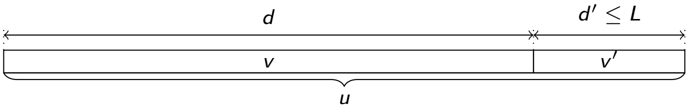


# Extensions

**Definition**

Let $v$ be a ground word. We say that a relevant word $u$ is *an extension* of $v$ if $u = vv'$ for some word $v'$.

- A relevant word is an extension of a unique ground word.

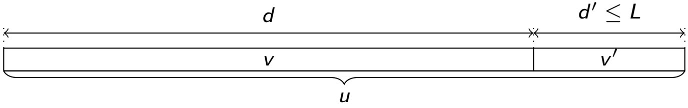


# Extensions

**Definition**

Let $v$ be a ground word. We say that a relevant word $u$ is *an extension* of $v$ if $u = vv'$ for some word $v'$.

- A relevant word is an extension of a unique ground word.
- A ground word has $\mathcal{O}(\sigma^{d'})$ extensions of length $d + d'$, which gives $\mathcal{O}(\sigma^{L})$ relevant extensions in total.

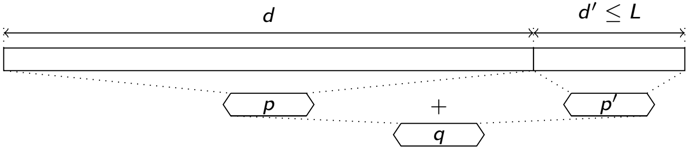



- The number of extensions of a word is exponential in $L$.

# Extensions

**Definition**

Let $p$ be a ground vector. We say that a relevant vector $q$ is *an extension* of $p$ if $q = p + p'$ for some Parikh vector $p'$.

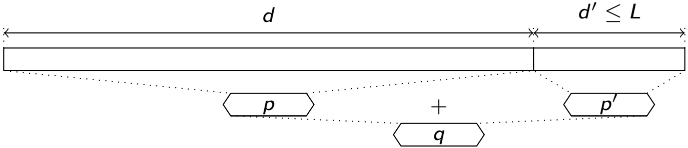


# Extensions

**Definition**

Let $p$ be a ground vector. We say that a relevant vector $q$ is *an extension* of $p$ if $q = p + p'$ for some Parikh vector $p'$.

- A relevant vector of length $d + d'$ is an extension of at most $\binom{d'+\sigma-1}{\sigma-1} = \mathcal{O}(d'^{\sigma-1})$ ground vectors.

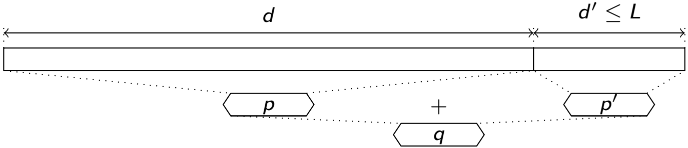


# Extensions

**Definition**

Let $p$ be a ground vector. We say that a relevant vector $q$ is *an extension* of $p$ if $q = p + p'$ for some Parikh vector $p'$.

- A relevant vector of length $d + d'$ is an extension of at most $\binom{d'+\sigma-1}{\sigma-1} = \mathcal{O}(d'^{\sigma-1})$ ground vectors.
- A ground vector has $\mathcal{O}(d'^{\sigma-1})$ extensions of length $d + d'$, which gives $\mathcal{O}(L^{\sigma})$ relevant extensions in total.




- The number of extensions of a vector is polynomial in $L$.

# The $(d, L)$-Layer: Overview

- We can store a list of occurrences for each ground vector present in $w$ (in a hash map)
  - $\mathcal{O}(n)$ space.

# The $(d, L)$-Layer: Overview

- We can store a list of occurrences for each ground vector present in $w$ (in a hash map)
  - $\mathcal{O}(n)$ space.
- We divide the ground vectors into *heavy* and *light*.
  - *Heavy* vectors have many occurrences, more than possible extensions: we have enough SPACE to store the extenstions present in $w$ in a hash set.
  - *Light* vectors have few occurrences: we have enough TIME to scan all of them within queries.
  - The threshold on the number of occurrences is set to $L^\sigma$.

# The $(d, L)$-Layer: Details

Components:

- a hash map $M$ assigning each light ground vector the list of its occurrences,
- a hash set $S$ of relevant vectors extending the occurrences of heavy ground vectors,
- Parikh vectors of prefixes of $w$.

# The $(d, L)$-Layer: Details

Components:

- a hash map $M$ assigning each light ground vector the list of its occurrences,
- a hash set $S$ of relevant vectors extending the occurrences of heavy ground vectors,
- Parikh vectors of prefixes of $w$.

Space usage: $\mathcal{O}(n)$ words

- Each vector in $S$ is an extension of a heavy ground vector, and each heavy ground vector has more occurrences than extensions in $S$, so $|S| = \mathcal{O}(n)$.
- Clearly the remaining components also take $\mathcal{O}(n)$ space.

# The $(d, L)$-Layer: Details

Components:

- a hash map $M$ assigning each light ground vector the list of its occurrences,
- a hash set $S$ of relevant vectors extending the occurrences of heavy ground vectors,
- Parikh vectors of prefixes of $w$.

Construction: $\mathcal{O}(nL)$ time

- Compute the Parikh vector of each ground factor.
- Generate the list of occurrences for each.
- Store the list for light vectors in $M$.
- For each occurrence of a heavy vector, add to $S$ the relevant vectors occurring at the same position.

# The $(d, L)$-Layer: Queries

The algorithm for a relevant vector $q$:

1. Check if $q$ is present in $S$.
2. For each light ground vector $p$ such that $q$ is an extension of $p$, and for each occurrence $i$ of $p$ (obtained from $M$) check whether $q = \mathcal{P}(w[1, i + |q| - 1]) - \mathcal{P}(w[1, i - 1])$.

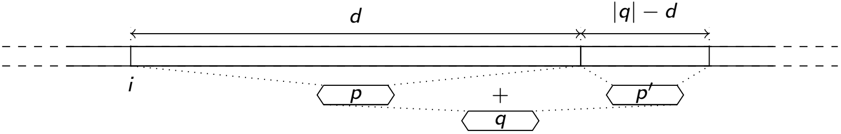
# The $(d, L)$-Layer: Queries

The algorithm for a relevant vector $q$:

1. Check if $q$ is present in $S$.
2. For each light ground vector $p$ such that $q$ is an extension of $p$, and for each occurrence $i$ of $p$ (obtained from $M$) check whether $q = \mathcal{P}(w[1, i + |q| - 1]) - \mathcal{P}(w[1, i - 1])$.

**Correctness.** Each occurrence of $q$ extends an occurrence of a ground vector $p$:

- if $p$ is heavy, then $q$ is detected in the first step,
- if $p$ is light, then $q$ is detected in the second step.



# The $(d, L)$-Layer: Queries

The algorithm for a relevant vector $q$:

1. Check if $q$ is present in $S$.
2. For each light ground vector $p$ such that $q$ is an extension of $p$, and for each occurrence $i$ of $p$ (obtained from $M$) check whether $q = \mathcal{P}(w[1, i + |q| - 1]) - \mathcal{P}(w[1, i - 1])$.

**Correctness.** Each occurrence of $q$ extends an occurrence of a ground vector $p$:

- if $p$ is heavy, then $q$ is detected in the first step,
- if $p$ is light, then $q$ is detected in the second step.

**Complexity.** A query is answered in $\mathcal{O}(L^{2\sigma-1})$ time:

- there are $\mathcal{O}(L^{\sigma-1})$ ground vectors $p$ whose extension is $q$,
- for the light ones, there are up to $L^{\sigma}$ occurrences,
- a single check takes $\mathcal{O}(\sigma) = \mathcal{O}(1)$ time.

# The Data Structure

**Theorem**

*For any $\delta \in (0,1)$ there exists an index with $\mathcal{O}(n^{2-\delta})$ size, $\mathcal{O}(m^{\delta(2\sigma-1)})$ query time, where $m$ is the norm of the pattern, and $\mathcal{O}(n^2)$ construction time.*

- We divide $\{1,\ldots,n\}$ greedily into layers with $L = \lfloor d^\delta \rfloor$, i.e. we build $(d_i, L_i)$-layers with $d_1 = 1$, $L_i = \lfloor d_i^\delta \rfloor$, $d_{i+1} = d_i + L_i + 1$.
- In total, this gives $\mathcal{O}(n^{1-\delta})$ layers.
- If $(d_i, L_i)$-layer is reponsible for $q$, then $L_i = \mathcal{O}(|q|^\delta)$, i.e. the query can be answered in $\mathcal{O}(|q|^{\delta(2\sigma-1)})$ time.

# Quick Overview of Subquadratic Construction

- The only bottleneck is finding relevant vectors which occur as extensions of heavy ground vectors.
- Set $L = \Theta(\frac{\log d}{\sigma \log \log d})$.
- Parikh vectors of norm $\leq L$ can be assigned integer identifiers, $L$ identifiers fit a single machine word (we call such word a *packed set*).
- For each *word* of length $\leq L$ (only $o(d)$) we precompute a packed set containing (Abelian) identifiers of its prefixes.
- We use bit-parallelism to efficiently compute the set-theoretic union of packed sets.
- For each heavy ground vector, we apply this operation for vectors occurring right after its occurrences.
- Finally we *unpack* the union and store the corresponding Parikh vectors in the hash set $S$.

# Summary

- The first Abelian index with $o(n^2)$ space and $o(n)$ query time for alphabets of constant but arbitrarily large size.

# Summary

- The first Abelian index with $o(n^2)$ space and $o(n)$ query time for alphabets of constant but arbitrarily large size.
- The size of the index is actually strongly subquadratic, i.e. $o(n^{2-\varepsilon})$, and the query time strongly sublinear in pattern size, i.e. $o(m^{1-\varepsilon})$.

# Summary

- The first Abelian index with $o(n^2)$ space and $o(n)$ query time for alphabets of constant but arbitrarily large size.
- The size of the index is actually strongly subquadratic, i.e. $o(n^{2-\varepsilon})$, and the query time strongly sublinear in pattern size, i.e. $o(m^{1-\varepsilon})$.
- An index with $o(n^2)$ construction time and $\mathcal{O}(polylog(m))$ queries.

Thank you for your attention!