Pattern Matching with Mismatches and Wildcards

Gabriel Bathie[1,2], *Panagiotis Charalampopoulos*[3], Tatiana Starikovskaya[1]

1. École normale supérieure de Paris, France

2. Université de Bordeaux, France

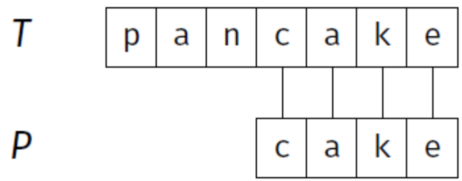
3. Birkbeck, University of London, UK

ESA 2024

The Problem

Pattern Matching

Given a text T and a pattern P, compute the occurrences of P in T.

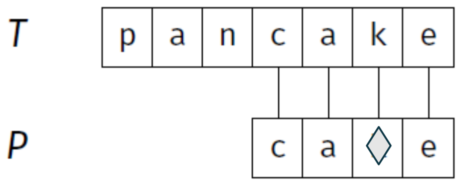


Easy; a linear-time algorithm is known since 1970 [Morris-Pratt]. However, looking for exact matches of P in T might be too restrictive: think of spelling mistakes and corrupt data.

The Problem

Pattern Matching with Wildcards

Given a text T and a pattern P, which may contain wildcards ($\lozenge$), compute the occurrences of P in T.

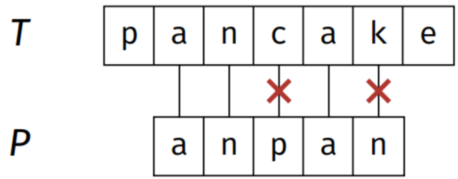


If we know the corrupt positions, we can replace their entries with wildcards ($\lozenge$) which match all letters of the alphabet and perform exact pattern matching. A long series of works has culminated in an elegant FFT-based $\mathcal{O}(|T| \log |P|)$-time algorithm [Clifford–Clifford; 2007].

The Problem

Pattern Matching with Mismatches

Given a text T, a pattern P, and an integer threshold k, compute the substrings of T that are at Hamming distance at most k from P.

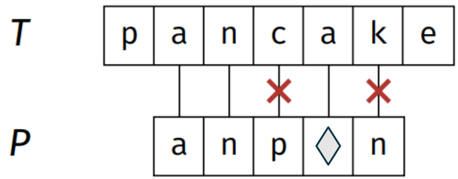


Alternatively, we can look for substrings of T that are close to P, e.g., under the Hamming distance. This is a much harder problem; it admits an $\tilde{\mathcal{O}}(|T| + k \cdot |T|/\sqrt{|P|})$-time solution [Gawrychowski–Uznanski; 2018].

The Problem

Pattern Matching with Mismatches and Wildcards

Given a text T, a pattern P, which may contain wildcards ($\Diamond$), and an integer threshold k, compute the substrings of T that are at Hamming distance at most k from P.



In this work, we revisit the variant of problem where some of the corrupt positions are known.

Previous Work and Our Result

$n = |T|, m = |P|$

$D = \#$ wildcards $= 9$

$G = \#$ groups of wildcards $= 4$

Previous Work and Our Result

P

$n = |T|$, $m = |P|$

$D = \#$ wildcards $= 9$

$G = \#$ groups of wildcards $= 4$

For wildcards in both P and T:

$\tilde{\mathcal{O}}(n\sqrt{m - D})$ [Amir-Lewenstein-Porat; 2004]

$\tilde{\mathcal{O}}(nk)$ [Clifford-Efremenko-Porat-Rothschild; 2010]

Previous Work and Our Result

P 

$n = |T|$, $m = |P|$

$D = \#$ wildcards $= 9$

$G = \#$ groups of wildcards $= 4$

For wildcards in both P and T:

$\tilde{\mathcal{O}}(n\sqrt{m-D})$ [Amir-Lewenstein-Porat; 2004]

$\tilde{\mathcal{O}}(nk)$ [Clifford-Efremenko-Porat-Rothschild; 2010]

For wildcards only in P:

$\tilde{\mathcal{O}}(n\sqrt[3]{mk})$ [Clifford-Porat; 2010]

$\tilde{\mathcal{O}}(n\sqrt{k} + n \cdot \min\{\sqrt[3]{Gk}, \sqrt{G}\})$ [Nicolae-Rajasekaran; 2017]

$\mathcal{O}(n + (n/m)(D+k)(G+k))$ [this work]

Previous Work and Our Result

P 

$n = |T|$, $m = |P|$

$D = \#$ wildcards $= 9$

$G = \#$ groups of wildcards $= 4$

For wildcards in both P and T:

$\tilde{\mathcal{O}}(n\sqrt{m-D})$	[Amir-Lewenstein-Porat; 2004]
$\tilde{\mathcal{O}}(nk)$	[Clifford-Efremenko-Porat-Rothschild; 2010]

For wildcards only in P:

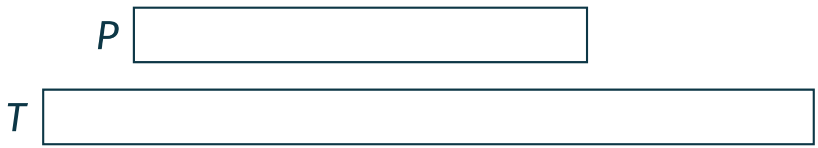
$\tilde{\mathcal{O}}(n\sqrt[3]{mk})$	[Clifford-Porat; 2010]
$\tilde{\mathcal{O}}(n\sqrt{k} + n \cdot \min\{\sqrt[3]{Gk}, \sqrt{G}\})$	[Nicolae-Rajasekaran; 2017]
$\mathcal{O}(n + (n/m)(D+k)(G+k))$	[this work]

Fast when D, G, and k are small relative to n. For $m = n/2$, $k = G = n^{2/5}$, and $D = n^{3/5}$, our algorithm takes $\mathcal{O}(n)$ time, improving over $\mathcal{O}(n^{6/5})$.

The Structure of Exact Pattern Matching

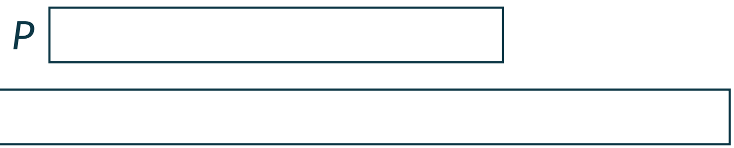
The Structure of Exact Pattern Matching

Fact [folklore] Given a pattern P of length m and a text T of length $n \leq 3/2\, m$ at least one of the following holds:



Fact [folklore] Given a pattern P of length m and a text T of length $n \leq 3/2\, m$ at least one of the following holds:

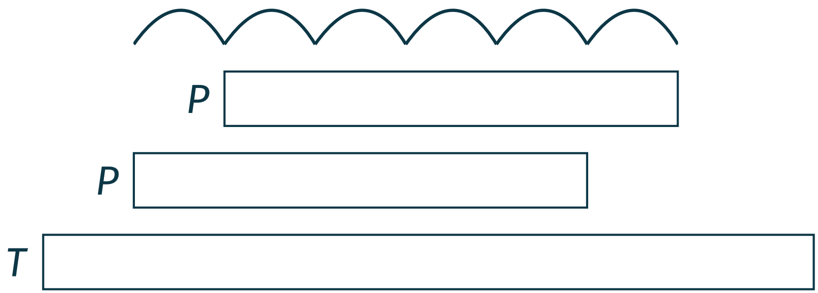
- The pattern P has at most one occurrence in T.



The Structure of Exact Pattern Matching

Fact [folklore] Given a pattern P of length m and a text T of length $n \leq 3/2\, m$ at least one of the following holds:

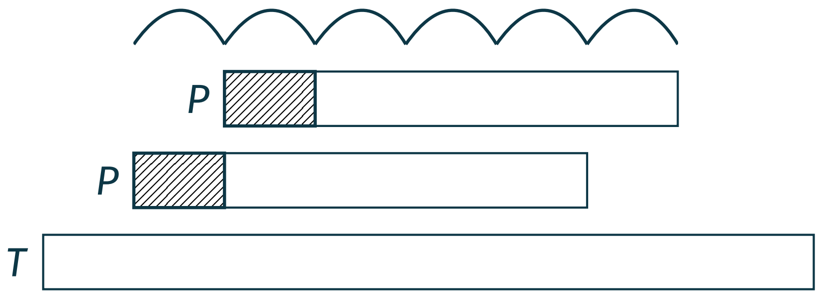
- The pattern P has at most one occurrence in T.
- The pattern P is periodic ($\text{per}(P) \leq |P|/2$).



The Structure of Exact Pattern Matching

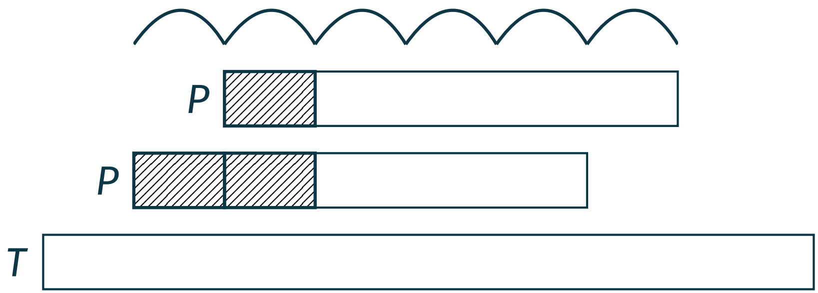
Fact [folklore] Given a pattern P of length m and a text T of length $n \leq {}^{3}\!/_{2}\, m$ at least one of the following holds:

- The pattern P has at most one occurrence in T.
- The pattern P is periodic ($\text{per}(P) \leq |P|/2$).



Fact [folklore] Given a pattern P of length m and a text T of length $n \leq {}^{3}\!/_{2}\, m$ at least one of the following holds:

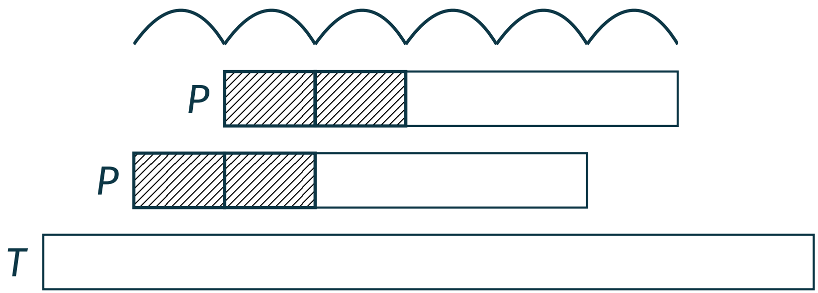
- The pattern P has at most one occurrence in T.
- The pattern P is periodic ($\text{per}(P) \leq |P|/2$).



The Structure of Exact Pattern Matching

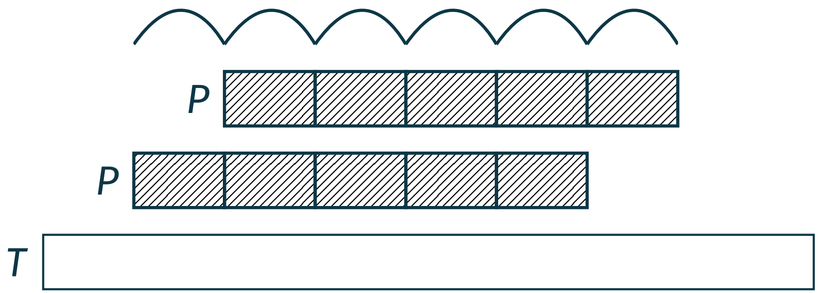
Fact [folklore] Given a pattern P of length m and a text T of length $n \leq {}^{3}\!/_{2}\, m$ at least one of the following holds:

- The pattern P has at most one occurrence in T.
- The pattern P is periodic ($\text{per}(P) \leq |P|/2$).



Fact [folklore] Given a pattern P of length m and a text T of length $n \leq \frac{3}{2} m$ at least one of the following holds:

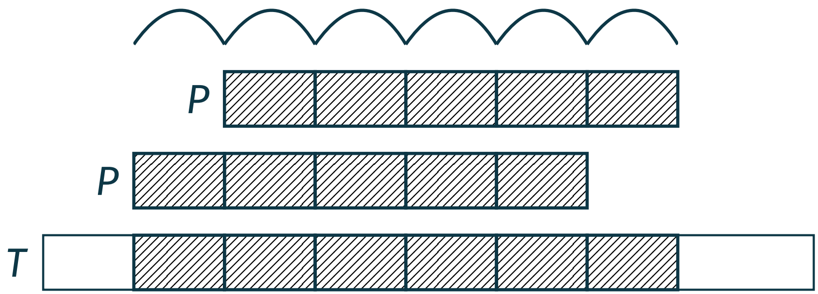
- The pattern P has at most one occurrence in T.
- The pattern P is periodic ($\text{per}(P) \leq |P|/2$).



The Structure of Exact Pattern Matching

Fact [folklore] Given a pattern P of length m and a text T of length $n \leq {}^{3}\!/_{2}\, m$ at least one of the following holds:

- The pattern P has at most one occurrence in T.
- The pattern P is periodic ($\mathrm{per}(P) \leq |P|/2$).

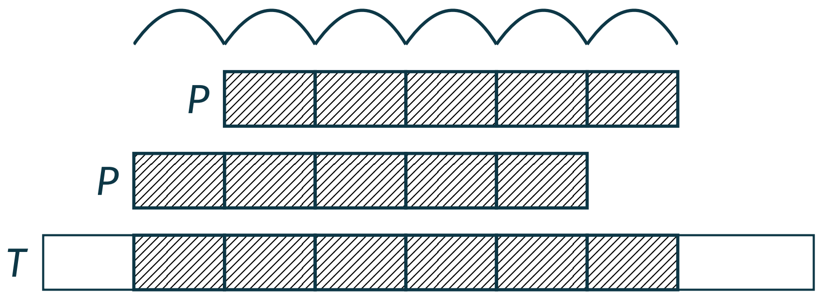


The fragment of T spanned by P's occurrences is periodic as well.

The Structure of Exact Pattern Matching

Fact [folklore] Given a pattern P of length m and a text T of length $n \leq 3/2\, m$ at least one of the following holds:

- The pattern P has at most one occurrence in T.
- The pattern P is periodic ($\text{per}(P) \leq |P|/2$).



The standard trick: Our assumption on the length of the text is not restrictive. If the text is much longer that the pattern, we can always consider separately $\mathcal{O}(n/m)$ fragments of T of length $\leq 3/2\, m$ that overlap by $m - 1$ positions.

The structure of Pattern Matching with Mismatches was understood only recently!

The structure of Pattern Matching with Mismatches was understood only recently!

Long history of algorithmic results. Some of these heavily exploited the periodic structure of P and T, and implied some (weak) structural results.

The structure of Pattern Matching with Mismatches was understood only recently!

Long history of algorithmic results. Some of these heavily exploited the periodic structure of P and T, and implied some (weak) structural results.

First explicit structural result due to Bringmann, Künnemann, and Wellnitz [BKW'19].

The structure of Pattern Matching with Mismatches was understood only recently!

Long history of algorithmic results. Some of these heavily exploited the periodic structure of P and T, and implied some (weak) structural results.

First explicit structural result due to Bringmann, Künnemann, and Wellnitz [BKW'19].

Tightened by C., Kociumaka, and Wellnitz in [CKW'20]; we showed that at least one of the following holds:

The structure of Pattern Matching with Mismatches was understood only recently!

Long history of algorithmic results. Some of these heavily exploited the periodic structure of P and T, and implied some (weak) structural results.

First explicit structural result due to Bringmann, Künnemann, and Wellnitz [BKW'19].

Tightened by C., Kociumaka, and Wellnitz in [CKW'20]; we showed that at least one of the following holds:

- The pattern P has $\mathcal{O}(k)$ k-mismatch occurrences in T.

The structure of Pattern Matching with Mismatches was understood only recently!

Long history of algorithmic results. Some of these heavily exploited the periodic structure of P and T, and implied some (weak) structural results.

First explicit structural result due to Bringmann, Künnemann, and Wellnitz [BKW'19].

Tightened by C., Kociumaka, and Wellnitz in [CKW'20]; we showed that at least one of the following holds:

- The pattern P has $\mathcal{O}(k)$ k-mismatch occurrences in T.
- The pattern is almost periodic: at Hamming distance $< 2k$ from a string with period $\mathcal{O}(m/k)$.

The structure of Pattern Matching with Mismatches was understood only recently!

Long history of algorithmic results. Some of these heavily exploited the periodic structure of P and T, and implied some (weak) structural results.

First explicit structural result due to Bringmann, Künnemann, and Wellnitz [BKW'19].

Tightened by C., Kociumaka, and Wellnitz in [CKW'20]; we showed that at least one of the following holds:

- The pattern P has $\mathcal{O}(k)$ k-mismatch occurrences in T.
- The pattern is almost periodic: at Hamming distance $< 2k$ from a string with period $\mathcal{O}(m/k)$.

This structural insight leads to an alternative $\mathcal{O}(n + k^2)$-time algorithm [CKW'20].

Our Result

Our Result

An easy reduction to the no-wildcards case yields an analogous combinatorial result and an algorithm with runtime $\mathcal{O}(n + (D + k)^2)$ for our problem.

Our Result

An easy reduction to the no-wildcards case yields an analogous combinatorial result and an algorithm with runtime $\mathcal{O}(n + (D + k)^2)$ for our problem.

We obtain an algorithm with runtime $\mathcal{O}(n + (D + k) \cdot (G + k))$ and a tighter combinatorial bound by opening the black box of [CKW'20].

Our Result

An easy reduction to the no-wildcards case yields an analogous combinatorial result and an algorithm with runtime $\mathcal{O}(n + (D + k)^2)$ for our problem.

We obtain an algorithm with runtime $\mathcal{O}(n + (D + k) \cdot (G + k))$ and a tighter combinatorial bound by opening the black box of [CKW'20].

The k-mismatch occurrences of P in T can be decomposed into $\mathcal{O}((D + k)(G + k))$ arithmetic progressions. Lower bound: $\Omega((D + k)(k + 1))$.

What is the right answer?

Our Result

An easy reduction to the no-wildcards case yields an analogous combinatorial result and an algorithm with runtime $\mathcal{O}(n + (D + k)^2)$ for our problem.

We obtain an algorithm with runtime $\mathcal{O}(n + (D + k) \cdot (G + k))$ and a tighter combinatorial bound by opening the black box of [CKW'20].

The k-mismatch occurrences of P in T can be decomposed into $\mathcal{O}((D + k)(G + k))$ arithmetic progressions. Lower bound: $\Omega((D + k)(k + 1))$.
What is the right answer?

Bonus: A simple $\mathcal{O}(n + DG)$-time algorithm for exact PM with wildcards.

Warm-up: Choosing a chunk and extending seeds

Warm-up: Choosing a chunk and extending seeds

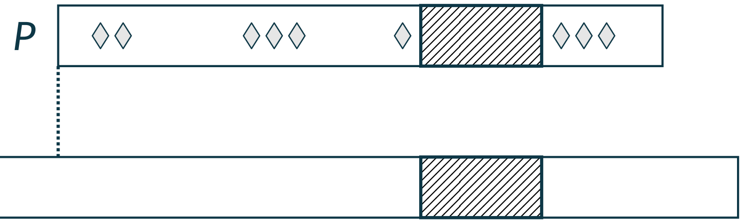
This technique is also useful in practice.

Warm-up: Choosing a chunk and extending seeds

This technique is also useful in practice.

Observation: P contains a chunk C of length $\Theta(m/G)$ that does not contain $\Diamond$s.

$G = 4, D = 9$



Warm-up: Choosing a chunk and extending seeds

This technique is also useful in practice.

Observation: P contains a chunk C of length $\Theta(m/G)$ that does not contain $\diamond$s.

$G = 4, D = 9$

Algorithm strategy: Find the exact matches of C in T and try to extend them to matches of P. We can verify in $\mathcal{O}(G)$ time after $\mathcal{O}(n)$-time preprocessing.

Warm-up: Choosing a chunk and extending seeds

This technique is also useful in practice.

Observation: P contains a chunk C of length $\Theta(m/G)$ that does not contain $\diamond$s.

$G = 4, D = 9$

Algorithm strategy: Find the exact matches of C in T and try to extend them to matches of P. We can verify in $\mathcal{O}(G)$ time after $\mathcal{O}(n)$-time preprocessing.

Observation: If the chunk C is aperiodic, its occurrences cannot overlap by more than $|C|/2$ positions $\Rightarrow$ at most $n/(|C|/2) = \mathcal{O}(G \cdot n/m) = \mathcal{O}(G)$ occurrences.

Algorithm’s Overview

Algorithm's Overview

Carefully select a chunk. – one of the main novelties of this work

Algorithm's Overview

Carefully select a chunk. – one of the main novelties of this work

If the chunk has $\mathcal{O}(D)$ occurrences in T, verify each of them in $\mathcal{O}(G)$ time. – easy

Algorithm's Overview

Carefully select a chunk. – one of the main novelties of this work

If the chunk has $\mathcal{O}(D)$ occurrences in T, verify each of them in $\mathcal{O}(G)$ time. – easy

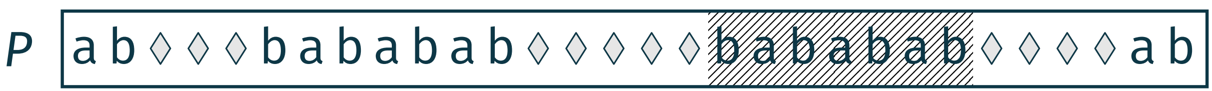
Else, the chunk has small period, that is, $\text{per}(C) = \mathcal{O}(m/D)$.

Carefully select a chunk. – one of the main novelties of this work

If the chunk has $\mathcal{O}(D)$ occurrences in T, verify each of them in $\mathcal{O}(G)$ time. – easy

Else, the chunk has small period, that is, $\text{per}(C) = \mathcal{O}(m/D)$.

- If the period extends to all of P, we use a sliding window approach. – easy



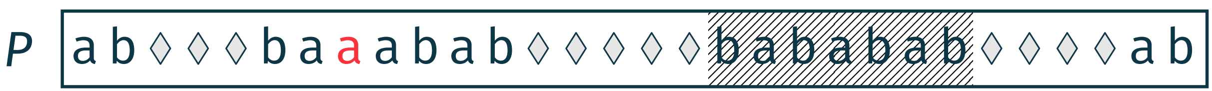
Algorithm's Overview

Carefully select a chunk. – one of the main novelties of this work

If the chunk has $\mathcal{O}(D)$ occurrences in T, verify each of them in $\mathcal{O}(G)$ time. – easy

Else, the chunk has small period, that is, $\mathrm{per}(C) = \mathcal{O}(m/D)$.

- If the period extends to all of P, we use a sliding window approach. – easy
- Else, we have to work a bit more. :)



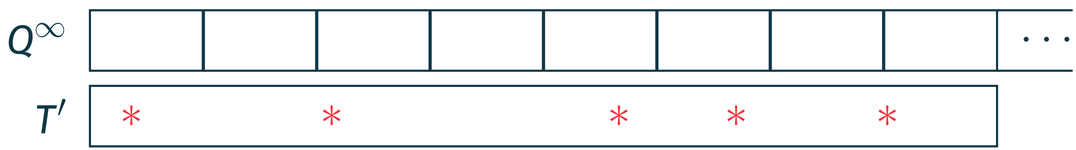
The Almost Periodic Case

The Almost Periodic Case

Setting: P matches a prefix of Q^{∞}, where Q is a string that does not contain wildcards and is of length $\mathcal{O}(m/D)$.

Setting: P matches a prefix of Q^∞, where Q is a string that does not contain wildcards and is of length $\mathcal{O}(m/D)$.

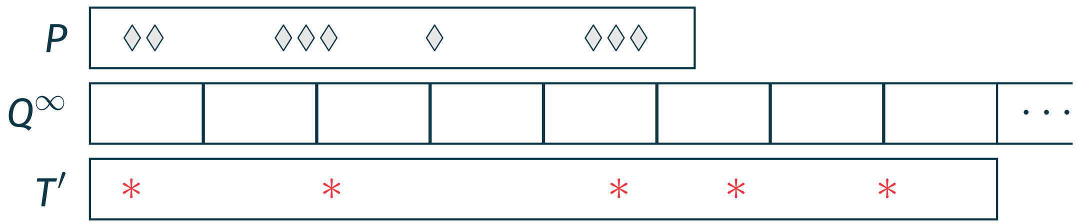
Adapted lemma from [CKW'20]: We can efficiently compute a substring T' of T that contains all occurrences of P and is at distance $\mathcal{O}(D)$ from a prefix of Q^∞.



The Almost Periodic Case

Setting: P matches a prefix of Q^∞, where Q is a string that does not contain wildcards and is of length $\mathcal{O}(m/D)$.

Adapted lemma from [CKW'20]: We can efficiently compute a substring T' of T that contains all occurrences of P and is at distance $\mathcal{O}(D)$ from a prefix of Q^∞.

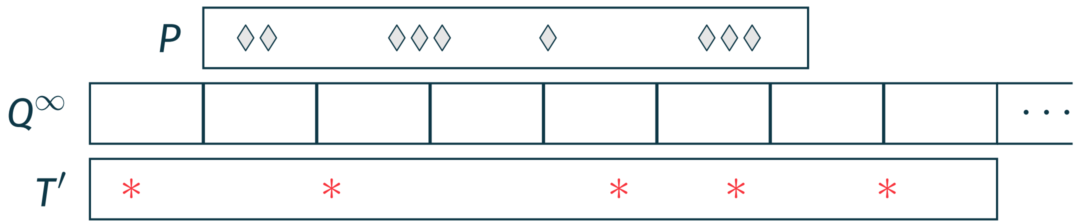


Conceptually, we slide P on T', $|Q|$ positions at a time. There is an exact occurrence whenever all the misperiods on the sliding window are aligned with $\lozenge$s.

The Almost Periodic Case

Setting: P matches a prefix of Q^∞, where Q is a string that does not contain wildcards and is of length $\mathcal{O}(m/D)$.

Adapted lemma from [CKW'20]: We can efficiently compute a substring T' of T that contains all occurrences of P and is at distance $\mathcal{O}(D)$ from a prefix of Q^∞.

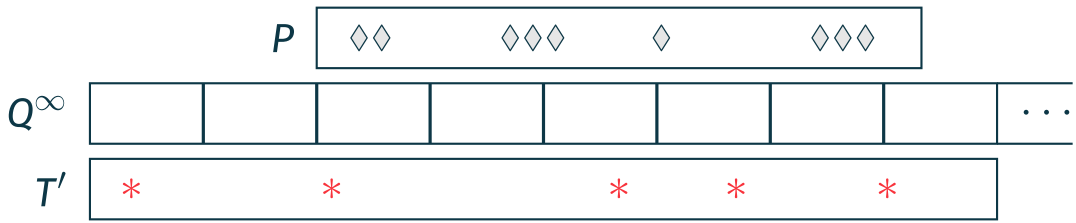


Conceptually, we slide P on T', $|Q|$ positions at a time. There is an exact occurrence whenever all the misperiods on the sliding window are aligned with $\diamond$s.

The Almost Periodic Case

Setting: P matches a prefix of Q^∞, where Q is a string that does not contain wildcards and is of length $\mathcal{O}(m/D)$.

Adapted lemma from [CKW'20]: We can efficiently compute a substring T' of T that contains all occurrences of P and is at distance $\mathcal{O}(D)$ from a prefix of Q^∞.

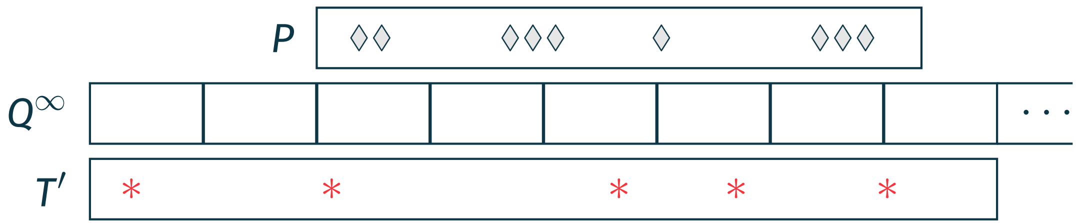


Conceptually, we slide P on T', $|Q|$ positions at a time. There is an exact occurrence whenever all the misperiods on the sliding window are aligned with $\diamond$s.

The Almost Periodic Case

Setting: P matches a prefix of Q^∞, where Q is a string that does not contain wildcards and is of length $\mathcal{O}(m/D)$.

Adapted lemma from [CKW'20]: We can efficiently compute a substring T' of T that contains all occurrences of P and is at distance $\mathcal{O}(D)$ from a prefix of Q^∞.



Conceptually, we slide P on T', $|Q|$ positions at a time. There is an exact occurrence whenever all the misperiods on the sliding window are aligned with $\diamond$s.

$\mathcal{O}(DG)$ events yielding $\mathcal{O}(DG)$ arithmetic progressions with difference $|Q|$.

Getting a feel for the hard case via [BKW'19]

Getting a feel for the hard case via [BKW'19]

Setting: The chunk C has period $\mathcal{O}(m/D)$, but this does not extend to all of P.

Getting a feel for the hard case via [BKW'19]

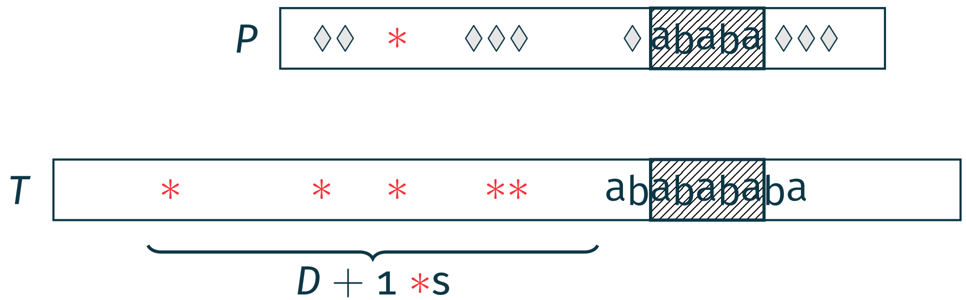
Setting: The chunk C has period $\mathcal{O}(m/D)$, but this does not extend to all of P.

Structure of exact pattern matching $\Rightarrow$ occurrences of C in T can be decomposed to $\mathcal{O}(G)$ disjoint arithmetic progressions (C-runs).

Getting a feel for the hard case via [BKW'19]

Setting: The chunk C has period $\mathcal{O}(m/D)$, but this does not extend to all of P.

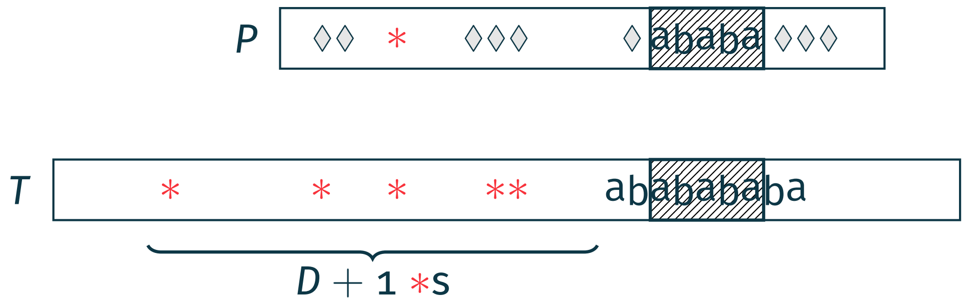
Structure of exact pattern matching $\Rightarrow$ occurrences of C in T can be decomposed to $\mathcal{O}(G)$ disjoint arithmetic progressions (C-runs).



Getting a feel for the hard case via [BKW'19]

Setting: The chunk C has period $\mathcal{O}(m/D)$, but this does not extend to all of P.

Structure of exact pattern matching $\Rightarrow$ occurrences of C in T can be decomposed to $\mathcal{O}(G)$ disjoint arithmetic progressions (C-runs).



Observation: The misperiod in P must be aligned with one of the first $D+1$ misperiods in T. We thus have $\mathcal{O}(D)$ candidates, and each can be verified in $\mathcal{O}(G)$ time. Total time: $\mathcal{O}(DG^2)$.

Improvement

Improvement

Goal: Try to generate less candidates per C-run (on average).

Improvement

Goal: Try to generate less candidates per C-run (on average).

A helpful assumption: The $\Diamond$s in P are well-spread around the chunk: every substring U of P that contains C has $\mathcal{O}(|U| \cdot D/m)$ $\Diamond$s.

Improvement

Goal: Try to generate less candidates per C-run (on average).

A helpful assumption: The $\Diamond$s in P are well-spread around the chunk: every substring U of P that contains C has $\mathcal{O}(|U| \cdot D/m)$ $\Diamond$s.

Exploiting it: As we compute misperiods in T one by one, we stop if they become too dense; our wildcards are sparse and hence cannot hide all of them.

Improvement

Goal: Try to generate less candidates per C-run (on average).

A helpful assumption: The $\Diamond$s in P are well-spread around the chunk: every substring U of P that contains C has $\mathcal{O}(|U| \cdot D/m)$ $\Diamond$s.

Exploiting it: As we compute misperiods in T one by one, we stop if they become too dense; our wildcards are sparse and hence cannot hide all of them.

Amortisation: If while extending a C-run with misperiods, we reach another run that is synchronised (i.e., their starting positions differ by a multiple of the period), we do not need to process the latter C-run.

Improvement

Goal: Try to generate less candidates per C-run (on average).

A helpful assumption: The $\Diamond$s in P are well-spread around the chunk: every substring U of P that contains C has $\mathcal{O}(|U| \cdot D/m)$ $\Diamond$s.

Exploiting it: As we compute misperiods in T one by one, we stop if they become too dense; our wildcards are sparse and hence cannot hide all of them.

Amortisation: If while extending a C-run with misperiods, we reach another run that is synchronised (i.e., their starting positions differ by a multiple of the period), we do not need to process the latter C-run.

A periodicity-based argument yields that we now need to verify $\mathcal{O}(D)$ candidates over all C-runs! Total time: $\mathcal{O}(DG)$.

What about the assumption?

What about the assumption?

Lemma: Let V be a binary vector of size N with $M := \|V\|$ 1s. We can efficiently compute a large set $U \subseteq [1\,..\,N]$ such that for each $i \in U$ and radius $r \in [1\,..\,N]$, $\|B_V(i, r)\| \leq 8r \cdot M/N$.

What about the assumption?

Lemma: Let V be a binary vector of size N with $M := \|V\|$ 1s. We can efficiently compute a large set $U \subseteq [1 \,..\, N]$ such that for each $i \in U$ and radius $r \in [1 \,..\, N]$, $\|B_V(i, r)\| \leq 8r \cdot M/N$.

We simply apply the above lemma with $\Diamond$s mapped to 1s and other letters mapped to 0s and then select the chunk so that it contains a position in U. We call such positions sparsifiers.

What about mismatches?

What about mismatches?

We open the black-box of [CKW'20], ensure that some of the considered substrings contain sparsifiers, and refine the analysis.

Lower Bound on the Arithmetic Progressions

Lower Bound on the Arithmetic Progressions

Large progression-free sets: For any sufficiently large M, there exists an integer $n_M = \mathcal{O}(M 2^{\sqrt{\log M}})$ and a progression-free set S such that S has cardinality M and $S \subseteq [n_M]$. [Elkin'22]

Lower Bound on the Arithmetic Progressions

Large progression-free sets: For any sufficiently large M, there exists an integer $n_M = \mathcal{O}(M 2^{\sqrt{\log M}})$ and a progression-free set S such that S has cardinality M and $S \subseteq [n_M]$. [Elkin'22]

We use such sets to construct P and T such that P has $\Omega((D+k)\cdot(k+1))$ k-mismatch occurrences in T and no three occurrences form an arithmetic progression.

Final Remarks

Final Remarks

We implement our algorithm in the PILLAR model; it is only based on a set of primitive operations of strings. We thus obtain efficient algorithms for the problem in scope in several other settings:

Final Remarks

We implement our algorithm in the PILLAR model; it is only based on a set of primitive operations of strings. We thus obtain efficient algorithms for the problem in scope in several other settings:

- when both strings are given in compressed form (e.g., as SLPs);

Final Remarks

We implement our algorithm in the PILLAR model; it is only based on a set of primitive operations of strings. We thus obtain efficient algorithms for the problem in scope in several other settings:

- when both strings are given in compressed form (e.g., as SLPs);
- when we maintain a dynamic collection of strings;

Final Remarks

We implement our algorithm in the PILLAR model; it is only based on a set of primitive operations of strings. We thus obtain efficient algorithms for the problem in scope in several other settings:

- when both strings are given in compressed form (e.g., as SLPs);
- when we maintain a dynamic collection of strings;
- in the quantum setting;

Final Remarks

We implement our algorithm in the PILLAR model; it is only based on a set of primitive operations of strings. We thus obtain efficient algorithms for the problem in scope in several other settings:

- when both strings are given in compressed form (e.g., as SLPs);
- when we maintain a dynamic collection of strings;
- in the quantum setting;
- etc.

Final Remarks

We implement our algorithm in the PILLAR model; it is only based on a set of primitive operations of strings. We thus obtain efficient algorithms for the problem in scope in several other settings:

- when both strings are given in compressed form (e.g., as SLPs);
- when we maintain a dynamic collection of strings;
- in the quantum setting;
- etc.

Open problems:

Final Remarks

We implement our algorithm in the PILLAR model; it is only based on a set of primitive operations of strings. We thus obtain efficient algorithms for the problem in scope in several other settings:

- when both strings are given in compressed form (e.g., as SLPs);
- when we maintain a dynamic collection of strings;
- in the quantum setting;
- etc.

Open problems:

- Is the algorithm optimal?

Final Remarks

We implement our algorithm in the PILLAR model; it is only based on a set of primitive operations of strings. We thus obtain efficient algorithms for the problem in scope in several other settings:

- when both strings are given in compressed form (e.g., as SLPs);
- when we maintain a dynamic collection of strings;
- in the quantum setting;
- etc.

Open problems:

- Is the algorithm optimal?
- Close the gap on the number of arithmetic progressions.

Final Remarks

We implement our algorithm in the PILLAR model; it is only based on a set of primitive operations of strings. We thus obtain efficient algorithms for the problem in scope in several other settings:

- when both strings are given in compressed form (e.g., as SLPs);
- when we maintain a dynamic collection of strings;
- in the quantum setting;
- etc.

Open problems:

- Is the algorithm optimal?
- Close the gap on the number of arithmetic progressions.
- Edit distance instead of Hamming?

Final Remarks

We implement our algorithm in the PILLAR model; it is only based on a set of primitive operations of strings. We thus obtain efficient algorithms for the problem in scope in several other settings:

- when both strings are given in compressed form (e.g., as SLPs);
- when we maintain a dynamic collection of strings;
- in the quantum setting;
- etc.

Open problems:

- Is the algorithm optimal?
- Close the gap on the number of arithmetic progressions.
- Edit distance instead of Hamming?
- More applications for sparsifiers?

The End

Thank you for your attention!

Questions?