Pattern Matching with Mismatches and Wildcards

Gabriel Bathie
DIENS, École normale supérieure de Paris, PSL Research University, France
LaBRI, Université de Bordeaux, Talence, France

Panagiotis Charalampopoulos
Birkbeck, University of London, UK

Tatiana Starikovskaya
DIENS, École normale supérieure de Paris, PSL Research University, France

Abstract

In this work, we address the problem of approximate pattern matching with wildcards. Given a pattern P of length m containing D wildcards, a text T of length n, and an integer k, our objective is to identify all fragments of T within Hamming distance k from P.

Our primary contribution is an algorithm with runtime $\mathcal{O}(n + (D + k)(G + k) \cdot n/m)$ for this problem. Here, $G \leq D$ represents the number of maximal wildcard fragments in P. We derive this algorithm by elaborating in a non-trivial way on the ideas presented by [Charalampopoulos, Kociumaka, and Wellnitz, FOCS'20] for pattern matching with mismatches (without wildcards). Our algorithm improves over the state of the art when D, G, and k are small relative to n. For instance, if $m = n/2$, $k = G = n^{2/5}$, and $D = n^{3/5}$, our algorithm operates in $\mathcal{O}(n)$ time, surpassing the $\Omega(n^{6/5})$ time requirement of all previously known algorithms.

In the case of *exact* pattern matching with wildcards ($k = 0$), we present a much simpler algorithm with runtime $\mathcal{O}(n + DG \cdot n/m)$ that clearly illustrates our main technical innovation: the utilisation of positions of P that do not belong to any fragment of P with a density of wildcards much larger than D/m as anchors for the sought (approximate) occurrences. Notably, our algorithm outperforms the best-known $\mathcal{O}(n \log m)$-time FFT-based algorithms of [Cole and Hariharan, STOC'02] and [Clifford and Clifford, IPL'04] if $DG = o(m \log m)$.

We complement our algorithmic results with a structural characterization of the k-mismatch occurrences of P. We demonstrate that in a text of length $\mathcal{O}(m)$, these occurrences can be partitioned into $\mathcal{O}((D + k)(G + k))$ arithmetic progressions. Additionally, we construct an infinite family of examples with $\Omega((D + k)k)$ arithmetic progressions of occurrences, leveraging a combinatorial result on progression-free sets [Elkin, SODA'10].

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases pattern matching, wildcards, mismatches, Hamming distance

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.20

Related Version *Full Version*: `https://arxiv.org/abs/2402.07732`

Funding A research visit during which part of the presented ideas were conceived was funded by a Royal Society International Exchanges Award.
Gabriel Bathie: Partially supported by the grant ANR-20-CE48-0001 from the French National Research Agency (ANR).
Tatiana Starikovskaya: Partially supported by the grant ANR-20-CE48-0001 from the French National Research Agency (ANR).

1 Introduction

Pattern matching is one of the most fundamental algorithmic problems on strings. Given a text T of length n and a pattern P of length m, both over an alphabet Σ, the goal is to compute all occurrences of P in T. This problem admits an efficient linear-time solution, for example using the seminal algorithm of Knuth, Morris, and Pratt [30]. However, looking for exact matches of P in T can be too restrictive for some applications, for example when working with potentially corrupted data, when accounting for mutations in genomic data, or when searching for an incomplete pattern. There are several ways to model and to work with corrupt or partial textual data for the purposes of pattern matching.

A natural and well-studied problem is that of computing fragments of the text that are close to the pattern with respect to some distance metric. One of the most commonly used such metrics is the Hamming distance. Abrahamson and Kosaraju [1, 32] independently developed an $\mathcal{O}(n\sqrt{m \log m})$-time algorithm that computes the Hamming distance between the pattern and every m-length substring of the text using convolutions via the Fast Fourier Transform (FFT). This complexity has only been recently improved with the state of the art being the randomised $\mathcal{O}(n\sqrt{m})$-time algorithm of Chan, Jin, Vassilevska Williams, and Xu [9] and the deterministic $\mathcal{O}(n\sqrt{m \log \log m})$-time algorithm of Jin and Xu [28]. In many applications, one is interested in computing substrings of the text that are *close* to the pattern instead of computing the distance to every substring. In this case, an integer threshold k is given as part of the input and the goal is to compute fragments of T that have at most k mismatches with P. Such a fragment is called a *k-mismatch occurrence* of P in T. The state-of-the-art for this problem are the $\mathcal{O}(n\sqrt{k \log k})$-time algorithm of Amir, Lewenstein, and Porat [3], the $\mathcal{O}(n + (n/m) \cdot k^2)$-time algorithm of Chan, Golan, Kociumaka, Kopelowitz, and Porat [8], and the $\mathcal{O}(n+kn/\sqrt{m})$-time algorithm of Chan et al. [9] that provides a smooth trade-off between the two aforementioned solutions, improving the bound for some range of parameters. Deterministic counterparts of the last two algorithms (which are randomised) at the expense of extra polylogarithmic factors were presented in [15, 24, 11].

The structure of the set of k-mismatch occurrences of P in T admits an insightful characterisation, shown by Charalampopoulos, Kociumaka, and Wellnitz [11] who tightened the result of Bringmann, Künnemann, and Wellnitz [7]: either P has $\mathcal{O}(k \cdot n/m)$ k-mismatch occurrences in T or P is at Hamming distance less than $2k$ from a string with period $q = \mathcal{O}(m/k)$; further, in the periodic case, the starting positions of the k-mismatch occurrences of P in T can be partitioned into $\mathcal{O}(k^2 \cdot n/m)$ arithmetic progressions with difference q. This characterisation can be exploited towards obtaining efficient algorithms in settings other than the standard one, e.g., in the setting where both P and T are given in compressed form [11], and, in combination with other ideas and techniques, in the streaming setting [31] and in the quantum setting [27].

In the case when the positions of the corrupt characters in the two strings are known in advance, one can use a more adaptive approach, by placing a *wildcard* $\lozenge \notin \Sigma$, a special character that matches any character in $\Sigma \cup \{\lozenge\}$, in each of these positions, and then performing exact pattern matching. Already in 1974, Fischer and Paterson [36] presented an $\mathcal{O}(n \log m \log \sigma)$-time algorithm for the pattern matching problem with wildcards. Subsequent works by Indyk [26], Kalai [29], and Cole and Hariharan [17] culminated in an $\mathcal{O}(n \log m)$-time deterministic algorithm [17]. A few years later, Clifford and Clifford [12] presented a very elegant algorithm with the same complexities. All the above solutions are based on fast convolutions via the FFT.

Unsurprisingly, the pattern matching problem in the case where we both have wildcards and allow for mismatches has also received significant attention. Conceptually, it covers the case where some of the corrupt positions are known, but not all of them. We denote by D the total number of wildcards in P and T, and by G the number of maximal fragments in P and T all of whose characters are wildcards. A summary of known results for the considered problem is provided in Table 1. Note that, as discussed below, any algorithm for the pattern

Table 1 Results on pattern matching with wildcards under the Hamming distance.

Time complexity	$\lozenge$ in	Reference
$\mathcal{O}(nk^2 \log^2 m)$	P and T	[14]
$\mathcal{O}(n(k + \log m \log k) \log n)$		[14]
$\mathcal{O}(nk \operatorname{polylog} m)$		[13, 14, 37]
$\mathcal{O}(n\sqrt{m \log m})$		follows from [1, 32], cf. [14]
$\mathcal{O}(n\sqrt{m - D} \log m)$		[3]
$\mathcal{O}(n \sqrt[3]{mk \log^2 m})$	one of P or T	[16]
$\mathcal{O}(n\sqrt{k \log m} + n \cdot \min\{\sqrt[3]{Gk \log^2 m}, \sqrt{G \log m}\})$	P	[38]
$\mathcal{O}(n + (n/m)(D + k)(G + k))$	P	**this work**

matching problem with at most k mismatches (without wildcards) can be applied to the setting where we have wildcards only in the pattern at the expense of allowing for $k + D$ mismatches instead of k (and hence replacing any factor of k in the complexity by a factor $k + D$). We chose to not include the implied results in the table to avoid clutter.

Note that, in practice, G may be much smaller than D. For example, DNA sequences have biologically important loci, which are characterised using the notion of structured motifs [35]: sequences of alternating conserved and non-conserved blocks. Conserved blocks are ones which are identical across intra- or inter-species occurrences of the structured motif, while non-conserved ones are not known to have biological significance and can vary significantly across such occurrences. Non-conserved blocks can be hence modelled with blocks of wildcards as in [34]. In this case, evidently, we have G being much smaller than D. This feature has been used in the literature before, e.g., for the problem of answering longest common compatible prefix queries over a string with wildcards. Crochemore et al. [19] showed a $\mathcal{O}(nG)$-time construction algorithm for a data structure that is capable of answering such queries in $\mathcal{O}(1)$ time, while the previously best known construction time was $\mathcal{O}(nD)$ [6].

In several applications, it is sufficient to only account for wildcards in one of P and T: in the application we just discussed, the text is a fixed DNA sequence, whereas the sought pattern, the structured motif, is modelled as a string with wildcards. In such cases, one can obtain more efficient solutions than those for the general case where both P and T have wildcards, such as the ones presented in [16, 38] and the one we present here.

Multi-framework algorithms with the PILLAR model. We describe our algorithms in the PILLAR model, introduced by Charalampopoulos et al. [11]. In this model, we account for the number of calls to a small set of versatile primitive operations on strings, called PILLAR operations, such as longest common extension queries or internal pattern matching queries, plus any extra time required to perform usual word RAM operations. The PILLAR model allows for a unified approach across several settings, due to known efficient implementations of PILLAR operations. These settings include the standard word RAM model, the compressed setting, where the strings are compressed as straight-line programs, the dynamic setting [23], and the quantum setting. Therefore, in essence, we provide meta-algorithms, that can be combined with efficient PILLAR implementations to give efficient algorithms for a variety of settings.

The standard trick. For reasons related to the periodic structure of strings, it is often convenient to assume that the length of the text is at most $3m/2$, where m is the length of the pattern. This does not pose any actual restrictions as one can cover T with $\mathcal{O}(n/m)$ fragments, each of length $3m/2$ (except maybe the last one) such that each two consecutive fragments overlap on $m-1$ positions. Then, any occurrence of P in T is contained in exactly one of these fragments. Thus, an algorithm with runtime $C(m)$ for a pattern of length m and a text of length at most $3m/2$, readily implies an algorithm with runtime $\mathcal{O}(C(m) \cdot n/m)$ for texts of length n, as one can run $\mathcal{O}(n/m)$ separate instances and aggregate the results.

Reduction to pattern matching with mismatches. The problem of k-mismatch pattern matching with D wildcards can be straightforwardly reduced to $(D+k)$-mismatch pattern matching in *solid strings*, i.e., strings without wildcards. In what follows we consider solid texts. Given the pattern P, construct the string $P_\#$ obtained by replacing every wildcard in P with a new character $\# \notin \Sigma$. Observe that a pattern P with D wildcards has a k-mismatch occurrence at a position i of a solid text T if and only if $P_\#$ has a $(D+k)$-mismatch occurrence at that position.

In [11], the authors present an efficient algorithm for the d-mismatch pattern matching problem for solid strings in the `PILLAR` model.

▶ **Theorem 1.1** ([11, Main Theorem 8]). *Let S and T be solid strings of respective lengths m and $n \le 3m/2$. We can compute a representation of the d-mismatch occurrences of S in T using $\mathcal{O}(d^2 \log\log d)$ time plus $\mathcal{O}(d^2)$ `PILLAR` operations.*

Applying Theorem 1.1 with $S = P_\#$ and $d = D + k$, we obtain an algorithm for k-mismatch pattern matching with D wildcards that runs in $\tilde{\mathcal{O}}((D+k)^2)$ time in the `PILLAR` model.[1]

Our results. We provide a more fine-grained result, replacing one D factor with a G factor. We also make an analogous improvement over the structural result for the set of k-mismatch occurrences obtained via the reduction to $(D+k)$-mismatch pattern matching. Our main result can be formally stated as follows.

▶ **Theorem 1.2.** *Let P be a pattern of length m with D wildcards arranged in G groups, T be a solid text of length $n \le 3m/2$, and k be a positive integer. We can compute a representation of the k-mismatch occurrences of P in T as $\mathcal{O}((D+k)G)$ arithmetic progressions with common difference and $\mathcal{O}((D+k)k)$ additional occurrences using $\mathcal{O}((D+k) \cdot (G+k) \log\log(D+k))$ time plus $\mathcal{O}((D+k) \cdot (G+k))$ `PILLAR` operations.*

In the usual word RAM model, by using known implementations of the `PILLAR` operations with $O(n)$ preprocessing time and $O(1)$ operation time, using the "standard trick", and observing that the loglogarithmic factor can be avoided at the cost of $\mathcal{O}(n)$ extra time, we obtain an algorithm with runtime $\mathcal{O}(n + (n/m)(D+k)(G+k))$ for texts of arbitrary length n. The full version of this work details the implementation of our algorithm in other settings, such as the dynamic and compressed settings. For example, given a solid text T and a pattern P with D wildcards represented as straight-line programs of sizes N and M respectively, we can compute the number of k-mismatch occurrences of P in T in time $\tilde{\mathcal{O}}(M + N \cdot (D+k)(G+k))$, without having to uncompress P and T.

We complement our structural result with a lower bound on the number of arithmetic progressions of occurrences of a pattern with mismatches and wildcards (Theorem 5.1), based on a neat construction that employs large sets that do not contain any arithmetic progression

[1] In this work, the notation $\tilde{\mathcal{O}}(\cdot)$ suppresses factors polylogarithmic in the length of the input strings.

of size 3 [21, 5, 20]. Informally, we show that there exist a pattern P and a text T of length at most $3|P|/2$ such that the set of k-mismatch occurrences of P in T cannot be covered with less than $\Omega((D+k)\cdot(k+1))$ arithmetic progressions. This implies, in particular, a lower bound of $\Omega(D)$ on the number of arithmetic progressions of exact occurrences for a pattern with D wildcards and a lower bound of $\Omega(k^2)$ on the number of arithmetic progressions of k-mismatch occurrences of a solid pattern, thus showing the tightness of the known upper bound [11].

When $k=0$, Theorem 1.2 readily implies an $\mathcal{O}(n+DG\cdot n/m)$-time algorithm for *exact* pattern matching. However, the techniques employed by this algorithm are rather heavy-handed, and this can be avoided. In Section 3, we present a much simpler algorithm that achieves the same time complexity and showcases the primary technical innovation of our approach: the utilization of carefully selected positions, termed *sparsifiers*, which exclusively belong to fragments F of P such that the ratio of the number of wildcards within them to their length is bounded by $\mathcal{O}(D/m)$. In the standard word RAM model, the implied $\mathcal{O}(n+DG\cdot n/m)$ time complexity for exact pattern matching outperforms the state-of-the-art $\mathcal{O}(n\log m)$ [12, 17] when $DG=o(m\log m)$.

Technical overview. To illustrate how sparsifiers help, consider our algorithm for exact pattern matching, which draws ideas from the work of Bringmann et al. [7]. We first compute a solid $\Omega(m/G)$-length fragment S of P that contains a sparsifier. We then compute its exact matches in T. If S only has a few occurrences, we straightforwardly verify which of those extend to occurrences of P. However, if S has many occurrences we cannot afford to do that, and we instead have to exploit the implied periodic structure of S. We distinguish between two cases. In the case when P matches a periodic string with the same period as S, denoted $\mathsf{per}(S)$, we take a sliding window approach as in [7], using the fact that the wildcards are organised in only G groups. The remaining case poses the main technical challenge. In that case, our goal is to align the maximal fragment $S' := P[i\,..\,j]$ of P that contains S and matches a solid string with period $\mathsf{per}(S)$ with a periodic fragment of T such that position $i-1$ is aligned with a position breaking the periodicity in T; a so-called *misperiod*. To this end, we compute $\mathcal{O}(G)$ maximal fragments of T, called S-runs, that have period $\mathsf{per}(S)$. The issue is, however, that up to D misperiods in T might be aligned with wildcards of S'. A straightforward approach would be to extend each S-run to the left, allowing for $D+1$ misperiods and to try aligning each such misperiod with $i-1$. This would yield an algorithm with runtime $\mathcal{O}(G^2D)$ in the `PILLAR` model, as we would have $\mathcal{O}(DG)$ candidate misperiods to align position $i-1$ with, and the verification time for each such alignment is $\mathcal{O}(G)$. The crucial observation is that since S' contains a sparsifier, we do not need to extend each S-run allowing for $D+1$ misperiods. Instead, we extend it while the ratio of the encountered misperiods to its length does not exceed $20\cdot D/m$. By skipping S-runs that are covered due to the extension of other S-runs, we ensure that the total number of misperiods with which we align $i-1$ is only $\mathcal{O}(D)$, obtaining the desired complexity.

Other related work. The pattern matching problem with wildcards under the edit distance has also been studied. A straightforward adaptation of the $\mathcal{O}(nk)$-time algorithm of Landau and Vishkin [33] for pattern matching under edit distance for solid P and T yields an algorithm with running time $\mathcal{O}(n(k+G))$. Akutsu [2] presented an algorithm running in time $\mathcal{O}(n\sqrt{mk}\operatorname{polylog} m)$. Recently, an algorithm with runtime $\mathcal{O}(n(k+\sqrt{Gk\log n}))$ was presented [4], improving over both previously known algorithms when $k \ll G \ll m$. The aforementioned algorithms can handle the case where both P and T contain wildcards.

2 Preliminaries

In this work, Σ denotes an alphabet that consists of integers polynomially bounded in the length of the input strings. The elements of Σ are called *(solid) characters*. Additionally, we consider a special character denoted by $\lozenge$ that is not in Σ and is called a *wildcard*. Let $\Sigma_\lozenge = \Sigma \cup \{\lozenge\}$. Two characters match if (a) they are identical or (b) at least one of them is a wildcard. Two equal-length strings match if and only if their i-th characters match for all i.

For an integer $n \geq 0$, we denote the set of all length-n strings over an alphabet A by A^n. The set of all strings over A is denoted by A^*. The unique empty string is denoted by ε. A string in $\Sigma_\lozenge^*$ is called *solid* if it only contains solid characters, i.e., it is in Σ^*.

For two strings $S, T \in \Sigma_\lozenge^*$, we use ST to denote their concatenation. For an integer $m > 0$, the string obtained by concatenating m copies of S is denoted by S^m. We denote by S^∞ the string obtained by concatenating infinitely many copies of S.

For a string $T \in \Sigma_\lozenge^n$ and an index $i \in [1\,..\,n]$,[2] the i-th character of T is denoted by $T[i]$. We use $|T| = n$ to denote the length of T. For indices $1 \leq i, j \leq n$, $T[i\,..\,j]$ denotes the *fragment* $T[i]T[i+1]\cdots T[j]$ of T if $i \leq j$ and the empty string otherwise. We extend this notation in a natural way to $T[i\,..\,j+1) = T[i\,..\,j] = T(i-1\,..\,j]$. When $i = 1$ or $j = n$, we omit these indices, i.e., $T[\,..\,j] = T[1\,..\,j]$ and $T[i\,..\,] = T[i\,..\,n]$. A string P is a *prefix* of T if there exists $j \in [1\,..\,n]$ such that $P = T[\,..\,j]$, and a *suffix* of T if there exists $i \in [1\,..\,n]$ such that $P = T[i\,..\,]$. A *ball* with a radius r and a center i, $B_T(i, r)$ is a fragment $T[\max\{1, i-r\}\,..\,\min\{i+r, n\}]$, where we often omit the subscript T if it is clear from the context. A position i of a string T is called an *occurrence* of a string P if $T[i\,..\,i+|P|) = P$.

A positive integer ρ is a *period* of a (solid) string $T \in \Sigma^n$ if $T[i] = T[i+\rho]$ for all $i \in [1\,..\,n-\rho]$. The smallest period of T is referred to as *the period* of T and is denoted by $\mathsf{per}(T)$. If $\mathsf{per}(T) \leq |T|/2$, T is called *periodic*. We exploit the following folklore fact, which is a straightforward corollary of the Fine–Wilf periodicity lemma [22]:

▶ **Corollary 2.1** (folklore). *Let $P, T \in \Sigma^*$ be solid strings such that $|T| < 2|P|$. The set of occurrences of P in T can be represented as one arithmetic progression (possibly, trivial) with difference equal to $\mathsf{per}(P)$.*

For an integer $s \in [1\,..\,n)$, we denote by $\mathsf{rot}^s(T)$ the string $T[s+1]\cdots T[n]T[1]\cdots T[s]$, while $\mathsf{rot}^0(T) := T$. More generally, for any $s \in \mathbb{Z}$, we denote by $\mathsf{rot}^s(T)$ the string $\mathsf{rot}^x(T)$, where $x = s \bmod |T|$. A non-empty (solid) string is called *primitive* if it is different from each of its non-trivial rotations.

▶ **Fact 2.2** ([18]). *For any solid string $T \in \Sigma^n$, the prefix $T[1\,..\,\mathsf{per}(T)]$ is primitive.*

An integer ρ is a *deterministic period* of a string $S \in \Sigma_\lozenge^*$ (that may contain wildcards), if there exists a solid string T that matches S and has period ρ.

The Hamming distance $\delta_H(S_1, S_2)$ between two equal-length strings S_1, S_2 in $\Sigma_\lozenge^*$ is the number of positions i such that $S_1[i]$ does not match $S_2[i]$. For two strings $U, Q \in \Sigma_\lozenge^*$, we slightly abuse notation and denote $\delta_H(U, Q^\infty[\,..\,|U|])$ by $\delta_H(U, Q^\infty)$. A position i of a string T is called a k-mismatch occurrence of a string P if $\delta_H(T[i, i+|P|), P) \leq k$, and the set of all k-mismatch occurrences of P in T is denoted by $\mathrm{Occ}_k(P, T)$.

The `PILLAR` model. The `PILLAR` model of computation, introduced in [11], abstracts away the implementation of a versatile set of primitive operations on strings. In this model, one is given a family of solid strings $\mathcal{X}$ for preprocessing. The elementary objects are fragments

[2] For integers $i, j \in \mathbb{Z}$, denote $[i\,..\,j] = \{k \in \mathbb{Z} : i \leq k \leq j\}$, $[i\,..\,j) = \{k \in \mathbb{Z} : i \leq k < j\}$.

$X[i\,..\,j]$ of strings $X \in \mathcal{X}$. Each such fragment S is represented via a handle, which is how S is passed as input to `PILLAR` operations. Initially, the model provides a handle to each $X \in \mathcal{X}$. Handles to other fragments can be obtained through an Extract operation:

- Extract(S, ℓ, r): Given a fragment S and positions $1 \le \ell \le r \le |S|$, extract $S[\ell\,..\,r]$.

Furthermore, given elementary objects S, S_1, S_2 the following primitive operations are supported in the `PILLAR` model:

- Access(S, i): Assuming $i \in [1\,..\,|S|]$, retrieve $S[i]$.
- Length(S): Retrieve the length $|S|$ of S.
- lcp(S_1, S_2): Compute the length of the longest common prefix of S_1 and S_2.
- $\text{lcp}^R(S_1, S_2)$: Compute the length of the longest common suffix of S_1 and S_2.
- Internal pattern matching IPM(S_1, S_2): Assuming that $|S_2| < 2|S_1|$, compute the set of the starting positions of occurrences of S_1 in S_2 represented as one arithmetic progression.

We use the following facts; the proof of Fact 2.4 is provided in the full version of this work.

▶ **Fact 2.3** ([10, proof of Lemma 12]). *The value* $\textit{lcp}(X^\infty, Z)$ *for a fragment* X *and a suffix* Z *of a solid string* Y *can be computed in* $\mathcal{O}(1)$ *time in the* `PILLAR` *model.*

We work in the `PILLAR` model despite considering strings with wildcards. We circumvent this by replacing each wildcard with a solid character $\# \notin \Sigma$ and using `PILLAR` operations over the obtained collection of (solid) strings. We ensure that for each string in the collection we can efficiently compute a linked list that stores the endpoints of groups of wildcards.

▶ **Fact 2.4.** *Let* P *be a pattern with* D *wildcards arranged in* G *groups and* T *be a solid text. For a position* p *and a given threshold* $k \ge 0$*, one can test whether* $\delta_H(P, T[i\,..\,i+m)) \le k$ *in* $\mathcal{O}(G + k)$ *time in the* `PILLAR` *model.*

Sparsifiers. In Section 1, we elucidated the pivotal role of the fragments of the pattern where wildcards exhibit a "typical" distribution. In this section, we formalize this concept.

▶ **Definition 2.5** (Sparsifiers). *Consider a string* $X \in \Sigma_\lozenge^m$ *containing* D *wildcards. We call a position* i *in* X *a* sparsifier *if* $X[i]$ *is a solid character and, for any* r*, the count of wildcards within the ball of radius* r *centered at* i *is at most* $8r \cdot D/m$.

In the following, we demonstrate that P contains a long fragment whose every position is a sparsifier. We start with an abstract lemma, the proof of which can be found in the full version of this work, where one can think of a binary vector V as the indicator vector for wildcards, and $\|V\|$ denotes the number of 1s in V. A *run of 1s (resp. 0s)* is a maximal fragment that consists only of 1s (resp. 0s).

▶ **Lemma 2.6.** *Let* V *be a binary vector of size* N*,* $M := \|V\|$ *and* R *be the number of runs of 1s in* V*. Assume* V *to be represented as a linked list of the endpoints of runs of 1s in* V*, arranged in the sorted order. There is an* $\mathcal{O}(R)$*-time algorithm that computes a set* $U \subseteq [1\,..\,N]$ *satisfying all of the following conditions:*

1. $|U| \ge N/2 - M$*,*
2. U *can be represented as a union of at most* $R + 1$ *disjoint intervals,*
3. *for each* $i \in U$ *and radius* $r \in [1\,..\,N]$*,* $\|B_V(i, r)\| \le 8r \cdot M/N$.

▶ **Corollary 2.7.** *Consider a string* $P \in \Sigma_\lozenge^m$ *containing* D *wildcards arranged in* G *groups. If* $D < m/4$*, then there is a fragment* S *of* P *of length* $L = \lfloor m/(8G) \rfloor$ *whose every position is a sparsifier, and one can compute* S *in* $\mathcal{O}(G)$ *time.*

Proof. An application of Lemma 2.6 to P with wildcards treated as 1s and solid characters treated as 0s returns $m/2 - D > m/4$ sparsifiers in the form of $G+1$ intervals in $\mathcal{O}(G)$ time. Thus, there is a fragment of size at least $m/(4(G+1)) \geq L$ whose every position is a sparsifier; the L-length prefix of this fragment satisfies the condition of the claim. ◀

3 Exact Pattern Matching in the PILLAR Model

In this section, we consider a pattern P of length m with $D \geq 1$ wildcards arranged in G groups and a solid text T of length n such that $n \leq 3m/2$. We then use the "standard trick" presented in the introduction to lift the result to texts of arbitrary length. We prove a structural result for the exact occurrences of P in T and show how to compute them efficiently when the product of D and G is small. In particular, we compute them in linear time when $DG = \mathcal{O}(m)$, thus improving by a logarithmic factor over the state-of-the-art $\mathcal{O}(n \log m)$-time algorithms in this case.

▶ **Definition 3.1** (Misperiods). *Consider a string V over alphabet $\Sigma_\diamond$. We say that a position x is a misperiod with respect to a solid fragment $V[i\,..\,j]$ when $V[x]$ does not match $V[y]$, where y is any position in $[i\,..\,j]$ such that $\mathsf{per}(V[i\,..\,j])$ divides $|y-x|$. Additionally, we consider positions 0 and $|V|+1$ as misperiods. We denote the set of the at most κ rightmost misperiods smaller than i with respect to $V[i\,..\,j]$ by $\mathsf{LeftMisper}(V,i,j,\kappa)$. Similarly, we denote the set of the at most κ leftmost misperiods larger than j with respect to $V[i\,..\,j]$ by $\mathsf{RightMisper}(V,i,j,\kappa)$.*

▶ **Example 3.2.** Consider string $V = \texttt{cc}\diamond\texttt{bd}\underline{\texttt{abcabcab}}\texttt{cab}$. The misperiods with respect to the underlined fragment $V[6\,..\,13]$, which has period 3, are positions 0, 1, 5, and $|V|+1 = 17$. We have $\mathsf{LeftMisper}(V,6,13,2) = \{1,5\}$ and $\mathsf{RightMisper}(V,6,13,2) = \{17\}$.

The next lemma states that the sets $\mathsf{LeftMisper}(V,i,j,k)$ and $\mathsf{RightMisper}(V,i,j,k)$ can be computed efficiently in an incremental fashion. Its proof, which can be found in the full version of this work, uses the kangaroo method and closely follows [7, 10].

▶ **Lemma 3.3.** *Consider a string V over an alphabet $\Sigma_\diamond$ and a solid periodic fragment $V[i\,..\,j]$ of V. The elements of either of $\mathsf{LeftMisper}(V,i,j,|V|)$ and $\mathsf{RightMisper}(V,i,j,|V|)$ can be computed in the increasing order with respect to their distance from position i so that:*

- *the first misperiod x can be computed in $\mathcal{O}(1+G_0)$ time in the PILLAR model, where G_0 denotes the number of groups of wildcards between positions x and i;*
- *given the t-th misperiod $x \notin \{0, |V|+1\}$, the $(t+1)$-th misperiod can be computed in $\mathcal{O}(1+G_t)$ time in the PILLAR model, where G_t denotes the number of groups of wildcards between said misperiods.*

A direct application of the above lemma yields the following fact.

▶ **Corollary 3.4.** *For any integer k, the sets $\mathsf{LeftMisper}(V,i,j,k)$ and $\mathsf{RightMisper}(V,i,j,k)$ can be computed in $\mathcal{O}(k+G)$ time in the PILLAR model.*

▶ **Definition 3.5.** *For two strings S and Q, let $\mathsf{MI}(S,Q)$ denote the set of positions of mismatches between S and Q^∞.*

▶ **Definition 3.6** (S-runs). *A fragment of a solid string V spanned by a set of occurrences of a solid string S in V whose starting positions form a maximal arithmetic progression with difference $\mathsf{per}(S)$ is called an S-run.*

▶ **Example 3.7.** Let $V = \texttt{cab}\underline{\texttt{abcabcabcab}}\texttt{c}$ and $S = \texttt{abcab}$. The underlined fragment $V[4\,..\,14]$ is the sole S-run in V; it is spanned by the occurrences of S at positions 4, 7, 10.

The following fact characterises the overlaps of S-runs; its proof can be found in the full version of this work.

▶ **Fact 3.8.** *Two S-runs can overlap by no more than $\mathsf{per}(S) - 1$ positions.*

We need a final ingredient before we prove the main theorem of this section. We state a more general variant of the statement than we need here that also accounts for k mismatches, as this will come handy in the subsequent section. For the purposes of this section one can think of k as 0. The following corollary follows from [11, Lemma 4.6] via the reduction to computing $(D+k)$-mismatch occurrences of $P_{\#}$ in T.

▶ **Corollary 3.9** (of [11, Lemma 4.6]). *Let S be a string of length m with D wildcards, let T be a solid string such that $|T| \leq 3|S|/2$, let $k \in [0\,..\,m]$ and $d \geq 2(D+k)$ be a positive integer, and let Q be a primitive solid string such that $|Q| \leq m/8d$ and $\delta_H(S, Q^\infty) \leq d$. Then, we can compute, in $\mathcal{O}(d)$ time in the* `PILLAR` *model, a fragment $T' = T[\ell\,..\,r]$ of T such that*
- *$\delta_H(T', Q^\infty) \leq 3d$, and*
- *all elements of $\mathrm{Occ}_k(S, T') = \{p - \ell : p \in \mathrm{Occ}_k(S, T)\}$ are equivalent to $0 \pmod{|Q|}$.*

▶ **Theorem 3.10.** *Consider a pattern P of length m with D wildcards arranged in G groups and a solid text T of length $n \leq 3m/2$. Either P has $\mathcal{O}(D)$ occurrences in T or P has a deterministic period $q = \mathcal{O}(m/D)$. A representation of the occurrences of P in T can be computed in $\mathcal{O}(DG \log\log D)$ time plus the time required to perform $\mathcal{O}(DG)$* `PILLAR` *operations. In the former case the occurrences are returned explicitly, while in the latter case they are returned as $\mathcal{O}(DG)$ arithmetic progressions with common difference q.*

Proof. First, observe that if $D = \Theta(m)$ the statement holds trivially as there can only be $\mathcal{O}(m)$ occurrences and we can compute them using $\mathcal{O}(mG)$ `PILLAR` operations, e.g., by applying Fact 2.4 for each position of the text. We thus henceforth assume that $D < m/4$.

We apply Corollary 2.7 to P, thus obtaining, in $\mathcal{O}(G)$ time, a fragment $S = P[x\,..\,y]$ of length $m/(8G)$ whose every position is a sparsifier. (As an implication, S is a solid fragment.) Then, we compute all occurrences of S in T in $\mathcal{O}(G)$ time in the `PILLAR` model, represented as $\mathcal{O}(G)$ arithmetic progressions with common difference $\mathsf{per}(S)$ (see Corollary 2.1).

Case (I): S has less than $384D$ occurrences in T. In this case, we try to extend each such occurrence to an occurrence of P in T using Fact 2.4 in $\mathcal{O}(G)$ time in the `PILLAR` model. This takes $\mathcal{O}(DG)$ time in total in the `PILLAR` model.

Case (II): S has at least $384D$ occurrences in T. In this case, we have two occurrences of S in T starting within $(3m/2)/(384D)$ positions of each other, and hence $\mathsf{per}(S) \leq m/(256D)$. Let $Q = S[1\,..\,\mathsf{per}(S)]$. By definition of $\mathsf{per}(S)$, S is a prefix of Q^∞ and by Fact 2.2, Q is primitive. Using Corollary 3.4, we compute the sets $\mathsf{LeftMisper}(P, x, y, 1)$ and $\mathsf{RightMisper}(P, x, y, 1)$ in $\mathcal{O}(G)$ time in the `PILLAR` model. In other words, we compute the maximal fragment V of P that contains S and matches exactly some substring of Q^∞.

Subcase (a): $V = P$. We conclude that $q := |Q| \leq m/(256D)$ is a deterministic period of P. We replace Q by its (possibly trivial) rotation Q_0 such that P is equal to a prefix of Q_0^∞ and then apply Corollary 3.9 to compute, in $\mathcal{O}(D)$ time in the `PILLAR` model, a fragment T' of T that contains the same number of occurrences of P as T, is at Hamming distance $\mathcal{O}(D)$ from a prefix of Q^∞, and only has occurrences of P at positions equivalent to $1 \pmod{q}$.

It now suffices to show how to compute the occurrences of P in T'. As in previous works [7, 11], we take a sliding window approach. Let W be the set of positions in P where we have a wildcard. For $i \in [1\,..\,|T'|-m+1]$, define $\mathsf{Hidden}(i)$ to be the size of the intersection of $\mathsf{MI}(T', Q) \cap [i \,.\,.\, i+m)$ with $i + W$.[3] Intuitively, this is the number of mismatches between $T'[i \,.\,.\, i+m)$ and Q^∞ that are aligned with a wildcard in P (and are hence "hidden") when we align P with $T'[i \,.\,.\, i+m)$. $\mathsf{Hidden}(\cdot)$ is a step function whose value changes $\mathcal{O}(DG)$ times as we increase i, since each mismatch enters or exits the window $[i \,.\,.\, i+m)$ at most once and whether it is hidden or not changes at most $2G$ times. We compute $\mathsf{Hidden}(1)$ and store the positions where the function changes (as well as by how much) as events in the increasing order; this sorting takes $\mathcal{O}(DG \log\log D)$ time [25].

For a position $i \le |T'| - m + 1$ with $i \equiv 1 \pmod q$, we have $d_i := \delta_H(T'[i \,.\,.\, i+m), P) = \mathsf{MI}(T'[i \,.\,.\, i+m), Q) - \mathsf{Hidden}(i)$. We maintain this value as we, intuitively, slide P along T, q positions at a time. If there are no events in $(i \,.\,.\, i+q] \subseteq [1 \,.\,.\, |T'|]$, then $d_i = d_{i+q}$. This allows us to report all occurrences of P in T efficiently as $\mathcal{O}(DG)$ arithmetic progressions with common difference q by processing all events in a left-to-right manner in $\mathcal{O}(DG)$ time.

Subcase (b): $V \neq P$. Our goal is to show that, in this case, the occurrences of P in T are $\mathcal{O}(D)$ and they can be computed in time $\mathcal{O}(GD)$. Without loss of generality, assume that V is not a prefix of P. This means that $\mathsf{LeftMisper}(P, x, y, 1) = \{\mu\} \neq \{0\}$. The occurrences of S in T give us a collection $\mathcal{S}$ of $\mathcal{O}(G)$ S-runs in T, any two of which can overlap by less than $\mathsf{per}(S) = q$ positions due to Fact 3.8. For each S-run R, extend R to the left until either of the following two conditions is satisfied, keeping track of the encountered misperiods:

(a) the ratio of encountered misperiods to the sum of $|R|$ and the number of prepended positions exceeds $20D/m$,

(b) the beginning of T has been reached.

Denote by E_R the resulting fragment of T and by $\mathcal{M}_R$ the set of misperiods in it. The following two claims are of crucial importance for the algorithm's performance. The proof of the first one can be found in the full version of this work along with an illustration.

▷ Claim 3.11. If $p+1$ is an occurrence of P in T that aligns S with an occurrence of S in an S-run $R = T[r \,.\,.\, r']$, then $p + \mu \in \mathcal{M}_R$.

Proof sketch. Assume towards a contradiction that $p + \mu \notin \mathcal{M}_R$. Intuitively, this can only be the case if the extension of R did not reach the beginning of T due to encountering too many misperiods. On the other hand, the fragment $P[\mu \,.\,.\, x)S$ of P contains only one misperiod and cannot contain many wildcards, since every position of S is a sparsifier. As a result, a misperiod in T will be aligned with a position of P that is neither a misperiod nor a wildcard, contradicting the fact that $p+1$ is an occurrence of P in T. ◁

▷ Claim 3.12. The set $\cup_{R \in \mathcal{S}} \mathcal{M}_R$ is of size $\mathcal{O}(D)$ and it can be computed using $\mathcal{O}(D)$ `PILLAR` operations given the set $\mathcal{S}$ of S-runs and q.

Proof. We start by initialising a set $\mathcal{R} := \mathcal{S}$, marking every element of $\mathcal{R}$ as unprocessed and a set $\mathcal{M} = \emptyset$. We then iteratively perform the following procedure for the rightmost unprocessed $R = T[r \,.\,.\, r'] \in \mathcal{R}$. Compute $\mathcal{M}_R$ using Lemma 3.3, set $\mathcal{M} := \mathcal{M} \cup \mathcal{M}_R$, and mark R as processed. This takes time proportional to the sum of $|\mathcal{M}_R|$ and the number of groups of wildcards contained in E_R. Let us say that two elements $R = T[r \,.\,.\, r']$ and $R' = T[t \,.\,.\, t']$ of $\mathcal{S}$ are *synchronised* if and only if $r = t \pmod q$. During the procedure, whenever we compute some $E_R = T[x \,.\,.\, r']$ that extends beyond an (unprocessed) S-run $R' = T[t \,.\,.\, t']$, that is, $x \le t \le t' < r'$, and R and R' are synchronised, we remove R' from $\mathcal{R}$ – the total time required for this step is $\mathcal{O}(G)$.

[3] For a set Y and an integer z, by $z + Y$ we denote the set $\{z + y : y \in Y\}$.

We now show the correctness of the algorithm. If, while extending a run $R = T[r\mathinner{.\,.} r'] \in \mathcal{R}$, we extend beyond a run $R' = T[t\mathinner{.\,.} t'] \in \mathcal{R}$ with $r = t \pmod{q}$, observe that the left endpoint of E_R cannot be to the right of the left endpoint of $E_{R'}$, since we have at least as big a budget for misperiods in the extension of R when we reach position t as in the extension of R' when we reach position t. This implies that $\mathcal{M}_{R'} \subseteq \mathcal{M}_R$ and hence the algorithm correctly computes $\mathcal{M} = \cup_{R\in\mathcal{S}}\mathcal{M}_R$. Additionally, it guarantees that any computed E_R and $E_{R'}$ for synchronised S-runs R and R' are disjoint.

Finally, we analyse the algorithm's time complexity. Henceforth, $\mathcal{R}$ denotes set of runs that were processed. Observe that the run extensions take $\mathcal{O}(\sum_{R\in\mathcal{R}} |\mathcal{M}_R|)$ time in total in the `PILLAR` model. As we have $\sum_{R\in\mathcal{R}} |\mathcal{M}_R| \le |\mathcal{R}| + \sum_{R\in\mathcal{R}} |E_R| \cdot 20D/m \le \mathcal{O}(G) + 20D/m \cdot \sum_{R\in\mathcal{R}} |E_R|$, proving that $\sum_{R\in\mathcal{R}} |E_R| = \mathcal{O}(m)$ directly yields that $\mathcal{M} = \mathcal{O}(D)$ and that the algorithm takes $\mathcal{O}(D)$ time.

In what follows, we ignore all E_R that are of length at most m/D as their total length is $\mathcal{O}(G \cdot m/D) = \mathcal{O}(m)$. Let us partition T into a collection $\mathcal{Q} = \{T[1 + iq\mathinner{.\,.} (i+1)q] : i \in [0\mathinner{.\,.} \lfloor n/q \rfloor - 1]\}$ of consecutive fragments of length q, with the last one potentially being shorter and in this case discarded. We say that an element $T[i\mathinner{.\,.} j]$ of $\mathcal{Q}$ is *synchronised* with an element $R = T[r\mathinner{.\,.} r']$ of $\mathcal{R}$ if no position in $[i\mathinner{.\,.} j]$ is a misperiod with respect to $T[r\mathinner{.\,.} r']$. For a run $R = T[r\mathinner{.\,.} r']$, let $\mathcal{Q}_R = \{T[i\mathinner{.\,.} j] \in \mathcal{Q} : r \le i \le j \le r'\}$ consist of all elements of $\mathcal{Q}$ that are fully contained in E_R and observe that $|\mathcal{Q}_R| \ge |E_R|/q - 2 \ge |E_R|/(m/256D) - 2 = |E_R| \cdot 256D/m - 2 \ge |E_R| \cdot 252D/m + 2$. Further, let $\mathcal{Q}^s_R = \{X \in \mathcal{Q}_R : X \text{ is synchronised with } R\}$. As E_R contains at most $|E_R| \cdot 20D/m + 1$ misperiods with respect to $T[r\mathinner{.\,.} r']$, we have $|\mathcal{Q}^s_R| \ge |\mathcal{Q}_R|/2$. This means that $|E_R| = \mathcal{O}(|\mathcal{Q}^s_R| \cdot q)$. Now, observe that if some element of $\mathcal{Q}$ is synchronised with two elements R and R' of $\mathcal{R}$, then R and R' are themselves synchronised. Since the computed extensions of synchronised runs are pairwise disjoint, the considered sets $\mathcal{Q}^s_R$ are pairwise disjoint and hence the bound follows: $\sum_{R\in\mathcal{R}} |E_R| = \mathcal{O}(m) + \sum_{R\in\mathcal{R}, |E_R|\ge m/D} |E_R| = \mathcal{O}(m + |\mathcal{Q}| \cdot q) = \mathcal{O}(m)$. ◁

We can now conclude the proof of the theorem. By Claim 3.11, the starting positions of occurrences of P in T are in the set $\{\nu - \mu + 1 : \nu \in \mathcal{M}\}$. This concludes the proof of the combinatorial bound, as the size of this set is $\mathcal{O}(D)$. As for the time complexity, we verify each candidate position using Fact 2.4 in total time $\mathcal{O}(DG)$ in the `PILLAR` model. ◀

4 Technical Overview for Theorem 1.2

In this section, we outline the main ideas for extending the results of Section 3 towards obtaining Theorem 1.2. We build upon the framework of Charalampopoulos et al. [11] for k-mismatch pattern matching on solid strings. They presented an efficient structural analysis of a solid pattern that returns a number of *breaks* or a number of *repetitive regions*, or concludes that the pattern is *almost periodic*. Then, they treated each case separately, exploiting the computed structure. We make several alterations to account for wildcards, such as ensuring that breaks are solid strings and adapting the sliding window approach. The primary technical challenge in achieving an efficient solution lies in limiting the number of occurrences of repetitive regions in T. The greater the number of repetitive region occurrences, the higher the number of potential starting positions for k-mismatch occurrences of P. By ensuring that each repetitive region contains a sparsifier, we force an upper bound on the number of wildcards in each repetitive region, which, in turn, allows us to bound the number of its approximate occurrences in T.

Computing structure in the pattern. We first show a *decomposition lemma*, that either extracts useful structure from the pattern or reveals that it is close to a periodic string. Our lemma is analogous to the decomposition lemma for the case where both strings are solid [11, Lemma 3.6]. We take extra care to ensure that the breaks in Case (I) are solid strings and that each computed repetitive region in Case (II) contains a sparsifier.

▶ **Lemma 4.1.** *Let P be a string of length m that contains $D \leq m/16$ wildcards arranged in G groups. Further, let $k \in [1\,..\,m]$ be an integer threshold, and let $\gamma := G + k$ and $\tau := D + k$. At least one of the following holds:*

(I) *P contains 2γ disjoint solid strings $B_1, \ldots, B_{2\gamma}$, that we call* breaks*, each having length $m/(16\gamma)$ and the period greater than $m/(512\tau)$.*

(II) *P contains disjoint* repetitive regions *$R_1, \ldots, R_r$ of total length $m_R \geq m/8$, such that, for every i: R_i contains a sparsifier, $|R_i| \geq m/16\gamma$, and, for a primitive string Q_i with $|Q_i| \leq m/(512\tau)$, we have $\delta_H(R_i, Q_i^\infty) = \lceil 32k/m \cdot |R_i| \rceil$.*

(III) *There exists a primitive string Q of length at most $m/(512\tau)$ such that $\delta_H(P, Q^\infty) \leq 32k$.*

Moreover, there is an algorithm that takes $\mathcal{O}(G + k)$ time in the `PILLAR` *model and distinguishes between the above cases, returning one of the following: either 2γ disjoint breaks, or repetitive regions $R_1, \ldots, R_r$ of total length at least $m/8$ along with primitive strings $Q_1, \ldots, Q_r$, or a primitive string Q along with $\mathsf{MI}(P, Q)$.*

Let us now outline how the three different cases are dealt with in the proof of Theorem 1.2.

Case (III). The almost periodic case is treated quite similarly to Case (II.a) of the exact pattern matching algorithm in Section 3. Specifically, we take a sliding window approach where we have an event for each misperiod/wildcard in P being aligned with a misperiod in T and an event for each misperiod in T entering or exiting the window. We have $\mathcal{O}((D+k)(G+k))$ events and hence obtain an algorithm running in time $\mathcal{O}((D + k)(G + k) \log\log(D + k))$ in the `PILLAR` model.

Case (I). In the case when the algorithm underlying Lemma 4.1 returns 2γ breaks, we perform a standard marking trick. As each break is long and has period greater than $m/(512\tau)$, it has $\mathcal{O}(\tau)$ exact occurrences in T, which we can compute efficiently using IPM queries. For each break $P[p_i\,..\,p_i + m/(16\gamma))$, for each occurrence of it at a position j in T, we place a mark at the corresponding position where an occurrence of P would start, that is, at position $j - p_i + 1$. Over all breaks, we place $\mathcal{O}(\tau\gamma)$ marks in total. Now, observe that in any k-mismatch occurrence of P in T, at most k breaks are not matched exactly and hence at least $2\gamma - k$ breaks are matched exactly. It is sufficient to verify each position that has at least $2\gamma - k$ marks; their number is $\mathcal{O}(\tau\gamma/(2\gamma - k)) = \mathcal{O}(D + k)$. As the verification of each position takes $\mathcal{O}(G + k)$ time in the `PILLAR` model due to Fact 2.4, the total time required in this case is $\mathcal{O}((D + k)(G + k))$.

Case (II). The case when the algorithm underlying Lemma 4.1 returns repetitive regions is the most challenging one. We tackle it using a more elaborate marking scheme. For a repetitive region R_i, let D_i denote the number of wildcards in R_i, let d_i denote $\lfloor 32(k + D)/m \cdot |R_i| \rfloor$, and let k_i denote $\lfloor 16k/m \cdot |R_i| \rfloor$. First, for each repetitive region R_i, we compute R_i's k_i-mismatch occurrences using our algorithm for Case (III); this takes $\tilde{\mathcal{O}}((D + k)(G + k))$ time in the `PILLAR` model. Then, for each k_i-mismatch occurrence of R_i, we put $|R_i|$ marks at the corresponding starting position for P. Importantly, we show that the number of

k_i-mismatch occurrences of R_i is $\mathcal{O}(D+k)$: this crucially relies on the fact that the repetitive region contains a sparsifier and hence $D_i \leq 16D/m \cdot |R_i|$. We can thus upper-bound the total number of placed marks by $\mathcal{O}((D+k) \cdot m_R)$. On the other hand, we show that each starting position of a k-mismatch occurrence of P in T must have at least $m_R - m/16$ marks. As in Case (I), we obtain $\mathcal{O}(D+k)$ candidate positions, which we verify in $\mathcal{O}((D+k)(G+k))$ time in total in the `PILLAR` model.

Concluding, the algorithm runs in $\tilde{\mathcal{O}}((D+k)(G+k))$ time in the `PILLAR` model. In Cases (I) and (II), we have shown that P has $\mathcal{O}(D+k)$ k-mismatch occurrences in T, which can be trivially paritioned into the same number of arithmetic progressions. Therefore, the bottleneck to obtain the refined result on the combinatorial structure of occurences is the sliding window scheme of Case (III). We consider events associated to misperiods and groups of wildcards of P separately: in the former case, they are $\mathcal{O}((D+k)k)$ and lead to "isolated" occurrences, whereas in the latter case, they are $\mathcal{O}((D+k)G)$ and result in arithmetic progressions of occurrences.

5 A Lower Bound on the Number of Arithmetic Progressions

In this section we show a lower bound on the number of arithmetic progressions covering the set of k-mismatch occurrences of a pattern in a text.

▶ **Theorem 5.1.** *There exist a pattern P of length $m = \Omega((D+k)^{1+o(1)}(k+1))$ and a text T of length $n \leq 3m/2$ such that the set of k-mismatch occurrences of P in T cannot be covered with less than $\Omega((D+k)\cdot(k+1))$ arithmetic progressions.*

Proof. We call a set $S \subseteq [1\,..\,n]$ progression-free if it contains no non-trivial arithmetic progression, that is, three distinct integers a, b, c such that $a+b-2c=0$.

▶ **Fact 5.2** ([20]). *For any sufficiently large M, there exists an integer $n_M = \mathcal{O}(M2^{\sqrt{\log M}})$ and a progression-free set S such that S has cardinality M and $S \subseteq [n_M]$.*

Let $M = D + k/2$ and $S \subseteq [n_M]$ be a progression-free set of cardinality M. We encode S in a string P_S of size n_M as follows: for every $i \notin S$ we set $P_S[i] = 0$, and we arbitrarily assign $k/2$ 1s and D wildcards to the remaining $D + k/2$ positions. We then consider the pattern $P = 0^\ell P_S 0^\ell$, where ℓ is a parameter to be determined later. In what follows, let $m := 2\ell + n_M$ denote the length of P.

Now, let $M' = k/2+1$ and $S' \subseteq [n_{M'}]$ be a progression-free set of cardinality M'. We set $T := 0^{m/2}B_1 \dots B_{n_{M'}}0^{m/2}$, where $B_i = 0^{t-1}1$ if $i \in S'$, $B_i = 0^t$ otherwise, and $t = \lfloor m/(2n_{M'})\rfloor$. We pick ℓ large enough such that $t \geq 10n_M$ and $2n_{M'}$ divides m. We have $t = \lfloor m/(2n_{M'})\rfloor = \lfloor (2\ell + n_M)/(2n_{M'})\rfloor \geq 10n_M$, which implies that it suffices for ℓ to be larger than $n_M n_{M'}$ by a constant factor, i.e., $m = \Omega((k+D)(k+1)2^{\sqrt{\log(k+D)}+\sqrt{\log(k+1)}})$. Observe that $i \in X$ if and only if there exists j such that $P[j] \in \{1, \diamond\}$ and $T[i+j] = 1$. Moreover, any pair of 1s in T are at least $t \geq 10n_M$ positions apart, while the 1s and wildcards of P all lie within an interval of size n_M. Therefore, for a given alignment of P and T, there can be at most one 1 of T that is aligned with a 1 or a wildcard of P; it follows that $\mathrm{Occ}_k(P,T)$ has cardinality $(D+k/2)\cdot(k/2+1) = \Omega((D+k)\cdot(k+1))$.

It remains to show that $\mathrm{Occ}_k(P,T)$ does not contain arithmetic progressions of length 3. Assume for a sake of contradiction that there exist $x, y, z \in \mathrm{Occ}_k(P,T)$ with $x < y < z$ that form an arithmetic progression, i.e., $y - x = z - y$. Let i_x denote the index of the block B_{i_x} of T that contains the leftmost 1 that is aligned with a 1 or a wildcard of P: this 1 is at position $m/2 + i_x t$ in T. Similarly, let d_x be such that the corresponding aligned character is at position $\ell + d_x$ in P. Define i_y, i_z, d_y, d_z similarly for y, z. We can express each $w \in \{x, y, z\}$ in terms of i_w and d_w as $w = m/2 + i_w t - d_w - \ell + 1$. Combining the above equations for x, y, and z, we get $y - x = (i_y - i_x)t - (d_y - d_x)$ and $z - y = (i_z - i_y)t - (d_z - d_y)$. By construction, $|d_y - d_x| \le n_M$. As $t \ge 10n_M$, the equality $y - x = z - y$ thus yields $i_y - i_x = i_z - i_y$ and $d_y - d_x = d_z - d_y$. However, as $x < y < z$, at least one of the above two equations involves non-zero values. In other words, there is a three-term arithmetic progression in either S or S', contradicting the fact that they are progression-free. ◀

References

1 Karl R. Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16(6):1039–1051, 1987.

2 Tatsuya Akutsu. Approximate string matching with don't care characters. *Inf. Process. Lett.*, 55(5):235–239, 1995. doi:10.1016/0020-0190(95)00111-O.

3 Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004. doi:10.1016/S0196-6774(03)00097-X.

4 Gabriel Bathie, Panagiotis Charalampopoulos, and Tatiana Starikovskaya. Longest common extensions with wildcards: Trade-off and applications. In *ESA*, 2024. to appear.

5 Felix A. Behrend. On sets of integers which contain no three terms in arithmetical progression. *Proceedings of the National Academy of Sciences*, 32(12):331–332, 1946.

6 Francine Blanchet-Sadri and Justin Lazarow. Suffix trees for partial words and the longest common compatible prefix problem. In *LATA*, volume 7810, pages 165–176. Springer, 2013. doi:10.1007/978-3-642-37064-9_16.

7 Karl Bringmann, Philip Wellnitz, and Marvin Künnemann. Few matches or almost periodicity: Faster pattern matching with mismatches in compressed texts. In *SODA*, pages 1126–1145, 2019. doi:10.1137/1.9781611975482.69.

8 Timothy M. Chan, Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. Approximating text-to-pattern hamming distances. In *STOC*, pages 643–656, 2020. doi:10.1145/3357713.3384266.

9 Timothy M. Chan, Ce Jin, Virginia Vassilevska Williams, and Yinzhan Xu. Faster algorithms for text-to-pattern hamming distances. In *FOCS*, pages 2188–2203, 2023. doi:10.1109/FOCS57990.2023.00136.

10 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, and Wiktor Zuba. Circular pattern matching with k mismatches. *J. Comput. Syst. Sci.*, 115:73–85, 2021. doi:10.1016/j.jcss.2020.07.003.

11 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *FOCS*, pages 978–989, 2020. doi:10.1109/FOCS46700.2020.00095.

12 Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Inf. Process. Lett.*, 101(2):53–54, 2007. doi:10.1016/j.ipl.2006.08.002.

13 Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. From coding theory to efficient pattern matching. In *SODA*, pages 778–784, 2009.

14 Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. Pattern matching with don't cares and few errors. *J. Comput. Syst. Sci.*, 76(2):115–124, 2010. doi:10.1016/j.jcss.2009.06.002.

15 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k-mismatch problem revisited. In *SODA*, pages 2039–2052, 2016. doi:10.1137/1.9781611974331.CH142.

16 Raphaël Clifford and Ely Porat. A filtering algorithm for k-mismatch with don't cares. *Inf. Process. Lett.*, 110(22):1021–1025, 2010. doi:10.1016/j.ipl.2010.08.012.

17 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *STOC*, pages 592–601, 2002. doi:10.1145/509907.509992.

18 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007. doi:10.1017/cbo9780511546853.

19 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Jakub Radoszewski, Wojciech Rytter, Bartosz Szreder, and Tomasz Walen. A note on the longest common compatible prefix problem for partial words. *J. Discrete Algorithms*, 34:49–53, 2015. doi:10.1016/J.JDA.2015.05.003.

20 Michael Elkin. An improved construction of progression-free sets. In *SODA*, pages 886–905, 2010. doi:10.1137/1.9781611973075.72.

21 Paul Erdös and Paul Turán. On some sequences of integers. *Journal of the London Mathematical Society*, s1-11(4):261–264, 1936. doi:10.1112/jlms/s1-11.4.261.

22 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.2307/2034009.

23 Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łącki, and Piotr Sankowski. Optimal dynamic strings. In *SODA*, pages 1509–1528, 2018. doi:10.1137/1.9781611975031.99.

24 Pawel Gawrychowski and Przemyslaw Uznański. Towards unified approximate pattern matching for Hamming and L_1 distance. In *ICALP*, pages 62:1–62:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.62.

25 Yijie Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. *J. Algorithms*, 50(1):96–105, 2004. doi:10.1016/j.jalgor.2003.09.001.

26 Piotr Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *FOCS*, pages 166–173, 1998. doi:10.1109/SFCS.1998.743440.

27 Ce Jin and Jakob Nogler. Quantum speed-ups for string synchronizing sets, longest common substring, and k-mismatch matching. In *SODA*, pages 5090–5121, 2023. doi:10.1137/1.9781611977554.ch186.

28 Ce Jin and Yinzhan Xu. Shaving logs via large sieve inequality: Faster algorithms for sparse convolution and more. In *STOC*, pages 1573–1584, 2024. doi:10.1145/3618260.3649605.

29 Adam Kalai. Efficient pattern-matching with don't cares. In *SODA*, pages 655–656, 2002.

30 Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

31 Tomasz Kociumaka, Ely Porat, and Tatiana Starikovskaya. Small-space and streaming pattern matching with k edits. In *FOCS*, pages 885–896, 2021. doi:10.1109/FOCS52979.2021.00090.

32 S. Rao Kosaraju. Efficient string matching. Unpublished manuscript, 1987.

33 Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *J. Algorithms*, 10(2):157–169, 1989. doi:10.1016/0196-6774(89)90010-2.

34 Udi Manber and Ricardo A. Baeza-Yates. An algorithm for string matching with a sequence of don't cares. *Inf. Process. Lett.*, 37(3):133–136, 1991. doi:10.1016/0020-0190(91)90032-D.

35 Gerhard Mehldau and Gene Myers. A system for pattern matching applications on biosequences. *Bioinformatics*, 9(3):299–314, 1993. doi:10.1093/bioinformatics/9.3.299.

36 Michael S Paterson Michael J Fischer. String-matching and other products. In *SCC*, pages 113–125, 1974.

37 Marius Nicolae and Sanguthevar Rajasekaran. On string matching with mismatches. *Algorithms*, 8(2):248–270, 2015. doi:10.3390/a8020248.

38 Marius Nicolae and Sanguthevar Rajasekaran. On pattern matching with k mismatches and few don't cares. *Inf. Process. Lett.*, 118:78–82, 2017. doi:10.1016/j.ipl.2016.10.003.