Generalised Pattern Matching Revisited

Bartlomiej Dudek, Pawel Gawrychowski, Tatiana Starikovskaya

Generalised Pattern Matching Revisited

Bartłomiej Dudek
Institute of Computer Science, University of Wrocław, Poland
bartlomiej.dudek@cs.uni.wroc.pl

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Tatiana Starikovskaya
DIENS, École normale supérieure, PSL Research University, Paris, France
tat.starikovskaya@gmail.com

Abstract

In the problem of Generalised Pattern Matching (GPM) [STOC'94, Muthukrishnan and Palem], we are given a text T of length n over an alphabet Σ_T, a pattern P of length m over an alphabet Σ_P, and a matching relationship $\subseteq \Sigma_T \times \Sigma_P$, and must return all substrings of T that match P (*reporting*) or the number of mismatches between each substring of T of length m and P (*counting*). In this work, we improve over all previously known algorithms for this problem:

- For $\mathcal{D}$ being the maximum number of characters that match a fixed character, we show two new Monte Carlo algorithms, a reporting algorithm with time $\mathcal{O}(\mathcal{D}\, n \log n \log m)$ and a $(1-\varepsilon)$-approximation counting algorithm with time $\mathcal{O}(\varepsilon^{-1}\mathcal{D}\, n \log n \log m)$. We then derive a $(1-\varepsilon)$-approximation deterministic counting algorithm for GPM with $\mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n \log^6 n)$ time.
- For $\mathcal{S}$ being the number of pairs of matching characters, we demonstrate Monte Carlo algorithms for reporting and $(1-\varepsilon)$-approximate counting with running time $\mathcal{O}(\sqrt{\mathcal{S}}\, n \log m \sqrt{\log n})$ and $\mathcal{O}(\sqrt{\varepsilon^{-1}\mathcal{S}}\, n \log m \sqrt{\log n})$, respectively, as well as a $(1-\varepsilon)$-approximation deterministic algorithm for the counting variant of GPM with $\mathcal{O}(\varepsilon^{-1}\sqrt{\mathcal{S}} n \log^{7/2} n)$ time.
- Finally, for $\mathcal{I}$ being the total number of disjoint intervals of characters that match the m characters of the pattern P, we show that both the reporting and the counting variants of GPM can be solved exactly and deterministically in $\mathcal{O}(n\sqrt{\mathcal{I} \log m} + n \log n)$ time.

At the heart of our new deterministic upper bounds for $\mathcal{D}$ and $\mathcal{S}$ lies a faster construction of superimposed codes, which solves an open problem posed in [FOCS'97, Indyk] and can be of independent interest.

To conclude, we demonstrate first lower bounds for GPM. We start by showing that any deterministic or Monte Carlo algorithm for GPM must use $\Omega(\mathcal{S})$ time, and then proceed to show higher lower bounds for combinatorial algorithms. These bounds show that our algorithms are almost optimal, unless a radically new approach is developed.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases pattern matching, superimposed codes, conditional lower bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.18

Related Version A full version of the paper is available at `https://arxiv.org/abs/2001.05976`.

Funding *Bartłomiej Dudek*: partially supported by the National Science Centre, Poland, grant number 2017/27/N/ST6/02719.

1 Introduction

Processing noisy data is a keystone of modern string processing. One possible approach to address this challenge is approximate pattern matching, where the task is to find all substrings of the text that are close to the pattern under some similarity measure, such as

Hamming or edit distance. The approximate pattern matching approach assumes that noise is arbitrary, i.e. that we can delete or replace any character of the pattern or of the text by any other character of the alphabet.

The assumption that the noise is completely arbitrary is not necessarily justified, as in practice we might have some predetermined knowledge about the structure of the errors. In this paper we focus on the Generalised Pattern Matching (GPM) problem that addresses this setting. We assume to be given a text T over an alphabet Σ_T, a pattern P over an alphabet Σ_P, and we allow each character of Σ_T to match a subset of characters of Σ_P. We must report all substrings of the text that match the pattern. This problem was introduced in STOC'94 [35] by Muthukrishnan and Palem to provide a unified approach for solving different extensions of the classical pattern matching question that has been considered as separate problems in the early 90s. Later, Muthukrishnan [34] considered a *counting variant* of GPM, where the task is to count the number of mismatches between substrings of the text and the pattern. Formally, the problem is defined as follows:

Generalised Pattern Matching (GPM)
Input: A text $T \in (\Sigma_T)^n$, a pattern $P \in (\Sigma_P)^m$, and a matching relationship $\subseteq \Sigma_T \times \Sigma_P$.
Output (Reporting): All $i \in [n-m+1]$ such that $T[i, i+m-1]$ matches P.
Output (Counting): For each $i \in [n-m+1]$, the number of positions $j \in [m]$ such that $T[i+j-1]$ does not match $P[j]$.

Muthukrishnan and Palem [35] and subsequent work [34, 36] considered three natural parameters describing the matching relationship ($\mathcal{D}, \mathcal{S}$) or the pattern ($\mathcal{I}$). Viewing the matching relationship as a bipartite graph with edges connecting pairs of matching characters from $\Sigma_T \times \Sigma_P$, $\mathcal{D}$ is the maximum degree of a node and $\mathcal{S}$ is the total number of edges in the graph. Next, the parameter $\mathcal{I}$ describes the pattern rather than the matching relationship. For each character $a \in \Sigma_P$, let $I(a)$ be the minimal set of disjoint sorted intervals that contain the characters that match a, and define $\mathcal{I} = \sum_{j \in [m]} |I(P[j])|$.

The maximum number of characters that match a fixed character, $\mathcal{D}$. For the reporting variant of GPM, Muthukrishnan [34] showed a Las Vegas algorithm with running time $\mathcal{O}(\mathcal{D}\, n \log n \log m)$. Indyk [27] used superimposed codes to show a deterministic algorithm with running time $\mathcal{O}(|\Sigma_P|\mathcal{D}^2 \log^2 n + \mathcal{D}\, n \log^3 n \log m)$. For the counting variant, Muthukrishnan [34] showed a $(\log m)$-approximation Las Vegas algorithm with time $\mathcal{O}(\mathcal{D}\, n \log n \log m)$. Indyk [27] gave a $(1-\varepsilon)$-approximation deterministic and Monte Carlo algorithm with running time $\mathcal{O}(\varepsilon^{-2}\mathcal{D}^2 n \log^3 n)$ and $\mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n \log^3 n)$, respectively.

The number of matching pairs of characters, $\mathcal{S}$. Muthukrishnan and Ramesh [36] gave an $\mathcal{O}((\mathcal{S}\, m \log^2 m)^{1/3} n)$-time algorithm for the reporting variant of GPM.

The number of intervals of matching characters, $\mathcal{I}$. For this parameter, Muthukrishnan [34] gave an $\mathcal{O}(\mathcal{I} + (m\mathcal{I})^{1/3} n\sqrt{\log m})$-time algorithm[1].

1.1 Our Contribution

We improve existing randomised and deterministic upper bounds for GPM, and demonstrate matching lower bounds. At heart of our deterministic algorithms for the counting variant of GPM is a solution to an open problem of Indyk [27] on construction of superimposed codes.

[1] [34, Theorem 9] claims $\mathcal{O}(n + \mathcal{I} + \mathcal{I}^{1/3}(nm)^{2/3}\sqrt{\log m})$, but the first sentence of the proof states that for $n \le 2m$ the algorithm takes $\mathcal{O}(\mathcal{I} + \mathcal{I}^{1/3} m^{4/3}\sqrt{\log m})$ time, where the first term is the time that we need to read the input. For a longer text, one needs to apply it n/m times for overlapping blocks of length $2m$, making the total time $\mathcal{O}(\mathcal{I} + n/m \cdot \mathcal{I}^{1/3} m^{4/3}\sqrt{\log m}) = \mathcal{O}(\mathcal{I} + (m\mathcal{I})^{1/3} n\sqrt{\log m})$.

Data-dependent superimposed codes. A z-superimposed code is a set of binary vectors such that no vector is contained in a Boolean sum (i.e. bitwise OR) of z other vectors. Superimposed codes find their main application in information retrieval (e.g. in compressed representation of document attributes), and optimizing broadcasting on radio networks [30], and have also proved to be useful in graph algorithms [1, 25]. Indyk [27] extended the notion of superimposed codes to the so-called *data-dependent superimposed codes*, and asked for a deterministic construction for such codes with a certain additional property that makes them useful for counting mismatches (see Section 2 for a formal definition). We provide such a construction algorithm in Theorem 10. We briefly describe the high-level idea below.

We need the concept of discrepancy minimization. Given a universe U, each of its elements is assigned one of two colours, red or blue. The *discrepancy* of a subset of U is defined as the difference between the number of red and blue elements in it, and the discrepancy of a family $\mathcal{F}$ of subsets is defined as the maximum of the absolute values of discrepancies of the subsets in $\mathcal{F}$. Discrepancy minimization is a fundamental notion with numerous applications, including derandomization, computational geometry, numerical integration, understanding the limits of models of computation, and so on (see e.g. [13]). A recent line of work showed a series of algorithms for constructing colourings of low discrepancy in various settings [5–10, 32, 33]. For our applications, we need to work under the assumption that the size of each subset in $\mathcal{F}$ is bounded by a given parameter k. In Theorem 7, we describe a fast deterministic algorithm that returns a colouring of small discrepancy for this case. We follow the algorithm described by Chazelle [13] that can be roughly summarized as based on the method of conditional expectations tweaked as to allow for an efficient implementation. In more detail, Chazelle's construction assumes infinite precision of computation and does not immediately translate into an efficient algorithm working in the Word RAM model of computation, thus requiring resolving some technical issues to bound the required precision and the overall complexity.

We apply discrepancy minimization to design in Lemma 9 a procedure that, given a family $\mathcal{F}$ of subsets of U, partitions the universe U into not too many parts such that the intersection of each part and each of the subsets in $\mathcal{F}$ is small. The procedure follows the natural idea of colouring the universe with two colours, and then recursing on the elements with the same colour. Every step of such construction introduces some penalty that needs to be carefully controlled as to guarantee the desired property in the end. Because of this penalty, we are only able to guarantee that the intersections are small, but not constant. To finish the construction, we combine the partition with a hash function into the ring of polynomials. We stress that this part of the construction is new and not simply a modification of Chazelle's (or Indyk's) method.

Upper bounds for GPM. Similar to previous work, we assume that the alphabets' sizes are polynomial in n and that the matching relationship is given as a graph M on the set of vertices $\Sigma_T \cup \Sigma_P$. We also assume to have access to three oracles that can answer the following questions in $\mathcal{O}(1)$ time:

1. Is there an edge between $a \in \Sigma_T$ and $b \in \Sigma_P$ (in other words, do a and b match)?
2. What is the degree of a character $a \in \Sigma_T$ or $b \in \Sigma_P$ (in other words, what is the number of characters that match a given character)?
3. What is the k-th neighbor of $a \in \Sigma_T$ (in other words, what is the k-th character $b \in \Sigma_P$ matching a)? We assume an arbitrary (but fixed) order of neighbors of every node.

Under these assumptions, we show the following upper bounds summarized in Tables 1 and 2:

Table 1 Generalised Pattern Matching (reporting).

Time	Det./Rand.	
$\mathcal{O}(\lvert\Sigma_P\rvert\mathcal{D}^2\log^2 n + \mathcal{D}\, n\log^3 n\log m)$	Det.	[27]
$\mathcal{O}(\mathcal{D}\, n\log^6 n)$	Det.	This work
$\mathcal{O}(\mathcal{D}\, n\log n\log m)$	Rand.	[34]
$\mathcal{O}(\mathcal{D}\, n\log n\log m)$	Rand.	This work
$\mathcal{O}((\mathcal{S}\, m\log^2 m)^{1/3}n)$	Det.	[36]
$\mathcal{O}(\sqrt{\mathcal{S}}\, n\log^{7/2} n)$	Det.	This work
$\mathcal{O}(\sqrt{\mathcal{S}}\, n\log m\sqrt{\log n})$	Rand.	This work
$\mathcal{O}(\mathcal{I} + (m\mathcal{I})^{1/3}n\sqrt{\log m})$	Det.	[34]
$\mathcal{O}(n\sqrt{\mathcal{I}\log m} + n\log n)$	Det.	This work

Table 2 Generalised Pattern Matching (counting).

Time	Det./Rand.	Approx. factor	
$\mathcal{O}(\varepsilon^{-2}\mathcal{D}^2 n\log^3 n)$	Det.	$(1-\varepsilon)$	[27]
$\mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n\log^6 n)$	Det.	$(1-\varepsilon)$	This work
$\mathcal{O}(\mathcal{D}\, n\log n\log m)$	Rand.	$\log m$	[34]
$\mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n\log^3 n)$	Rand.	$(1-\varepsilon)$	[27]
$\mathcal{O}(\varepsilon^{-1}\mathcal{D}\, n\log n\log m)$	Rand.	$(1-\varepsilon)$	This work
$\mathcal{O}(\varepsilon^{-1}\sqrt{\mathcal{S}}\, n\log^{7/2} n)$	Det.	$(1-\varepsilon)$	This work
$\mathcal{O}(\sqrt{\varepsilon^{-1}\mathcal{S}}\, n\log m\sqrt{\log n})$	Rand.	$(1-\varepsilon)$	This work
$\mathcal{O}(\mathcal{I} + (m\mathcal{I})^{1/3}n\sqrt{\log m})$	Det.	–	[34]
$\mathcal{O}(n\sqrt{\mathcal{I}\log m} + n\log n)$	Det.	–	This work

1. We start by showing a new Monte Carlo algorithm for the parameter $\mathcal{D}$ with running time $\mathcal{O}(\mathcal{D}\, n\log m\log n)$ (Theorem 11). While its running time is the same as that of [34], it encapsulates a novel approach to the problem that serves as a basis for other algorithms. We then derive a Monte Carlo algorithm for the parameter $\mathcal{S}$ with running time $\mathcal{O}(\sqrt{\mathcal{S}}\, n\log m\sqrt{\log n})$ (Theorem 12). As a corollary, we show a $(1-\varepsilon)$-approximation Monte Carlo algorithm that solves the counting variant of GPM in time $\mathcal{O}(\min\{\varepsilon^{-1}\mathcal{D}\log n, \sqrt{\varepsilon^{-1}\mathcal{S}\log n}\}\cdot n\log m)$ (Corollary 13). All three algorithms have inverse-polynomial error probability.
2. Next, using the data-dependent superimposed codes, we construct $(1-\varepsilon)$-approximation deterministic algorithms for the counting variant of GPM. The first algorithm requires $\mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n\log^6 n)$ time (Theorem 14), and the second algorithm $\mathcal{O}(\varepsilon^{-1}\sqrt{\mathcal{S}}\, n\log^{7/2} n)$ time (Theorem 15). By taking $\varepsilon = 1/2$, we immediately obtain deterministic algorithms for the reporting variant of the problem with the same complexities.
3. Finally, we show that both the reporting and the counting variants of GPM can be solved exactly and deterministically in $\mathcal{O}(n\sqrt{\mathcal{I}\log m} + n\log n)$ time (Theorem 17).

Lower bounds for GPM. We also show first lower bounds for GPM (see Section 4). We start with a simple adversary-based argument that shows that any deterministic algorithm or any Monte Carlo algorithm with constant error probability that solves GPM must use $\Omega(\mathcal{S})$ time (Lemma 19 and 20). We then proceed to show higher lower bounds for combinatorial

algorithms by reduction from Boolean matrix multiplication[2] parameterized by $\mathcal{D}, \mathcal{S}, \mathcal{I}$ (Lemma 21 and Corollary 23). All the lower bounds are presented for the reporting variant of GPM, so they immediately apply also to the counting variant. These bounds show that our algorithms are almost optimal, unless a radically new approach is developed.

1.2 Related Work

Degenerate string matching. A more general approach to dealing with noise in string data is degenerate string matching, where the set of matching characters is specified for every position of the text or of the pattern (as opposed to every character of the alphabets). Abrahamson [3] showed the first efficient algorithm for a degenerate pattern and a standard text. Later, several practically efficient algorithms were shown [26, 37].

Pattern matching with don't cares. In this problem, we assume $\Sigma_T = \Sigma_P = \Sigma$, where Σ contains a special character – "don't care". We assume that two characters of Σ match if either one of them is the don't care character, or they are equal. The study of this problem commenced in [21], where a $\mathcal{O}(n \log m \log |\Sigma|)$-time algorithm was presented. The time complexity of the algorithm was improved in subsequent work [18, 28, 29], culminating in an elegant $\mathcal{O}(n \log m)$-time deterministic algorithm of Clifford and Clifford [15]. Clifford and Porat [17] also considered the problem of identifying all alignments where the number of mismatching characters is at most k.

Threshold pattern matching. In the threshold pattern matching problem, we are given a parameter δ, and we say that two characters a, b match if $|a - b| < \delta$. The threshold pattern matching problem has been studied both in reporting and counting variants [4, 11, 12, 16, 19, 20, 22, 38]. The best algorithm for the reporting variant of the threshold pattern matching problem is deterministic and takes linear time (after the pattern has been preprocessed). The best deterministic algorithm for the counting variant of threshold pattern matching has time $\mathcal{O}((\log \delta + 1)n\sqrt{m \log m})$, while the best randomised algorithm has time $\mathcal{O}((\log \delta + 1)n \log m)$ [38].

In threshold pattern matching the matching relationship is described with a single interval per character, so $\mathcal{I} = m$. Hence from Theorem 17 immediately follows a faster deterministic algorithm for the counting variant of the threshold pattern matching problem (Corollary 18).

2 Data-Dependent Superimposed Codes

We start by solving the open problem posed by Indyk [27]: provide a deterministic algorithm for construction of a variant of data-dependent superimposed codes that is particularly suitable for the counting variant of GPM. The algorithm that we present is rather involved, a reader more interested in pattern matching applications can skip this section on the first reading.

▶ **Definition 1.** *Let $S_1, \ldots, S_z$ be subsets of a universe U. A family of sets $\mathcal{C} = \{C_1, \ldots, C_{|U|}\}$, where $C_u \subseteq [\ell]$ and $|C_u| = w$ for $u \in U$ is called an $(\{S_i\}, \tau)$-superimposed code if for every S_i and $u \notin S_i$ we have $|C_u - \bigcup_{v \in S_i} C_v| \geq \tau$. We call ℓ and w respectively the length and the weight of the code $\mathcal{C}$.*

[2] It is not clear what combinatorial means precisely. However, FFT and Boolean convolution often used in algorithms on strings are considered not to be combinatorial.

Suppose that the size of each S_i is at most k, where k is some fixed integer. Indyk asked if there exists a deterministic $\tilde{\mathcal{O}}((zk)/\varepsilon^{\mathcal{O}(1)})$-time algorithm that computes an $(\{S_i\}, (1-\varepsilon)w)$-superimposed code of some weight w and length $\ell = \mathcal{O}(k \operatorname{polylog}(zk))$. It can be seen that we cannot hope to construct such a code with ℓ independent of ε. In the following lemma we show that even if we restrict to the case of $k = 1$ we still need that ℓ significantly depends on ε.

▶ **Lemma 2.** *For every constant $\delta \in (0,1)$, function $f(z) = \mathcal{O}(\operatorname{polylog} z)$, and z large enough, there exists a family of singleton sets $S_1, S_2, \ldots, S_z$ and $0 < \varepsilon < 1$ such that any $(\{S_i\}, (1-\varepsilon)w)$-superimposed code of weight w must have length length $\ell > f(z)/\varepsilon^{\delta}$.*

Proof. Consider sets $S_i = \{i\}$ for $i \in [z]$, where z will be determined later. Let $\varepsilon = 1/(2f(z))^{\frac{1}{1-\delta}}$ and suppose that there is a $(\{S_i\}, (1-\varepsilon)w)$-superimposed code $\mathcal{C}$. Then, by definition of superimposed codes and from $w \leq \ell$, for $i \neq j$ it holds

$$\begin{cases} |C_i - C_j| \geq (1-\varepsilon)w = w - \varepsilon w \geq w - \varepsilon\ell, \\ \varepsilon\ell \leq \varepsilon f(z)/\varepsilon^{\delta} = \varepsilon^{1-\delta} f(z) = 1/2 \end{cases}$$

so $|C_i - C_j| > w - 1$. Hence, $|C_i - C_j| = w$ and every C_i and C_j must be disjoint, and therefore $\ell \geq zw \geq z$. Assume towards a contradiction that $\ell \leq f(z)/\varepsilon^{\delta}$. We obtain

$$\ell \leq f(z)/\varepsilon^{\delta} = f(z) \cdot (2f(z))^{\frac{\delta}{1-\delta}} = f(z)^{\frac{1}{1-\delta}} \cdot 2^{\frac{\delta}{1-\delta}} = \mathcal{O}(\operatorname{polylog} z) \cdot 2^{\frac{\delta}{1-\delta}} < z$$

where the last inequality holds for sufficiently large z. This leads to a contradiction and the claim follows. ◀

Therefore, one should allow $\ell = \mathcal{O}(k \operatorname{polylog}(zk)/\varepsilon^{\mathcal{O}(1)})$. We give a positive answer to this natural relaxation. We start by showing an efficient deterministic algorithm for discrepancy minimization that will play an essential role in our approach.

2.1 Discrepancy Minimization

Let us start with a formal definition of discrepancy.

▶ **Definition 3** (Discrepancy). *Consider a family $\mathcal{F}$ of z sets $S_i \subseteq U$, $i \in [z]$. We call a function $\chi : U \to \{-1, +1\}$* a colouring. *The* discrepancy *of a set S_i is defined as $\chi(S_i) = \sum_{u \in S_i} \chi(u)$, and the* discrepancy *of $\mathcal{F}$ is defined as $\max_{i \in [z]} |\chi(S_i)|$.*

In [13, Section 1.1], Chazelle presented a construction of a colouring of small discrepancy assuming infinite precision of computation. Our deterministic algorithm will follow the outline of this construction (although crucial modifications are required in order to overcome the infinite precision assumption), so we quickly restate Chazelle's construction below. The main idea is to assign colours so as to minimize the value of an objective function $G = G(\chi, \{S_i\})$ defined as follows: let ε be chosen so that $\log \frac{1+\varepsilon}{1-\varepsilon} = \alpha \cdot \sqrt{\log(3z)/k}$ for some constant $\alpha > 2$, and let p_i (respectively, n_i) be the number of $u \in S_i$ such that $\chi(u) = +1$ (respectively, $\chi(u) = -1$) for $i \in [z]$. Define

$$G_i = (1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i} + (1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i} \quad \text{and} \quad G = \sum_{i \in [z]} G_i$$

Chazelle's construction assigns colours to one element of U at a time, without ever backtracking. To assign a colour to an element u, it performs the following three simple steps. First, it computes G^+, the value of G assuming $\chi(u) = +1$. Second, it computes G^-, the

value of G assuming $\chi(u) = -1$. Finally, if $G^+ \leq G^-$, it sets $\chi(u) = +1$ and $G = G^+$, and otherwise it sets $\chi(u) = -1$ and $G = G^-$. Note that for each $i \in [z]$, we have

$$(1+\varepsilon)^{p_i+1}(1-\varepsilon)^{n_i} + (1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i+1} + (1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i+1} + (1+\varepsilon)^{n_i+1}(1-\varepsilon)^{p_i} = 2 \cdot ((1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i} + (1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i})$$

and therefore the value of G can only decrease. This implies an important property of Chazelle's construction: since at initialization we have $n_i = p_i = 0$ for all $i \in [z]$ and therefore $G = 2z$, we have $G_i \leq G \leq 2z$ for $i \in [z]$ at any moment of the construction. Let us show that small values of G_i's imply small discrepancy. In order to do this, we follow the outline of [13], but use a slightly higher bound for G_i's to be able to apply this lemma later.

▶ **Lemma 4** ([13]). *If after all elements of U have been assigned a colour we have $G_i \leq 3z$ for all $i \in [z]$, then the discrepancy of the resulting colouring is at most $\alpha \cdot \sqrt{k \log(3z)}$ for any constant $\alpha > 2$.*

We will show a deterministic algorithm that computes a colouring for which the values G_i are bounded by $3z$. By Lemma 4, we therefore obtain that the discrepancy is bounded by $\alpha \cdot \sqrt{k \log(3z)}$. We must overcome several crucial issues: first, we must explain how to compute ε. Second, we must design an algorithm that uses only multiplications and additions so as to be able to control the accumulated precision error. And finally, we must explain how to remove the assumption of infinite precision and to ensure that we never operate on numbers that are too small.

▶ **Proposition 5.** *Assume $k > \log(3z)$. There is a deterministic algorithm that computes $\varepsilon \in (0,1)$ such that $\log \frac{1+\varepsilon}{1-\varepsilon} = \alpha \cdot \sqrt{\log(3z)/k}$ for some constant $\alpha > 2$ in $\mathcal{O}(\log(zk))$ time. Both ε and $1-\varepsilon$ are bounded from below by $1/(kz)^{\mathcal{O}(1)}$.*

We can implement Chazelle's construction to use only multiplications and additions via segment trees.

▶ **Proposition 6.** *Assume that $(1+\varepsilon)$ and $(1-\varepsilon)$ are known. Chazelle's construction can be implemented via $\mathcal{O}(zk \log z)$ addition and multiplication operations.*

Proof. We maintain a complete binary tree on top of $\{1, 2, \ldots, 2^t\}$, where $2^{t-1} < z \leq 2^t$. At any moment, the $(2i-1)$-th leaf stores $(1+\varepsilon)^{p_i}(1-\varepsilon)^{n_i}$ and the $(2i)$-th leaf stores $(1+\varepsilon)^{n_i}(1-\varepsilon)^{p_i}$ for all $i \in [z]$, while all the other leaves store value 0. Each internal node stores the sum of the values in the leaves of its subtree. In particular, the root stores the value G. To update G after setting $\chi(u)$ for $u \in U$, we must update the values stored in the $(2i-1)$-th and $(2i)$-th leaves for all i such that $u \in S_i$, as well as the sums in the $\mathcal{O}(\log z)$ internal nodes above these leaves. For each leaf, we use one multiplication operation (we must multiply the value by $(1+\varepsilon)$ or $(1-\varepsilon)$ as appropriate), and for each internal node we use one addition operation. In total, we need $\mathcal{O}(\sum_{i \in [z]} |S_i| \log z) = \mathcal{O}(zk \log z)$ addition and multiplication operations. ◀

We are now ready to remove the infinite precision assumption and to show the final result of this section. Our algorithm will follow the outline of Proposition 6, but the addition and the multiplication operations will be implemented with precision Δ. Moreover, we will guarantee that the algorithm only works with values in $[\Delta, \mathcal{O}(z)]$, which will imply that both arithmetic operations can be performed in constant time and that the algorithm takes $\mathcal{O}(zk \log z)$ time.

▶ **Theorem 7.** *Given a family of z sets $S_i \subseteq U$ where $|S_i| \leq k$ and $|U| = zk$, one can find deterministically in $\mathcal{O}(zk \log z)$ time a colouring $\chi : U \to \{-1, +1\}$ such that $\max_{i \in [z]} |\chi(S_i)| \leq \alpha \cdot \sqrt{k \log(3z)}$ for some constant $\alpha > 2$.*

Theorem 7 can be used to partition the universe U into a small number of subsets such that the intersection of every subset of the partition and every set S_i is small. We start with a simple technical lemma.

▶ **Lemma 8.** *Consider a process that starts with $x_0 = x$, and keeps computing $x_{i+1} := \lfloor x_i(1/2 + 1/\sqrt{x_i}) \rfloor$ as long as $x_i > 4$. The process ends after at most $\log x + \mathcal{O}(\log^* x)$ steps.*

▶ **Lemma 9.** *Given a family of z sets $S_i \subseteq U$ where $|S_i| \leq k$ and $|U| = zk$, one can construct deterministically in $\mathcal{O}(|U| \log z \log k)$ time a function $f : U \to [k \cdot 2^{\mathcal{O}(\log^* k)}]$ such that for each $c \in [k \cdot 2^{\mathcal{O}(\log^* k)}]$ and for each S_i, the intersection of $\{u \in U \mid f(u) = c\}$ and S_i contains $\mathcal{O}(\log z)$ elements.*

Proof. We can reformulate the statement of the lemma as follows. We must show that there is a partitioning of U into subsets $X_c = \{u \in U : f(u) = c\}$ such that for every S_i, the intersection $X_c \cap S_i$ has size at most $\mathcal{O}(\log z)$.

We partition U recursively using the procedure from Theorem 7. We start with a single set $X = U$. Suppose that after several steps we have a partitioning of U into sets X_c such that $|S_i \cap X_c| \leq y$ for all i and c and some integer y. We then apply Theorem 7 to the sets X_c. Using the colouring output by the lemma, we partition each set X_c into sets X_{c_0} and X_{c_1}, where the former contains all the elements of X_c of colour -1 and the latter all the elements of X_c of colour $+1$. For $j \in \{0, 1\}$ we choose c_j (and also the value of $f(x)$ for $x \in X_{c_j}$) so that its binary representation equals the binary representation of c appended with j. By Theorem 7, there is a constant α such that

$$|S_i \cap X_{c_0}|, |S_i \cap X_{c_1}| \leq y/2 + \frac{1}{2}\alpha \cdot \sqrt{y \log(3z)} \leq y(1/2 + 1/\sqrt{y/\alpha^2 \log(3z)}).$$

We continue this process until $|S_i \cap X_c| \leq 4\alpha^2 \log(3z)$ for all i and c.

It remains to bound the number of iterations. By setting $x = k/\alpha^2 \log(3z)$ in Lemma 8, we obtain that we need at most $\log x + \mathcal{O}(\log^* x) \leq \log k + \mathcal{O}(1) + \mathcal{O}(\log^* k) = \log k + \mathcal{O}(\log^* k) = t$ recursive applications of the partition procedure implemented with Theorem 7 to ensure that every set S_i has at most $4\alpha^2 \log(3z) = \mathcal{O}(\log z)$ elements in common with every X_c. Therefore, the size of the image of f is bounded by $2^t = k2^{\mathcal{O}(\log^* k)}$. The overall construction time is $\mathcal{O}(|U| \log z \log k)$. ◀

2.2 Superimposed Codes

We are now ready to show an efficient construction algorithm for data-dependent superimposed codes (see Definition 1). At a high level, we will construct a family of functions which, combined with the partition f from Lemma 9, will give us the superimposed code.

▶ **Theorem 10.** *Given a family of z sets $S_i \subseteq U$ where $|S_i| \leq k$ and $|U| = zk$, one can construct an $(\{S_i\}, (1 - \varepsilon)w)$-superimposed code of weight $w = \mathcal{O}(\varepsilon^{-1} \log^2 |U|)$ and $\ell = \mathcal{O}(\varepsilon^{-2} k \log^5 |U|)$ in $\mathcal{O}(\varepsilon^{-1}|U| \log^2 |U|)$ time and space.*

Proof. By applying Lemma 9, we obtain in $\mathcal{O}(|U| \log z \log k) = \mathcal{O}(|U| \log^2 |U|)$ time a function $f : U \to [k \cdot 2^{\mathcal{O}(\log^* k)}]$ which gives a partitioning of U into subsets $X_c = \{u \in U \mid f(u) = c\}$, such that for some constant α, for every c and i holds $|X_c \cap S_i| \leq \alpha \log z$.

Consider the ring of polynomials $\mathbb{Z}_2[x]$. Let $U = \{u_1, u_2, \ldots, u_{zk}\}$. We define a mapping $\textsc{pol} : U \to \mathbb{Z}_2[x]$ as follows. Let $u = u_q$ and $q = \overline{q_t q_{t-1} \ldots q_0}$ be the binary representation of q, where $t = \lfloor \log |U| \rfloor$, then $\textsc{pol}(u) = \sum_{i=0}^{t} q_i x^i$.

Let $\mathcal{H}(U, d)$ be the family of functions $h_p : U \to \mathbb{F}_{2^d}$ of the form $h_p(u) = (\textsc{pol}(u) \bmod p)$ for all irreducible polynomials p of degree d. By Gauss's formula [14, 23], there are $\Theta(2^d/d)$ irreducible polynomials of degree d over $\mathbb{Z}_2$, and so is the size of the family $\mathcal{H}(U, d)$. Consider two distinct polynomials x, y of degree t. Observe that there are at most t/d irreducible polynomials p that hash both x and y to the same value $h_p(x) = h_p(y)$, because $\mathbb{Z}_2[x]$ is a unique factorization domain [23]. We choose d in such a way that the probability that x, y are hashed to the same value while choosing a hash function uniformly at random from $\mathcal{H}(U, d)$ is bounded by $\varepsilon/(\alpha \log z)$: $\frac{t/d}{\Theta(2^d/d)} \leq \frac{\varepsilon}{\alpha \log z}$ and hence we can choose $d = \Theta(\log \frac{t \log z}{\varepsilon})$.

If $d > t$, then $\varepsilon < \frac{\log^2 |U|}{|U|}$ and we can take $\ell = |U|, w = 1$ and set $C_{u_q} = \{q\}$. From now on, assume $d \leq t$. Let f be as in Lemma 9. Consider $u \in U$ such that $u \in X_c$, where $c = f(u) \in [k \cdot 2^{\mathcal{O}(\log^* k)}]$. We define C_u as follows:

$$C_u = \{H_p(u) = \textsc{num}(h_p(u)) + 2^d \cdot \textsc{num}(p) + 4^d \cdot c \mid h_p \in \mathcal{H}(U, d)\},$$

where the mapping $\textsc{num}(q)$ treats a polynomial $q = \sum_{i=0}^{d-1} q_i x^i$ as a d-bit number $\overline{q_{d-1} \ldots q_0}$. Clearly, $w = |C_u| = \mathcal{O}(2^d/d) = \mathcal{O}(2^d) = \mathcal{O}(\frac{t \log z}{\varepsilon}) = \mathcal{O}(\varepsilon^{-1} \log^2 |U|)$ and $C_u \subseteq [l]$ where:

$$\ell = 2^d \cdot 2^d \cdot k 2^{\mathcal{O}(\log^* k)} = \frac{t^2 \log^2 z}{\varepsilon^2} \cdot k \cdot 2^{\mathcal{O}(\log^* k)} = \mathcal{O}(\varepsilon^{-2} k \log^5 |U|).$$

We claim that the obtained code is a $(\{S_i\}, (1-\varepsilon)w)$-superimposed code. Consider any S_i and $u \notin S_i$. We count elements of C_u that do not belong to any C_v, for $v \in S_i$. Let $c = f(u) \in [k \cdot 2^{\mathcal{O}(\log^* k)}]$ and so $u \in X_c$. By construction, $|X_c \cap S_i| \leq \alpha \log z$. Thus, by the union bound, the probability that $h_p(u) = h_p(x)$ for some $x \in X_c \cap S_i$ is at most ε for h_p chosen uniformly at random from $\mathcal{H}(U, d)$. Recall that C_u consists of elements $H_p(u) = \textsc{num}(h_p(u)) + 2^d \cdot \textsc{num}(p) + 4^d \cdot c$ for $h_p \in \mathcal{H}(U, d)$. The number of irreducible polynomials p such that $H_p(u) = H_p(x)$ for some $x \in X_c \cap S_i$ is at most $\varepsilon \cdot w$. Consequently, at least $w - \varepsilon \cdot w = (1-\varepsilon)w$ elements of C_u do not belong to any C_v, for $v \in S_i$.

We now show that we can construct the above superimposed codes in $\mathcal{O}(|U|w)$ time. To this end, we need to generate all irreducible polynomials of degree d and to explain how we compute remainders modulo these polynomials. Note first that as we only operate on polynomials of degree $\leq t = \mathcal{O}(\log |U|)$, they fit in a machine word and hence we can subtract two polynomials or multiply a polynomial by any power of x in constant time. We can now use this to generate the irreducible polynomials and compute the sets C_u at the same time. We maintain a bit vector I that for each polynomial p of degree $\leq d$ stores an indicator bit equal to 1 iff p, i.e. iff its remainder modulo any polynomial of degree smaller than $\deg(p)$ is not zero. We consider the polynomials of degree $0, 1, 2, \ldots, d$ in order. For every irreducible polynomial p, we compute a table $\texttt{Mod}_p[q] = (q \bmod p)$ for all polynomials q of degree $\leq t$ in overall $\mathcal{O}(|U|)$ time using dynamic programming with the following recursive formula:

$$\texttt{Mod}_p[q] = \begin{cases} q, & \text{if } \texttt{deg}(q) < \texttt{deg}(p) \\ \texttt{Mod}_p[q - p \cdot x^{\texttt{deg}(q) - \texttt{deg}(p)}], & \text{otherwise} \end{cases}$$

We use the table to compute $H_p(u)$ for all $u \in U$. Also, if for a polynomial q the remainder is zero, we zero out the corresponding bit in I. Here we use the fact that $d \leq t$ to guarantee that we will find all irreducible polynomials of degree $\leq d$ in this way.

As there are w irreducible polynomials, in total we spend $\mathcal{O}(|U|w) = \mathcal{O}(\varepsilon^{-1}|U| \log^2 |U|)$ time. At any moment, we use $\mathcal{O}(|U|)$ space to store the table and $\mathcal{O}(\varepsilon^{-1}|U| \log^2 |U|)$ space to store the codes. ◀

3 Upper Bounds for Generalised Pattern Matching

In this section, we present new algorithms for the parameters $\mathcal{D}$, $\mathcal{S}$ and $\mathcal{I}$. Our algorithms for the parameters $\mathcal{D}$ and $\mathcal{S}$ share similar ideas, so we present them together in Section 3.1. The algorithm for $\mathcal{I}$ is presented in Section 3.2.

We start by recalling the formal statement of the Pattern Matching with Don't Cares problem that will be used throughout this section.

> Pattern Matching with Don't Cares (counting, binary alphabet)
> **Input:** A text $T \in \{0,1,?\}^n$ and a pattern $P \in \{0,1,?\}^m$, where "?" is a *don't care character* that matches any character of the alphabet.
> **Output:** For each $i \in [n-m+1]$, the number of positions $j \in [m]$ such that $T[i+j-1]$ does not match $P[j]$.

Clifford and Clifford [15] showed that this problem can be solved in $\mathcal{O}(n \log m)$ time.

3.1 Parameters $\mathcal{D}$ and $\mathcal{S}$

We first show Monte Carlo algorithms for the reporting and counting variants of GPM, and then de-randomise them using the data-dependent superimposed codes of Section 2.

3.1.1 Randomised Algorithms

We start by presenting a new reporting algorithm for the parameter $\mathcal{D}$. It does not improve over the algorithm of [34], but encapsulates a novel idea that will be used by all our algorithms for the parameters $\mathcal{D}$ and $\mathcal{S}$. Essentially, we use hashing to reduce Σ_T to a smaller set of characters of size $p = \Theta(\mathcal{D})$ while preserving occurrences of the pattern in the text with constant probability, and then show that this smaller instance of GPM can be reduced to $p = \Theta(\mathcal{D})$ instances of Pattern Matching with Don't Cares.

▶ **Theorem 11.** *Let $\mathcal{D}$ be the maximum degree in the matching graph M and c be any constant fixed in advance. There is a Monte Carlo algorithm that solves the reporting variant of GPM in $\mathcal{O}(\mathcal{D}\, n \log m \log n)$ time. The error is one-sided (only false positives are allowed), and the error probability is at most $1/n^c$.*

Proof. If $\mathcal{D} > m$, we can use a naive algorithm that compares the pattern and each m-length substring of the text character-by-character and uses $\mathcal{O}(mn) = \mathcal{O}(\mathcal{D}\, n)$ time in total. Below we assume $\mathcal{D} \leq m$. We can also assume $|\Sigma_T| \leq n$.

We first choose a 2-wise independent hash function $h : \Sigma_T \to [2\mathcal{D}]$ of the form $h(x) = ((a \cdot x + b) \mod p) \mod (2\mathcal{D}) + 1$, where $p \geq |\Sigma_T|$ is a prime, and a, b are chosen independently and uniformly from $\mathbb{F}_p$. Note that we can find a prime p such that $n \leq p \leq 2n$, in $\mathcal{O}(n)$ time. Consider a matching graph M' on the set of vertices $[p] \cup \Sigma_P$. For every character $b = P[j]$ and for every character $a \in \Sigma_T$ in the adjacency list of b, we add an edge $(h(a), b)$ to M'. Overall, it takes $\mathcal{O}(\mathcal{D}\, m) = \mathcal{O}(\mathcal{D}\, n)$ time.

We claim that if M does not contain an edge (a, b), then the probability of M' to contain an edge $(h(a), b)$ is at most $1/2$. By definition, if $(h(a), b)$ belongs to M', then there exists a character $a' \in \Sigma_T$ such that (a', b) is in M and $h(a') = h(a)$. Since h is 2-wise independent, for a fixed character a' the probability of $h(a') = h(a)$ is $1/(2\mathcal{D})$. Because the degree of b is at most $\mathcal{D}$, the probability of such event is at most $1/2$ by the union bound.

Consider a text T', where $T'[i] = h(T[i])$. If $T[i, i+m-1]$ does not match P under M, then $T'[i, i+m-1]$ does not match P under M' with probability $\geq 1/2$. Indeed, suppose that for some $j \in [m]$, $T[i+j-1]$ and $P[j]$ do not match under M, or equivalently, an

edge $(T[i+j-1], P[j])$ does not belong to M. From above, with probability at least $1/2$, $h(T[i+j-1])$ and $P[j]$ do not match under M'. It follows that we can use the GPM algorithm for M', T', and P to eliminate every non-occurrence of P in T with probability at least $1/2$. We can amplify the probability in a standard way, i.e. by independently repeating the algorithm $c \log n$ times.

It remains to explain how to solve GPM for M', T', and P. We use the fact that the size of the alphabet of T' is $\mathcal{O}(\mathcal{D})$. For every $a \in [2\mathcal{D}]$ we create a new text $T'_a[1,n]$ and a new pattern $P_a[1,m]$ as follows:

$$T'_a[j] = \begin{cases} \texttt{0} & \text{if } T'[j] = a, \\ \texttt{?} & \text{otherwise.} \end{cases} \qquad P_a[j] = \begin{cases} \texttt{0} & \text{if } a \text{ matches } P[j] \text{ under } M', \\ \texttt{1} & \text{otherwise.} \end{cases}$$

We can construct T'_a and P_a in $\mathcal{O}(n+m) = \mathcal{O}(n)$ time, or in $\mathcal{O}(\mathcal{D}\,n)$ total time for all $a \in [2\mathcal{D}]$. It is not hard to see that $T'[i, i+m-1]$ matches P if and only if $T'_a[i, i+m-1]$ matches P_a for all $a \in [2\mathcal{D}]$. Therefore, to solve GPM for M', T', and P, it suffices to solve the $2\mathcal{D}$ instances of Pattern Matching with Don't Cares. By [15], this can be done in total $\mathcal{O}(\mathcal{D}\,n \log m)$ time. As we repeat the algorithm $c \log n$ times, the theorem follows. ◀

We now show a new randomised algorithm for the parameter $\mathcal{S}$. At a high level, we divide Σ_P into heavy and light characters based on their degree in M (a character of Σ_P is called heavy when it matches many characters of Σ_T, and light otherwise). The number of heavy characters is relatively small, and we can eliminate all substrings of T that do not match P because of heavy characters by running an instance of Pattern Matching with Don't Cares for each of them. For light characters, we apply Theorem 11.

▶ **Theorem 12.** *Let $\mathcal{S}$ be the number of edges in the matching graph M and c be any constant fixed in advance. There is a Monte Carlo algorithm that solves the reporting variant of GPM in $\mathcal{O}(\sqrt{\mathcal{S}}\,n \log m \sqrt{\log n})$ time. The error is one-sided (only false positive are allowed), and the error probability is at most $1/n^c$.*

Combining the techniques of Theorems 11, 12 and the approach of Kopelowitz and Porat [31], we obtain the following corollary.

▶ **Corollary 13.** *Let c be any constant fixed in advance, $\mathcal{D}$ be the maximum degree and $\mathcal{S}$ be the number of edges in the matching graph M. There is a $(1-\varepsilon)$-approximation Monte Carlo algorithm that solves the counting variant of GPM in $\mathcal{O}(\min\{\varepsilon^{-1}\mathcal{D}\log n, \sqrt{\varepsilon^{-1}\mathcal{S}\log n}\} \cdot n \log m)$ time. The error probability is at most $1/n^c$.*

3.1.2 Deterministic Algorithms

We are now ready to give $(1-\varepsilon)$-approximation deterministic algorithms for the counting variant of GPM for the parameters $\mathcal{D}$ and $\mathcal{S}$. By taking $\varepsilon = 1/2$, the algorithms for the reporting variant follow immediately. We first remind the definition of superimposed codes, which we will use throughout this section.

▶ **Definition 1.** *Let $S_1, \ldots, S_z$ be subsets of a universe U. A family of sets $\mathcal{C} = \{C_1, \ldots, C_{|U|}\}$, where $C_u \subseteq [\ell]$ and $|C_u| = w$ for $u \in U$ is called an $(\{S_i\}, \tau)$-superimposed code if for every S_i and $u \notin S_i$ we have $|C_u - \bigcup_{v \in S_i} C_v| \geq \tau$. We call ℓ and w respectively the length and the weight of the code $\mathcal{C}$.*

▶ **Theorem 14.** *Let $\mathcal{D}$ be the maximum degree in the matching graph M. There is an $(1-\varepsilon)$-approximation deterministic algorithm that solves the counting variant of GPM in $\mathcal{O}(\varepsilon^{-2}\mathcal{D}\,n \log^6 n)$ time.*

Proof. First, note that we can assume $\mathcal{D} \leq m$ and $\varepsilon \geq 1/m$. If this is not the case, we can run a naive algorithm that compares each m-length substring of the text T and the pattern character-by-character in $\mathcal{O}(mn) = \mathcal{O}(\mathcal{D}\, n)$ time.

For each distinct character b of the pattern P, consider a set S_b containing all characters in Σ_T that match b. By definition, $|S_b| \leq \mathcal{D}$. We define the universe $U = (\bigcup_{b\in\Sigma_P} S_b) \cup \{\$\}$, where $\$ \notin \Sigma_T$ is a special character that we will need later, $|U| = \mathcal{O}(n)$. We apply Theorem 10 that constructs $(\{S_b\}, (1-\varepsilon)w)$-superimposed code for the universe U and sets S_b in $\mathcal{O}(\varepsilon^{-1} n \log^2 n)$ time, where the weight $w = \mathcal{O}(\varepsilon^{-1}\log^2 n)$ and the length $\ell = \mathcal{O}(\varepsilon^{-2}\mathcal{D}\log^5 n)$.

We define the code of a character $a \in U$ to be a binary vector of length ℓ such that its j-th bit equals 1 if C_a contains j, and 0 otherwise. For a character $a' \in \Sigma_T \setminus U$, we define its code to be equal to the code of \$. We define the code of a character $b \in \Sigma_P$ to be a binary vector of length ℓ such that its j-th bit equals 1 if $\bigcup_{a\in S_b} C_a$ contains j, and 0 otherwise. Next, we create a text $T'[1, n\ell]$ and a pattern $P'[1, m\ell]$ by replacing the characters in respectively T and P by their codes. To finish this step, we replace each 1 in P' with the don't care character and run the algorithm of Clifford and Clifford [15] for T' and P' that takes $\mathcal{O}(n\ell\log(m\ell)) = \mathcal{O}(\varepsilon^{-2}\mathcal{D}\, n\log^6 n)$ time (here we use $\varepsilon \geq 1/m$).

Let h' be the number of mismatching characters between P' and $T'[(i-1)\cdot\ell+1, (i+m-1)\cdot\ell]$, and h be the number of mismatches between P and $T[i, i+m-1]$. We claim that $(1-\varepsilon)wh \leq h' \leq wh$. Indeed, if $P[j]$ matches $T[i+j-1]$, then $C_{T[i+j-1]}$ is a subset of $\bigcup_{a\in S_{P[j]}} C_a$. Therefore, if the code of $T[i+j-1]$ contains 1 in position k, the code of $P[j]$ will have 1 in position k as well. By replacing all 1s in P' with the don't care characters, we ensure that the corresponding fragments of P' and T' match. On the other hand, if $P[j]$ does not match $T[i+j-1]$, then from the definition of the code it follows that the distance between the corresponding chunks of P' and T' will be at least $(1-\varepsilon)w$ and at most w. ◀

To show a deterministic algorithm for the parameter $\mathcal{S}$, we again consider the partition of the alphabet Σ_P into heavy and light characters. To count the mismatches caused by some heavy character, we create an instance of Pattern Matching with Don't Cares. As the number of heavy characters is small, the total number of the created instances is small as well. For light characters, we use the superimposed codes similarly as in Theorem 14.

▶ **Theorem 15.** *Let $\mathcal{S}$ be the number of edges in the matching graph M. There is an $(1-\varepsilon)$-approximation deterministic algorithm that solves the counting variant of GPM in $\mathcal{O}(\varepsilon^{-1}\sqrt{\mathcal{S}}\, n\log^{7/2} n)$ time.*

3.2 Parameter $\mathcal{I}$

In this section, we show a deterministic GPM algorithm for the parameter $\mathcal{I}$. The algorithm solves the counting variant of the problem exactly, and we can immediately derive an algorithm for the reporting version with the same complexities as a corollary. We will need the following technical lemma.

▶ **Lemma 16.** *Let b be a parameter, $S = \{x_1, x_2, \ldots, x_\ell\}$ be a sequence of integers, and $s = \sum_{i\in[\ell]} x_i$. Then S can be partitioned into $\mathcal{O}(s/b+1)$ ranges $S_1, S_2, \ldots$ such that, for every i, either S_i is a singleton or the sum of all elements in S_i is at most b.*

We are now ready to show the main result of the section.

▶ **Theorem 17.** *For each character $a \in \Sigma_P$ consider a minimal set $I(a)$ of disjoint sorted intervals that contain the characters that match a, and define $\mathcal{I} = \sum_{j\in[m]} |I(P[j])|$. There is a deterministic algorithm that solves the counting version of GPM in $\mathcal{O}(n\sqrt{\mathcal{I}\log m} + n\log n)$ time.*

Proof. If $\mathcal{I} > m^2$, we can use the naive algorithm that compares each m-length substring with the pattern character-by-character and takes $\mathcal{O}(mn)$ time in total.

We first make a pass over T and retrieve the set of distinct characters $a_1, a_2, \ldots, a_l$ of Σ_T that occur in it, as well as their frequencies. This can be done in $\mathcal{O}(n \log n)$ time using a binary search tree. We partition $a_1, a_2, \ldots, a_l$ into ranges as follows. Let $\mathsf{count}(c)$, for $c \in \Sigma_T$, be the frequency (i.e. the number of occurrences) of c in T. We apply Lemma 16 for $b > 1$ that will be specified later and the sequence $\mathsf{count}(a_1), \mathsf{count}(a_2), \ldots, \mathsf{count}(a_l)$ which sums up to n.

Let Σ'_T be a new alphabet obtained by creating a character for every range in the partition, where $|\Sigma'_T| = \mathcal{O}(n/b + 1)$. For $c \in \Sigma'_T$ we denote by $\mathsf{range}(c)$ the range of Σ_T corresponding to c, and for $a \in \Sigma_T$ we denote by $\mathsf{range}^{-1}(a)$ the character of Σ'_T corresponding to the range containing a. We create a new text $T'[1, n]$ and pattern $P'[1, m]$ as follows. For every $i \in [n]$, we set $T'[i] = \mathsf{range}^{-1}(T[i])$. For every $j \in [m]$, we set $P'[j] = \{c \in \Sigma'_T \mid \mathsf{range}(c) \text{ contains a character that matches } P[j]\}$. As the number of the ranges is $\mathcal{O}(n/b + 1)$, the size of the set $P'[j]$ is $\mathcal{O}(n/b + 1)$. We represent it as a binary vector of length $\mathcal{O}(n/b + 1)$. Furthermore, we can construct T' in $\mathcal{O}(n)$ time, and P' in $\mathcal{O}(\mathcal{I} + m(n/b + 1))$ time.

After this initial step the algorithm consists of two phases. First, we solve the Subset Pattern Matching for T' and P' that consists of counting, for every $i \in [n - m + 1]$, all positions $j \in [m]$ such that $T'[i + j - 1] \notin P'[j]$. To this end, we create an instance of Pattern Matching with Don't Cares for every $c \in \Sigma'_T$, namely, we create a text $T'_c[1, n]$ and a pattern $P'_c[1, m]$ as follows:

$$T'_c[i] = \begin{cases} \mathtt{0} & \text{if } T'[i] = c, \\ \mathtt{?} & \text{otherwise.} \end{cases} \qquad P'_c[j] = \begin{cases} \mathtt{0} & \text{if } c \in P'[j], \\ \mathtt{1} & \text{otherwise.} \end{cases}$$

We can solve all these instances in $\mathcal{O}(|\Sigma'_T| n \log m) = \mathcal{O}((n/b+1)n \log m)$ time [15]. Summing up the results, we obtain the result for the subset matching problem.

In the second phase, we slightly adjust the results obtained for Subset Pattern Matching to obtain the results for GPM. Consider a substring $T[i, i + m - 1]$ that does not match P because of a mismatch in position j of the pattern, i.e. $T[i + j - 1]$ does not match $P[j]$. We have two possible cases. The first case is when $T'[i + j - 1] \notin P'[j]$. In this case, the mismatch is detected by the Subset Pattern Matching algorithm. The second case is when $T'[i + j - 1] \in P'[j]$. Observe that in this case, $\mathsf{range}(T'[i + j - 1])$ cannot be a singleton and must contain an endpoint of some interval of characters that match $P[j]$.

To detect such mismatches, we run the following algorithm. For each $j \in [m]$, we consider the intervals $I(P[j])$ of the characters that match $P[j]$. For every endpoint $c \in \Sigma_T$ of the intervals in $I(P[j])$, we iterate over all $a \in \mathsf{range}^{-1}(c)$ such that a does not match $P[j]$ and all occurrences of a in the text. Summing over all j and a, there are in total $\mathcal{O}(\mathcal{I} \cdot b)$ of the occurrences due to the properties of the partition and the fact that $\mathsf{range}(T'[i + j - 1])$ is not a singleton. We can find the occurrences in $\mathcal{O}(\mathcal{I} \cdot b + n + mn/b)$ time as follows. First we find the ranges containing the endpoints in $\mathcal{O}(\mathcal{I} + m(n/b+1))$ time similarly to above, and we can generate the lists of occurrences of every character $a \in \Sigma_T$ in T by one pass over T in $\mathcal{O}(n \log n)$ time. For each such occurrence $T[k] = a$ that does not match $P[j]$, we increment the number of mismatches for the substring $T[k - j + 1, k + m - j]$. This correctly detects every mismatch that has not been accounted for in the first phase, and hence allows counting all mismatches in $\mathcal{O}(\mathcal{I} + m(n/b + 1) + (n/b + 1)n \log m + n \log n + \mathcal{I} \cdot b) = \mathcal{O}(n^2 \log(m)/b + n \log n + \mathcal{I} \cdot b)$ total time. Substituting $b = n\sqrt{\log m/\mathcal{I}}$ gives us the claim of the theorem. ◀

▶ **Corollary 18.** *There is a deterministic algorithm that solves the counting variant of the threshold pattern matching problem in* $\mathcal{O}(n(\sqrt{m \log m} + \log n))$ *time.*

4 Lower Bounds for GPM

In this section we give lower bounds for GPM algorithms. All the lower bounds are presented for the reporting variant of GPM, so they immediately apply also to the counting variant. Recall that we assume to have access to three oracles that can answer the following questions about the matching graph M in $\mathcal{O}(1)$ time:

1. Is there an edge between $a \in \Sigma_T$ and $b \in \Sigma_P$?
2. What is the degree of a character $a \in \Sigma_T$ or $b \in \Sigma_P$?
3. What is the k-th neighbor of $a \in \Sigma_T$?

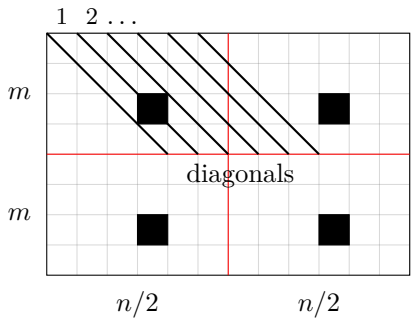

Figure 1 The adjacency matrix of the matching graph M. We show diagonals (solid lines) and a quadruple of related cells (black). Note that among any quadruple of related cells, only one can belong to a diagonal.

We first use an adversary-based argument to show an $\Omega(\mathcal{S})$ time lower bound.

▶ **Lemma 19.** *Any deterministic algorithm for GPM requires* $\Omega(\mathcal{S})$ *time.*

Proof. We will show that any deterministic algorithm checking if there exists at least one occurrence needs to inspect $\Omega(\mathcal{S})$ entries of M in the worst case by an adversary-based argument. In particular, this implies a lower bound of $\Omega(nm)$ when $\mathcal{S} = \Theta(nm)$. The main difficulty in the argument is to design the input so that the second oracle is essentially useless.

It will be convenient for us to think in terms of the adjacency matrix of the matching graph M that we denote by $\mathcal{M}$. Let us assume that $n \geq 2m$ is even, $\Sigma_T = [n]$, and $\Sigma_P = [2m]$. We split both alphabets into halves. For every $a \in [n/2]$ and $b \in [m]$ we will choose one of the following two possibilities:

1. $\mathcal{M}[a, b] = \mathcal{M}[n/2 + a, m + b] = 1$ and $\mathcal{M}[n/2 + a, b] = \mathcal{M}[a, m + b] = 0$,
2. $\mathcal{M}[a, b] = \mathcal{M}[n/2 + a, m + b] = 0$ and $\mathcal{M}[n/2 + a, b] = \mathcal{M}[a, m + b] = 1$.

We call $\mathcal{M}[a, b]$, $\mathcal{M}[n/2 + a, b]$, $\mathcal{M}[a, m + b]$ and $\mathcal{M}[n/2 + a, m + b]$ *related*. Observe that, irrespectively of all such choices, the second oracle returns the same number for every $b \in \Sigma_P$ and every $a \in \Sigma_T$, and so the algorithm only needs to query the first oracle.

We choose the text $T = 1\ 2 \ldots n/2\ 1\ 2 \ldots n/2$ and the pattern $P = 1\ 2 \ldots m$. Clearly, P occurs in T when, for some $a \in [n/2]$, we have $M[1 + (a + b - 2) \bmod n/2, b] = 1$ for every $b \in [m]$. We call the set of corresponding entries of M a *diagonal* (see Fig. 1).

Note that among any quadruple of related entries exactly one can belong to the diagonals. Furthermore, suppose that an algorithm retrieves the values in a quadruple of related entries $\mathcal{M}[a,b]$, $\mathcal{M}[n/2+a,m+b]$, $\mathcal{M}[n/2+a,b]$, $\mathcal{M}[a,m+b]$. This can be done by one of the following queries: ask for the value of any of these four entries, or retrieve the particular neighbor of one of the nodes a, $n/2+a$, b, or $m+b$. In both cases, we retrieve only the related entries and spend $\Omega(1)$ time for any of the retrieved quadruples.

The adversary proceeds as follows. If the algorithm retrieves a quadruple containing $\mathcal{M}[a,b]$, for $a \in [n/2]$ and $b \in [m]$, such that the value of $\mathcal{M}[a,b]$ is not yet determined, the adversary checks if setting $\mathcal{M}[a,b]=1$ would result in creating a diagonal containing only 1s. If so, the adversary sets $\mathcal{M}[a,b]=0$, and otherwise the adversary sets $\mathcal{M}[a,b]=1$. In other words, the adversary sets $M[a,b]=0$ when it is the last undecided entry on its diagonal.

The algorithm can report an occurrence only after having verified that the corresponding diagonal contains only 1s, and the adversary makes sure that this is never the case. On the other hand, if the algorithm terminates without having reported an occurrence while there exists a diagonal that has not been fully verified then the adversary could set its remaining entries to 1s and obtain an instance that does contain an occurrence. Consequently, the algorithm needs to retrieve all the entries in all the diagonals, and as we showed, it requires $\Omega(mn)=\Omega(\mathcal{S})$ time.

Note that above $\mathcal{S}=nm/2$. The proof can be extended to $\mathcal{S}<nm/2$ as follows. If $\mathcal{S}\geq m$ we set $n'=\lfloor \mathcal{S}/m \rfloor$ and choose the text to be the prefix of length n of $(1\ 2\ldots n')^\infty$ (the string $1\ 2\ldots n'$ repeated infinitely many times). Then the above argument shows that any algorithm needs to inspect $n'm \geq \mathcal{S}/2$ entries of $\mathcal{M}$. If $\mathcal{S}<m$ we choose the pattern to be the prefix of length m of $(1\ 2\ldots \mathcal{S})^\infty$ (the string $1\ 2\ldots \mathcal{S}$ repeated infinitely many times), the text to be 1^n (1 repeated n times) and proceed as above to argue that one must inspect $\Omega(\mathcal{S})$ entries of $\mathcal{M}$. ◀

A similar argument can be used to show that this bound holds for Monte Carlo algorithms with constant error probability as well.

▶ **Lemma 20.** *Any Monte Carlo algorithm for GPM with constant error probability $\varepsilon < 1/2$ requires $\Omega(\mathcal{S})$ time.*

We now show lower bounds for GPM conditional on hardness of Boolean matrix multiplication.

▶ **Conjecture** ([2]). *For any $\alpha,\beta,\gamma,\varepsilon>0$, there is no combinatorial[3] algorithm for multiplying two Boolean matrices of size $N^\alpha \times N^\beta$ and $N^\beta \times N^\gamma$ in time $\mathcal{O}(N^{\alpha+\beta+\gamma-\varepsilon})$.*

A simple adaptation of the folklore lower bound for computing the Hamming distance (cf. [24]) yields the following lower bounds.

▶ **Lemma 21.** *For any $\alpha \geq 1$, and $1 \geq \beta,\varepsilon > 0$, there is no combinatorial algorithm that solves GPM in time $\mathcal{O}(\mathcal{S}^{0.5-\varepsilon}n)$, for $n=\Theta(m^{(1+\alpha)/2})$ and $\mathcal{S}=\Theta(m^\beta)$.*

Proof. We show a reduction from Boolean matrix multiplication. Consider a matrix A of size $x \times y$ and a matrix B of size $y \times z$, where $x=N^\alpha$, $y=N^\beta$, $z=N$. We transform the matrix A by replacing every 1 by the number of the column it belongs to and every 0 by the don't care character ?. Similarly, we replace each 1 in B by the number of the row it belongs to and every 0 by the don't care character ?.

[3] It is not clear what combinatorial means precisely, but fast matrix multiplication is definitely non-combinatorial. Arguably neither is FFT used in our algorithms, thus making them non-combinatorial.

▶ **Example 22.** Consider $A = ((0,0,1),(1,0,1),(0,1,0))$ and $B = ((1,0,1),(0,1,0),(1,1,0))$. After the transform, they become $((?,?,3),(1,?,3),(?,2,?))$ and $((1,?,1),(?,2,?),(3,3,?))$, respectively.

We define the text $T = ?^{z^2} A_1 ?^{z-y+1} A_2 ?^{z-y+1} \dots ?^{z-y+1} A_x ?^{z^2}$, where A_i is the i-th row of A, and the pattern $P = B_1 ?^{z-y} B_2 ?^{z-y} \dots ?^{z-y} B_z$, where B_j is the j-th column of the matrix B. The length of T is $n = 2z^2 + (x-1)(z-y+1) + xy = \mathcal{O}(N^{1+\alpha})$, and the length of P is $m = yz + (z-y)(z-1) = \mathcal{O}(N^2)$. Next, we define the matching relationship as follows. Every character different than the don't care is defined to match all characters of the alphabet but itself, and the don't care character matches all characters of the alphabet. Consequently, the alphabet has size $y+1$ and the matching relationship matrix contains $\mathcal{S} = \Theta(y^2) = \Theta(N^{2\beta})$ set bits.

Let $C = A \times B$. By definition, $C[i,j] = 1$ iff $\bigvee_{k=1}^{y} (A_i[k] \wedge B_j[k]) = 1$. We claim that this is the case iff, aligning A_i in the text and B_j in the pattern does not yield an occurrence of the pattern. Suppose first that $\bigvee_{k=1}^{y} (A_i[k] \wedge B_j[k]) = 1$. Then there is k_0 such that $A_i[k_0] = B_j[k_0] = 1$. In the text and in the pattern they are both encoded by the same $k_0 \neq ?$ and aligned, and k_0 does not match itself. Therefore, we do not have an occurrence. Assume otherwise. We need to show that for every character $a \neq ?$, a is not aligned with itself. For B_j it follows from the fact that $\bigvee_{k=1}^{y} (A_i[k] \wedge B_j[k]) \neq 1$. For other columns of B it follows from the shift caused by the don't care characters.

It follows that a combinatorial algorithm that correctly outputs all occurrences of P in T in $\mathcal{O}(\mathcal{S}^{0.5-\varepsilon} n)$ time implies a combinatorial algorithm for Boolean matrix multiplication of matrices of size $N^\alpha \times N^\beta$ and $N^\beta \times N$ in time $\mathcal{O}(\mathcal{S}^{0.5-\varepsilon} n) = \mathcal{O}(N^{1+\alpha+2\beta(0.5-\varepsilon)}) = \mathcal{O}(N^{\alpha+1+\beta-2\varepsilon\beta})$, which contradicts the combinatorial matrix multiplication conjecture. The lower bound follows. ◀

▶ **Corollary 23.** *For any $\alpha \geq 1$, and $1 \geq \beta, \varepsilon > 0$, there is no combinatorial algorithm that solves GPM in time $\mathcal{O}(\mathcal{D}^{1-\varepsilon} n)$, for $n = \Theta(m^{(1+\alpha)/2})$ and $\mathcal{D} = \Theta(m^\beta)$. For any $\alpha \geq 1$, and $1 \geq \varepsilon > 0$, there is no combinatorial algorithm that solves GPM in time $\mathcal{O}(\mathcal{I}^{0.5-\varepsilon} n)$, for $n = \Theta(m^{(1+\alpha)/2})$ and $\mathcal{I} = \Theta(m)$.*

Proof. To show the first part of the claim, note that in the constructed instance of generalized pattern matching $\mathcal{D} = \Theta(m^{\beta/2})$. For the second part, we take $\beta = 1$. Then $\mathcal{I} = \mathcal{O}(m)$, and therefore a combinatorial algorithm that correctly outputs all occurrences of P in T in $\mathcal{O}(\mathcal{I}^{0.5-\varepsilon} n)$ time implies a combinatorial algorithm for Boolean matrix multiplication of matrices of size $N^\alpha \times N$ and $N \times N$ in time $\mathcal{O}(\mathcal{I}^{0.5-\varepsilon} n) = \mathcal{O}(N^{1+\alpha+2(0.5-\varepsilon)}) = \mathcal{O}(N^{\alpha+2-2\varepsilon})$, which contradicts the combinatorial matrix multiplication conjecture. ◀

References

1 Amir Abboud, Loukas Georgiadis, Giuseppe F. Italiano, Robert Krauthgamer, Nikos Parotsidis, Ohad Trabelsi, Przemyslaw Uznanski, and Daniel Wolleb-Graf. Faster algorithms for all-pairs bounded min-cuts. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*, ICALP, pages 7:1–7:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.7.

2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, FOCS, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.

3 Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987. doi:10.1137/0216067.

4 Mikhail J. Atallah and Timothy W. Duket. Pattern matching in the Hamming distance with thresholds. *Information Processing Letters*, 111(14):674–677, 2011. doi:10.1016/j.ipl.2011.04.004.

5 Nikhil Bansal. Constructive algorithms for discrepancy minimization. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, FOCS, pages 3–10, 2010. doi:10.1109/FOCS.2010.7.

6 Nikhil Bansal, Moses Charikar, Ravishankar Krishnaswamy, and Shi Li. Better algorithms and hardness for broadcast scheduling via a discrepancy approach. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 55–71, 2014. doi:10.1137/1.9781611973402.5.

7 Nikhil Bansal, Daniel Dadush, and Shashwat Garg. An algorithm for Komlós conjecture matching Banaszczyk's bound. *SIAM J. Comput.*, 48(2):534–553, 2019. doi:10.1137/17M1126795.

8 Nikhil Bansal, Daniel Dadush, Shashwat Garg, and Shachar Lovett. The Gram-Schmidt walk: A cure for the Banaszczyk blues. In *Proceedings of the Annual ACM SIGACT Symposium on Theory of Computing*, STOC, pages 587–597, 2018. doi:10.1145/3188745.3188850.

9 Nikhil Bansal and Shashwat Garg. Algorithmic discrepancy beyond partial coloring. In *Proceedings of the Annual ACM SIGACT Symposium on Theory of Computing*, STOC, pages 914–926, 2017. doi:10.1145/3055399.3055490.

10 Nikhil Bansal and Joel Spencer. Deterministic discrepancy minimization. *Algorithmica*, 67(4):451–471, 2013. doi:10.1007/s00453-012-9728-1.

11 Emilios Cambouropoulos, Maxime Crochemore, Costas S. Iliopoulos, Laurent Mouchard, and Yoan J. Pinzon. Algorithms for computing approximate repetitions in musical sequences. *International Journal of Computer Mathematics*, 79(11):1135–1146, 2002. doi:10.1080/00207160213939.

12 Domenico Cantone, Salvatore Cristofaro, and Simone Faro. An efficient algorithm for δ-approximate matching with α-bounded gaps in musical sequences. In *Proceedings of the International Conference on Experimental and Efficient Algorithms*, WEA, pages 428–439, 2005. doi:10.1007/11427186_37.

13 Bernard Chazelle. *The discrepancy method - randomness and complexity*. Cambridge University Press, 2001. doi:10.1017/CBO9780511626371.

14 Sunil Chebolu and Jan Minac. Counting irreducible polynomials over finite fields using the inclusion-exclusion principle. *Mathematics Magazine*, 84(5):369–371, 2011. doi:10.4169/math.mag.84.5.369.

15 Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Information Processing Letters*, 101(2):53–54, 2007. doi:10.1016/j.ipl.2006.08.002.

16 Peter Clifford, Raphaël Clifford, and Costas Iliopoulos. Faster algorithms for δ,γ-matching and related problems. In *Proceedings on the Annual Symposium on Combinatorial Pattern Matching*, CPM, pages 68–78, 2005. doi:10.1007/11496656_7.

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

18 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, STOC, pages 592–601, 2002. doi:10.1145/509907.509992.

19 Richard Cole, Costas Iliopoulos, Thierry Lecroq, Wojciech Plandowski, and Wojciech Rytter. On special families of morphisms related to δ-matching and don't care symbols. *Information Processing Letters*, 85(5):227–233, 2003. doi:10.1016/S0020-0190(02)00430-1.

20 Maxime Crochemore, Costas S. Iliopoulos, Thierry Lecroq, Yoan J. Pinzon, Wojciech Plandowski, and Wojciech Rytter. Occurrence and substring heuristics for δ-matching. *Fundamenta Informaticae*, 56(1,2):1–21, October 2002.

21 Michael John Fischer and Michael Stewart Paterson. String-matching and other products. Technical report, Massachusetts Institute of Technology, 1974.

22 Kimmo Fredriksson and Szymon Grabowski. Efficient algorithms for (δ, γ, α) and $(\delta, k_\delta, \alpha)$-matching. *International Journal of Foundations of Computer Science*, 19(01):163–183, 2008. doi:10.1142/S0129054108005607.

23 Carl Friedrich Gauss. *Untersuchungen über höhere Arithmetik. (Disquisitiones arithmeticae. Theorematis arithmetici demonstratio nova. Summatio quarundam serierum singularium ó.)*. Deutsch hrsg. von H. Mas, Berlin, 1889.

24 Paweł Gawrychowski and Przemysław Uznański. Towards unified approximate pattern matching for Hamming and L_1 distance. In *Procedings of the International Colloquium on Automata, Languages and Programming*, ICALP, pages 62:1–62:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.62.

25 Loukas Georgiadis, Daniel Graf, Giuseppe F. Italiano, Nikos Parotsidis, and Przemyslaw Uznanski. All-pairs 2-reachability in $O(n^w \log n)$ time. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming*, ICALP, pages 74:1–74:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.74.

26 Jan Holub, William F. Smyth, and Shu Wang. Fast pattern-matching on indeterminate strings. *J. of Discrete Algorithms*, 6(1):37–50, March 2008. doi:10.1016/j.jda.2006.10.003.

27 Piotr Indyk. Deterministic superimposed coding with applications to pattern matching. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, FOCS, pages 127–136, 1997. doi:10.1109/SFCS.1997.646101.

28 Piotr Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, FOCS, pages 166–173, 1998. doi:10.1109/SFCS.1998.743440.

29 Adam Kalai. Efficient pattern-matching with don't cares. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 655–656, 2002.

30 William Kautz and Richard Singleton. Nonrandom binary superimposed codes. *IEEE Trans. Inf. Theor.*, 10(4):363–377, September 2006. doi:10.1109/TIT.1964.1053689.

31 Tsvi Kopelowitz and Ely Porat. A simple algorithm for approximating the text-to-pattern Hamming distance. In *Proceedings of the SIAM Symposium on Simplicity in Algorithms*, volume 61 of *OASICS*, pages 10:1–10:5, 2018. doi:10.4230/OASIcs.SOSA.2018.10.

32 Kasper Green Larsen. Constructive discrepancy minimization with hereditary L2 guarantees. In *Proceedings of the International Symposium on Theoretical Aspects of Computer Science*, STACS, pages 48:1–48:13, 2019. doi:10.4230/LIPIcs.STACS.2019.48.

33 Shachar Lovett and Raghu Meka. Constructive discrepancy minimization by walking on the edges. *SIAM Journal on Computing*, 44(5):1573–1582, 2015. doi:10.1137/130929400.

34 Shan Muthukrishnan. New results and open problems related to non-standard stringology. In *Proceedings of the Annual Symposium on Combinatorial Pattern Matching*, CPM, pages 298–317, 1995. doi:10.1007/3-540-60044-2_50.

35 Shan Muthukrishnan and Krishna Palem. Non-standard stringology: Algorithms and complexity. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, STOC, pages 770–779. ACM, 1994. doi:10.1145/195058.195457.

36 Shan Muthukrishnan and Hariharan Ramesh. String matching under a general matching relation. *Information and Computation*, 122(1):140–148, 1995. doi:10.1007/3-540-56287-7_118.

37 Gonzalo Navarro. NR-grep: A fast and flexible pattern-matching tool. *Softw. Pract. Exper.*, 31(13):1265–1312, October 2001. doi:10.1002/spe.411.

38 Peng Zhang and Mikhail J. Atallah. On approximate pattern matching with thresholds. *Information Processing Letters*, 123:21–26, 2017. doi:10.1016/j.ipl.2017.03.001.