Pattern Masking for Dictionary Matching

Panagiotis Charalampopoulos
The Interdisciplinary Center Herzliya, Israel

Huiping Chen
King's College London, UK

Peter Christen
Australian National University, Canberra, Australia

Grigorios Loukides
King's College London, UK

Nadia Pisanti
University of Pisa, Italy

Solon P. Pissis
CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands

Jakub Radoszewski
University of Warsaw, Poland

Abstract

Data masking is a common technique for sanitizing sensitive data maintained in database systems, and it is also becoming increasingly important in various application areas, such as in record linkage of personal data. This work formalizes the Pattern Masking for Dictionary Matching (PMDM) problem. In PMDM, we are given a dictionary $\mathcal{D}$ of d strings, each of length ℓ, a query string q of length ℓ, and a positive integer z, and we are asked to compute a smallest set $K \subseteq \{1, \ldots, \ell\}$, so that if $q[i]$ is replaced by a wildcard for all $i \in K$, then q matches at least z strings from $\mathcal{D}$. Solving PMDM allows providing data utility guarantees as opposed to existing approaches.

We first show, through a reduction from the well-known k-Clique problem, that a decision version of the PMDM problem is NP-complete, even for strings over a binary alphabet. We thus approach the problem from a more practical perspective. We show a combinatorial $\mathcal{O}((d\ell)^{|K|/3} + d\ell)$-time and $\mathcal{O}(d\ell)$-space algorithm for PMDM for $|K| = \mathcal{O}(1)$. In fact, we show that we cannot hope for a faster combinatorial algorithm, unless the combinatorial k-Clique hypothesis fails [Abboud et al., SIAM J. Comput. 2018; Lincoln et al., SODA 2018]. We also generalize this algorithm for the problem of masking multiple query strings simultaneously so that every string has at least z matches in $\mathcal{D}$.

Note that PMDM can be viewed as a generalization of the decision version of the dictionary matching with mismatches problem: by querying a PMDM data structure with string q and $z = 1$, one obtains the minimal number of mismatches of q with any string from $\mathcal{D}$. The query time or space of all known data structures for the *more restricted* problem of dictionary matching with at most k mismatches incurs some exponential factor with respect to k. A simple exact algorithm for PMDM runs in time $\mathcal{O}(2^\ell d)$. We present a data structure for PMDM that answers queries over $\mathcal{D}$ in time $\mathcal{O}(2^{\ell/2}(2^{\ell/2} + \tau)\ell)$ and requires space $\mathcal{O}(2^\ell d^2/\tau^2 + 2^{\ell/2} d)$, for any parameter $\tau \in [1, d]$.

We complement our results by showing a two-way polynomial-time reduction between PMDM and the Minimum Union problem [Chlamtáč et al., SODA 2017]. This gives a polynomial-time $\mathcal{O}(d^{1/4+\epsilon})$-approximation algorithm for PMDM, which is tight under a plausible complexity conjecture.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases string algorithms, dictionary matching, wildcards, record linkage, query term dropping

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2021.65

Related Version *Extended Version*: `https://arxiv.org/abs/2006.16137`

Funding This paper is part of the PANGAIA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 872539. This paper is also part of the ALPACA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 956229.
Panagiotis Charalampopoulos: Supported by the Israel Science Foundation grant 592/17.
Huiping Chen: Supported by a CSC Scholarship.
Grigorios Loukides: Supported in part by the Leverhulme Trust RPG-2019-399 project.
Nadia Pisanti: Supported by the University of Pisa under the "PRA – Progetti di Ricerca di Ateneo" (Institutional Research Grants) – Project no. PRA_2020-2021_26.
Jakub Radoszewski: Supported by the Polish National Science Center, grant number 2018/31/D/ST6/03991.

1 Introduction

Let us start with a true incident to illustrate the essence of the computational problem formalized in this work. In the Netherlands, water companies bill the non-drinking and drinking water separately. The 6th author of this paper had direct debit for the former but not for the latter. When he tried to set up the direct debit for the latter, he received the following masked message by the company:

```
Is this you?
Initial: S.  Name: P****s  E-mail address: s******13@g***l.com
Bank account number: NL10RABO********11.
```

The rationale of the data masking is: the client should be able to identify themselves, to help the companies *link* the client's profiles, but not infer the identity of any other client, so that clients' privacy is preserved. Thus, the masked version of the data is required to conceal as few symbols as possible but correspond to a sufficient number of other clients.

This requirement can be formalized as the Pattern Masking for Dictionary Matching (PMDM) problem: Given a dictionary $\mathcal{D}$ of d strings, each of length ℓ, a query string q of length ℓ, and a positive integer z, PMDM asks to compute a smallest set $K \subseteq \{1, \dots, \ell\}$, so that if $q[i]$, for all $i \in K$, is replaced by a wildcard, q matches at least z strings from $\mathcal{D}$. The PMDM problem applies data masking, a common operation to sanitize personal data maintained in database systems [1, 26, 56]. In particular, PMDM lies at the heart of record linkage of databases containing personal data [23, 39, 40, 52, 59, 61], which is the main application we consider in this work.

Record linkage is the task of identifying records that refer to the same entities across databases, in situations where no entity identifiers are available in these databases [22, 32, 47]. This task is of high importance in various application domains featuring personal data, ranging from the health sector and social science research, to national statistics and crime and fraud detection [23, 36]. In a typical setting, the task is to link two databases that contain names or other attributes, known collectively as quasi-identifiers (QIDs) [60]. The similarity between each pair of records (a record from one of the databases and a record from the other) is calculated with respect to their values in QIDs, and then all compared record pairs are classified into matches (the pair is assumed to refer to the same person), non-matches (the two records in the pair are assumed to refer to different people), and potential matches (no decision about whether the pair is a match or non-match can be made) [22, 32]. Unfortunately, potential matches happen quite often [9]. A common approach [52, 59] to deal with potential matches is to conduct a manual clerical review, where a domain expert looks at the attribute values in record pairs and then makes a manual match or non-match decision. At the same time, to comply with policies and legislation, one needs to prevent domain experts from inferring the identity of the people represented in the manually assessed record pairs [52]. The challenge is to achieve desired data protection/utility guarantees; i.e. enabling a domain expert to make good decisions without inferring peoples' identities.

To address this challenge, we can solve PMDM twice, for a potential match (q_1, q_2). The first time we use as input the query string q_1 and a reference dictionary (database) $\mathcal{D}$ containing personal records from a sufficiently large population (typically, much larger than the databases to be linked). The second time, we use as input q_2 instead of q_1. Since each masked q derived by solving PMDM matches at least z records in $\mathcal{D}$, the domain expert would need to distinguish between at least z individuals in $\mathcal{D}$ to be able to infer the identity of the individual corresponding to the masked string. The underlying assumption is that $\mathcal{D}$ contains one record per individual. Also, some wildcards from one masked string can be superimposed on another to ensure that the expert does not gain more knowledge from combining the two strings, and the resulting strings would still match at least z records in $\mathcal{D}$. Thus, by solving PMDM in this setting, we provide privacy guarantees alike z-map [57]; a variant of the well-studied z-anonymity [54] privacy model.[1] In z-map, each record of a dataset must match at least z records in a reference dataset, from which the dataset is derived. In our setting, we consider a pattern that is not necessarily contained in the reference dataset. Offering such privacy is desirable in real record linkage systems where databases containing personal data are being linked [23, 40, 61]. On the other hand, since each masked q contains the minimum number of wildcards, the domain expert is still able to use the masked q to meaningfully classify a record pair as a match or as a non-match. Offering such utility is again desirable in record linkage systems [52]. Record linkage is an important application for our techniques, because no existing approach can provide privacy and utility guarantees when releasing linkage results to domain experts [41]. In particular, existing approaches [40, 41] recognize the need to offer privacy by preventing the domain expert from distinguishing between a small number of individuals, but they provide *no algorithm* for offering such privacy, let alone an algorithm offering utility guarantees as we do.

A secondary application where PMDM is of importance is *query term dropping*, an information retrieval task that seeks to drop keywords (terms) from a query, so that the remaining keywords retrieve a sufficiently large number of documents. This task is performed by search engines, such as Google [8], and by e-commerce platforms such as e-Bay [42], to improve users' experience [29, 58] by making sufficiently many search results available to users. For example, e-Bay applies query term dropping, removing one term, in our test query:

```
Query: vacuum database cleaner
Query results: 0 results found for vacuum database cleaner
               42 results found for vacuum cleaner
```

We could perform query term dropping by solving PMDM in a setting where strings in a dictionary correspond to document terms and a query string corresponds to a user's query. Then, we provide the user with the masked query, after removing all wildcards, and with its matching strings from the dictionary. Query term dropping is a relevant application for our techniques, because existing techniques [58] do not minimize the number of dropped terms. Rather, they drop keywords randomly, which may unnecessarily shorten the query,

[1] The notation used for such privacy models is generally k instead of z, e.g. k-anonymity [55, 57].

or drop keywords based on custom rules, which is not sufficiently generic to deal with all queries. More generally, our techniques can be applied to drop terms from any top-z database query [33] to ensure there are z results in the query answer.

Related Algorithmic Work. Let us denote the wildcard symbol by $\star$ and provide a brief overview of works related to PMDM, the main problem considered in this paper.

- *Partial Match*: Given a dictionary $\mathcal{D}$ of d strings over an alphabet $\Sigma = \{0, 1\}$, each of length ℓ, and a string q over $\Sigma \sqcup \{\star\}$ of length ℓ, the problem asks whether q matches any string from $\mathcal{D}$. This is a well-studied problem [14, 18, 35, 46, 50, 51, 53]. Patrascu [50] showed that any data structure for the Partial Match problem with cell-probe complexity t must use space $2^{\Omega(\ell/t)}$, assuming the word size is $\mathcal{O}(d^{1-\epsilon}/t)$, for any constant $\epsilon > 0$. The key difference to PMDM is that the wildcard positions in the query strings are fixed.
- *Dictionary Matching with k-errors*: A similar line of research to that of Partial Match has been conducted under the Hamming and edit distances, where, in this case, k is the maximum allowed distance between the query string and a dictionary string [11, 12, 15, 17, 25, 64]. The structure of Dictionary Matching with k-errors is very similar to Partial Match as each wildcard in the query string gives $|\Sigma|$ possibilities for the corresponding symbol in the dictionary strings. On the other hand, in Partial Match the wildcard positions are fixed. The PMDM problem is a generalization of the decision version of the Dictionary Matching with k-errors problem (under Hamming distance): by querying a data structure for PMDM with string q and $z = 1$, one obtains the minimum number of mismatches of q with any string from $\mathcal{D}$, which suffices to answer the decision version of the Dictionary Matching with k-errors problem. The query time or space of all known data structures for Dictionary Matching with k-mismatches incurs some exponential factor with respect to k. In [24], Cohen-Addad et al. showed that, in the pointer machine model, for the reporting version of the problem, one cannot avoid exponential dependency on k either in the space or in the query time. In the word-RAM model, Rubinstein showed that, conditional on the Strong Exponential Time Hypothesis [16], any data structure that can be constructed in time polynomial in the total size $||\mathcal{D}||$ of the strings in the dictionary cannot answer queries in time strongly sublinear in $||\mathcal{D}||$.

We next provide a brief overview of other algorithmic works related to PMDM.

- *Dictionary Matching with k-wildcards*: Given a dictionary $\mathcal{D}$ of total size N over an alphabet Σ and a query string q of length ℓ over $\Sigma \sqcup \{\star\}$ with up to k wildcards, the problem asks for the set of matches of q in $\mathcal{D}$. This is essentially a parameterized variant of the Partial Match problem. The seminal paper of Cole et al. [25] proposed a data structure occupying $\mathcal{O}(N \log^k N)$ space allowing for $\mathcal{O}(\ell + 2^k \log\log N + |\mathsf{output}|)$-time querying. This data structure is based on recursively computing a heavy-light decomposition of the suffix tree and copying the subtrees hanging off light children. Generalizations and slight improvements have been proposed in [13], [43], and [28]. In [13] the authors also proposed an alternative data structure that instead of a $\log^k N$ factor in the space complexity has a multiplicative $|\Sigma|^{k^2}$ factor. Nearly-linear-sized data structures that essentially try all different combinations of letters in the place of wildcards and hence incur a $|\Sigma|^k$ factor in the query time have been proposed in [13, 44]. On the lower bound side, Afshani and Nielsen [3] showed that, in the pointer machine model, essentially any data structure for the problem in scope must have exponential dependency on k in either the space or the query time, explaining the barriers hit by the existing approaches.
- *Enumerating Motifs with k-wildcards*: Given an input string s of length n over an alphabet Σ and positive integers k and z, this problem asks to enumerate all motifs over $\Sigma \sqcup \{\star\}$ with up to k wildcards that occur at least z times in s. As the size of the output is exponential in k, the enumeration problem has such a lower bound. Several approaches

exist for efficient motif enumeration, all aimed at reducing the impact of the output's size: efficient indexing to minimise the output delay [7, 30]; exploiting a hierarchy of wildcards positions according to the number of occurrences [10]; or defining a subset of motifs of fixed-parameter tractable size (in k or z) that can generate all the others [5, 48, 49].

Our Contributions. We consider the word-RAM model of computations with w-bit machine words, where $w = \Omega(\log(d\ell))$, for stating our results. We make the following contributions:

1. (Section 3) A reduction from the k-Clique problem to a decision version of the PMDM problem, which implies that PMDM is NP-hard, even for strings over a binary alphabet.
2. (Section 4) A combinatorial $\mathcal{O}((d\ell)^{k/3} + d\ell)$-time and $\mathcal{O}(d\ell)$-space algorithm for PMDM if $k = |K| = \mathcal{O}(1)$, which is optimal if the combinatorial k-Clique hypothesis is true.
3. (Section 5) We consider a generalized version of PMDM, referred to as MPMDM: we are given a collection $\mathcal{M}$ of m query strings (instead of one query string) and we are asked to compute a smallest set K so that, for every q from $\mathcal{M}$, if $q[i]$, for all $i \in K$, is replaced by a wildcard, then q matches at least z strings from dictionary $\mathcal{D}$. We show an $\mathcal{O}((d\ell)^{k/3} z^{m-1} + d\ell)$-time algorithm for MPMDM, for $k = |K| = \mathcal{O}(1)$ and $m = \mathcal{O}(1)$.
4. (Section 6) A data structure for PMDM that answers queries over $\mathcal{D}$ in $\mathcal{O}(2^{\ell/2}(2^{\ell/2} + \tau)\ell)$ time and requires space $\mathcal{O}(2^{\ell} d^2/\tau^2 + 2^{\ell/2} d)$, for any parameter $\tau \in [1, d]$.
5. (Section 7) A polynomial-time $\mathcal{O}(d^{1/4+\epsilon})$-approximation algorithm for PMDM, which we show to be tight under a plausible complexity conjecture.

Let us now discuss why our data structure results (Section 6) cannot be directly obtained using data structures for Dictionary Matching with k-wildcards. Conceivably, one could construct such a data structure, and then iterate over all subsets of $\{1, \ldots, \ell\}$, querying for the masked string. Existing data structures for dictionary matching with wildcards (cf. [13, Table 1], [44], and [28]), that allow querying a pattern with at most ℓ wildcards, have

(a) either $\Omega(\min\{\sigma^{\ell}, d\})$ query time, thus yielding $\Omega(2^{\ell} \cdot \min\{\sigma^{\ell}, d\})$ query time for our problem, and space $\Omega(d\ell)$, a trade-off dominated by the Small-ℓ algorithm (cf. our Table 1);
(b) or $\Omega(\ell)$ query time, thus yielding $\Omega(2^{\ell}\ell)$ query time for our problem, and $\Omega(d\ell \log^{\ell} \log(d\ell))$ space, a trade-off dominated by the DS Simple (cf. our Table 1).

2 Definitions and Notation

Strings. An *alphabet* Σ is a finite nonempty set whose elements are called *letters*. We assume throughout an integer alphabet $\Sigma = [1, |\Sigma|]$. Let $x = x[1] \cdots x[n]$ be a *string* of length $|x| = n$ over Σ. For two indices $1 \le i \le j \le n$, $x[i\,..\,j] = x[i] \cdots x[j]$ is the *substring* of x that starts at position i and ends at position j of x. By ε we denote the *empty string* of length 0. A *prefix* of x is a substring of x of the form $x[1\,..\,j]$, and a *suffix* of x is a substring of x of the form $x[i\,..\,n]$. A *dictionary* is a collection of strings. We also consider alphabet $\Sigma_{\star} = \Sigma \sqcup \{\star\}$, where $\star$ is a *wildcard* letter that is not in Σ and *matches* all letters from $\Sigma_{\star}$. Then, given a string x over $\Sigma_{\star}$ and a string y over Σ with $|x| = |y|$, we say that x *matches* y if and only if $x[i] = y[i]$ or $x[i] = \star$, for all $1 \le i \le |x|$. Given a string x of length n and a set $S \subseteq \{1, \ldots, n\}$, we denote by $x_S = x \otimes S$ the string obtained by first setting $x_S = x$ and then $x_S[i] = \star$, for all $i \in S$. We then say that x is *masked* by S.

The main problem considered in this paper is the following.

Pattern Masking for Dictionary Matching (PMDM)
Input: A dictionary $\mathcal{D}$ of d strings, each of length ℓ, a string q of length ℓ, and a positive integer z.
Output: A smallest set $K \subseteq \{1, \ldots, \ell\}$ such that $q_K = q \otimes K$ matches at least z strings from $\mathcal{D}$.

We refer to the problem of computing only the size k of a smallest set K as PMDM-Size. We also consider the data structure variant of the PMDM problem in which $\mathcal{D}$ is given for preprocessing, and q, z queries are to be answered on-line. Throughout, we assume that $k \geq 1$ as the case $k = 0$ corresponds to the well-studied dictionary matching problem for which there exists a classic optimal solution [4]. We further assume $z \leq d$; otherwise the PMDM has trivially no solution. In what follows, we use N to denote $d\ell$.

Tries. Let $\mathcal{M}$ be a finite set containing $m > 0$ strings over Σ. The *trie* of $\mathcal{M}$, denoted by $\mathcal{R}(\mathcal{M})$, contains a node for every distinct prefix of a string in $\mathcal{M}$; the root node is ε; the set of leaf nodes is $\mathcal{M}$; and edges are of the form $(u, \alpha, u\alpha)$, where u and $u\alpha$ are nodes and $\alpha \in \Sigma$ is the label. The *compacted trie* of $\mathcal{M}$, denoted by $\mathcal{T}(\mathcal{M})$, contains the root, the branching nodes, and the leaf nodes of $\mathcal{R}(\mathcal{M})$. Each maximal branchless path segment from $\mathcal{R}(\mathcal{M})$ is replaced by a single edge, and a fragment of a string $M \in \mathcal{M}$ is used to represent the label of this edge in $\mathcal{O}(1)$ space. The size of $\mathcal{T}(\mathcal{M})$ is thus $\mathcal{O}(m)$. The most well-known example of a compacted trie is the suffix tree of a string: the compacted trie of all the suffixes of the string [62]. To access the children of a trie node by the first letter of their edge label in $\mathcal{O}(1)$ time we use perfect hashing [27]. In this case, the claimed complexities hold *with high probability* (w.h.p., for short), that is, with probability at least $1 - N^{-c}$ (recall that $N = d\ell$), where $c > 0$ is a constant fixed at construction time. Assuming that the children of every trie node are sorted by the first letters of their edge labels, randomization can be avoided at the expense of a $\log |\Sigma|$ factor incurred by binary searching for the appropriate child.

3 PMDM-Size is NP-hard

We show that the following decision version of PMDM-Size is NP-complete.

k-PMDM
Input: A dictionary $\mathcal{D}$ of d strings, each of length ℓ, a string q of length ℓ, and positive integers $z \leq d$ and $k \leq \ell$.
Output: Is there a set $K \subseteq \{1, \ldots, \ell\}$ of size k, such that $q_K = q \otimes K$ matches at least z strings from $\mathcal{D}$?

Our reduction is from the well-known NP-complete k-Clique problem [37]: Given an undirected graph G on n nodes and a positive integer k, decide whether G contains a clique of size k (a *clique* is a subset of the nodes of G that are pairwise adjacent).

▶ **Theorem 1.** *Any instance of the k-Clique problem for a graph with n nodes and m edges can be reduced in $\mathcal{O}(nm)$ time to a k-PMDM instance with $\ell = n$, $d = m$ and $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.*

Proof. Let $G = (V, E)$ be an undirected graph on $n = |V|$ nodes numbered 1 through n, in which we are looking for a clique of size k. We reduce k-Clique to k-PMDM as follows. Consider the alphabet $\{\mathtt{a}, \mathtt{b}\}$. Set $q = \mathtt{a}^n$, and for every edge $(u, v) \in E$ such that $u < v$, add string $\mathtt{a}^{u-1}\mathtt{b}\mathtt{a}^{v-u-1}\mathtt{b}\mathtt{a}^{n-v}$ to $\mathcal{D}$. Set $z = k(k-1)/2$. Then G contains a clique of size k, if and only if k-PMDM returns a positive answer. This can be seen by the fact that cliques of

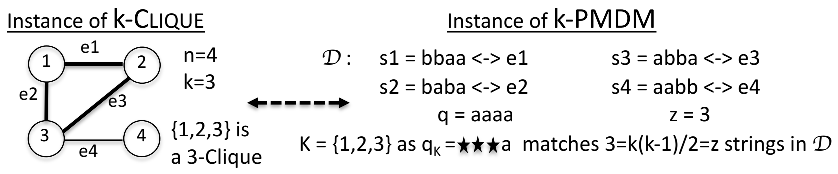

Figure 1 An example of the reduction from k-Clique to k-PMDM. The solution for both is $\{1, 2, 3\}$ as shown. Note that, for $k = 4$, the instance of 4-PMDM would need $z = 6$ matches; neither this many matches can be found in $\mathcal{D}$ nor a 4-clique can be found in the graph.

size k in G are in one-to-one correspondence with subsets $K \subseteq \{1, \ldots, n\}$ of size k for which q_K matches z strings from $\mathcal{D}$: the elements of K correspond to the nodes of a clique and the z strings correspond to its edges. k-PMDM is clearly in NP and the result follows. ◀

An example of the reduction from k-Clique to k-PMDM is shown in Figure 1.

▶ **Corollary 2.** *k-PMDM is NP-complete for strings over a binary alphabet.*

Our reduction (Theorem 1) shows that solving k-PMDM efficiently even for strings over a binary alphabet would imply a breakthrough for the k-Clique problem for which it is known that, in general, no fixed-parameter tractable algorithm with respect to parameter k exists unless the Exponential Time Hypothesis (ETH) fails [19, 34]. That is, k-Clique has no $f(k)n^{o(k)}$ time algorithm, and is thus W[1]-complete (again, under the ETH hypothesis). On the upper bound side, k-Clique can be trivially solved in $\mathcal{O}(n^k)$ time (enumerating all subsets of nodes of size k), and this can be improved to $\mathcal{O}(n^{\omega k/3})$ time for k divisible by 3 using square matrices multiplication (ω is the exponent of square matrix multiplication). However, for general $k \geq 3$ and any constant $\epsilon > 0$, the k-Clique hypothesis states that there is no $\mathcal{O}(n^{(\omega/3-\epsilon)k})$-time algorithm and no combinatorial $\mathcal{O}(n^{(1-\epsilon)k})$-time algorithm [2, 45, 63].

In particular, assuming that the k-Clique hypothesis is true, due to Theorem 1, we cannot hope to devise a combinatorial algorithm for k-PMDM requiring $\mathcal{O}((d\ell)^{(1-\epsilon)k/3})$ time, for any $k \geq 3$ and $\epsilon > 0$, since $d\ell = nm$. In Section 4, we show a combinatorial $\mathcal{O}((d\ell)^{k/3})$-time algorithm, for constant $k \geq 3$, for the optimization version of k-PMDM (seeking to maximize the matches), which can then be trivially applied to solve k-PMDM in the same time complexity, thus matching the above conditional lower bound.

Given an undirected graph G, an *independent set* is a subset of nodes of G such that no two distinct nodes of the subset are adjacent. Let us note that the problem of computing a maximum clique in a graph G, which is equivalent to that of computing the maximum independent set in the complement of G, cannot be $n^{1-\epsilon}$-approximated in polynomial time, for any $\epsilon > 0$, unless P = NP [31, 65].

Any algorithm solving PMDM-Size can be trivially applied to solve k-PMDM.

▶ **Corollary 3.** *PMDM-Size is NP-hard for strings over a binary alphabet.*

4 Exact Algorithms for a Bounded Number k of Wildcards

We consider the following problem, which we solve by exact algorithms.

Heaviest k-PMDM
Input: A dictionary $\mathcal{D}$ of d strings, each of length ℓ, a string q of length ℓ, and a positive integer $k \leq \ell$.
Output: A set $K \subseteq \{1, \ldots, \ell\}$ of size k such that $q_K = q \otimes K$ matches the maximum number of strings in $\mathcal{D}$.

We will show the following result, which we will employ to solve the PMDM problem.

▶ **Theorem 4.** *Heaviest k-PMDM for $k = \mathcal{O}(1)$ can be solved in $\mathcal{O}(N + \min\{N^{k/3}, \ell^k\})$ time, where $N = d\ell$.*

A *hypergraph* H is a pair (V, E), where V is the set of nodes of H and E is a set of non-empty subsets of V, called *hyperedges* – in order to simplify terminology we will simply call them edges. Hypergraphs are a generalization of graphs in the sense that an edge can connect more than two nodes. Recall that the size of an edge is the number of nodes it contains. The *rank* of H, denoted by $r(H)$, is the maximum size of an edge of H.

We refer to a hypergraph $H \times K = (K, \{e : e \in E, e \subseteq K\})$, where K is a subset of V, as a $|K|$-*section*. $H \times K$ is the hypergraph induced by H on the nodes of K, and it contains all edges of H whose elements are all in K. A hypergraph is *weighted* when each of its edges is associated with a weight. We define *the weight* of a weighted hypergraph as the sum of the weights of all of its edges. In what follows, we also refer to weights of nodes for conceptual clarity; this is equivalent to having a singleton edge of equal weight consisting of that node.

We define the following auxiliary problem on hypergraphs (see also [20]).

Heaviest k-Section
Input: A weighted hypergraph $H = (V, E)$, with E given as a list, and an integer $k > 0$.
Output: A subset K of size k of V such that $H \times K$ has maximum weight.

When $k = \mathcal{O}(1)$, we preprocess the edges of H as follows in order to have $\mathcal{O}(1)$-time access to any queried edge. We represent each edge as a string, whose letters correspond to its elements in increasing order. Then, we sort all such strings lexicographically using radix sort in $\mathcal{O}(|E|)$ time and construct a trie over them. An edge can then be accessed in $\mathcal{O}(k \log k) = \mathcal{O}(1)$ time by a forward search starting from the root node of the trie.

A polynomial-time $\mathcal{O}(n^{0.697831+\epsilon})$-approximation for Heaviest k-Section, for any $\epsilon > 0$, for the case when all hyperedges of H have size at most 3 was shown in [20] (see also [6]).

Two remarks are in place. First, we can focus on edges of size up to k as larger edges cannot, by definition, exist in any k-section. Second, Heaviest k-Section is a generalization of the problem of deciding whether a (c, k)-hyperclique (i.e. a set of k nodes whose subsets of size c are all in E) exists in a graph, which in turn is a generalization of k-Clique. Unlike k-Clique, the (c, k)-hyperclique problem is not known to benefit from fast matrix multiplication in general; see [45] for a discussion on its hardness.

▶ **Lemma 5.** *Heaviest k-PMDM can be reduced to Heaviest k-Section for a hypergraph with ℓ nodes and d edges in $\mathcal{O}(N)$ time, where $N = d\ell$.*

Proof. We first compute the set M_s of positions of mismatches of q with each string $s \in \mathcal{D}$. We ignore strings from $\mathcal{D}$ that match q exactly, as they will match q after changing any set of letters of q to wildcards. This requires $\mathcal{O}(d\ell) = \mathcal{O}(N)$ time in total.

Let us consider an empty hypergraph (i.e. with no edges) H on ℓ nodes, numbered 1 through ℓ. Then, for each string $s \in \mathcal{D}$, we add M_s to the edge-set of H if $|M_s| \leq k$; if this edge already exists, we simply increment its weight by 1.

We set the parameter k of Heaviest k-Section to the parameter k of Heaviest k-PMDM. We now observe that for $K \subseteq V$ with $|K| = k$, the weight of $H \times K$ is equal to the number of strings that would match q after replacing with wildcards the k letters of q at the positions corresponding to elements of K. The statement follows. ◀

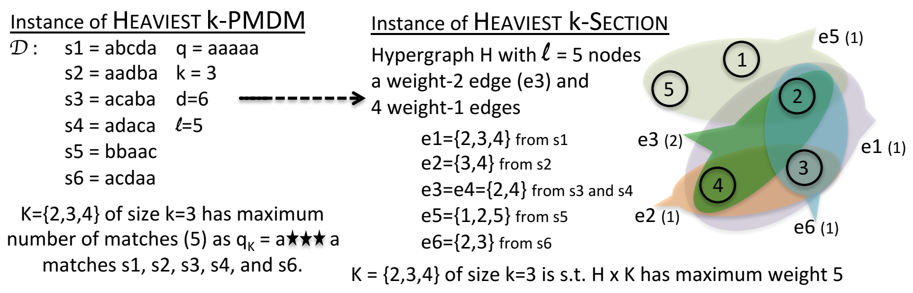
Instance of HEAVIEST k-PMDM
𝒟: s1 = abcda q = aaaaa
s2 = aadba k = 3
s3 = acaba d=6
s4 = adaca ℓ=5
s5 = bbaac
s6 = acdaa
K={2,3,4} of size k=3 has maximum number of matches (5) as q_K = a★★★ a matches s1, s2, s3, s4, and s6.
Instance of HEAVIEST k-SECTION
Hypergraph H with ℓ = 5 nodes a weight-2 edge (e3) and 4 weight-1 edges
e1={2,3,4} from s1
e2={3,4} from s2
e3=e4={2,4} from s3 and s4
e5={1,2,5} from s5
e6={2,3} from s6
K = {2,3,4} of size k=3 is s.t. H x K has maximum weight 5
e5 (1)
e1 (1)
e3 (2)
e2 (1)
e6 (1)
1
2
3
4
5

Figure 2 An example of the reduction from HEAVIEST k-PMDM to HEAVIEST k-SECTION. The solutions are at the bottom. Each edge has its weight in brackets and the total weight is $d = 6$.

An example of the reduction in Lemma 5 is shown in Figure 2.

The next lemma gives a straightforward solution to HEAVIEST k-SECTION. It is analogous to algorithm SMALL-ℓ, presented in Section 6, but without the optimization in computing sums of weights over subsets. It implies a linear-time algorithm for HEAVIEST 1-SECTION.

▶ **Lemma 6.** *HEAVIEST k-SECTION, for any constant k, can be solved in $\mathcal{O}(|V|^k + |E|)$ time and $\mathcal{O}(|V| + |E|)$ space.*

Proof. For every subset $K \subseteq V$ of size at most k, we sum the weights of all edges corresponding to its subsets. There are $\binom{|V|}{k} = \mathcal{O}(|V|^k)$ choices for $|K|$, each having $2^k - 1$ non-empty subsets: for every subset, we can access the corresponding edge (if it exists) in $\mathcal{O}(1)$ time. ◀

We next show that for the cases $k = 2$ and $k = 3$, there exist more efficient solutions. In particular, we provide a linear-time algorithm for HEAVIEST 2-SECTION.

▶ **Lemma 7.** *HEAVIEST 2-SECTION can be solved in $\mathcal{O}(|V| + |E|)$ time.*

Proof. Let K be a set of nodes of size 2 such that $H \times K$ has maximum weight. We decompose the problem in two cases. For each of the cases, we give an algorithm that considers several 2-sections such that the heaviest of them has weight equal to that of $H \times K$.

Case 1. There is an edge $e = K$ in E. For each edge $e \in E$ of size 2, i.e. edge in the classic sense, we compute the sum of its weight and the weights of the nodes that it is incident to. This step requires $\mathcal{O}(|E|)$ time.

Case 2. There is no edge equal to K in E. We compute $H \times \{v_1, v_2\}$, where v_1, v_2 are the two nodes with maximum weight, i.e. max and second-max. This step takes $\mathcal{O}(|V|)$ time.

In the end, we return the heaviest 2-section among those returned by the algorithms for the two cases, breaking ties arbitrarily. ◀

We next show that for $k = 3$ the result of Lemma 6 can be improved when $|E| = o(|V|^2)$.

▶ **Lemma 8.** *HEAVIEST 3-SECTION can be solved in time $\mathcal{O}(|V| \cdot |E|)$ using $\mathcal{O}(|V| + |E|)$ space.*

Proof. Let K be a set of nodes of size 3 such that $H \times K$ has maximum weight. We decompose the problem into the following three cases.

Case 1. There is an edge $e = K$ in E. We go through each edge $e \in E$ of size 3 and compute the weight of $H \times e$ in $\mathcal{O}(1)$ time. This takes $\mathcal{O}(|E|)$ time in total. Let the edge yielding the maximum weight be $e_{\max}$.

Case 2. There is no edge of size larger than one in $H \times K$. We compute $H \times \{v_1, v_2, v_3\}$, where v_1, v_2, v_3 are the three nodes with maximum weight, i.e. max, second-max and third-max. This step takes $\mathcal{O}(|V|)$ time.

Case 3. There is an edge of size 2 in $H \times K$. We can pick an edge e of size 2 from E in $\mathcal{O}(|E|)$ ways and a node v from V in $\mathcal{O}(|V|)$ ways. We compute the weight of $H \times (e \cup \{v\})$ for all such pairs. Let the pair yielding maximum weight be (e', u').

Finally, the maximum weight of $H \times K'$ for $K' \in \{\, e_{\max}, \{v_1, v_2, v_3\}, e' \cup \{u'\} \,\}$ is equal to the weight of $H \times K$, breaking ties arbitrarily. ◀

▶ **Lemma 9.** *Heaviest k-Section for an arbitrarily large constant $k \geq 4$ can be solved in time $\mathcal{O}((|V| \cdot |E|)^{k/3})$ using $\mathcal{O}(|V| + |E|)$ space.*

Proof. If $|E| > |V|^2$, then the simple algorithm of Lemma 6 solves the problem in time

$$\mathcal{O}(|V|^k + |E|) = \mathcal{O}(|V|^{k/3}(|V|^2)^{k/3} + |E|) = \mathcal{O}((|V| \cdot |E|)^{k/3})$$

and linear space. We can thus henceforth assume that $|E| \leq |V|^2$.

Let K be a set of nodes of size at most k such that $H \times K$ has maximum weight. If $H \times K$ contains isolated nodes (i.e. nodes not contained in any edge), they can be safely deleted without altering the result. We can thus assume that $H \times K$ does not contain isolated nodes, and that $|V| \leq k|E|$ since otherwise the hypergraph H would contain isolated nodes.

We first consider the case that the rank $r(H \times K) > 1$, i.e. there is an edge of $H \times K$ of size at least 2. We design a branching algorithm that constructs several candidate sets; the ones with maximum weight will have weight equal to that of $H \times K$. We will construct a set of nodes X, starting with $X := \emptyset$. For each set X that we process, let Z_X be the superset of X of size at most k such that $H \times Z_X$ has maximum weight. We have the following two cases:

Case 1. There is an edge e in $H \times Z_X$ that contains at least two nodes from $Z_X \setminus X$. To account for this case, we select every possible such edge e, set $X := X \cup e$, and continue the branching algorithm.

Case 2. Each edge in $H \times Z_X$ contains at most one node from $Z_X \setminus X$. In this case we conclude the branching algorithm as follows. For every node $v \in V \setminus X$ we compute its weight as the total weight of edges $Y \cup \{v\} \in E$ for $Y \subseteq X$ in $\mathcal{O}(2^k) = \mathcal{O}(1)$ time. Finally, in $\mathcal{O}(|V|k) = \mathcal{O}(|V|)$ time we select $k - |X|$ nodes with largest weights and insert them into X. The total time complexity of this step is $\mathcal{O}(|V|)$. This case also works if $|X| = k$ and then its time complexity is only $\mathcal{O}(1)$.

The correctness of this branching algorithm follows from an easy induction, showing that at every level of the branching tree there is a subset of K.

Let us now analyze the time complexity of this branching algorithm. Each branching in Case 1 takes $\mathcal{O}(|E|)$ time and increases the size of $|X|$ by at least 2. At every node of the branching tree we call the procedure of Case 2. It takes $\mathcal{O}(|V|)$ time if $|X| < k$.

If the procedure of Case 2 is called in a non-leaf node of the branching tree, then its $\mathcal{O}(|V|)$ running time is dominated by the $\mathcal{O}(|E|)$ time that is required for further branching since we have assumed that $|V| \leq k|E|$. Hence, it suffices to bound (a) the total time complexity of calls to the algorithm for Case 2 in leaves that correspond to sets X such that $|X| < k$ and (b) the total number of leaves that correspond to sets X such that $|X| = k$.

If k is even, (a) is bounded by $\mathcal{O}(|E|^{(k-2)/2}|V|)$ and (b) is bounded by $\mathcal{O}(|E|^{k/2})$. Hence, (b) dominates (a) and we have

$$\mathcal{O}(|E|^{k/2}) = \mathcal{O}(|E|^{k/3}|E|^{k/6}) = \mathcal{O}(|E|^{k/3}|V|^{k/3}). \tag{1}$$

If k is odd, (a) is bounded by $\mathcal{O}(|E|^{(k-1)/2}|V|)$ and (b) is bounded by $\mathcal{O}(|E|^{(k-1)/2})$, which is dominated by (a). By using (1) for $k-3$ we also have:

$$\mathcal{O}(|E|^{(k-1)/2} \cdot |V|) = \mathcal{O}(|E|^{(k-3)/2} \cdot |E| \cdot |V|) = \mathcal{O}((|E| \cdot |V|)^{(k-3)/3} \cdot |E| \cdot |V|) = \mathcal{O}((|E| \cdot |V|)^{k/3}).$$

We now consider the case that $r(H \times K) = 1$. We use the algorithm for Case 2 above that works in $\mathcal{O}(|V|)$ time, which is $\mathcal{O}(|V| \cdot |E|)$. ◀

Lemmas 5-9 imply Theorem 4, which we employ iteratively to obtain the following result.

▶ **Theorem 10.** *PMDM can be solved in time $\mathcal{O}(N + \min\{N^{k/3}, \ell^k\})$ using space $\mathcal{O}(N)$ if $k = \mathcal{O}(1)$, where $N = d\ell$.*

Proof. We apply Lemma 5 to obtain a hypergraph with $|V| = \ell$ and $|E| = d$. Starting with $k = 1$ and for growing values of k, we solve HEAVIEST k-SECTION until we obtain a solution of weight at least z, employing either only Lemma 6, or Lemmas 6, 7, 8, 9 for $k = 1, 2, 3$ and $k \geq 4$, respectively. We obtain $\mathcal{O}(N + \min\{N^{k/3}, \ell^k\})$ time and $\mathcal{O}(N)$ space. ◀

5 Exact Algorithms for a Bounded Number m of Query Strings

Recall that masking a potential match (q_1, q_2) in record linkage can be performed by solving PMDM twice and superimposing the wildcards (see Section 1). In this section, we consider the following generalized version of PMDM to perform the masking simultaneously. The advantage of this approach is that it minimizes the final number of wildcards in q_1 and q_2.

MULTIPLE PATTERN MASKING FOR DICTIONARY MATCHING (MPMDM)
Input: A dictionary $\mathcal{D}$ of d strings, each of length ℓ, a collection $\mathcal{M}$ of m strings, each of length ℓ, and a positive integer z.
Output: A smallest set $K \subseteq \{1, \ldots, \ell\}$ such that, for every q from $\mathcal{M}$, $q_K = q \otimes K$ matches at least z strings from $\mathcal{D}$.

Let $N = d\ell$. We show the following theorem.

▶ **Theorem 11.** *MPMDM can be solved in time $\mathcal{O}(N + \min\{N^{k/3} z^{m-1}, \ell^k\})$ if $k = \mathcal{O}(1)$ and $m = \mathcal{O}(1)$, where $N = d\ell$.*

We use a generalization of HEAVIEST k-SECTION in which the weights are m-tuples that are added and compared component-wise, and we aim to find a subset K such that the weight of $H \times K$ is at least $(z, \ldots, z)$. An analogue of Lemma 6 holds without any alterations, which accounts for the $\mathcal{O}(N + \ell^k)$-time algorithm. We adapt the proof of Lemma 9 as follows. The branching remains the same, but we have to tweak the final step, that is, what happens when we are in Case 2. For $m = 1$ we could simply select a number of largest weights, but for $m > 1$ multiple criteria need to be taken into consideration. All in all, the problem reduces to a variation of the classic Multiple-Choice Knapsack problem [38], which we solve using dynamic programming. Overall, we pay an additional $\mathcal{O}(z^{m-1})$ factor in the complexity of handling of Case 2, which yields the complexity of Theorem 11.

6 A Data Structure for PMDM Queries

We next show algorithms and data structures for the PMDM problem under the assumption that 2^ℓ is reasonably small. We measure space in terms of w-bit machine words, where $w = \Omega(\log(d\ell))$, and focus on showing space vs. query-time trade-offs for answering q, z PMDM queries over $\mathcal{D}$. A summary of the complexities of the data structures is shown in Table 1. Specifically, algorithm SMALL-ℓ and data structure SIMPLE are used as building blocks in the more involved data structure SPLIT underlying the following theorem.

▶ **Theorem 12.** *There exists a data structure that answers q, z PMDM queries over $\mathcal{D}$ in time $\mathcal{O}(2^{\ell/2}(2^{\ell/2} + \tau)\ell)$ w.h.p. and requires space $\mathcal{O}(2^\ell d^2/\tau^2 + 2^{\ell/2} d)$, for any $\tau \in [1, d]$.*

Table 1 Basic complexities of the data structures from Section 6.

Data structure	Space	Query time
Algorithm SMALL-ℓ	$\mathcal{O}(d\ell)$	$\mathcal{O}(2^\ell \ell + d\ell)$
DS SIMPLE	$\mathcal{O}(2^\ell d)$	$\mathcal{O}(2^\ell \ell)$
DS SPLIT, any τ	$\mathcal{O}(2^\ell d^2/\tau^2 + 2^{\ell/2} d)$	$\mathcal{O}(2^{\ell/2} \cdot (2^{\ell/2} + \tau)\ell)$
DS SPLIT for $\tau = 2^{\ell/4}\sqrt{d}$	$\mathcal{O}(2^{\ell/2} d)$	$\mathcal{O}(2^\ell \ell + 2^{3\ell/4}\sqrt{d}\ell)$

Algorithm SMALL-ℓ: $\mathcal{O}(d\ell)$ Space, $\mathcal{O}(2^\ell \ell + d\ell)$ Query Time. No data structure on top of the dictionary $\mathcal{D}$ is stored. In the query algorithm, we initialize an array A of size 2^ℓ with zeros. For an ℓ-bit vector m, by $K_m \subseteq \{1, \ldots, \ell\}$ let us denote the set of the positions of set bits of m. Now for every possible ℓ-bit vector m we want to compute the number of strings in $\mathcal{D}$ that match $q_{K_m} = q \otimes K_m$.

To this end, for every string $s \in \mathcal{D}$, we compute the set K of positions in which s and q differ. For m that satisfies $K = K_m$, we increment $A[m]$. This computation takes $\mathcal{O}(d\ell)$ time and $\mathcal{O}(1)$ extra space. Then we apply a folklore dynamic-programming-based approach to compute array B, which is defined as follows:

$$B[m] = \sum_{j \in S(m)} A[j], \text{ where } S(m) = \{j \in [1, 2^\ell] : K_j \subseteq K_m\}.$$

In other words, $B[m]$ stores the number of strings from $\mathcal{D}$ that match q_{K_m}. It takes $\mathcal{O}(\ell 2^\ell)$ time and $\mathcal{O}(2^\ell)$ extra space. Thus, overall, the (query) time required by algorithm SMALL-ℓ is $\mathcal{O}(\ell 2^\ell + d\ell)$, the data structure space is $\mathcal{O}(d\ell)$, and the extra space is $\mathcal{O}(2^\ell)$.

We first present SIMPLE, an auxiliary data structure, which we will apply later on to construct DS SPLIT, a data structure with the space/query-time trade-off of Theorem 12.

DS SIMPLE: $\mathcal{O}(2^\ell d)$ Space, $\mathcal{O}(2^\ell \ell)$ Query Time. We initialize an empty set $\mathcal{Q}$. For each possible subset of $\{1, \ldots, \ell\}$ we do the following. We mask the corresponding positions in all strings from $\mathcal{D}$ and then sort the masked strings lexicographically. By iterating over the lexicographically sorted list of the masked strings, we count how many copies of each distinct (masked) string we have in our list. We insert each such (masked) string to $\mathcal{Q}$ along with its count. After processing all 2^ℓ subsets, we construct a compacted trie for the strings in $\mathcal{Q}$; each leaf corresponds to a unique element of $\mathcal{Q}$, and stores this element's count. The total space occupied by this compacted trie is thus $\mathcal{O}(2^\ell d)$. Upon an on-line query q (of length ℓ) and z, we apply all possible 2^ℓ masks to q and read the count for each of them from the compacted trie in $\mathcal{O}(\ell)$ time per mask. Next, we show how to decrease the exponential dependency on ℓ in the space complexity when $2^\ell = o(d)$, incurring extra time in the query.

DS SPLIT: $\mathcal{O}(2^\ell d^2/\tau^2 + 2^{\ell/2} d)$ Space, $\mathcal{O}(2^{\ell/2} \cdot (2^{\ell/2} + \tau)\ell)$ Query Time, for any τ. This trade-off is relevant when $\tau = \omega(\sqrt{d})$; otherwise the DS SIMPLE is better. We split each string $p \in \mathcal{D}$ roughly in the middle, to prefix p_L and suffix p_R; specifically, $p = p_L p_R$ and $|p_L| = \lceil \ell/2 \rceil$. We create dictionaries $\mathcal{D}_L = \{p_L : p \in \mathcal{D}\}$ and $\mathcal{D}_R = \{p_R : p \in \mathcal{D}\}$. Let us now explain how to process $\mathcal{D}_L$; we process $\mathcal{D}_R$ analogously. Let $\lambda = \lceil \ell/2 \rceil$. We construct DS SIMPLE over $\mathcal{D}_L$. This requires space $\mathcal{O}(2^{\ell/2} d)$. Let τ be an input parameter, intuitively used as the minimum frequency threshold. For each of the possible 2^λ masks, we can have at most $\lfloor d/\tau \rfloor$ (masked) strings with frequency at least τ. Over all masks, we thus have at most $2^\lambda \lfloor d/\tau \rfloor$ such strings, which we call *τ-frequent*. For every pair of τ-frequent strings, one from $\mathcal{D}_L$ and one from $\mathcal{D}_R$, we store the number of occurrences of their concatenation in $\mathcal{D}$ using a compacted trie as in DS SIMPLE. This requires space $\mathcal{O}(2^\ell d^2/\tau^2)$.

			Counts of Simple	
$\mathcal{D}$	$\mathcal{D}_L$	$\mathcal{D}_R$	$\mathcal{D}_L$	$\mathcal{D}_R$
`abba`	`ab`	`ba`	`a`$\star$ 4	`b`$\star$ 2
`acba`	`ac`	`ba`	$\star$`b` 2	$\star$`a` 3
`acca`	`ac`	`ca`	$\star$`c` 2	$\cdots$
`abac`	`ab`	`ac`	$\star\star$ 4	$\star\star$ 4

Compacted trie of Split

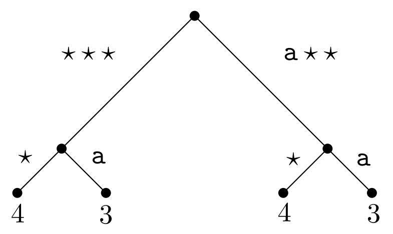

Figure 3 Let $\tau = 3$. If both q'_L and q'_R are 3-frequent (we check this using the counts of DS Simple), we read the count for $q'_L q'_R$ from the compacted trie of DS Split. If q'_L is 3-infrequent, we apply Small-ℓ on q_R and the dictionary consisting of at most $\tau = 3$ strings from $\mathcal{D}_R$ corresponding to the right halves of strings in $\mathcal{D}_L$ that match q'_L.

Consider $\mathcal{D}_L$. For each mask i and each string $p_L \in \mathcal{D}_L$, we can afford to store the list of all strings in $\mathcal{D}_L$ that match $p_L \otimes i$. Note that we have computed this information when sorting for constructing DS Simple over $\mathcal{D}_L$. This information requires space $\mathcal{O}(2^{\ell/2}d)$. Thus, DS Split requires $\mathcal{O}(2^{\ell}d^2/\tau^2 + 2^{\ell/2}d)$ space overall.

Let us now show how to answer an on-line q, z query. Let $q = q_L q_R$ with $|q_L| = \lceil \ell/2 \rceil$. We iterate over all possible 2^{ℓ} masks.

For a mask i, let $q' = q \otimes i$. We split q' into two halves, q'_L and q'_R with $q' = q'_L q'_R$ and $|q'_L| = \lceil \ell/2 \rceil$. First, we check whether each of q'_L and q'_R is τ-infrequent using the DS Simple we have constructed for $\mathcal{D}_L$ and D_R, respectively, in time $\mathcal{O}(\ell)$. We have the following two cases (inspect also Figure 3).

- If both halves are τ-frequent, we can read the frequency of their concatenation using the stored compacted trie in time $\mathcal{O}(\ell)$.
- Else, at least one of the two halves is τ-infrequent. Assume without loss of generality that q'_L is τ-infrequent. Let $\mathcal{F}$ be the dictionary consisting of at most τ strings from $\mathcal{D}_R$ that correspond to the right halves of strings in $\mathcal{D}_L$ that match q'_L. Naïvely counting how many elements of $\mathcal{F}$ match q'_R could require $\Omega(\tau\ell)$ time, and thus $\Omega(2^{\ell}\tau\ell)$ overall. Instead, we apply algorithm Small-ℓ on q_R and $\mathcal{F}$. The crucial point is that if we ever come across q'_L again (for a different mask on q), we will not need to do anything. We can maintain whether q'_L has been processed by temporarily marking the leaf corresponding to it in DS Simple for $\mathcal{D}_L$. Thus, overall, we perform the Small-ℓ algorithm $\mathcal{O}(2^{\ell/2})$ times, each time in $\mathcal{O}((2^{\ell/2} + \tau)\ell)$ time. This completes the proof of Theorem 12.

Efficient Construction. For completeness, we next show how to construct DS Split in $\mathcal{O}(d\ell \log(d\ell) + 2^{\ell}d\ell + 2^{\ell}\ell d^2/\tau^2)$ time. We preprocess $\mathcal{D}$ by sorting its letters in $\mathcal{O}(d\ell \log(d\ell))$ time. The DS Simple for $\mathcal{D}_L$ and $\mathcal{D}_R$ can then be constructed in $\mathcal{O}(2^{\ell/2}d\ell)$ time. We then create the compacted trie for pairs of τ-frequent strings. For each of the 2^{ℓ} possible masks, say i, and each string $p \in \mathcal{D}$, we split $p' = p \otimes i$ in the middle to obtain p'_L and p'_R. If both p'_L and p'_R are τ-frequent then p' will be in the set of strings for which we will construct the compacted trie for pairs of τ-frequent strings. The counts for each of those strings can be read in $\mathcal{O}(\ell)$ time from a DS Simple over $\mathcal{D}$, which we can construct in time $\mathcal{O}(2^{\ell}d\ell)$ – this data structure is then discarded. The compacted trie construction requires time $\mathcal{O}(2^{\ell}\ell d^2/\tau^2)$.

Comparison of the Data Structures. DS Simple has lower query time than algorithm Small-ℓ. However, its space complexity can be much higher. DS Split can be viewed as an intermediate option. For τ as in Table 1, it has lower query time than algorithm Small-ℓ for

$d = \omega(2^{3\ell/2})$, while keeping moderate space complexity. DS Split always has higher query time than DS Simple, but its space complexity is lower by a factor of $2^{\ell/2}$. For example, for $d = 2^{2\ell}$ we get the complexities shown in Table 2.

Table 2 Basic complexities of the data structures from Section 6 for $d = 2^{2\ell}$.

Data structure	Space	Query time
Algorithm Small-ℓ	$\mathcal{O}(2^{2\ell}\ell)$	$\mathcal{O}(2^{2\ell}\ell)$
DS Simple	$\mathcal{O}(2^{3\ell})$	$\mathcal{O}(2^{\ell}\ell)$
DS Split for $\tau = 2^{5\ell/4}$	$\mathcal{O}(2^{5\ell/2})$	$\mathcal{O}(2^{7\ell/4}\ell)$

7 Approximation Algorithm for PMDM

Clearly, PMDM is at least as hard as PMDM-Size because it also outputs the positions of the wildcards (set K). Thus, PMDM is also NP-hard. In what follows, we show existence of a polynomial-time approximation algorithm for PMDM whose approximation factor is given with respect to d. Specifically, we show the following approximation result for PMDM.

▶ **Theorem 13.** *For any constant $\epsilon > 0$, there is an $\mathcal{O}(d^{1/4+\epsilon})$-approximation algorithm for PMDM, whose running time is polynomial in N, where $N = d\ell$.*

Our result is based on the Minimum Union (MU) problem [21], which we define next.

Minimum Union (MU)
Input: A collection $\mathcal{S}$ of d sets over a universe U and a positive integer $z \leq d$.
Output: A collection $\mathcal{T} \subseteq \mathcal{S}$ with $|\mathcal{T}| = z$ such that the size of $\cup_{S\in\mathcal{T}} S$ is minimized.

To illustrate the MU problem, consider an instance of it where $U = \{1,2,3,4,5\}$, $\mathcal{S} = \{\{1\},\{1,2,3\},\{1,3,5\},\{3\},\{3,4,5\},\{4\},\{4,5\},\{5\}\}$, with $d = |\mathcal{S}| = 8$, and $z = 4$. Then $\mathcal{T} = \{\{3\},\{3,4,5\},\{4\},\{4,5\}\}$ is a solution because $|\mathcal{T}| = z = 4$ and $|\cup_{S\in\mathcal{T}} S| = 3$ is minimum. The MU problem is NP-hard and the following approximation result is known.

▶ **Theorem 14** ([21]). *For any constant $\epsilon > 0$, there is an $\mathcal{O}(d^{1/4+\epsilon})$-approximation algorithm for MU, whose running time is polynomial in the size of $\mathcal{S}$.*

▶ **Theorem 15.** *PMDM can be reduced to MU in time polynomial in N.*

Proof. We reduce the PMDM problem to MU in polynomial time as follows. Given any instance $\mathcal{I}_{\text{PMDM}}$ of PMDM, we construct an instance $\mathcal{I}_{\text{MU}}$ of MU in time $\mathcal{O}(d\ell)$ by performing the following steps:

1. The universe U is set to $\{1,\ldots,\ell\}$.
2. We start with an empty collection $\mathcal{S}$. Then, for each string s_i in $\mathcal{D}$, we add member S_i to $\mathcal{S}$, where S_i is the set of positions where string q and string s_i have a mismatch. This can be done trivially in time $\mathcal{O}(d\ell)$ for all strings in $\mathcal{D}$.
3. Set the z of the MU problem to the z of the PMDM problem.

Thus, the total time $\mathcal{O}(d\ell)$ needed for Steps 1 to 3 above is clearly polynomial in the size of $\mathcal{I}_{\text{PMDM}}$.

▷ Claim 16. For any solution $\mathcal{T}$ to $\mathcal{I}_{\text{MU}}$ and any solution K to $\mathcal{I}_{\text{PMDM}}$, $|K| = |\cup_{S\in\mathcal{T}} S|$.

Proof. Let $\mathcal{F} \subseteq \mathcal{D}$ consist of z strings that match q_K. Further, let the set $\mathcal{F}^*$ consist of the elements of $\mathcal{S}$ corresponding to strings in $\mathcal{F}$. We have $|\cup_{S \in \mathcal{T}} S| \leq |\cup_{S \in \mathcal{F}^*} S| \leq |K|$.

Now, let $C = \cup_{S \in \mathcal{T}} S$. Then, $q_C = q \otimes C$ matches at least z strings from $\mathcal{D}$ and hence $|K| \leq |C| = |\cup_{S \in \mathcal{T}} S|$. ◁

To conclude the proof, it remains to show that given a solution $\mathcal{T}$ to $\mathcal{I}_{\text{MU}}$ we can obtain a solution K to $\mathcal{I}_{\text{PMDM}}$ in time polynomial in the size of $\mathcal{I}_{\text{MU}}$. This readily follows from the proof of the above claim: it suffices to set $K = \cup_{S \in \mathcal{T}} S$. ◀

Proof of Theorem 13. The reduction in Theorem 15 implies that there is a polynomial-time approximation algorithm for PMDM. In particular, Theorem 14 provides an approximation guarantee for MU that depends on the number of sets of the input $\mathcal{S}$. In Step 2 of the reduction of Theorem 15, we construct *one* set for the MU instance per *one* string of the dictionary $\mathcal{D}$ of the PMDM instance. Also, from the constructed solution $\mathcal{T}$ to the MU instance, we obtain a solution K to the PMDM instance by simply substituting the positions of q corresponding to the elements of the sets of $\mathcal{T}$ with wildcards. This construction implies the approximation result of Theorem 13 that depends on the size of $\mathcal{D}$. ◀

Sanity Check. Theorem 1 (reduction from k-Clique to k-PMDM) and Theorem 13 (approximation algorithm for PMDM) do not contradict the inapproximability results for the maximum clique problem (see Section 3), since our reduction from k-Clique to k-PMDM cannot be adapted to a reduction from maximum clique to PMDM-Size.

Two-Way Reduction. Chlamtáč et al. [21] also show that their polynomial-time $\mathcal{O}(d^{1/4+\epsilon})$-approximation algorithm for MU is tight under a plausible conjecture for the so-called Hypergraph Dense vs Random problem. In what follows, we also show that approximating the MU problem can be reduced to approximating PMDM in polynomial time and hence the same tightness result applies to PMDM.

▶ **Theorem 17.** *MU can be reduced to PMDM in time polynomial in the size of $\mathcal{S}$.*

Proof. Let $||\mathcal{S}||$ denote the total number of elements in the d members of $\mathcal{S}$. We reduce the MU problem to the PMDM problem in polynomial time as follows. Given any instance $\mathcal{I}_{\text{MU}}$ of MU, we construct an instance $\mathcal{I}_{\text{PMDM}}$ of PMDM by performing the following steps:

1. Sort the union of all elements of members of $\mathcal{S}$, and assign to each element j a unique rank $\mathsf{rank}(j) \in \{1, \ldots, |U|\}$. Set $\ell = |U|$. This can be done in $\mathcal{O}(||\mathcal{S}|| \log ||\mathcal{S}||)$ time.
2. Set the query string q equal to the string $\mathtt{a}^\ell$ of length ℓ. For each set S_i in $\mathcal{S}$, construct a string $s_i = \mathtt{a}^\ell$, set $s_i[\mathsf{rank}(j)] := \mathtt{b}$ if and only if $j \in S_i$, and add s_i to dictionary $\mathcal{D}$. This can be done in $\mathcal{O}(d\ell)$ time.
3. Set the z of the PMDM problem equal to the z of the MU problem. This can be done in $\mathcal{O}(1)$ time.

Thus, the total time $\mathcal{O}(d\ell \log(d\ell))$ needed for Steps 1 to 3 above is clearly polynomial in the size of $\mathcal{I}_{\text{MU}}$ as $\ell \leq ||\mathcal{S}||$.

A proof of the following claim is analogous to that of Claim 16.

▷ Claim 18. For any solution $\mathcal{T}$ to $\mathcal{I}_{\text{MU}}$ and any solution K to $\mathcal{I}_{\text{PMDM}}$, $|K| = |\cup_{S \in \mathcal{T}} S|$.

To conclude the proof, it remains to show that, given a solution K to $\mathcal{I}_{\text{PMDM}}$, we can obtain a solution $\mathcal{T}$ to $\mathcal{I}_{\text{MU}}$ in time polynomial in the size of $\mathcal{I}_{\text{PMDM}}$. It suffices to pick z sets in $\mathcal{S}$ that are subsets of K. Their existence is guaranteed by construction, because such sets correspond to the at least z strings in $\mathcal{D}$ that have b in a subset of the positions in K. This selection can be done naïvely in $\mathcal{O}(||\mathcal{S}||)$ time. Finally, the above claim guarantees that they indeed form a solution to $\mathcal{I}_{\text{MU}}$. ◀

References

1 Secure critical data with Oracle Data Safe (white paper). `https://www.oracle.com/a/tech/docs/dbsec/data-safe/wp-security-data-safe.pdf`, September 2020.

2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018. `doi:10.1137/16M1061771`.

3 Peyman Afshani and Jesper Sindahl Nielsen. Data structure lower bounds for document indexing problems. In *43rd International Colloquium on Automata, Languages and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 93:1–93:15, 2016. `doi:10.4230/LIPIcs.ICALP.2016.93`.

4 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975. `doi:10.1145/360825.360855`.

5 Alberto Apostolico and Laxmi Parida. Incremental paradigms of motif discovery. *Journal of Computational Biology*, 11(1):15–25, 2004. `doi:10.1089/106652704773416867`.

6 Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. *SIAM Journal on Computing*, 42(5):2008–2037, 2013. `doi:10.1137/120884857`.

7 Hiroki Arimura and Takeaki Uno. An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. *Journal of Combinatorial Optimization*, 13(3):243–262, 2007. `doi:10.1007/s10878-006-9029-1`.

8 David R. Bailey, Alexis J. Battle, Benedict A. Gomes, and P. Pandurang Nayak. Estimating confidence for query revision models, U.S. Patent US7617205B2 (granted to Google), 2009.

9 Martha Bailey, Connor Cole, Morgan Henderson, and Catherine Massey. How well do automated linking methods perform? Lessons from U.S. historical data. NBER Working Papers 24019, National Bureau of Economic Research, Inc, 2017. `doi:10.3386/w24019`.

10 Giovanni Battaglia, Davide Cangelosi, Roberto Grossi, and Nadia Pisanti. Masking patterns in sequences: A new class of motif discovery with don't cares. *Theoretical Computer Science*, 410(43):4327–4340, 2009. `doi:10.1016/j.tcs.2009.07.014`.

11 Djamal Belazzougui. Faster and space-optimal edit distance "1" dictionary. In *20th Annual Symposium on Combinatorial Pattern Matching (CPM 2009)*, volume 5577 of *Lecture Notes in Computer Science*, pages 154–167. Springer, 2009. `doi:10.1007/978-3-642-02441-2_14`.

12 Djamal Belazzougui and Rossano Venturini. Compressed string dictionary search with edit distance one. *Algorithmica*, 74(3):1099–1122, 2016. `doi:10.1007/s00453-015-9990-0`.

13 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and Søren Vind. String indexing for patterns with wildcards. *Theory of Computing Systems*, 55(1):41–60, 2014. `doi:10.1007/s00224-013-9498-4`.

14 Allan Borodin, Rafail Ostrovsky, and Yuval Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *31st ACM Symposium on Theory of Computing (STOC 1999)*, pages 312–321, 1999. `doi:10.1145/301250.301330`.

15 Gerth Stølting Brodal and Srinivasan Venkatesh. Improved bounds for dictionary look-up with one error. *Information Processing Letters*, 75(1-2):57–59, 2000. `doi:10.1016/S0020-0190(00)00079-X`.

16 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Computation, 4th International Workshop (IWPEC 2009)*, volume 5917 of *Lecture Notes in Computer Science*, pages 75–85. Springer, 2009. `doi:10.1007/978-3-642-11269-0_6`.

17 Ho-Leung Chan, Tak Wah Lam, Wing-Kin Sung, Siu-Lung Tam, and Swee-Seong Wong. Compressed indexes for approximate string matching. *Algorithmica*, 58(2):263–281, 2010. doi:10.1007/s00453-008-9263-2.

18 Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *29th International Colloquium on Automata, Languages and Programming (ICALP 2002)*, pages 451–462, 2002. doi:10.1007/3-540-45465-9_39.

19 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007.

20 Eden Chlamtáč, Michael Dinitz, Christian Konrad, Guy Kortsarz, and George Rabanca. The densest k-subhypergraph problem. *SIAM Journal on Discrete Mathematics*, 32(2):1458–1477, 2018. doi:10.1137/16M1096402.

21 Eden Chlamtáč, Michael Dinitz, and Yury Makarychev. Minimizing the union: Tight approximations for small set bipartite vertex expansion. In *28th ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 881–899, 2017. doi:10.1137/1.9781611974782.56.

22 Peter Christen. *Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Data-Centric Systems and Applications. Springer, Heidelberg, 2012. doi:10.1007/978-3-642-31164-2.

23 Peter Christen, Thilina Ranbaduge, and Rainer Schnell. *Linking Sensitive Data.* Springer, Heidelberg, 2020. doi:10.1007/978-3-030-59706-1.

24 Vincent Cohen-Addad, Laurent Feuilloley, and Tatiana Starikovskaya. Lower bounds for text indexing with mismatches and differences. In *30th ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 1146–1164, 2019. doi:10.1137/1.9781611975482.70.

25 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *36th ACM Symposium on Theory of Computing (STOC 2004)*, pages 91–100, 2004. doi:10.1145/1007352.1007374.

26 Alfredo Cuzzocrea and Hossain Shahriar. Data masking techniques for nosql database security: A systematic review. In *2017 IEEE International Conference on Big Data (BigData 2017)*, pages 4467–4473, 2017. doi:10.1109/BigData.2017.8258486.

27 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.

28 Pawel Gawrychowski, Moshe Lewenstein, and Patrick K. Nicholson. Weighted ancestors in suffix trees. In *Algorithms - 22th Annual European Symposium (ESA 2014)*, volume 8737 of *Lecture Notes in Computer Science*, pages 455–466. Springer, 2014. doi:10.1007/978-3-662-44777-2_38.

29 Sreenivas Gollapudi, Samuel Ieong, Alexandros Ntoulas, and Stelios Paparizos. Efficient query rewrite for structured web queries. In *20th ACM International Conference on Information and Knowledge Management (CIKM 2011)*, pages 2417–2420, 2011. doi:10.1145/2063576.2063981.

30 Roberto Grossi, Giulia Menconi, Nadia Pisanti, Roberto Trani, and Søren Vind. Motif trie: An efficient text index for pattern discovery with don't cares. *Theoretical Computer Science*, 710:74–87, 2018. doi:10.1016/j.tcs.2017.04.012.

31 Johan Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999. doi:10.1007/BF02392825.

32 Thomas N. Herzog, Fritz J. Scheuren, and William E. Winkler. *Data quality and record linkage techniques.* Springer, 2007.

33 Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4), 2008. doi:10.1145/1391729.1391730.

34 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

35 T. S. Jayram, Subhash Khot, Ravi Kumar, and Yuval Rabani. Cell-probe lower bounds for the partial match problem. *Journal of Computer and System Sciences*, 69(3):435–447, 2004. doi:10.1016/j.jcss.2004.04.006.

36 Dimitrios Karapiperis, Aris Gkoulalas-Divanis, and Vassilios S. Verykios. Summarizing and linking electronic health records. *Distributed and Parallel Databases*, pages 1–40, 2019. doi:10.1007/s10619-019-07263-0.

37 Richard M. Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2010. doi:10.1007/978-3-540-68279-0_8.

38 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *The Multiple-Choice Knapsack Problem*, pages 317–347. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-24777-7_11.

39 Pradap Konda, Sanjib Das, Paul Suganthan G.C., Philip Martinkus, Adel Ardalan, Jeffrey R. Ballard, Yash Govind, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. Technical perspective: Toward building entity matching management systems. *SIGMOD Record*, 47(1):33–40, 2018. doi:10.1145/3277006.3277015.

40 Hye-Chung Kum, Ashok Krishnamurthy, Ashwin Machanavajjhala, Michael K. Reiter, and Stanley Ahalt. Privacy preserving interactive record linkage (PPIRL). *Journal of the American Medical Informatics Association*, 21(2):212–220, 2014. doi:10.1136/amiajnl-2013-002165.

41 Hye-Chung Kum, Eric D. Ragan, Gurudev Ilangovan, Mahin Ramezani, Qinbo Li, and Cason Schmit. Enhancing privacy through an interactive on-demand incremental information disclosure interface: Applying privacy-by-design to record linkage. In *Fifteenth USENIX Conference on Usable Privacy and Security*, pages 175–189, 2019. doi:10.5555/3361476.3361489.

42 Prathyusha Senthil Kumar, Praveen Arasada, and Ravi Chandra Jammalamadaka. Systems and methods for generating search query rewrites, U.S. Patent US10108712B2 (granted to ebay), 2018.

43 Moshe Lewenstein, J. Ian Munro, Venkatesh Raman, and Sharma V. Thankachan. Less space: Indexing for queries with wildcards. *Theoretical Computer Science*, 557:120–127, 2014. doi:10.1016/j.tcs.2014.09.003.

44 Moshe Lewenstein, Yakov Nekrich, and Jeffrey Scott Vitter. Space-efficient string indexing for wildcard pattern matching. In *31st Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, pages 506–517, 2014. doi:10.4230/LIPIcs.STACS.2014.506.

45 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *29th ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 1236–1252, 2018. doi:10.1137/1.9781611975031.80.

46 Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998. doi:10.1006/jcss.1998.1577.

47 George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys*, 53(2), 2020. doi:10.1145/3377455.

48 Laxmi Parida, Isidore Rigoutsos, Aris Floratos, Daniel E. Platt, and Yuan Gao. Pattern discovery on character sets and real-valued data: Linear bound on irredundant motifs and an efficient polynomial time algorithm. In *11th ACM-SIAM Symposium on Discrete Algorithms (SODA 2000)*, pages 297–308, 2000. doi:10.1145/338219.338266.

49 Nadia Pisanti, Maxime Crochemore, Roberto Grossi, and Marie-France Sagot. Bases of motifs for generating repeated patterns with wild cards. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(1):40–50, 2005. doi:10.1109/TCBB.2005.5.

50 Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal on Computing*, 40(3):827–847, 2011. doi:10.1137/09075336X.

51 Mihai Pătraşcu and Mikkel Thorup. Higher lower bounds for near-neighbor and further rich problems. *SIAM Journal on Computing*, 39(2):730–741, 2009. doi:10.1137/070684859.

52 Eric D. Ragan, Hye-Chung Kum, Gurudev Ilangovan, and Han Wang. Balancing privacy and information disclosure in interactive record linkage with visual masking. In *ACM Conference on Human Factors in Computing Systems (CHI 2018)*, 2018. doi:10.1145/3173574.3173900.

53 Ronald L. Rivest. Partial-match retrieval algorithms. *SIAM Journal on Computing*, 5(1):19–50, 1976. doi:10.1137/0205003.

54 Pierangela Samarati and Latanya Sweeney. Generalizing data to provide anonymity when disclosing information (abstract). In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1998)*, page 188. Association for Computing Machinery, 1998. doi:10.1145/275487.275508.

55 Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, Computer Science Laboratory, SRI International, 1998.

56 Ricardo Jorge Santos, Jorge Bernardino, and Marco Vieira. A data masking technique for data warehouses. In *15th International Database Engineering and Applications Symposium (IDEAS 2011)*, pages 61–69, 2011. doi:10.1145/2076623.2076632.

57 Latanya Sweeney. *Computational disclosure control: a primer on data privacy protection*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2001. URL: http://hdl.handle.net/1721.1/8589.

58 Zehong Tan, Canran Xu, Mengjie Jiang, Hua Yang, and Xiaoyuan Wu. Query rewrite for null and low search results in ecommerce. In *SIGIR Workshop On eCommerce*, volume 2311 of *CEUR Workshop Proceedings*, 2017.

59 Yufei Tao. Entity matching with active monotone classification. In *37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2018)*, pages 49–62, 2018. doi:10.1145/3196959.3196984.

60 Dinusha Vatsalan and Peter Christen. Scalable privacy-preserving record linkage for multiple databases. In *23rd ACM International Conference on Information and Knowledge Management (CIKM 2014)*, pages 1795–1798, 2014. doi:10.1145/2661829.2661875.

61 Dinusha Vatsalan, Ziad Sehili, Peter Christen, and Erhard Rahm. Privacy-preserving record linkage for Big Data: Current approaches and research challenges. In *Handbook of Big Data Technologies*, pages 851–895. Springer, 2017. doi:10.1007/978-3-319-49340-4.

62 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory (SWAT 1973)*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.

63 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *2018 International Congress of Mathematicians (ICM)*, pages 3447–3487, 2019. doi:10.1142/9789813272880_0188.

64 Andrew Chi-Chih Yao and Frances F. Yao. Dictionary look-up with one error. *Journal of Algorithms*, 25(1):194–202, 1997. doi:10.1006/jagm.1997.0875.

65 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007. doi:10.4086/toc.2007.v003a006.