Partial Permutations Comparison, Maintenance and Applications

Avivit Levy
Department of Software Engineering, Shenkar College, Ramat-Gan, Israel

Ely Porat
Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel

B. Riva Shalom
Department of Software Engineering, Shenkar College, Ramat-Gan, Israel

Abstract

This paper focuses on the concept of *partial permutations* and their use in algorithmic tasks. A *partial permutation* over Σ is a bijection $\pi_{par} : \Sigma_1 \mapsto \Sigma_2$ mapping a subset $\Sigma_1 \subset \Sigma$ to a subset $\Sigma_2 \subset \Sigma$, where $|\Sigma_1| = |\Sigma_2|$ ($|\Sigma|$ denotes the size of a set Σ). Intuitively, two partial permutations *agree* if their mapping pairs do not form *conflicts*. This notion, which is formally defined in this paper, enables a consistent as well as informatively rich comparison between partial permutations. We formalize the *Partial Permutations Agreement* problem (PPA), as follows. Given two sets A_1, A_2 of partial permutations over alphabet Σ, each of size n, output all pairs (π_i, π_j), where $\pi_i \in A_1, \pi_j \in A_2$ and π_i *agrees* with π_j. The possibility of having a data structure for efficiently maintaining a dynamic set of partial permutations enabling to retrieve agreement of partial permutations is then studied, giving both negative and positive results. Applying our study enables to point out fruitful versus futile methods for efficient genes sequences comparison in database or automatic color transformation data augmentation technique for image processing through neural networks. It also shows that an efficient solution of strict Parameterized Dictionary Matching with One Gap (PDMOG) over general dictionary alphabets is not likely, unless the Strong Exponential Time Hypothesis (SETH) fails, thus negatively answering an open question posed lately.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Partial permutations, Partial words, Genes comparison, Color transformation, Dictionary matching with gaps, Parameterized matching, SETH hypothesis

Digital Object Identifier 10.4230/LIPIcs.CPM.2022.10

Acknowledgements We would like to thank Amihood Amir for his valuable suggestions while reading a former version of the paper.

1 Introduction

Permutations are a classical mathematical concept widely used in computer science: playing a role in analyzing sorting algorithms [29], being a basic building block in randomization [19], and appearing in various fields, such as Computational Biology (e.g. [8, 24]), Pattern Matching (e.g. [4, 32]), Cryptography (e.g. [21]), and more. A permutation over an alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ is a bijection $\pi : \Sigma \mapsto \Sigma$ mapping every symbol $\sigma_i \in \Sigma$ to a distinct symbol $\sigma_j \in \Sigma$ (where it may be that $i = j$). In this paper, we focus on the concept of *partial* permutations and their use in algorithmic tasks.

A *partial permutation* over Σ is a bijection $\pi_{par} : \Sigma_1 \mapsto \Sigma_2$ mapping a subset $\Sigma_1 \subset \Sigma$ to a subset $\Sigma_2 \subset \Sigma$, where the sizes of the sets Σ_1, Σ_2 are equal, i.e., $|\Sigma_1| = |\Sigma_2|$.[1] Partial permutations are closely related to *partial words*, defined as follows. A *partial word* over Σ

[1] The subscript par is only used in this paragraph to distinguish a partial permutation from a permutation, however, throughout the paper we omit it for convenience and denote a partial permutation by π.

is a word (string) over the alphabet $\Sigma \cup \{\Diamond\}$, where the symbol $\Diamond$ is treated as a *hole*[2]. In the study of partial words, the holes are usually treated as gaps that may be filled by an arbitrary letter of Σ. Note that, a partial permutation is a partial word π such that each symbol of Σ appears in π exactly once, and all the remaining symbols of π are holes [17].

The study of partial words was initiated by [11, 34] for comparing genes, where alignment can be viewed as a construction of two partial words that are compatible in the sense defined in [11]. However, for the task of comparing genes sequences, partial permutations were suggested as an appropriate model due to diversity of genes and the incompleteness nature of such sequences [45]. Partial permutations play a role also in computational tasks other than computational biology. For example, it can be used for representing color transformations as a data augmentation technique in image processing through neural networks [26, 37]. In addition, in pattern matching algorithms strings may be mapped to other strings, as in the well-known parameterized matching and related problems [9, 35, 40].

Combinatorial aspects of partial words that have been studied include periods in partial words [11, 41], avoidability/unavoidability of sets of partial words [12, 13], squares in partial words [22], and overlap-freeness [23]. Combinatorial questions regarding partial permutations were also studied, e.g., pattern avoidance [17], enumeration [42, 31] or restricted forms [17, 14].

In this paper, we study algorithmic aspects of maintaining partial permutations. To this end, we next discuss the basic operation of comparing partial permutations, formally define the concept of their *agreement* and describe a condition on partial permutations representations that naturally perceive the agreement between two partial permutations.

1.1 Partial Permutations Comparison

Let R be any representation of a *permutation* π over an alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$. Since R represents the bijection where the domain and codomain of π are identical, it should only specify the mapped pairs of symbols. Then, it holds that $R(\pi_1) = R(\pi_2)$ if and only if the permutations π_1 and π_2 are equal. We refer to this property as *the comparison axiom.*

Assume any representation R of a *partial permutation.* Since R represents a bijection having non-obvious domain and codomain, it should specify the domain and codomain sets of π (denoted by D_R and C_R, respectively) as well as the set of symbols pairs mapped by the bijection (denoted by M_R). A comparison of partial permutations based on such a representation R is more complicated. We may wish to know if two partial permutations are identical and enforce their representations to be equal, but then we put a rigid limitation on our notion of comparison. Considering the nature of partial permutations, we may rather prefer a way to compare the "agreement" between two given partial permutations. Formally,

▶ **Definition 1** (Conflict and Agreement of Partial Permutations). *Two partial permutations π_1, π_2 are* conflicting *(alternatively, contain a* conflict*) if either:*

1. *There exist $\sigma_i, \sigma_j \in \Sigma_2$, $\sigma_i \neq \sigma_j$, such that there exists $\sigma_k \in \Sigma_1$ where $\pi_1(\sigma_k) = \sigma_i$ and $\pi_2(\sigma_k) = \sigma_j$, or*
2. *There exist $\sigma_i, \sigma_j \in \Sigma_1$, $\sigma_i \neq \sigma_j$, such that there exists $\sigma_k \in \Sigma_2$ where $\pi_1(\sigma_i) = \pi_2(\sigma_j) = \sigma_k$.*

We say that π_1, π_2 agree *if they do not contain any conflict.*

[2] The hole symbol $\Diamond$ is not treated as a don't care symbol as is common in pattern matching, but rather as a don't know symbol.

Definition 1 enables to establish a comparison between two given partial permutations aimed at revealing wether they agree or not. Since the representation is the key to the comparison process, we have to take it into account. Note, however, that Definition 1 is independent of the chosen representation. We may, therefore, derive from it a universal condition on any representation enabling comparison of agreement between partial permutations.

▶ **Lemma 2** (A Universal Condition of Partial Permutations Agreement)**.** *Let R be any representation of partial permutation bijections, and let π_1, π_2 be two partial permutations. Then, π_1, π_2 agree if and only if the following conditions hold on $R(\pi_1) = \langle D_R(\pi_1), C_R(\pi_1), M_R(\pi_1)\rangle, R(\pi_2) = \langle D_R(\pi_2), C_R(\pi_2), M_R(\pi_2)\rangle$:*

1. *For every $\sigma_k \in D_R(\pi_1) \cap D_R(\pi_2)$ and $\sigma_i \in C_R(\pi_1)$, if $(\sigma_k, \sigma_i) \in M_R(\pi_1)$ then $\sigma_i \in C_R(\pi_2)$ and $(\sigma_k, \sigma_i) \in M_R(\pi_2)$.*
2. *For every $\sigma_k \in C_R(\pi_1) \cap C_R(\pi_2)$ and $\sigma_i \in D_R(\pi_1)$, if $(\sigma_i, \sigma_k) \in M_R(\pi_1)$ then $\sigma_i \in D_R(\pi_2)$ and$(\sigma_i, \sigma_k) \in M_R(\pi_2)$.*

Proof. Obviously, the first condition of the lemma avoids a conflict of the first type of Definition 1, and the second condition avoids a conflict of the second type in Definition 1. ◀

▶ **Example.** *Let $\Sigma = \{0, 1, 2, 3\}$, $D_R(\pi_1) = \{1, 2\}$, $C_R(\pi_1) = \{0, 3\}$, $M_R(\pi_1) = \{(1, 3), (2, 0)\}$, $D_R(\pi_2) = \{0, 1\}$, $C_R(\pi_2) = \{1, 3\}$, $M_R(\pi_2) = \{(0, 1), (1, 3)\}$, then π_i* agrees *with π_j.*

The following classification of partial permutations representations will be useful for the discussion of algorithms complexity.

▶ **Definition 3** (Good Representation)**.** *A representation R for partial permutations is called a* good representation *if, assuming word-RAM model, for every π_i,*

1. *the size of $R(\pi_i)$ is $O(|\Sigma|)$, and*
2. *for every π_j, determining whether the universal agreement condition between $R(\pi_i)$ and $R(\pi_j)$ holds can be done in $O(|\Sigma|)$ time.*

In the paper, we discuss the existence of a data structure for maintaining a dynamic set of partial permutations supporting the operations of insert, delete and search agreement with the query over the set. A *static* partial permutations set situation is first investigated and then supporting a *dynamic* set is referred to. We employ a fine-grained complexity analysis, which have recently become an important tool (e.g., in [28, 20, 25]).

1.2 Fine-Grained Complexity Analysis

In traditional computer science theory, the typical problems considered "hard" are $\mathcal{NP}$-Hard and maybe even require exponential time to solve. Problems having polynomial time algorithms are considered "easy". The best known algorithms for many such "easy" problems have high run-times, thus, are impractical, and their improvement has been a longstanding open problem with little to no progress. It may be that these algorithms are optimal, however, deriving unconditional lower bounds seems beyond current techniques.

A new, conditional theory of hardness has recently been developed, based around several plausible conjectures. The theory develops reductions between seemingly very different problems, showing that the reason why the known algorithms have been difficult to improve is likely the same, even though the known run-times of the problems might be very different. This direction of study has been termed "fine-grained complexity" theory (see e.g. [44]).

Much of fine-grained complexity is based on hypotheses of the time complexity of infamous problems, e.g., CNF-SAT, All-Pairs Shortest Paths (APSP) and 3-SUM. The hypotheses are about the word-RAM model with $O(\log n)$ bit words, where n is the input size. [27] introduced the Strong Exponential Time Hypothesis (SETH) to address CNF-SAT complexity.

The Strong Exponential Time Hypothesis (SETH) [27]. For every $\epsilon > 0$ there exists an integer $k \geq 3$ such that CNF-SAT on formulas with clause size at most k (called k-SAT) and n variables cannot be solved in $O(2^{(1-\epsilon)n})$ time even by a randomized algorithm.

Orthogonal Vectors. The Orthogonal Vectors (OV) problem is a core problem in the basis of many fine-grained hardness results for problems in $\mathcal{P}$. The problem is formally defined as follows.

▶ **Definition 4** (The OV Problem). *Let $d = \omega(\log n)$; given two sets $S_1, S_2 \in \{0,1\}^d$ with $|S_1| = |S_2| = n$, determine whether there exist $a \in S_1$, $b \in S_2$ so that $a \cdot b = 0$, where $a \cdot b = \sum_{i=1}^{d} a[i] \cdot b[i]$.*

It is not hard to solve OV in $O(n^2 d)$ time by exhaustive search. The fastest known algorithms for the problem run in time $n^{2-1/\Theta(\log(d/\log n))}$ [1, 15]. It seems that $n^{2-o(1)}$ is necessary. This motivates the now widely used OV Hypothesis.

OV Hypothesis. No randomized algorithm can solve OV on instances of size n in $n^{2-\epsilon} poly(d)$ time for constant $\epsilon > 0$.

We describe the connection between OV and *Partial Permutation Agreement* (PPA) (formally defined in Definition 7, Section 3) problems. In fact, we show that they are equivalent, leading to both negative and positive results for PPA, derived also for the dynamic setting . Algorithmic applications to problems in computational biology, image processing and pattern matching are further described in Section 4.

This Paper Contributions. The main contributions of this paper are:

- Giving the first formal discussion from algorithmic point of view of efficient partial permutations maintenance enabling consistent as well as informatively rich comparison.
- Showing that a data structure for efficiently maintaining a dynamic set of partial permutations is not likely to exist, unless the SETH hypothesis fails.
- Describing positive results on maintaining a dynamic set of partial permutations: (1) an improvement in the general case derived via online matrix-factor multiplication, and (2) an efficient solution in a special case termed *almost full partial permutations*.
- Applying the study to reason about fruitful versus futile methods for efficient gene sequences comparison, enabling a formal understanding of this challenge, hinted in [45].
- Applying the study to form an *automatic* process of redundant augmented-data removal to avoid over-fitting of a neural network training set for image processing tasks and increasing its capability to generalize to unseen invariant data [43].
- Applying the study of partial permutation maintenance to answer negatively (unless the SETH hypothesis is false) for an open question regarding a solution over a general alphabet dictionary for the online strict PDMOG problem presented by [35] (see formal definition in Sect. 4.2), while supplying new tools for efficient solution in a special case termed *a k-saturated dictionary* (see formal definition in Sect. 4.2).

Paper Organization. The paper is organized as follows. Section 2 gives the needed preliminary discussion on a good representation of partial permutations. Section 3 studies the possibility of having a data structure for efficiently maintaining a dynamic set of partial permutations as follows. Subsection 3.1 describes the reduction from the OV problem. Subsection 3.2 shows the equivalence to the OV problem via a connection to the *Partial*

Match problem (PM) (formally defined there), which also enables to exploit some positive results for the PM problem to our purposes. Subsection 3.3 describes how to achieve an efficient solution in the special case of almost full partial permutations. Section 4 describes the applications to genes sequences comparison (Subsection 4.1), to color transformation data augmentation and to online strict PDMOG problems (Subsection 4.2). Section 5 concludes the paper with some open questions.

2 Preliminaries: A Good Representation of Partial Permutations

A permutation π over an alphabet $\Sigma = \{\sigma_1, \dots, \sigma_{|\Sigma|}\}$ can be represented as the string $s_\pi = \pi(\sigma_1)\pi(\sigma_2)\dots\pi(\sigma_{|\Sigma|})$ of length $|\Sigma|$. For example, let $\Sigma = \{a, b, c\}$ and $\pi = \{a \mapsto b, b \mapsto c, c \mapsto a\}$, then $s_\pi = bca$. It trivially holds that s_{π_1} matches s_{π_2} if and only if the permutations π_1 and π_2 are equal. A straightforward approach to achieve a good representation for partial permutations is to adjust the above by enabling the use of a don't care symbol $\star$ whenever an alphabet symbol of Σ does not belong to the partial permutation domain. For example, let $\Sigma = \{a, b, c, d, e, f\}$ and $M_R(\pi) = \{(a, e), (b, c), (c, d)\}$, where $D_R(\pi) = \{a, b, c\}$ and $C_R(\pi) = \{c, d, e\}$, then this approach suggests using the string $s_\pi = ecd \star\star\star$ as a representation for π. Note, that an exact string comparison of the strings representing two given partial permutations according to this suggestion still allows two partial permutations to be exactly the same. The use of a don't care symbol in order to broaden the equality scope to partial permutations that *agree* requires using approximate string matching with don't care, where this special symbol indeed matches any symbol.

Nonetheless, even when string matching with don't care is applied, the comparison axiom does not hold for this representation. To see this, consider the partial permutation π_1 where, $M_R(\pi_1) = \{(a, e), (b, c), (e, d)\}$, $D_R(\pi_1) = \{a, b, e\}$ and $C_R(\pi_1) = \{c, d, e\}$. By the above suggestion the representation of π_1 would be $s_{\pi_1} = ec\star\star d\star$. Let π_2 be the partial permutation from the above example, thus $s_{\pi_2} = ecd\star\star\star$. It holds that s_{π_2} matches s_{π_1}, because the don't care symbol $\star$ matches any symbol. However, the two partial permutations π_1 and π_2 contain a *conflict*. Note that, though the requirement of approximately matching the strings s_{π_1} and s_{π_2} exclude the possibility of a conflict of the first type in Definition 1, it does not exclude a conflict of the second type. Thus, it does not satisfy the universal condition for partial permutations agreement.

Correcting this flaw involves the use of the *inverse permutation*, defined as follows.

▶ **Definition 5** (Inverse of Partial Permutation). *Given a partial permutation π over Σ, mapping the subset $\Sigma_1 \subset \Sigma$ to the subset $\Sigma_2 \subset \Sigma$, where $|\Sigma_1| = |\Sigma_2|$, the* inverse partial permutation *π^{-1} of π is a bijection $\pi^{-1} : \Sigma_2 \mapsto \Sigma_1$ such that for every $\sigma_i \in \Sigma_2$, $\pi^{-1}(\sigma_i) = \sigma_j$ if and only if $\pi(\sigma_j) = \sigma_i$.*

For example, let $\Sigma = \{a, b, c, d, e, f\}$ and $\pi = \{a \mapsto e, b \mapsto c, c \mapsto d\}$ be a partial permutation, where $\Sigma_1 = \{a, b, c\}$ and $\Sigma_2 = \{c, d, e\}$, then the inverse partial permutation of π is $\pi^{-1} = \{c \mapsto b, d \mapsto c, e \mapsto a\}$.

Now, a partial permutations representation enabling a distinction between two different partial permutations that agree and two partial permutation that disagree is simply the string $s_\pi \cdot s_{\pi^{-1}}$, where $\cdot$ denotes strings concatenation. Note that the size of this representation is $\Theta(|\Sigma|)$. Lemma 6 below ensures that this representation satisfies the comparison axiom.

▶ **Lemma 6.** *Given two partial permutations π_1, π_2, then $s_{\pi_1} \cdot s_{\pi_1^{-1}}$ matches $s_{\pi_2} \cdot s_{\pi_2^{-1}}$ if and only if the partial permutations π_1 and π_2 do not contain any conflict.*

The Don't Care Representation of Partial Permutations. Based on Lemma 6, given a partial permutation π we may call the string $s_\pi \cdot s_{\pi^{-1}}$ *the don't care representation of* π. We make the distinction between the don't care representation of π, which is a specific representation described in this section, and the universal condition on any representation described in Subsection 1.1, in order to enable the discussion in the next section to be independent of the representation, when necessary.

3 Maintaining a Dynamic Set of Partial Permutations

In this section we study the possibility of having a data structure for keeping a set of partial permutations enabling the operations: search, insert and delete on the set.

Following the discussion on Subsection 1.1, though there is exactly one partial permutation having the *same* representation as a given partial permutation π, there can be many partial permutations with a representation that satisfy the *universal condition for agreement* with the representation of π. All such partial permutations agree with the given partial permutation, though they are not identical, and may not agree with each other.

Therefore, we make a distinction between these two kinds of search operations: searching the same permutation or searching agreeing permutations. Specifically, given a partial permutation representation, we would like to support a query that returns all the partial permutations in the data structure that have a representation satisfying the universal condition for agreement, i.e., permutations that agree with the query permutation.

We will also refer to an offline batch version of this problem, formally defined as follows.

▶ **Definition 7** (The Partial Permutations Agreement Problem (PPA)).
Input: *Sets A_1, A_2 of partial permutations over alphabet Σ, each of size n.*
Output: *All pairs (π_i, π_j), $\pi_i \in A_1, \pi_j \in A_2$ and π_i* agrees *with π_j.*

In the non-batch version of the problem the size of the two sets is different: A_1 has size n, where A_2, the query, has size 1. We call this problem the *single query PPA* problem, denoted as SPPA. The following observation immediately follows.

▶ **Observation 8.** *If SPPA can be solved in query time $O(q)$ and $O(\mathcal{S})$ space for a set of n partial permutations, then PPA can be solved in $O(nq)$ time and $O(\mathcal{S})$ space.*

Note, that by Definition 3, for any *good representation R*, SPPA can be naively solved in $q = O(n \cdot |\Sigma|)$ time and $\mathcal{S} = O(n \cdot |\Sigma|)$ space. Therefore, by Observation 8, PPA can be naively solved in $O(n^2 \cdot |\Sigma|)$ time and $O(\mathcal{S})$ space for any good representation of partial permutations.

3.1 Orthogonal Vectors and Partial Permutations Agreement

In this subsection, we show that PPA is not likely to be solved in $O(n^{2-\epsilon} \cdot |\Sigma|)$ time. We describe a reduction from the orthogonal vectors problem (OV). Theorem 10 follows. Corollary 12 then follows from Observation 8. The proofs are postponed to the full version.

The Reduction. Let S_1, S_2, n, d be an instance of the Orthogonal Vectors problem, we reduce it to an instance A_1, A_2, Σ of the Partial Permutations Agreement problem, where there are $v_i \in S_1, v_j \in S_2$ such that v_i, v_j are orthogonal if and only if there are $\pi_i \in A_1, \pi_j \in A_2$, such that π_i *agrees* with π_j.

We construct a permutation gadget for every binary vector v_i as follows. Let $v_i = (b^i_1, b^i_2, \ldots, b^i_d)$. We define a partial permutation π_i over alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_{d+1}\}$ ($|\Sigma| = d + 1$), where π_i includes the mapping of $\sigma_\ell \in \Sigma$ to a symbol from Σ if and only if $b^i_\ell = 1$,

$\forall 1 \leq \ell \leq d$. However, $\forall 1 \leq \ell \leq d$, where $b^i_\ell = 0$, σ_ℓ does not participate in any pair defining the permutation gadget. Hence, for any representation R of the partial permutation π, we have that $\sigma_\ell \in D_R(\pi)$ if and only if $b_\ell = 1$.

The specific transformation to permutations is asymmetric, i.e., the transformation of S_1 vectors differs from that of S_2 vectors, as follows.

- For π_i associated with $v_i \in S_1$, a symbol that participates in the mapping pairs is mapped to itself. This means that for any representation R of the partial permutation π_i, we have that $\sigma_\ell \in D_R(\pi_i)$, $\sigma_\ell \in C_R(\pi_i)$ and $(\sigma_\ell, \sigma_\ell) \in M_R(\pi_i)$ if and only if $b_\ell = 1$. The additional symbol σ_{d+1} does not participate in any mapping pair. We regard it as if v_i has an additional bit $b^i_{d+1} = 0$. Therefore, for any representation R we get $\sigma_{d+1} \notin D_R(\pi_i)$, $\sigma_{d+1} \notin C_R(\pi_i)$.
- For π_j associated with $v_j \in S_2$, a symbol that participates in the mapping pairs is mapped to the symbol cyclicly to its right in the sorting of Σ_1 – the symbols that participate in the mapping pairs. This means that for any representation R of the partial permutation π_j, we have that $\sigma_\ell \in D_R(\pi_j)$, $\sigma_{\ell'} \in C_R(\pi_j)$ and $(\sigma_\ell, \sigma_{\ell'}) \in M_R(\pi_j)$, where $\sigma_{\ell'}$ is the symbol that is cyclically to the right of σ_ℓ in the sorting of Σ_1, if and only if $b_\ell = 1$. The additional symbol σ_{d+1} is included in the mapping pairs of π_j. Thus, $\sigma_{d+1} \in \Sigma_1$ and assumed to be ordered last. We regard it as if v_j has an additional bit $b^j_{d+1} = 1$. Therefore, for any representation R we get $\sigma_{d+1} \in D_R(\pi_j)$, $\sigma_{d+1} \in C_R(\pi_j)$.

See Figure 1 for example.

▶ **Lemma 9.** *OV is reducible to PPA in $O(n \cdot d)$ time and space.*

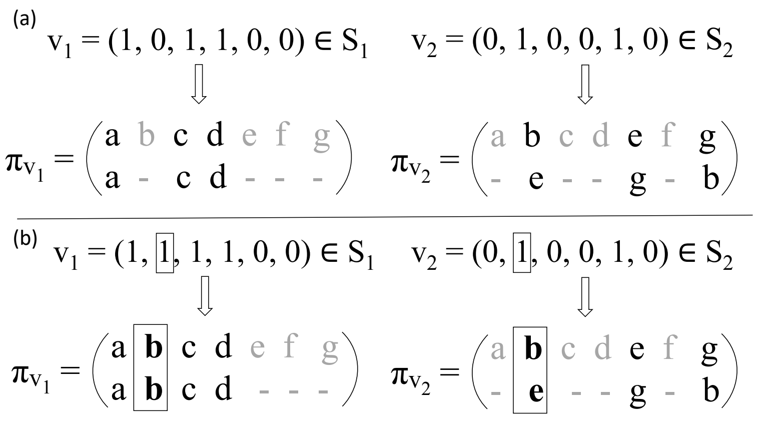
(a)
$v_1 = (1, 0, 1, 1, 0, 0) \in S_1$
$v_2 = (0, 1, 0, 0, 1, 0) \in S_2$
$\pi_{v_1} = \begin{pmatrix} a & b & c & d & e & f & g \\ a & - & c & d & - & - & - \end{pmatrix}$
$\pi_{v_2} = \begin{pmatrix} a & b & c & d & e & f & g \\ - & e & - & - & g & - & b \end{pmatrix}$
(b)
$v_1 = (1, \boxed{1}, 1, 1, 0, 0) \in S_1$
$v_2 = (0, \boxed{1}, 0, 0, 1, 0) \in S_2$
$\pi_{v_1} = \begin{pmatrix} a & \mathbf{b} & c & d & e & f & g \\ a & \mathbf{b} & c & d & - & - & - \end{pmatrix}$
$\pi_{v_2} = \begin{pmatrix} a & \mathbf{b} & c & d & e & f & g \\ - & \mathbf{e} & - & - & g & - & b \end{pmatrix}$

Figure 1 An example of the asymmetric transformation of vectors from S_1 and S_2 into permutations in A_1 and A_2, respectively. (a) The transformation for a pair of orthogonal vectors gives a pair of permutations that agree. (b) The transformation for a pair of non-orthogonal vectors gives a pair of permutations that do not agree.

Theorem 10 follows.

▶ **Theorem 10.** *Let R be any good representation of partial permutations. If there exists $\epsilon > 0$ such that for any $c > 0$, PPA is solvable in $O(n^{2-\epsilon})$ time and R is used to represent the partial permutations in the sets A_1, A_2, then the Strong Exponential Time Hypothesis (SETH) is false.*

Observation 11 below states a weaker version of Theorem 10, which refers explicitly also to the space complexity and includes the $|\Sigma|$-parameter of the PPA problem. This will be useful for the applications, especially in Section 4.

▶ **Observation 11.** *Let R be any good representation of partial permutations. If there exists $\epsilon > 0$ such that for any $c > 0$, PPA is solvable in $O(n^{2-\epsilon} \cdot |\Sigma|)$ time and $O(n \cdot |\Sigma|)$ space, where $|\Sigma| = c \log n$ and R is used to represent the partial permutations in the sets A_1, A_2, then the Strong Exponential Time Hypothesis (SETH) is false.*

▶ Remark. We include the dependence on $|\Sigma|$ in the time complexity in order to make explicit the role of this parameter. In the low-dimensional setting, where $|\Sigma|$ is slightly larger than logarithmic, it could be dropped (since sub-polynomial), where for moderate dimension even $O(n^{2-\epsilon} poly(\Sigma))$ algorithms can be ruled out under the OV Hypothesis.

Corollary 12 then follows from Theorem 10 and Observation 8.

▶ **Corollary 12.** *Let R be any good representation of partial permutations. If there exists $\epsilon > 0$ such that for any $c > 0$, SPPA query q can be answered in $O(n^{1-\epsilon})$ time and R is used to represent the partial permutations in the set A_1 and the query q, then the Strong Exponential Time Hypothesis (SETH) is false.*

3.2 The Partial Match Problem and Partial Permutations Agreement

In this subsection we discuss the connection between the Partial Permutations Agreement problem and another important problem – the Partial Match, formally defined as follows.

▶ **Definition 13** (The Partial Match Problem (PM)).
Preprocess: *A set D of n binary vectors of dimension d.*
Query: *A vector q of dimension d over the set $\{0, 1, \star\}$, where $\star$ is a "don't care" symbol.*
Output: *All vectors $v \in D$, such that v* matches *the query vector q.*

In the batch version of the Partial Match problem, denoted by BPM, we have instead of a single query vector, a set Q of n vectors over the set $\{0, 1, \star\}$, and the requested output is all pairs of vectors (v, q), where $v \in D$ and $q \in Q$, such that v *matches* q.

The PM problem has been thoroughly studied for decades (e.g. Rivest's PhD thesis [39]). However, there has been only minor algorithmic progress beyond the two obvious solutions of storing $2^{\Omega(d)}$ space for all possible queries, or taking $\Omega(n)$ time to try all points in the database. It was generally believed that PM is intractable for sufficiently large dimension d – this is one version of the "curse of dimensionality" hypothesis. The best known data structures for answering partial match queries are due to [16] for the general case, and [18] for queries with a bounded number of don't care symbols. Finally, [1] point out some evidence that batch partial match (BPM) is not solvable in sub-quadratic time due to its equivalence to the OV problem. Consequently, it gives some evidence to the difficulty of the PM problem: it is not likely to be solved in $O(n^{1-\epsilon} \cdot d)$ time and space due to an observation similar to Observation 8.

The Two-Sided BPM and PPA Problems. Note that, in the definitions of the PM and BPM problems don't care symbols are only allowed in the query vectors, but the database vectors are over $\{0, 1\}$. The good representation for partial permutations described in Section 2 gives a version of the PPA problem for the don't care representation which can be viewed as a generalization of the BPM problem, where both database and query vectors are over the set $\{0, 1, \star\}$. We call this problem the *two-sided Batch Partial Match* problem (two-sided

BPM). The result presented in Theorem 10 is strong in the sense that it is *independent of the representation*, and means that the difficulty of PPA is not due to a choice of a too rich representation. Theorem 10, specifically, applies also for the don't care representation of partial permutations, for which the PPA problem becomes exactly the two-sided BPM problem. We, thus, get from Theorem 10 the following corollary.

▶ **Corollary 14.** *If there exists $\epsilon > 0$ such that for any $c > 0$, two-sided BPM is solvable in $O(n^{2-\epsilon})$ time, then the Strong Exponential Time Hypothesis (SETH) is false.*

Equivalence of PM, Two-Sided PM and SPPA Problems. In fact, we now show that these three problems: the partial match (PM), two-sided partial match (two-sided PM) and single-query partial permutations agreement (SPPA), are actually equivalent. As mentioned above, the good representation for partial permutations described in Section 2 gives a version of the PPA problem for the don't care representation which is exactly the two-sided BPM problem, where both database and query vectors are over the set $\{0, 1, \star\}$. Moving to the non-batch versions of these problems, we therefore get, that SPPA and two-sided PM are equivalent. It is, thus, enough to show that the PM and two-sided PM problems are equivalent. Since PM is a special case of two-sided PM, where no don't care symbol appears in the database vectors set D, and may appear only in the query vector q, we need only show how to convert an input of the two-sided PM to an input of PM. Such a transformation can be achieved by a special coding of the symbols of the two-sided PM problem input. Symbols of the dictionary D vectors are coded in the following way:"0" is coded by "01","1" is coded by "10" and "$\star$ is coded by "00". Symbols of the query vector are coded in the following way: "0" is coded by "$0\star$", "1" is coded by "$\star 0$" and "$\star$" is coded by "$\star\star$". This is concluded in Lemma 15.

▶ **Lemma 15.** *There exists a linear time and space transformation from the two-sided PM problem d-dimensional input vectors $v \in D$ to PM problem $2d$-dimensional input vectors $t_d(v) \in t_d(D)$, such that a d-dimensional input vector $v \in D$ matches a given d-dimensional query vector q of the two-sided PM problem if and only if the $2d$-dimensional input vector $t_d(v) \in t_d(D)$ matches a given $2d$-dimensional query vector $t_q(q)$ of the PM problem.*

We have, therefore, proven Corollary 16.

▶ **Corollary 16.** *Any algorithm $\mathcal{A}lg$ that solves PM in query time $O(q)$ and $O(\mathcal{S})$ space can be used to solve the two-sided PM and SPPA problems in $O(q)$ query time and $O(\mathcal{S})$ space.*

Remark on a Computational Difference of PM and Two-Sided PM. Note that the transformation from two-sided PM to PM has a blow-up in the number of don't care symbols, which is linear in the size of the vectors. Thus, despite Corollary 16, algorithms solving PM efficiently assuming a bounded number of don't cares (such as [18]) cannot be used to efficiently solve the two-sided PM or SPPA problems.

Corollary 16 enables to apply positive results on PM (e.g. [16], which is independent of the number of don't care symbols) on both the two-sided PM and SPPA problems. We are specifically interested in the following result of [33][3].

[3] [33] refer to their result as a solution to PM, however, they actually solve two-sided PM.

▶ **Theorem 17** (Theorem 1.2 of [33]). *Let $\Sigma = \{\sigma_1, \dots, \sigma_k\}$ and let $\star$ be an element such that $\star \notin \Sigma$. For any set of strings $x_1, \dots, x_n \in (\Sigma \cup \star)^d$ with $d \leq n$, after $\tilde{O}(nd)$-time preprocessing, there is an $O(nd)$ space data structure such that, for every query string $q \in (\Sigma \cup \star)^d$, it is possible to answer online whether q matches x_i, for every $i = 1, \dots n$, in $O(nd \log k / 2^{\Omega(\sqrt{\log d})})$ amortized time over $2^{\omega(\log d)}$ queries.*

Moreover, we observe that the same bounds of Theorem 17 can be achieved for a *dynamic* set of strings, supporting updates (insertion and deletions) in $\tilde{O}(d)$ time. We give a brief description of the [33] solution and explain our observation next.

A Dynamic Two-Sided PM Solution. First, for simplicity assume that $n = d$. The case $d \leq n$ is handled using n/d-splitting technique. Build an $n \times n$ matrix A over $(\Sigma \cup \{\star\})^{n \times n}$ such that $A[i, j] = x_i[j]$. The matrix A is then transformed to a boolean matrix, as follows. Let $S_1, T_1, \dots, S_k, T_k \in [2 \log k]$ be a collection of subsets such that for all i, $|S_i \cap T_i| = \phi$, yet for all $i \neq j$, $|S_i \cap T_j| \neq \phi$. Such a collection exists, by simply taking (for example) S_i to be the ith subset of $[2 \log k]$ having exactly $\log k$ elements (in some ordering on sets), and taking T_i to be the complement of S_i. Extend the matrix A to an $n \times (2n \log k)$ boolean matrix B, by replacing every occurrence of σ_i with the $(2 \log k)$-dimensional row vector corresponding to S_i, and every occurrence of $\star$ with the $(2 \log k)$-dimensional row vector which is all-zeroes.

When a query vector $q \in (\Sigma \cup \{\star\})^n$ is received, convert q into a boolean (column) vector v by replacing each occurrence of σ_i with the $(2 \log k)$-dimensional (column) vector corresponding to T_i, and every occurrence of $\star$ by the $(2 \log k)$-dimensional (column) vector which is all-zeroes. Compute Av using the Online Matrix Vector multiplication algorithm of [33]. For all $i = 1, \dots, n$, q matches x_i if and only if the ith row of B is *orthogonal* to v. The two vectors are orthogonal if and only if for all $j = 1, \dots, n$, either the ith row of B contains the all-zero vector in entries $(j-1)(2 \log k) + 1, \dots, j(2 \log k)$, or in those entries B contains the indicator vector for a set S_ℓ and correspondingly v contains either $\star$ or a set $T_{\ell'}$ such that $S_\ell \cap T_{\ell'} \neq \phi$, i.e., x_i and q match in the jth symbol. That is, the two vectors are orthogonal if and only if q matches x_i. Therefore, Av reports for all $i = 1, \dots, n$ whether q matches x_i or not.

The important observation is that the transformation of each string to a matrix row is independent. Therefore, strings/vectors can be added/deleted from the set in time proportional to the transformation time, which is $\tilde{O}(d)$. The above solution can be still used for the dynamic set as long as we have enough (at least d) strings/vectors in the set. When the set of strings is less than d, we may use a naive solution instead.

This leads to Corollary 18.

▶ **Corollary 18** (Dynamic SPPA Online Computation). *Let A be a set of n partial permutations. After $\tilde{O}(n|\Sigma|)$-time preprocessing, there is an $O(n|\Sigma|)$ space dynamic data structure supporting update operation (insertion or deletion to A) in $\tilde{O}(|\Sigma|)$ time, such that, for every query partial permutation π, it possible to answer online whether π agrees with $\pi_i \in A$, for every $i = 1, \dots n$, in $O(n|\Sigma| \log |\Sigma| / 2^{\Omega(\sqrt{\log |\Sigma|})})$ amortized time over $2^{\omega(\log |\Sigma|)}$ queries.*

3.3 Almost Full Permutations

In this subsection we consider the PPA and SPPA problems in a special case, where there are only a few symbols in Σ that don't participate in the bijection pairs set. Formally,

▶ **Definition 19** (Almost Full Partial Permutation). *Let π be a partial permutation over Σ, i.e., a bijection mapping a subset $\Sigma_1 \subset \Sigma$ to a subset $\Sigma_2 \subset \Sigma$, where $|\Sigma_1| = |\Sigma_2|$. We call π an* almost full *partial permutation if $|\Sigma| - |\Sigma_1| = k$, where $k! = O(poly(|\Sigma|))$ and $poly(|\Sigma|)$ is some polynomial in the size of Σ.*

We first describe an efficient solution for the problems over permutations. Formally,

▶ **Definition 20** (The Equal Permutations Problem (EP)).
Input: *Sets B_1, B_2 of size n, of permutations over alphabet Σ.*
Output: *All pairs (π_i, π_j), where $\pi_i \in B_1, \pi_j \in B_2$ and $\pi_i = \pi_j$.*

As above, in the non-batch version of the problem the size of the two sets is different: B_1 has size n, where B_2, the query, has size 1. We refer to it as the *single query EP* problem, denoted as SEP.

Efficient Solution for the SEP and EP Problems. The SEP problem can be easily solved by using a *dimension reduction* in the representation of the permutation from $|\Sigma|$ dimensions to a single dimension, assigning each permutation a unique number in $O(|\Sigma|)$ time, assuming RAM model with $O(\log |\Sigma|)$ word size (as done in [35]). The unique numbers representing the permutations $\pi_i \in B_1$ are saved in a hash table, in which we look for the unique number assigned to the query permutation. The assignment of a unique number to a permutation, $num(\pi)$, is forming a $|\Sigma|$-radix number representing π. It can be done in several ways, each with complexity $O(|\Sigma|)$, as follows.

1. Assuming, without loss of generality, that $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_{|\Sigma|-1}\}$, and define $|\sigma_i| = i$. Let $\pi = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_{|\Sigma|}}$ then: $num(\pi) = |\sigma_{i_1}| \cdot |\Sigma|^{|\Sigma|-1} + |\sigma_{i_2}| \cdot |\Sigma|^{|\Sigma|-2} + \ldots + |\sigma_{i_{|\Sigma|}}|$.
2. By using the technique suggested by [38], where permutations are ranked according to the indices that are swapped in the process of converting the current permutation to the identity permutation.

Consequently, the EP problem can be solved in $q = O(n \cdot |\Sigma|)$ time, and $\mathcal{S} = O(n)$ space due to an observation similar to Observation 8.

Now, an efficient solution for SPPA and PPA in the almost-full partial permutations special case can be achieved via reduction of SPPA to the SEP problem. This is done by creating for each almost full partial permutation π in A_1, the $k!$ possible permutations derived from π by specifying all the choices to add the symbols that do not already appear in π. This is also done for the single query almost full partial permutation.

For example, let $\Sigma = \{a, b, c, d\}$, $k = 2$, $S_1 = \{\pi_1 = (a \mapsto b, b \mapsto a), \pi_2 = (a \mapsto c, c \mapsto a)\}$ and $q = (a \mapsto c, b \mapsto b)$. Denote the set of full permutations derived from a partial permutation π by $full(\pi)$. Hence, $full(\pi_1) = \{bacd, badc\}$, $full(\pi_2) = \{cbad, cdab\}$. Thus, $full(S_1) = \{bacd, badc, cbad, cdab\}$ and $full(q) = \{cbad, cbda\}$. Therefore, $full(q)$ and $full(S_1)$ have a matching pair due to *cbad*.

The SPPA is then solved using the above SEP solution with $O(k! \cdot |\Sigma| \cdot n)$ preprocessing time, $q = O(k! \cdot |\Sigma|)$ query time and $\mathcal{S} = O(k! \cdot n)$ space, where a hash table is used for the numbers of the $O(k! \cdot n)$ permutations derived from the n partial permutations of A_1. Consequently, PPA can be solved in preprocessing $O(k! \cdot |\Sigma| \cdot n)$ time, $q = O(k! \cdot |\Sigma| \cdot n)$ query time and $\mathcal{S} = O(k! \cdot n)$ space due to an observation similar to Observation 8. Moreover, the solution described above supports maintenance of the database set A_1 dynamically, as each partial permutation can be deleted from or inserted to A_1 in $O(k! \cdot |\Sigma|)$ time. This gives Theorem 21.

▶ **Theorem 21.** *SPPA for almost full partial permutations can be solved in preprocessing $O(k! \cdot |\Sigma| \cdot n) = O(poly(|\Sigma|) \cdot n)$ time, insertion and deletion in $O(k! \cdot |\Sigma|) = O(poly(|\Sigma|))$ time, $q = O(k! \cdot |\Sigma|) = O(poly(|\Sigma|))$ query time and $\mathcal{S} = O(k! \cdot n) = O(poly(|\Sigma|) \cdot n)$ space.*

4 Algorithmic Applications of Partial Permutations

In this section we describe specific computational tasks stemming from computational biology, image processing and pattern matching, for which our study of partial permutations applies. Due to space limitations, we give here only the applications to genes sequences comparison and solving string PDMOG. The description of the application to color transformation data augmentation is postponed to the full version.

4.1 Application to Genes Sequences Comparison

In this subsection, we describe the application of the results in Section 3 to genes sequences comparison.

Genes Sequences Comparison. A family of genomes is often modeled as a set of permutations on genes that are common to all organisms of the family, as in [10]. A limitation in this type of models, comes from the difficulty to identify a large number of genes that are common even to a relatively small set of organisms. This is due, in part, to incompleteness in functional annotations of genes in public websites, and also to the difficulty of determining orthology relationships among genes in different genomes, since this relationship is known to be many-to-many. On the contrary, ubiquitous genes, such as ribosomal genes are often the ones best preserved in different species both in sequence and order and thus provide little valuable information in a gene-order based analysis [45].

Therefore, classifying species based on genes order in case of missing genes and, thus, incomplete permutations, is suggested as a better approach [45]. Furthermore, [45] point out that the occurrence of incomplete permutations with missing elements renders the classification problem more computationally challenging and has received limited attention. The study of partial permutations in this paper, enables to address this computational problem formalization as well as better understand its algorithmic computational challenge.

Incompleteness is formalized as partial words and studied in comparing genes, where the "alphabet" is small and, therefore, repetitiveness is expected. The *domain* $D(w)$ of a partial word w is the set of all positions i such that $w[i]$ is defined, i.e., is not a hole. An alignment of two sequences can be viewed as a construction of two partial words that are *compatible* in the following sense [11]. Given two partial words x and y of the same length, we say that x is contained in y or that y contains x, and we write $x \subset y$, if $D(x) \subset D(y)$ and $x[k] = y[k]$ for all $k \in D(x)$. Two words x and y are *compatible* if there exists a word z that contains both x and y. In this case, the smallest word containing x and y is defined by $D(x) \cup D(y)$.

Note that, two equal length partial words x and y that are compatible must agree on the positions in $D(x) \cap D(y)$, however, there is no requirement on the positions in $D(x) \cup D(y) \setminus (D(x) \cap D(y))$. In particular, it may have repeating symbols. Thus, applying the notion of compatibility in order to compare words in the special case of partial permutations suffers from the following inconsistency: given two partial permutations x and y, we have that the smallest word that contains both x and y, $D(x) \cup D(y)$, is not necessarily also a partial permutation. It also has the undesirable side-effect of generating artificial sequences.

The definition of agreement between partial permutations gives a consistent comparison for genes sequences as well as preserves the original input sequences. Thus, our study enables to point out possibly fruitful versus futile methods for efficient genes sequences comparison.

Our Results Applied to Genes Sequences Comparison. Note, that the basic building block for classification tasks is the comparison operation between a pair of genes sequences. Given a set of n gene sequences over a set of d identified family of genes. Considering the formalization of gene sequences as partial permutations explained above, our definition of *agreement* between partial permutations not only suggests such a building block, but also enables to differentiate situations where the problem can be efficiently computed from situations it probably cannot.

Specifically, the application of the results in Section 3 to genes sequences comparison gives the following. For a family of d identified genes, and a set of n partial permutations representing genes sequences over d, we have that:

- For any $\epsilon > 0$, there exists $c > 0$ such that, if $d = c \log n$, then finding the genes sequences that agree with a query genes sequence is not likely to be answered in $O(n^{1-\epsilon} \cdot d)$ time using $O(n \cdot d)$ (linear) space (unless the Strong Exponential Time Hypothesis (SETH) is false). This follows from Corollary 12 and Observation 11 in Subsection 3.1.
- If $d = \Theta(\log n)$, after $\tilde{O}(n \cdot d)$-time preprocessing, there is an $O(n \cdot d)$ space dynamic data structure supporting updates (insertion and deletion) in $\tilde{O}(d)$ time, such that, finding genes sequences that agree with a query gene sequence can be done in $O(n \cdot d \log d / 2^{\Omega(\sqrt{\log d})})$ amortized time over $2^{\omega(\log d)}$ queries. This follows from Corollary 18 in Subsection 3.2.
- If the gene sequences of both the database and query are almost full, then there is an $O(k! \cdot n) = O(poly(d) \cdot n)$ space dynamic data structure supporting updates (insertion and deletion) in $O(k! \cdot d) = O(poly(d))$ time, such that finding genes sequences that agree with a query genes sequence can be done in $O(k! \cdot d) = O(poly(d))$ time. This follows from Theorem 21 in Subsection 3.3.

4.2 Application to Solving Strict PDMOG

In this subsection, we describe the application of the results in Section 3 to solving the strict PDMOG problem.

Strict PDMOG. Two equal-length strings are a parameterized match, denoted by *p-match*, if there exists a bijection on their alphabet symbols under which one string matches the other. The PDMOG problem is motivated by the critical modern concern of cyber security. Network intrusion detection systems (NIDS) perform protocol analysis, content searching and content matching, in order to detect harmful software that may appear on several packets requiring *gapped matching* [30]. A *gapped pattern* P is one of the form $lp\ \{\alpha, \beta\}\ rp$, where each sub-pattern lp, rp is a string over alphabet Σ, and $\{\alpha, \beta\}$ matches any substring of length at least α and at most β . Several versions of gapped dictionary matching problems were studied recently (see [5, 6, 3, 2, 35, 7, 36]). The *Parameterized Dictionary Matching with One Gap problem* (PDMOG) is defined as follows [40]. Preprocess a dictionary D of d single-gap gapped patterns $P_1, \ldots, P_d$ over alphabet $\Sigma' \cup \Sigma$, such that $\Sigma' \cap \Sigma = \emptyset$, so that given a query text T of length n over alphabet $\Sigma' \cup \Sigma$, $\Sigma' \cap \Sigma = \emptyset$, output all locations ℓ in T, where there exist bijections $f_1, f_2 : \Sigma \to \Sigma$ and the following hold for any $P_i \in D$, and a gap length $g \in [\alpha_i, \beta_i]$, where α_i, β_i are the gap boundaries of P_i :

1. $\forall lp_i[j] \in \Sigma'$, $lp_i[j] = T[\ell - |lp_i| - g - |rp_i| + j]$.
2. $\forall lp_i[j] \in \Sigma$, $f_1(lp_i[j]) = T[\ell - |lp_i| - g - |rp_i| + j]$.
3. $\forall rp_i[j] \in \Sigma'$, $rp_i[j] = T[\ell - |rp_i| + j]$.
4. $\forall rp_i[j] \in \Sigma$, $f_2(rp_i[j]) = T[\ell - |rp_i| + j]$.

The *strict* PDMOG problem enforces both left and right sub-patterns to have the same parameterized matching (p-match) function, i.e., that $f_1 = f_2$, which is more reasonable if the encodings of both sub-patterns of a dictionary pattern are done simultaneously [35].

Online strict PDMOG was studied by [35] obtaining algorithms that are fast for some practical inputs called *alphabet saturated dictionary*, where dictionary sub-patterns contain the same alphabet symbols enabling to represent mapping functions of dictionary sub-patterns as permutations. While this assumption is reasonable if the alphabet size is relatively small and the dictionary sub-patterns are not very short, it is still a rigid restriction. Dealing with general alphabet dictionary requires a tool for efficient partial permutations representation and manipulation, which its existence is excluded in this paper (unless the SETH hypothesis is false), showing that an efficient solution for the strict PDMOG problem over a general dictionary alphabet is not likely. Thus, we answer negatively to an open question posed by [35].

Our Results Applied to strict PDMOG. The core issue of the strict PDMOG solution is that while scanning the text, the algorithm locates p-matches of the left sub-patterns of the dictionary D and maintains the partial permutations via which they were p-matched to the text. The algorithm also locates a set of right sub-patterns of the dictionary patterns in D which p-match the current text location. The algorithm needs to verify which of the right sub-patterns are p-matched via a partial permutation that *agrees* with any of the (dynamically changing) set of partial permutations that were used to p-match left sub-patterns that were located within an *active window* of locations determined by the relevant gap bounds of the dictionary D.

The online strict PDMOG was studied by [35] obtaining algorithms that are fast for some practical inputs called *alphabet saturated dictionary*, where dictionary sub-patterns contain the same alphabet symbols. Therefore, the algorithms of [35] represent mapping functions of dictionary sub-patterns as permutations and can efficiently maintain the dynamically changing set of permutations that were used in order to p-match the left sub-patterns of the dictionary D basically using the dimension reduction idea described in Subsection 3.3 for the representation of permutations[4].

While this assumption is reasonable if the alphabet size is relatively small and the dictionary sub-patterns are not very short, it is still a rigid restriction. Dealing with general alphabet dictionary requires a tool for efficient maintenance of partial permutations. The discussion and results of Subsections 3.1, 3.2, 3.3 can be, therefore, applied to conclude regarding the possibility to efficiently solve the online strict PDMOG problem. In order to simplify the discussion and avoid getting into unnecessary details ([35] use various techniques and several parameters to specify complexity), we summarize the application of the discussion above using the following parameters: s_L - the size of the set S_L of p-matched left sub-patterns of dictionary gapped patterns within the current active window of the text, $|\Sigma|$ - the dictionary D and text T alphabet size. We also need the following definition.

► **Definition 22** (A k-Saturated Dictionary). *Let D be a gapped patterns dictionary over alphabet Σ. We call D a k-*saturated dictionary *if every sub-pattern in D is over Σ_1, such that $\Sigma_1 \subseteq \Sigma$ and $k = |\Sigma| - |\Sigma_1|$, where $k! = O(poly(|\Sigma|))$ and $poly(|\Sigma|)$ is some polynomial in the size of Σ.*

[4] The application of this idea in [35] is slightly more involved, since it is combined with the use of range reporting data structures and other details of their algorithms.

The applications to the solution of strict PDMOG that we have shown can, therefore, be summarized as follows:

- For any $\epsilon > 0$, there exists $c > 0$ such that, for a general alphabet Σ with size $|\Sigma| = c \log s_L$, finding partial permutations, via which the sub-patterns in S_L were p-matched to the text, that agree with a partial permutation p-matching of a currently located right sub-pattern is not likely to be answered in $O(s_L^{1-\epsilon} \cdot |\Sigma|)$ time using $O(s_L \cdot |\Sigma|)$ (linear) space (unless the Strong Exponential Time Hypothesis (SETH) is false). This follows from Corollary 12 and Observation 11 in Subsection 3.1.
- For a general alphabet Σ with size $|\Sigma| = \Theta(\log s_L)$, after $\tilde{O}(s_L \cdot |\Sigma|)$-time preprocessing, there is an $O(s_L \cdot |\Sigma|)$ space dynamic data structure supporting updates (insertion and deletion) in $\tilde{O}(|\Sigma|)$ time, such that, finding partial permutations, via which the sub-patterns in S_L were p-matched to the text, that agree with a partial permutation p-matching of a currently located right sub-pattern can be done in $O(s_L|\Sigma| \log |\Sigma| / 2^{\Omega(\sqrt{\log |\Sigma|})})$ amortized time over $2^{\omega(\log |\Sigma|)}$ queries. This follows from Corollary 18 in Subsection 3.2.
- For a k-saturated dictionary D, there is an $O(k! \cdot s_L) = O(poly(|\Sigma|) \cdot s_L)$ space dynamic data structure supporting updates (insertion and deletion) in $O(k! \cdot |\Sigma|) = O(poly(|\Sigma|))$ time, such that finding partial permutations, via which the sub-patterns in S_L were p-matched to the text, that agree with a partial permutation p-matching of a currently located right sub-pattern can be done in $O(k! \cdot |\Sigma|) = O(poly(|\Sigma|))$ time. This follows from Theorem 21 in Subsection 3.3.

Note that, that if $|\Sigma| = \Theta(\log s_L)$ and $k! = O(|\Sigma|)$ (i.e., $poly(|\Sigma|)$ is actually linear in $|\Sigma|$), then the third result gives a linear (up to a logarithmic factor) space dynamic data structure for maintaining partial permutations with update and query time logarithmic in s_L.

5 Conclusion

This paper examined the use of *partial permutations* in algorithmic tasks. Some interesting related open questions are:

- Can an efficient solution for PPA/SPPA be achieved for other (practically interesting) special cases?
- What other applications require (possibly hidden) maintenance of partial permutations?

It is our belief that being a relatively basic mathematical concept, partial permutations play a hidden role in more applications. We, therefore, expect more research on the topic in order to explore their algorithmic use.

References

1 A. Abboud, R. Williams, and H. Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 218–230, 2015.

2 A. Amir, T. Kopelowitz, A. Levy, S. Pettie, E. Porat, and B. R. Shalom. Mind the gap! online dictionary matching with one gap. *Algorithmica*, 81(6):2123–2157, 2019.

3 A. Amir, T. Kopelowitz, A. Levy, S. Pettie, E. Porat, and B. R. Shalom. Mind the gap: Essentially optimal algorithms for online dictionary matching with one gap. In *27th International Symposium on Algorithms and Computation, ISAAC*, pages 12:1–12:12, Sydney, Australia, December 12-14, 2016.

4 A. Amir and A. Levy. String rearrangement metrics: A survey. In *Algorithms and Applications*, volume 6060 of *Lecture Notes in Computer Science*, pages 1–33. Springer, 2010.

5 A. Amir, A. Levy, E. Porat, and B. R. Shalom. Dictionary matching with one gap. In Alexander S. Kulikov, Sergei O. Kuznetsov, and Pavel A. Pevzner, editors, *Combinatorial Pattern Matching - 25th Annual Symposium, CPM 2014, Moscow, Russia, June 16-18, 2014. Proceedings*, volume 8486 of *Lecture Notes in Computer Science*, pages 11–20. Springer, 2014.

6 A. Amir, A. Levy, E. Porat, and B. R. Shalom. Dictionary matching with a few gaps. *Theor. Comput. Sci.*, 589:34–46, 2015.

7 A. Amir, A. Levy, E. Porat, and B. R. Shalom. Online recognition of dictionary with one gap. *Information and Computation*, 275:104633, 2020.

8 V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25:272–289, 1996.

9 B. S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 71–80, San Diego, CA, USA, May 16-18, 1993.

10 A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. *Journal of Computational Biology*, 13(7):1340–1354, 2006.

11 J. Berstel and L. Boasson. Partial words and a theorem of Fine and Wilf. *Theoretical Computer Science*, 218(1):135–141, 1999.

12 B. Blakeley, F. Blanchet-Sadri, J. Gunter, and N. Rampersad. *Developments in Language Theory*, volume 5583 of *LNCS*, chapter On the Complexity of Deciding Avoidability of Sets of Partial Words, pages 113–124. Springer, 2009.

13 F. Blanchet-Sadri, N. C. Brownstein, A. Kalcic, J. Palumbo, and T. Weyand. Unavoidable sets of partial words. *Theory of Computing Systems*, 45(2):381–406, 2009.

14 A. Burstein and I. Lankham. Restricted patience sorting and barred pattern avoidance. *Permutation patterns*, 376:233–257, 2010.

15 T. M. Chan and R. Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1246–1255, 2016.

16 M. Charikar, P. Indyk, and R. Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 451–462, 2002.

17 A. Claesson, V. Jelínek, E. Jelínková, and S. Kitaev. Pattern avoidance in partial permutations. *Electronic Journal of Combinatorics*, 18(1), 2011.

18 R. Cole, L. Gottlieb, and M. Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proceedings of 36 Annual ACM Symposium on Theory of Computing (STOC)*, pages 91–100, 2004.

19 T. H. Cormen, C. E. Leiserson, L. R. Rivest, and C. Stein. *Introduction to Algorithms*, chapter 14.3: Interval Trees, pages 348–353. MIT Press and McGraw-Hill, 3rd edition, 2009.

20 M. Equi, V. Mäkinen, and A. I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails. In *Proceedings of the 47th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, volume 12607 of *LNCS*, pages 608–622, 2021.

21 B. Esslinger. Cryptography and mathematics, 2011.

22 V. Halava, T. Harju, and T. Kärki. Square-free partial words. *Information Processing Letters*, 108(5):290–292, 2008.

23 V. Halava, T. Harju, T. Kärki, and P. Séébold. Overlap-freeness in infinite partial words. *Theoretical Computer Science*, 410(8-10):943–948, 2009.

24 S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *Proceedings of the 27th annual ACM symposium on the theory of computing*, pages 178–187, 1995.

25 G. Hoppenworth, J. W. Bentley, D.Gibney, and S. V. Thankachan. The fine-grained complexity of median and center string problems under edit distance. In *28th Annual European Symposium on Algorithms, ESA*, volume 173 of *LIPIcs*, pages 61:1–61:19, 2020.

26 A. G. Howard. Some improvements on deep convolutional neural network based image classification. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

27 R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

28 K.Bingmann, P. Gawrychowski, S. Mozes, and O. Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless APSP can). *ACM Trans. Algorithms*, 16(4):48:1–48:22, 2020.

29 D. E. Knuth. *The Art of Computer Programming*, volume 1-3 Boxed Set. Reading, Massachusetts: Addison-Wesley, 2nd edition, 1998.

30 M. Krishnamurthy, E. S. Seagren, R. Alder, A. W. Bayles, J. Burke, S. Carter, and E. Faskha. *How to cheat at securing linux*. Syngress Publishing, Inc., Elsevier, Inc., 2008.

31 C. Y. Ku and I. Leader. An Erdös-Ko-Rado theorem for partial permutations. *Discrete Mathematics*, 306(1):74–86, 2006.

32 G. M. Landau, A. Levy, and I. Newman. LCS approximation via embedding into locally non-repetitive strings. *Information and Computation*, 209(4):705–716, 2011.

33 K. G. Larsen and R. Williams. Faster online matrix-vector multiplication. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2182–2189, 2017.

34 P. Leupold. Partial words for DNA coding. In *10th International Meeting on DNA Computing (DNA 10)*, volume 3384 of *LNCS*, pages 224–234. Springer-Verlag, Berlin, 2005.

35 A. Levy and B. R. Shalom. Online parameterized dictionary matching with one gap. *Theoretical Computer Science*, 845(14):208–229, 2020.

36 A. Levy and B. R. Shalom. A comparative study of dictionary matching with gaps: Limitations, techniques and challenges. *Algorithmica*, 84:590–638, 2022.

37 Y. Li, G. Hu, Y. Wang, T. M. Hospedales, N. M. Robertson, and Y. Yang. DADA: differentiable automatic data augmentation. *CoRR*, abs/2003.03780, 2020. arXiv:2003.03780.

38 W. J. Myrvold and F. Ruskey. Ranking and unranking permutations in linear time. *Inf. Process. Lett.*, 79(6):281–284, 2001.

39 R. L. Rivest. *Analysis of Associative Retrieval Algorithms*. PhD thesis, Stanford University, 1974.

40 B. R. Shalom. Parameterized dictionary matching with one gap. *Theoretical Computer Science*, 854(1):1–16, 2021.

41 A. M. Shur and Y. V. Konovalova. On the periods of partial words. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 2136 of *LNCS*, pages 657–665. Springer-Verlag, 2001.

42 H. Straubing. A combinatorial proof of the Cayley-Hamilton theorem. *Discrete Mathematics*, 43(2-3):273–279, 1983.

43 L. Taylor and G. Nitschke. Improving deep learning with generic data augmentation. In *IEEE Symposium Series on Computational Intelligence, SSCI 2018, Bangalore, India, November 18-21, 2018*, pages 1542–1547. IEEE, 2018.

44 V. V. Williams. On some fine-grained quesions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians (ICM)*, pages 3447–3487, 2019.

45 X. Zhou, A. Amir, C. Guerra, G. M. Landau, and J. Rossignac. EDoP distance between sets of incomplete permutations: Application to bacteria classification based on gene order. *Journal of Computational Biology*, 25(11):1193–1202, 2018. doi:10.1089/cmb.2018.0063.