Pattern Matching under Weighted Edit Distance

Panagiotis Charalampopoulos
King's College London
United Kingdom

Tomasz Kociumaka
Max Planck Institute for Informatics
Saarland Informatics Campus
Saarbrücken, Germany

Philip Wellnitz
National Institute of Informatics
The Graduate University for Advanced Studies, SOKENDAI
Tokyo, Japan

Abstract

In Pattern Matching with *Weighted* Edits (PMwWE), we are given a pattern P of length m, a text T of length n, a positive threshold k, and oracle access to a weight function that specifies the costs of edits (depending on the involved characters, and normalized so that the cost of each edit is at least 1). The goal is to compute the starting positions of all fragments of T that can be obtained from P with edits of total cost at most k. PMwWE captures typical real-world applications more accurately than its unweighted variant (PMwE), where all edits have unit costs.

Indeed, the textbook $O(nm)$-time algorithm of Sellers [J. Algorithms'80], devised in the context of bioinformatics, already accounts for weights. Surprisingly, the understanding of PMwWE has not advanced in the last 45 years. In contrast, significant milestones for PMwE include an $O(nk)$-time algorithm by Landau and Vishkin [STOC'86, J. Algorithms'89], an $O(n + k^4 \cdot n/m)$-time algorithm by Cole and Hariharan [SODA'98, SICOMP'02], and a recent $\widetilde{O}(n + k^{3.5} \cdot n/m)$-time solution by Charalampopoulos, Kociumaka, and Wellnitz [FOCS'22].

In this work, we examine whether these results can be lifted to PMwWE even though (1) the underlying algorithms rely on combinatorial properties specific to the unweighted edit distance, and (2) under standard fine-grained complexity assumptions, computing the weighted edit distance is strictly harder than computing the unweighted edit distance [Cassis, Kociumaka, and Wellnitz; FOCS'23]. We obtain three main results:

- a conceptually simple $\widetilde{O}(nk)$-time algorithm for PMwWE, very different from that of Landau and Vishkin;
- a significantly more complicated $\widetilde{O}(n + k^{3.5} \cdot W^4 \cdot n/m)$-time algorithm for PMwWE under the assumption that the weight function is a metric with integer values between 0 and W; and
- an $\widetilde{O}(n + k^4 \cdot n/m)$-time algorithm for PMwWE for the case of arbitrary weights.

In the setting of metrics with small integer values, we nearly match the state of the art for PMwE where $W = 1$.

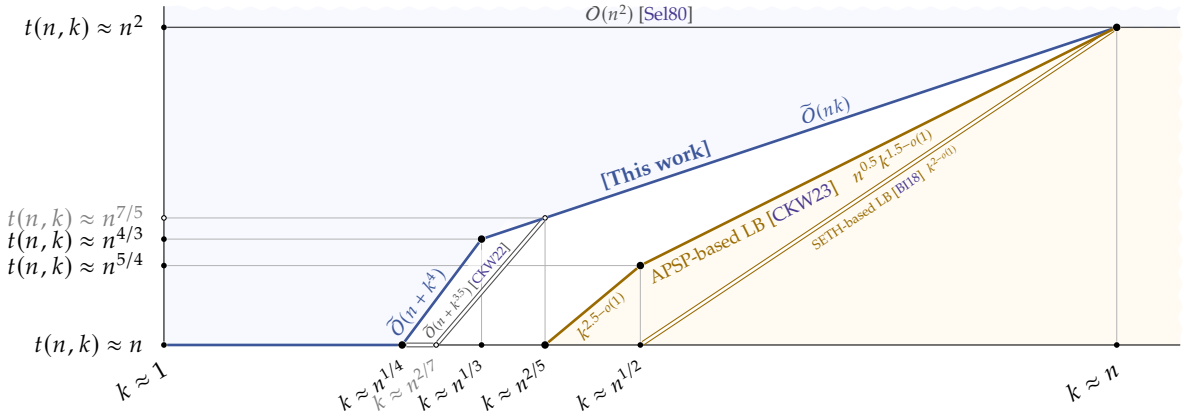


Figure 1. The bounds on the running time $t(n,k)$ of algorithms for the PMwWE problem as a function of k for the important case when $m = \Theta(n)$. The scales are logarithmic and sub-polynomial factors are hidden. Doubled lines represent bounds for PMwE, included only when they differ by a polynomial factor from their PMwWE counterparts. The bounds for PMwE remain valid for PMwWE with integer edit weights between 0 and $W = n^{o(1)}$.

Contents

1 Introduction

Pattern matching (string matching) models an action most of us perform daily when we search for a term on the web or a word in a document: find all fragments of a text T that are identical to a pattern P. While time-optimal solutions for this fundamental algorithmic problem date back to the 1970s [MP70, KMP77], exact pattern matching remains very restrictive. For example, a single typographical error in the pattern results in all occurrences being missed. Hence, both the theory and the practice communities have made great efforts to devise algorithms that efficiently compute *approximate occurrences* of P in T—fragments of T that are "close" to P.

Among many ways to quantify the notion of "closeness", one of the most natural and well-studied is to impose an upper bound k on the edit distance between the pattern and its sought approximate occurrences. The (weighted) edit distance $\mathsf{ed}^w(X \to Y)$ from a string X to a string Y is the minimum cost of transforming X into Y by character edits (insertions, deletions, and substitutions). The cost of each edit depends on the edit type and the characters involved, and is specified through a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}$, where $\bar{\Sigma}$ denotes the union of the alphabet Σ and a special symbol ε representing the empty string. For $a, b \in \Sigma$, the cost of deleting a is $w(a \mapsto \varepsilon)$, the cost of inserting b is $w(\varepsilon \mapsto b)$, and the cost of substituting a for b is $w(a \mapsto b)$.

Problem Pattern Matching with Weighted Edits, PMwWE(P, T, k, w).
Input. A pattern P of length m, a text T of length n, an integer $k > 0$, and oracle access to a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}$ normalized so that $w(a \mapsto a) = 0$ and $w(a \mapsto b) \geq 1$ for $a \neq b \in \bar{\Sigma}$.[1]
Output. The set $\mathrm{Occ}_k^w(P, T) := \{i : \exists_{j \geq i}\ \mathsf{ed}^w(P \to T[\,i\,..\,j\,)) \leq k\}$.

Applications of approximate pattern matching often incorporate domain-specific knowledge in the weight function. For example, in bioinformatics, scoring matrices are used to capture the frequency with which edits have been observed to happen in nucleotide and peptide sequences (for instance, due to mutations). Further scenarios where some edits are much more likely than others include optical character recognition and accounting for spelling mistakes.

Unfortunately, apart from the initial $O(nm)$-time algorithm for PMwWE [Sel80], algorithmic results and improvements were—perhaps surprisingly—limited to the important, but significantly easier and somewhat theoretical special case of the *unweighted* edit distance $\mathsf{ed}(X, Y)$, where the weight $w(a \mapsto b)$ for every two elements $a \neq b$ of $\bar{\Sigma}$ is uniformly set to 1.[2]

Problem Pattern Matching with Edits, PMwE(P, T, k).
Input. A pattern P of length m, a text T of length n, and a positive integer k.
Output. The set $\mathrm{Occ}_k(P, T) := \{i : \exists_{j \geq i}\ \mathsf{ed}(P, T[\,i\,..\,j\,)) \leq k\}$.

Inspired by the landmark $O(nk)$ algorithm for PMwE [LV89], we ask the following natural question:

Does PMwWE admit an $\widetilde{O}(nk)$-time solution?[3]

Using a completely different and more complicated approach compared to [LV89], we provide an affirmative answer.

Main Theorem 1. *PMwWE can be solved in $O((n \log m + m \log^2 m) \cdot k)$ time.*

[1] Consistently with the literature [CH02, DGH+23, CKW23, MBCT23], our normalization allows the complexity of algorithms to depend on k. Without this assumption, the weight function could be scaled arbitrarily.
[2] This variant is often also named after Levenshtein [Lev65].
[3] The $\widetilde{O}(\star)$ notation suppresses factors that are polylogarithmic in the length of the input strings.

While it may seem surprising that a result such as Main Theorem 1 is obtained only now, good reasons prevented earlier, similar results. Let us focus on the regime of fairly large distances, $m^{0.5} \leq k \leq m$, where the $O(nk)$-time complexity remains the state of the art for PMwE. Without much oversimplification, the Landau–Vishkin algorithm [LV89] can be thought of as performing $O(n/k)$ unweighted edit distance computations, in $O(k^2)$ time each, between P and various fragments of T.

To adapt this approach to PMwWE, one needs to compute weighted edit distances instead. Applying the $O(mk)$-time algorithm [Ukk85, Mye86], which was the fastest known until very recently, one merely recovers the $O(nm)$ running time of [Sel80]. The $O(m + k^5)$-time algorithm of Das, Gilbert, Hajiaghayi, Kociumaka, and Saha [DGH+23] is the earliest one that could be adapted to yield a running time of $o(nm)$ for PMwWE using this approach. A subsequent $\widetilde{O}(m + \sqrt{mk^3})$-time solution by Cassis, Kociumaka, and Wellnitz [CKW23] means that we can now expect to solve PMwWE in $\widetilde{O}(n\sqrt{mk})$ time. This complexity, however, constitutes a natural barrier due to the fine-grained conditional lower bound of $m^{0.5}k^{1.5-o(1)}$ for computing the weighted edit distance when $m^{0.5} \leq k \leq m$ [CKW23].

Our key result behind Main Theorem 1 is an algorithm that, after $\widetilde{O}(mk)$-time preprocessing of one string (P in our application), computes the weighted edit distance to any other string in $\widetilde{O}(k^2)$ time. Crucially, we use the SMAWK algorithm [AKM+87] for the (min, +)-multiplication of a Monge matrix with a vector as well as the Multiple-Source Shortest Paths (MSSP) data structure of Klein [Kle05] for efficient computations of distances in planar graphs. Consult Section 2.1 for a more detailed description of our ideas and Section 4 for the full technical details.

A surprising consequence of Main Theorem 1 is that, for the regime of $m^{0.5} \leq k \leq m$, the time complexity of PMwWE is strictly better understood than that of PMwE: Whereas the state-of-the-art upper bounds essentially match, the fine-grained lower bound for PMwWE exceeds its PMwE counterpart by a polynomial factor. These two lower bounds, which we discuss next, are inherited from the problems of computing the weighted and unweighted edit distance, respectively.

As proved in [CKW22, Section 1], the optimality of the $\Theta(m + k^2)$-time algorithm by Landau and Vishkin [LV88] for unweighted edit distance (up to sub-polynomial factors and conditioned on the Orthogonal Vector Hypothesis) implies an $n + k^{2-o(1)} \cdot n/m$ lower bound for PMwE. The $m^{0.5}k^{1.5-o(1)}$ lower bound for computing weighted edit distance [CKW23] is conditioned on the All-Pairs Shortest Paths (APSP) Hypothesis [VW18] and can be formally stated as follows.

Theorem 10.2 [CKW23, Main Theorem 2]. *For real parameters $\delta > 0$ and $0.5 \leq \kappa \leq 1$, assuming the APSP Hypothesis, no algorithm, given strings X, Y of length at most m, a real threshold $1 \leq k \leq m^{\kappa}$, and oracle access to a normalized weight function w, decides if $\mathsf{ed}^{w}(X \to Y) \leq k$ in time $O(m^{0.5+1.5\kappa-\delta})$.*

Theorem 10.2 readily yields a $k^{1.5-o(1)} \cdot n/m^{0.5}$ lower bound for PMwWE when $m^{0.5} \leq k \leq m$.

Corollary 10.3. *For real parameters $\delta > 0$ and $0 < \kappa \leq 1$, assuming the APSP Hypothesis, there is no algorithm that solves all instances of PMwWE satisfying $k \leq m^{\kappa}$ in time $O(n \cdot m^{\min(1.5\kappa-0.5,\, 2.5\kappa-1)-\delta})$.*

In fact, the $O(nk)$-time algorithm of Landau and Vishkin [LV89] remains the fastest known for PMwE also in the case of $m^{0.4} \leq k \leq m^{0.5}$. In this parameter range, Corollary 10.3 yields a non-trivial conditional lower bound for PMwWE. In contrast, no such non-trivial lower bound is known for PMwE—once again, the time complexity of PMwWE is better understood than that of PMwE. Naturally, we thus ask:

Are there algorithms for PMwWE that nearly match the state of the art for PMwE when $k \leq m^{0.4}$?

Before providing a (partially positive) answer to this question, let us highlight milestone results for PMwE beyond the $O(nk)$-time Landau–Vishkin algorithm [LV89]. We refer the interested reader to the thorough survey of Navarro [Nav01] for a review of other early results. By exploiting the periodic structure of the involved strings and employing deterministic coin tossing, Sahinalp and Vishkin [SV96]

developed an $\widetilde{O}(n + k^{25/3} \cdot n/m^{1/3})$-time algorithm, which improves upon $O(nk)$ for $k \ll m^{1/22}$. Cole and Hariharan [CH02] extended this range to $k < m^{1/3}$ by presenting an $O(n + k^4 \cdot n/m)$-time solution shortly afterward.[4] The $O(nk)$-time algorithm of Landau and Viskhin [LV89] and the algorithm of Cole and Hariharan [CH02] were the state of the art for over two decades, until the recent breakthrough of Charalampopoulos, Kociumaka, and Wellnitz [CKW22], who presented an algorithm for PMwE that runs in time $\widetilde{O}(n + k^{3.5} \cdot n/m)$ and remains the fastest known solution for PMwE when $k \ll m^{0.4}$.

The second main contribution of this work is an algorithm for PMwWE that matches the $O(n+k^4 \cdot n/m)$ time complexity of [CH02] up to a small polylogarithmic factor. Consequently, our solutions for PMwWE essentially recover the state-of-the-art running times for PMwE as they were before 2022. Hence, $m^{0.25} \ll k \ll m^{0.4}$ is currently the only parameter regime when PMwE has a significantly faster ($\widetilde{O}(n + k^{3.5} \cdot n/m)$-time) solution than PMwWE; see Figure 1 for a visualization.

Main Theorem 3. *PMwWE can be solved in $O(n + k^4 \cdot n/m \cdot \log^2(mk))$ time.*

In order to prove Main Theorem 3, we exploit the structural characterization provided by Charalampopoulos, Kociumaka, and Wellnitz [CKW20] for the approximate occurrences of P in T under the *unweighted* edit distance: P has few approximate occurrences or it is almost periodic.

As shown in [CKW20], in the first case, $\mathrm{Occ}_k(P,T)$ can be computed in $\widetilde{O}(n + k^4 \cdot n/m)$ time and covered with $O(k \cdot n/m)$ intervals of size $O(k)$ each. Due to the assumption that the weight function is normalized, we have $\mathrm{Occ}^w_k(P,T) \subseteq \mathrm{Occ}_k(P,T)$, so it suffices to focus on the intervals covering $\mathrm{Occ}_k(P,T)$. One of the main results of Cassis, Kociumaka, and Wellnitz [CKW23] is that $\mathrm{ed}^w(P \to T) \le k$ can be decided in $\widetilde{O}(k^3)$ time assuming that one can check in $\widetilde{O}(1)$ time whether any two fragments of P or T match perfectly. We generalize this result (leveraging the underlying structural insights) and show that, if $|P| \le |T| + O(k)$, then $\widetilde{O}(k^3)$ time is sufficient to report all fragments of T at weighted edit distance at most k from P. For each interval J in the cover of $\mathrm{Occ}_k(P,T)$, we apply this procedure for an auxiliary text obtained by trimming T to length $m + O(k)$. This way, after the standard $O(n)$-time preprocessing for substring equality queries, $\mathrm{Occ}^w_k(P,T) \cap J$ can be computed in $\widetilde{O}(k^3)$ time, for a total of $\widetilde{O}(n + k^4 \cdot n/m)$.

The main challenge then lies in the almost periodic case, which is also the bottleneck for the PMwE problem. The main contribution of [CKW22] is how to compute $\mathrm{Occ}_k(P,T)$ in that case in $\widetilde{O}(n + k^{3.5} \cdot n/m)$ time using *dynamic puzzle matching*. We generalize an $\widetilde{O}(n + k^4 \cdot n/m)$-time algorithm provided in [CKW22] as a warm-up application of that technique.

The underlying idea is to preprocess a few subgraphs of the alignment graph (interpreted as *puzzle pieces*), and then stitch these subgraphs appropriately to reconstruct the distances between P and relevant fragments of T. Every piece can be associated with an $O(k) \times O(k)$ matrix so that stitching two pieces corresponds to computing the (min, +)-product of the underlying matrices. Unweighted edit distance enjoys strong combinatorial properties that allow representing each matrix in $O(k)$ space and computing the (min, +)-product in $\widetilde{O}(k)$ time [Tis15]. With weights, we have to resort to the naive $O(k^2)$-space representation and the $O(k^2)$-time SMAWK [AKM+87] algorithm for computing the (min, +)-product of two Monge matrices. In order to prevent the overall running time from increasing to $\widetilde{O}(n + k^5 \cdot n/m)$, we refine the approach of [CKW22] so that most matrix-matrix multiplications are replaced by matrix-vector multiplications; the latter can be performed in $O(k)$ time using the SMAWK algorithm [AKM+87].

[4] Cole and Hariharan [CH02] incorrectly claimed that their result generalizes to PMwWE. Specifically, they wrote that "the only change needed is to the Landau–Vishkin algorithm to take into account the differing costs". However, in the presence of weights, there is no monotonicity in the diagonals of the dynamic programming table, and hence the Landau–Vishkin algorithm [LV86, LV89] does not work due to its greedy nature. This is confirmed by the lower bound of [CKW23] for the computation of bounded weighted edit distance, which implies that one cannot hope for a procedure that computes $J \cap \mathrm{Occ}^w_k(P,T)$ for a given interval J of size $O(k)$ in time $O(k^2)$ after a linear-time preprocessing.

Another challenge of a more technical nature is that the dynamic puzzle matching algorithm of [CKW22] is formalized in terms of restricted permutation matrices, meaningful only for the unweighted edit distance. Instead, we would ideally like to directly use distance matrices but, due to the bounded width of the puzzle pieces, we must work with distances in various subgraphs that are difficult to describe explicitly. In order to abstract away these technicalities, we introduce a notion of a *fern matrix*, which can be interpreted as storing distances in the entire alignment graph but potentially failing to truthfully represent distances that are too large to be relevant for our computations. Further ideas behind our $\widetilde{O}(n + k^4 \cdot n/m)$-time algorithm are presented in Section 2.2.

A recent work of Gorbachev and Kociumaka [GK25] describes faster algorithms for weighted edit distance with small integer weights: they achieve a running time of $\widetilde{O}(m + W \cdot k^2)$ if the cost of each edit belongs to $[1\,..\,W]$. For small integer weights, such as when $W = \widetilde{O}(1)$, this complexity recovers (up to a polylogarithmic factor) the running time of the Landau–Vishkin algorithm [LV88] for unweighted edit distance. This gives rise to the following question.

Is it possible to solve PMwWE in $\widetilde{O}(n + k^{3.5} \cdot n/m)$ time when edit cost are integers between 0 and $W = O(1)$?

In Sections 6, 7 and 9, we incorporate the techniques of [GK25] into a more complex version of our procedure behind Main Theorem 3 (building upon the $\widetilde{O}(n + k^{3.5} \cdot n/m)$-time algorithm for PMwE [CKW22], rather than its simpler $\widetilde{O}(n + k^4 \cdot n/m)$-time warm-up version) to derive the following result under the extra assumption that w is an integer metric weight function.[5]

Main Theorem 4. *PMwWE can be solved in $\widetilde{O}(n + k^{3.5} \cdot W^4 \cdot n/m)$ time for metric weight functions w with values in $[0\,..\,W]$.*

It is worth noting that our solutions are meta-algorithms relying on a small set of primitive operations. Specifically, we work in the PILLAR model of computation, introduced in [CKW20] to unify approximate pattern matching across several settings. An efficient PILLAR algorithm, together with an implementation of the primitive operations in a given setting, yields an efficient algorithm for that setting. Main Theorems 3 and 4 are corollaries of the following result.

Main Theorem 2. *PMwWE on strings with $n < 3/2\, m + k$ can be solved in the PILLAR model*
- *in time $O(k^4 \log^2(mk))$ for general weights; and*
- *in time $\widetilde{O}(k^{3.5} \cdot W^4)$ if w is a metric weight function with values in $[0\,..\,W]$.*

The output set $\mathrm{Occ}_k^w(P, T)$ is represented as a collection of disjoint arithmetic progressions.

In Section 10, we combine Main Theorem 2 with known efficient implementations of the PILLAR model to obtain efficient algorithms for approximate pattern matching in a plethora of settings, such as: (1) the compressed setting, where P and T are given in compressed form, and we wish to compute $\mathrm{Occ}_k^w(P, T)$ without decompressing them; (2) the dynamic setting; and (3) the quantum setting.

Future Directions and Open Problems

Let us briefly discuss potential future directions.

- The most obvious yet the most important open problem stemming from this work is to narrow the gap between the upper and the lower bounds for PMwWE. In particular, it is unsatisfactory that the best lower bounds for both PMwE and PMwWE are direct corollaries of tight lower bounds for edit distance computation and weighted edit distance computation, respectively. It would be exciting to obtain a separation between PMwWE and the problem of weighted edit distance computation.

[5] We did not try to optimize the exponent of W in Main Theorem 4.

- Regarding faster algorithms, the most natural question is whether the $\widetilde{O}(n + k^{3.5} \cdot n/m)$ time complexity of [CKW22] for PMwE can be lifted to PMwWE. To derive this from an $\widetilde{O}(k^{3.5} \cdot n/m)$-time PILLAR algorithm (as in [CKW22]), we would need a better combinatorial characterization of the set $\mathrm{Occ}_k^w(P,T)$: As shown in [CKW20], the set $\mathrm{Occ}_k(P,T)$ can be represented as the union of $O(k^3 \cdot n/m)$ arithmetic progressions; for $\mathrm{Occ}_k^w(P,T)$, Main Theorem 2 yields a representation using $O(k^4 \cdot n/m)$ progressions.
- Gorbachev and Kociumaka [GK25] also showed how to compute weighted edit distance in $\widetilde{O}(m+k^{2.5})$ time for arbitrarily large integer weights. It remains open whether PMwWE can be solved faster in this regime compared to the general setting allowing fractional weights.
- The output of PMwWE, $\mathrm{Occ}_k^w(P,T)$, consists of the *starting positions* of all fragments $T[\,i\,..\,j\,)$ such that $\mathrm{ed}^w(P \to T[\,i\,..\,j\,)) \le k$.[6] A natural harder variant asks for all *fragments* $T[\,i\,..\,j\,)$ such that $\mathrm{ed}^w(P \to T[\,i\,..\,j\,)) \le k$. As noted in Section 6, a straightforward application of our tools yields an $\widetilde{O}(nk^2)$-time solution of this problem. Can we improve this to $\widetilde{O}(nk)$? For unweighted edit distance, the analogous variant can be solved in $O(nk)$ time [LMS98].
- Recently, Kociumaka, Nogler, and Wellnitz [KNW24, KNW25] investigated the communication complexity of PMwE. Their techniques also led to a quantum algorithm for PMwE with almost optimal query complexity and (for $k \ll m^{1/6}$) almost optimal running time. It would be interesting to investigate whether (any of) these results can be lifted to PMwWE. Our quantum algorithm for PMwWE (see Corollary 10.8) can be viewed as a first step in this direction.

2 Technical Overview

To describe our new algorithms, let us first briefly introduce crucial notation. A matrix $M \in \mathbb{R}^{p\times q}$ is *Monge* if $M[\,i,j\,] + M[\,i+1,j+1\,] \le M[\,i,j+1\,] + M[\,i+1,j\,]$ holds for all integers $0 \le i < p-1$ and $0 \le j < q-1$. Given $O(1)$-time access to the entries of a matrix $M \in \mathbb{R}^{p\times q}$ and a vector $v \in \mathbb{R}^q$, the SMAWK algorithm [AKM+87] computes their $(\min,+)$-product $M \oplus v$ in $O(p+q)$ time.

For strings $X, Y \in \Sigma^*$ and a weight function $w : \bar{\Sigma}^2 \to \mathbb{R}_{\ge 0}$, we define the *alignment graph* $\mathsf{AG}^w(X,Y)$ as a grid graph with vertices $[\,0\,..\,|X|\,] \times [\,0\,..\,|Y|\,]$ and the following edges:

- vertical edges $(x,y) \to (x+1,y)$ of cost $w(X[\,x\,] \mapsto \varepsilon)$ for $(x,y) \in [\,0\,..\,|X|\,) \times [\,0\,..\,|Y|\,]$;
- horizontal edges $(x,y) \to (x,y+1)$ of cost $w(\varepsilon \mapsto Y[\,y\,])$ for $(x,y) \in [\,0\,..\,|X|\,] \times [\,0\,..\,|Y|\,)$;
- diagonal edges $(x,y) \to (x+1,y+1)$ of cost $w(X[\,x\,] \mapsto Y[\,y\,])$ for $(x,y) \in [\,0\,..\,|X|\,) \times [\,0\,..\,|Y|\,)$.

An *alignment* of X onto Y, denoted by $\mathcal{A} : X \rightsquigarrow Y$, is a path from $(0,0)$ to $(|X|,|Y|)$ in $\mathsf{AG}^w(X,Y)$. In such a path, vertical edges correspond to deletions, horizontal edges correspond to insertions, whereas diagonal edges correspond to matches and substitutions. The weighted edit distance $\mathrm{ed}^w(X \to Y)$ is defined as the minimum cost (length) of an alignment $\mathcal{A} : X \rightsquigarrow Y$. We regard $(0,0)$ as the top-left vertex of the grid and $(|X|,|Y|)$ as the bottom-right vertex of the grid, and we refer to the $|X|+1$ rows and the $|Y|+1$ columns of the grid in a natural way. As each insertion and deletion costs at least 1, any path of cost at most k intersects at most $k+1$ diagonals.

2.1 The $\widetilde{O}(nk)$-time Algorithm

Consider a collection $\mathcal{S}$ of $O(n/k)$ fragments of the form $T[\,ik\,..\,\max\{(i+2)k+m, n\}\,)$; observe that each (k,w)-error occurrences of P and, in general, each fragment of T of length at most $m+k$ is fully contained in at least one fragment $S \in \mathcal{S}$. Hence, it suffices to compute the (k,w)-error occurrences of P in all fragments $S \in \mathcal{S}$ and then merge the partial results.

[6] The algorithms underlying Main Theorems 1, 3 and 4 can also compute, for each $i \in \mathrm{Occ}_k^w(P,T)$, the minimum distance $\min_j \mathrm{ed}^w(P \to T[\,i\,..\,j\,))$ and the position j for which this minimum is attained.

6 Pattern Matching under Weighted Edit Distance

Let us focus on computing $\mathrm{Occ}_k^w(P,S)$ for an arbitrary $S \in \mathcal{S}$. One may use the MSSP data structure of Klein [Kle05] on a width-$O(k)$ diagonal band that is a subgraph of $\mathsf{AG}^w(P,S)$ for this purpose (see Figure 2), that is, in order to compute each top-to-bottom shortest-path distance in this band and check if it is at most k.[7] However, this diagonal band contains $O(mk)$ vertices—hence, this approach costs $O(mk\log m)$ time per $S \in \mathcal{S}$, or $O(mn\log m)$ time in total across all $S \in \mathcal{S}$, which is prohibitive when $m \gg k$.

We improve upon this basic approach by making two observations.

- Instead of computing the shortest-path distances all at once, we may equivalently split the (diagonal band subgraph of the) alignment graph along a separator V, compute shortest-path distances from the sources (top vertices of the band) to the separator and from the separator to the sinks (bottom vertices of the band), and then compose these distances using min-plus multiplication. Specifically, two instances of the the MSSP data structure provide $O(\log m)$-time access to, respectively, a distance matrix that captures all pairwise distances from the sources to the vertices of V and a distance matrix that captures all pairwise distances from the vertices of V to the sinks. Crucially, these distance matrices are *Monge*, and this allows for min-plus multiplying them efficiently using the SMAWK algorithm [AKM+87].[8]
- In the non-trivial case when $m \gg k$, any cost-k (weighted) alignment has to match exactly large parts of P and S. We show that, with an $O(mk\log^2 m)$-time preprocessing on P, we may compute (a representation of) suitable distance matrices for any such pairs of equal parts for all $S \in \mathcal{S}$ at once. In particular, for a single $S \in \mathcal{S}$, the task then reduces to recomputing distances for graphs of total size $O(k^2)$ (the part corresponding to actual edits; we can use Klein's algorithm [Kle05] here, which takes time $O(k^2\log m)$) plus querying $O(k)$ of our precomputed distance matrices $O(k)$ times each (the exactly matched parts between edits; where each such query costs $O(\log m)$).

For a more detailed description of our algorithm, first observe that if there is a (k,w)-error occurrence of P in the fragment S, then $\mathsf{ed}(P,S) \le 4k$: We may extend an alignment $\mathcal{A} : P \rightsquigarrow S[\,f\,..\,f'\,)$ with at most k edits to an alignment $\mathcal{B} : P \rightsquigarrow S$ by inserting at most $3k$ excess characters, namely those comprising $S[\,0\,..\,f\,)$ and $S[\,f'\,..\,|S|\,)$.

Now, after an $O(m)$-time preprocessing of P and T, we check in $O(k^2)$ time whether $\mathsf{ed}(P,S) \le 4k$ and, if so, compute an optimal unweighted alignment. Let us henceforth assume that we have such an alignment $\mathcal{B} : P \rightsquigarrow S$ of unweighted cost at most $4k$—which lets us identify fragments of P that are matched exactly to fragments of S.

For a simplified overview, let us assume that $\mathcal{B} : P \rightsquigarrow S$ consists only of substitutions; this case illustrates the main ideas of the algorithm whilst avoiding technicalities.

Now, observe that $\mathsf{ed}^w(P \to S[\,a\,..\,b\,)) \le k$ and $|S| = |P|$ imply $a \in [\,0\,..\,k\,]$ and $b \in [\,m-k\,..\,m\,]$. Hence, we consider $k+1$ source vertices and $k+1$ target vertices in $\mathsf{AG}^w(P,S)$ as shown in Figure 2. Moreover, the weighted edit distance of two strings X and Y is lower-bounded by $\big||X|-|Y|\big|$, so it suffices to consider paths confined in a band B comprised by $2k+1$ diagonals of $\mathsf{AG}^w(P,S)$ as shown in Figure 2.

Now, the alignment $\mathcal{B} : P \rightsquigarrow S$ yields at most $4k$ positions $i_0 < i_1 < \cdots < i_{r-1}$ such that

$$S = P[\,0\,..\,i_0\,)S[\,i_0\,]P(\,i_0\,..\,i_1\,)\cdots S[\,i_r\,]P(\,i_r\,..\,m\,).$$

---

[7] Given a planar graph with N vertices and a distinguished infinite face, the multiple-source shortest paths (MSSP) data structure [Kle05] can be constructed in $O(N\log N)$ time and can report the distance from (or to) any vertex on the infinite face to (or from) any other vertex in the graph in $O(\log N)$ time.

[8] As matrix-matrix multiplications are not fast enough for our purposes, we show how to make do with just matrix-vector multiplications which take time linear in the output size in our case.

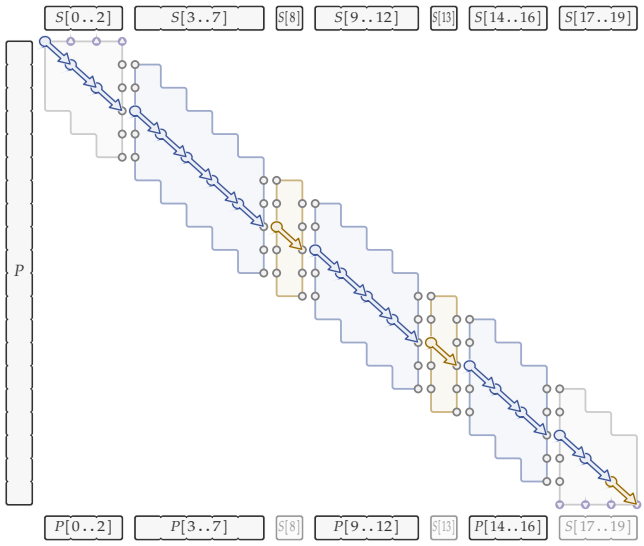


Figure 2. The band B of the alignment graph $\mathsf{AG}^w(P,S)$ for a pattern P of length 20, a threshold $k=3$, and a fragment $S = P[0..8)\mathtt{a}P[9..13)\mathtt{c}P[14..19)\mathtt{c}$ of T. The source vertices $(0,0),(0,1),(0,2),(0,3)$ in the top left and the target vertices $(20,17),(20,18),(20,19),(20,20)$ in the bottom right are marked in light purple. The distinguished band B is decomposed to seven parts. The left and right parts are colored gray; B_i that also occur in $\mathsf{AG}^w(P,P)$ are merged into *pure parts* and are colored blue. We highlight the portal vertices on the boundaries of each part of B.

Let us also decompose the band B into parts $B_0, B_1, \ldots, B_s$ so that B_0 corresponds to the first k columns (the left part) and B_s corresponds to the last k columns (the right part), whereas the intermediate parts correspond to substrings of P and single characters according to the decomposition of S; see Figure 2. Further, let V_j be the set of vertices shared by parts B_j and B_{j+1}—these vertices indeed form a separator of the band B.

For a vertex u, let $d(u)$ denote the minimum distance (within the band B) from u to any target vertex. We compute $d(u)$ for the vertices of the rightmost separator V_{s-1} using the algorithm of Klein [Kle05] on B_s. As B_s has a size of $O(k^2)$, Klein's algorithm runs in time $O(k^2 \log m)$.

Next, we process the separators from right to left. We compute the values $d(u)$ for all vertices $u \in V_{j-1}$ given $d(v)$ for all $v \in V_j$ differently based on whether B_j is a subgraph of $\mathsf{AG}^w(P,P)$.

- If B_j is not a subgraph of $\mathsf{AG}^w(P,P)$, we can again use the algorithm of Klein [Kle05] on B_j to obtain $O(\log n)$-time random access to a Monge matrix $D_{j-1,j}$ that stores the distances from the vertices of V_{j-1} to the vertices of V_j in B_j. We then min-plus multiply $D_{j-1,j}$ with a vector that stores $d(u)$ for $u \in V_j$ to obtain (a vector stores) $d(u)$ for each $u \in V_{j-1}$. Recall that, using the SMAWK algorithm, such a multiplication can be done in the time required for $O(k)$ accesses (at arbitrary positions) to the input matrices (interpreting the vector that stores $d(u)$ for $u \in V_j$ as an $m \times 1$ matrix).
- If B_j is subgraph of $\mathsf{AG}^w(P,P)$, we rely on a global $\widetilde{O}(mk)$-time preprocessing of P that constructs several instances of the MSSP data structure [Kle05] for subgraphs of $\mathsf{AG}^w(P,P)$. In particular, we obtain $O(\log m)$-time random access to two Monge matrices $D_{j-1,j-0.5}$ and $D_{j-0.5,j}$ with a min-plus product of $D_{j-1,j}$. We first min-plus multiply $D_{j-0.5,j}$ with a vector that stores $d(u)$ for $u \in V_j$ (to obtain an intermediate vector w) and then min-plus multiply w with $D_{j-1,j-0.5}$ to obtain (a vector that stores) $d(u)$ for each $u \in V_{j-1}$. Recall that, using the SMAWK algorithm, such a multiplication can be done in the time required for $O(k)$ accesses (at arbitrary positions) to the input matrices.

Finally, we again rely in the algorithm of Klein [Kle05] for the part between the leftmost separator and the sources.

Over all $O(k)$ parts $B_{s-1}, \ldots, B_1$, the above procedure takes $\widetilde{O}(k^2)$ time. The final complexity is then $O(n) + \widetilde{O}(mk) + |\mathcal{S}| \cdot \widetilde{O}(k^2) = \widetilde{O}(n + mk + n/k \cdot k^2) = \widetilde{O}(nk)$.

2.2 The $\widetilde{O}(n + k^4 \cdot n/m)$-time Algorithm

In this section, we provide an overview of our $\widetilde{O}(n + k^4)$-time algorithm solving PMwWE under the simplifying assumption that $n < {}^3\!/_2\, m + k$ (as in Main Theorem 2). The reduction from the general case relies on the *standard trick*, which yields $\Theta(n/m)$ independent instances of the restricted problem.

The starting point of our solution is the structural characterization of [CKW20] for PMwE: unless P is almost periodic, it has few approximate occurrences in T.

The Non-Periodic Case

In the first of the two cases of [CKW20], the set $\{\lfloor i/k \rfloor : i \in \mathrm{Occ}_k(P,T)\}$ is of size $O(k)$ and can be computed in $O(n + k^4)$ time. This gives $O(k)$ intervals of length k whose union is a superset of $\mathrm{Occ}_k(P,T) \supseteq \mathrm{Occ}_k^w(P,T)$. The remaining task is to compute $\mathrm{Occ}_k^w(P,T) \cap I$ for each interval I in this collection. For this, we design an efficient solution of the following Verify problem.

Problem Verify$(P,\ T,\ k,\ I,\ w)$.

Input. A text T of length n, a pattern P of length m, an integer threshold $k > 0$, an integer interval I, and oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [\,0\,,\, W\,]$.

Output. $\mathrm{Occ}_k^w(P,T) \cap I$ and, for each $i \in \mathrm{Occ}_k^w(P,T) \cap I$, the distance $\min_{j \in [\,i\,..\,n\,]} \mathsf{ed}^w(P \to T[\,i\,..\,j\,))$.

Our approach for Verify computes more than we need *here*: it reports all relevant (k,w)-error occurrences of P in T (rather than just their starting positions) and, for each of them, outputs the corresponding weighted edit distance from P. These extra features are useful later and come essentially for free.

In this overview, we present an $\widetilde{O}(k^3)$-time PILLAR model algorithm for Verify for the case when $n \le m + 2k$. In general, we partition I into $\lceil |I|/k \rceil$ pieces of size at most k, and, for each piece I', we trim the text to $T[\,\min I'\,..\,\min\{n, \max I' + m - k\}\,]$. Now, our problem generalizes the task considered in [CKW23]: instead of computing the edit distance from P to the entire T, we need to consider $O(k^2)$ fragments $T[\,i\,..\,j\,)$ with $j - i \ge m - k$.

In our solution, we apply the notion of *self-edit distance*, introduced in [CKW23] to quantify the locality of edit distance. The self-edit distance of a string X is defined as the distance from $(0,0)$ to $(|X|,|X|)$ in the *unweighted* alignment graph $\mathsf{AG}(X,X)$ with edges on the main diagonal removed. The algorithm of [CKW23] consists of a solution for the special case of $\mathsf{self\text{-}ed}(P) = O(k)$ and a divide-and-conquer recursive procedure that constitutes a reduction to this case. The first of these two components generalizes to our setting quite easily, so we focus on the case when $\mathsf{self\text{-}ed}(P) = \omega(k)$.

In this case, we identify vertices (p,t) and (p',t') in $\mathsf{AG}^w(P,T)$ with $\mathsf{self\text{-}ed}(P[\,0\,..\,p\,)) = O(k)$ and $\mathsf{self\text{-}ed}(P[\,p'\,..\,m\,)) = O(k)$ that lie on a w-optimal alignment $P \rightsquigarrow T[\,i\,..\,j\,)$ for every (k,w)-error occurrence of P in T. For every such (k,w)-error occurrence, the value $\mathsf{ed}^w(P \to T[\,i\,..\,j\,))$ decomposes to

$$\mathsf{ed}^w(P[\,0\,..\,p\,) \to T[\,i\,..\,t\,)) + \mathsf{ed}^w(P[\,p\,..\,p'\,) \to T[\,t\,..\,t'\,)) + \mathsf{ed}^w(P[\,p'\,..\,m\,) \to T[\,t'\,..\,j\,)).$$

We compute the middle term using the algorithm of [CKW23]; for the remaining two terms, we use our subroutine restricted to patterns of self-edit distance $O(k)$.

It is not obvious that such vertices (p,t) and (p',t') exist. Next, we show that the existence of such vertices follows from the central property of self-edit distance: If alignments $\mathcal{A}, \mathcal{B} : X \rightsquigarrow Y$ of cost at most k do not share diagonal edges, then $\mathsf{self\text{-}ed}(X) \le 2k$. (The path in $\mathsf{AG}(X,X)$ witnessing $\mathsf{self\text{-}ed}(X) \le 2k$ is obtained as the composition of $\mathcal{A}$ and $\mathcal{B}^{-1}$; see [CKW23].)

Fix an arbitrary (k,w)-error occurrence $T[\,i\,..\,j\,)$ of P in T and a w-optimal alignment $\mathcal{A} : P \rightsquigarrow T[\,i\,..\,j\,)$. We pick $(p,t),(p',t') \in \mathcal{A}$ so that $\mathsf{self\text{-}ed}(P[\,0\,..\,p\,)) = \mathsf{self\text{-}ed}(P[\,p'\,..\,m\,)) = 9k$. We have $p \ge 9k - 1$ as for any $q \in [\,1\,..\,m\,]$, the $(0,0) \to (0,1) \to (1,2) \to \cdots \to (q-1,q) \to (q,q)$ path in $\mathsf{AG}(P[\,0\,..\,q\,), P[\,0\,..\,q\,))$ has cost at most $q + 1$.

To show that this selection is valid, consider another (k,w)-error occurrence $T[\,i'\,..\,j'\,)$ of P in T and a w-optimal alignment $\mathcal{A}' : P \rightsquigarrow T[\,i'\,..\,j'\,)$. We claim that $\mathcal{A}$ intersects $\mathcal{A}'$ both before (p,t) and after (p',t'). The subpaths of both $\mathcal{A}$ and $\mathcal{A}'$ are shortest paths between these two intersection points, so we may reroute $\mathcal{A}'$ (without increasing its cost) using $\mathcal{A}$ to pass through both (p,t) and (p',t').

Due to $n \le m + 2k$, both of the (unique) extensions of $\mathcal{A}$ and $\mathcal{A}'$ to alignments $\mathcal{B}, \mathcal{B}' : P \rightsquigarrow T$ have *unweighted* cost at most $4k$. Let $(\bar{p},\bar{t})$ be the first point in the intersection $\mathcal{B} \cap \mathcal{B}'$ with $\bar{p} > 0$. If $\bar{p} = 1$, then, in particular, $\bar{p} < 9k - 1 \le p$, and hence $(\bar{p},\bar{t}) \in \mathcal{A} \cap \mathcal{A}'$ lies before (p,t). Otherwise, $\mathcal{B}$ and $\mathcal{B}'$ do not share any diagonal edge between $(0,0)$ and $(\bar{p},\bar{t})$ and, restricted to this segment, still have unweighted costs of at most $4k$ each. Thus, $\mathsf{self\text{-}ed}(P[\,0\,..\,\bar{p}\,)) \le 8k < 9k = \mathsf{self\text{-}ed}(P[\,0\,..\,p\,))$, which implies $\bar{p} < p$ as $\mathsf{self\text{-}ed}(\cdot)$ is monotone.

The construction in Section 6 follows the same spirit but cannot rely on the (unknown) arbitrary (k,w)-error occurrence $T[\,i\,..\,j\,)$ of P in T. Instead, we define (p,t) and (p',t') using (k,w)-error occurrences of a prefix P_p and a suffix P_s of P, respectively, with small self-edit distance; see Figure 8. These two (k,w)-error occurrences can be efficiently computed using the subroutine restricted to patterns of self-edit distance $O(k)$.

The Periodic Case

In the almost-periodic case, the characterization of [CKW20] yields a primitive string Q of length $|Q| \ll m/k$ and an alignment from P to a substring of Q^∞ with unweighted cost $O(k)$. Further results of [CKW20] let us trim T (without losing k-error occurrences) so that it admits a similar alignment.

To keep this overview simple, we assume that both substrings of Q^∞ are integer powers of Q, that is, $\mathsf{ed}(P,Q^p) = O(k)$ and $\mathsf{ed}(T,Q^t) = O(k)$ hold for some integer exponents $p,t \gg k$. We decompose $P = P_1 \cdots P_p$ and $T = T_1 \cdots T_t$, where the P_is and T_js are potentially edited copies of the string Q. In this case, all elements of $\mathrm{Occ}_k(P,T)$ must be within distance $O(k)$ from the start of some T_j. Following [CKW22], we then think of shifting P over T in steps of roughly $|Q|$ positions at a time, in order to align the starting position of P with the start of each T_j. Intuitively, for subsequent $j \in [\,0\,..\,t-p\,]$, we wish to compute $\mathrm{Occ}_k(P_1P_2\cdots P_p, T_jT_{j+1}\cdots T_{j+p-1})$ (up to extending T_j to the left and T_{j+p-1} to the right by $O(k)$ positions).

Applying DynamicPuzzleMatching to PMwE. The solution of [CKW22] is formalized through the DynamicPuzzleMatching problem, whose (oversimplified) formulation asks to maintain a dynamic sequence $\mathcal{I} = (U_1,V_1)(U_2,V_2)\cdots(U_z,V_z)$ of *puzzle pieces* subject to updates (insertions, deletions, and substitutions of pieces) so that, upon a query, one can efficiently compute $\mathrm{Occ}_k(U,V)$ for $U = U_1U_2\cdots U_z$ and $V = V_1V_2\cdots V_z$. In reality, the strings V_i overlap, but we mostly ignore these overlaps in this overview.

A naive reduction to DPM constructs and queries $\mathcal{I}_j := (P_1,T_{j+1})(P_2,T_{j+2})\cdots(P_p,T_{j+p})$ for subsequent integers $j \in [\,0\,..\,t-p\,]$. Since all but $O(k)$ strings P_i and T_j are equal to Q, updating $\mathcal{I}_j$ to $\mathcal{I}_{j+1}$ requires $O(k)$ substitutions. Each of these substitutions can be processed in $\widetilde{O}(k)$ time each. Unfortunately, this is too much as $t-p$ can be $\Theta(n/k)$.

To lay ground for further improvements, the algorithm of [CKW22] issues different updates, most of which increment or decrement the exponents of *runs of plain pieces* of the form $(Q,Q)^e$ for $e \in \mathbb{Z}_{\ge 0}$.

Example 2.1. Suppose that $P = Q^{99}P_{100}Q^{99}$ and $T = Q^{149}T_{150}Q^{149}$. Then,

$$\mathcal{I}_j = \begin{cases} (Q,Q)^{99}\cdot(P_{100},Q)\cdot(Q,Q)^{49-j}\cdot(Q,T_{150})\cdot(Q,Q)^{49+j} & \text{if } j \in [\,0\,..\,49\,],\\ (Q,Q)^{99}\cdot(P_{100},T_{150})\cdot(Q,Q)^{99} & \text{if } j = 50,\\ (Q,Q)^{149-j}\cdot(Q,T_{150})\cdot(Q,Q)^{j-51}\cdot(P_{100},Q)\cdot(Q,Q)^{99} & \text{if } j \in [\,51\,..\,100\,]. \end{cases}$$

For $j \in [\,0\,..\,48\,] \cup [\,51\,..\,99\,]$, one can update $\mathcal{I}_j$ to $\mathcal{I}_{j+1}$ by incrementing or decrementing the exponents of two runs of plain pieces. Effectively, this allows moving the special piece (Q,T_{150}) to the left.

Formally, as we issue updates to iterate over $\mathcal{I}_j$ for subsequent indices $j \in [\,0\,..\,t-p\,]$, all but $O(k^2)$ updates increment or decrement the exponent of a run of plain pieces. Furthermore, each of the $O(k^2)$ runs existing throughout the lifetime of the algorithm admits a simple structure of updates: its exponent is first incremented for some subsequent indices j, then it remains fixed, and then it is decremented.

This insight allows reducing the number of updates when combined with the following observation: For every run $(Q, Q)^e$ of plain pieces, the exponent can be capped to $e = \min\{e, \alpha\}$ for $\alpha = O(k)$ without affecting the output of DPM queries. If, instead of $\mathcal{I}_j$, we maintain sequences $\mathcal{I}'_j$ with the exponents capped at α, each run needs to be updated only $O(\alpha)$ times, for a total of $O(k^2\alpha)$ updates that can be processed in $\widetilde{O}(k^3\alpha) = \widetilde{O}(k^4)$ time in total.

The key insight to achieve $\widetilde{O}(k^{3.5})$ time in [CKW22] is that picking $\alpha \le k$ preserves answers to DPM queries for all but $O(k^2/\alpha)$ sequences $\mathcal{I}_j$. Unfortunately, the underlying combinatorial arguments do not seem to generalize to the case of arbitrary weight functions, so we fix $\alpha = \Theta(k)$ for the remainder of this discussion.

Applying BleachCommitDPM to PMwWE. The aforementioned reduction to DynamicPuzzleMatching translates from PMwE to PMwWE with minor changes. Unfortunately, the overall running time increases to $\widetilde{O}(k^5)$ because each DPM update takes $\widetilde{O}(k^2)$ time instead of $\widetilde{O}(k)$ time.

Intuitively, for each puzzle piece (X, Y), the DPM data structure of [CKW22] stores a matrix $D_{X,Y}$ whose entries represent distances $\mathsf{ed}(X, Y[\,i\,..\,j\,))$ for all k-error occurrences $Y[\,i\,..\,j\,)$ of X in Y. Since $|Y| \le |X| + O(k)$, these are matrices of size $O(k) \times O(k)$. The composition of two pieces (X, Y) and (X', Y') can be implemented by $(\min, +)$-multiplying matrices $D_{X,Y}$ and $D_{X',Y'}$, due to the fact that Y and Y' actually have a sufficiently large overlap. Under mild technical assumptions, the matrix $D \coloneqq \bigoplus_{i=1}^{z} D_{U_i,V_i}$ can be used to identify all k-error occurrences of $U = U_1 \cdots U_z$ in $V = V_1 \cdots V_z$. For the starting positions, that is, $\mathrm{Occ}_k(U, V)$, it suffices to compute the $(\min, +)$-product of D and a $\mathbf{0}$-vector of appropriate dimension.

For unweighted edit distance (or, to be precise, for deletion distance, which admits a simple isometric embedding from unweighted edit distance), all the matrices in question are *unit-Monge matrices* and the results of Tiskin [Tis15] allow storing each of them in $\widetilde{O}(k)$ space so that the $(\min, +)$-products can be computed in $\widetilde{O}(k)$ time. Thus, the global product D can be maintained in $\widetilde{O}(k)$ time assuming the individual matrices $D_{X,Y}$ have been precomputed for all plausible puzzle pieces (X, Y). In the presence of weights, the matrices are merely Monge matrices, which means that we have to resort to an $O(k^2)$-space representation and the $O(k^2)$-time $(\min, +)$-multiplication algorithm of [AKM+87].

What comes to the rescue is that the queries require $D \oplus \mathbf{0}$ and the SMAWK algorithm is able to compute a matrix-vector product in $O(k)$ time. We can offload some update cost to the query algorithm: if we lazily represent D as a product of d matrices, the query time increases from $O(k)$ to $O(kd)$. The challenge is to avoid matrix-matrix products for most updates without compromising the query time.

For this, recall that all but $O(k^2)$ updates adjust the exponent of a run of plain pieces, and each such run is active (meaning that its exponent changes between subsequent queries) for two blocks of $\alpha = O(k)$ queries each. Thus, we precompute the matrix for each possible exponent of a run of plain pieces and, in the maintained decomposition of D, we make sure that each active run is represented with a separate matrix that can be substituted in $\widetilde{O}(1)$ time when the exponent changes. The remaining $O(k^2)$ updates take $\widetilde{O}(k^2)$ time each. The query time becomes $O(k \cdot (1 + r))$, where r is the number of active runs at the moment, but each of the $O(k^2)$ runs is active during $O(k)$ queries, so the total query time is $O(k^4)$.

In Section 7, we state the BleachCommitDPM problem (a variant of DPM with appropriately extended interface) and provide an efficient reduction from PMwWE. The solution to BCDPM is described in Section 9. The main challenge there is to formally define the matrices $D_{X,Y}$ so that they can be efficiently constructed (using the techniques behind Verify) and so that their $(\min, +)$-product is meaningful. In [CKW22], a custom *restriction* operation is introduced so that $D_{X,Y}$ is uniquely defined yet computable

in $O(k^2)$ time even if X and Y are very long. Unfortunately, this operation relies on the *seaweed monoid* of Tiskin [Tis15], which loses its meaning in the presence of weights. In this work, we instead introduce a notion of a *fern matrix* that leaves a lot of freedom in the particular choice of $D_{X,Y}$ (in particular, for the values that do not correspond to (k,w)-error occurrences of X in Y) and is nevertheless sufficient for us to recover the (k,w)-error occurrences of $U = U_1 \cdots U_z$ in $V = V_1 \cdots V_z$ from the (min, +)-product of fern matrices D_{U_i,V_i}.

2.3 The $\widetilde{O}(n + k^{3.5} \cdot W^4 \cdot n/m)$-time Algorithm

We obtain Main Theorem 4 by combining the approach outlined in Section 2.2 with a generalization of techniques from [CKW22] and several results and insights from the recent (static and dynamic) algorithms for edit distance with small integer weights [GK25]. On the way to achieving this result, we make the following technical contributions. First, we devise an $\widetilde{O}(k^2W)$-time `PILLAR` algorithm for Verify, generalizing both the conditionally optimal $O(k^2)$-time `PILLAR` algorithm for Verify in the unweighted setting and the aforementioned algorithm of Gorbachev and Kociumaka [GK25]. This solution to Verify is instrumental in both the non-periodic case and the periodic case of the algorithm underlying Main Theorem 4. For instance, in the periodic case, it allows us to compute optimal weighted alignments from P and T to substrings of Q^∞; the algorithm outlined in Section 2.2 utilizes their unweighted counterparts. Second, we adapt an algorithm from [CKW20] for the computation of so-called *locked fragments* to the weighted setting; this might be of independent interest as locked fragments have also recently been used in obtaining solutions for a circular variant of PMwE [CPR+24]. Third, we lift combinatorial insights for PMwE from [CKW22] to PMwWE—our assumption that the weight function is a metric is required in this part of the solution since it ensures that the weighted edit distance satisfies the triangle inequality.

3 Preliminaries

Sets and Arithmetic Progressions

For integers i and j, we write $[\,i\,..\,j\,]$ to denote the set $\{i,\ldots,j\}$, and we write $[\,i\,..\,j\,)$ to denote the set $\{i,\ldots,j-1\}$. Similarly, we define $(\,i\,..\,j\,]$ and $(\,i\,..\,j\,)$.

For integers a, d, and ℓ with $d \neq 0$ and $\ell \geq 0$, the set $\{a + j \cdot d : j \in [\,0\,..\,\ell\,)\}$ is an *arithmetic progression* with starting value a, difference d, and length ℓ. Whenever we use arithmetic progressions in an algorithm, we store them as triples (a,d,ℓ) consisting of their starting value, difference, and length.

For a set $X \subseteq \mathbb{Z}$ and an integer $s \in \mathbb{Z}$, we write $s + X$ and $X + s$ for the set $\{s + x : x \in X\}$ of all elements of X incremented by s.

Strings

We write $T = T[\,0\,]\,T[\,1\,]\cdots T[\,n-1\,]$ to denote a *string* of length $|T| = n$ over an alphabet Σ. The elements of Σ are called *characters*. We write ε to denote the *empty string*. In a slight abuse of notation, we write $\bar{\Sigma} := \Sigma \cup \{\varepsilon\}$ for the alphabet Σ that is extended by ε (which we use to represent the lack of a character).

A string P is a *substring* of a string T if we have $P = T[\,i\,]\cdots T[\,j-1\,]$ for some integers i, j with $0 \leq i \leq j \leq |T|$. In this case, we say that there is an *exact occurrence* of P at position i in T, or, more simply, that P *exactly occurs in* T. We write $T[\,i\,..\,j\,)$ for this particular occurrence of P in T, which is formally a *fragment* of T specified by the two endpoints i, j. For notational convenience, we may also refer to this fragment as $T[\,i\,..\,j-1\,]$, $T(\,i-1\,..\,j-1\,]$, or $T(\,i-1\,..\,j\,)$. Two fragments (perhaps of different strings) *match* if they are occurrences of the same strings.

A *prefix* of a string T is a fragment that starts at position 0 (that is, a prefix is a fragment of the form $T[\,0\,..\,j\,)$ for some $j \in [\,0\,..\,|T|\,]$). A *suffix* of a string T is a fragment that ends at position $|T|-1$ (that is, a suffix is a fragment of the form $T[\,i\,..\,|T|\,)$ for some $i \in [\,0\,..\,|T|\,]$).

For two strings U and V, we write UV or $U \odot V$ to denote their concatenation; we also write $\bigodot_{i=1}^{m} X_i$ for $X_1 \odot \cdots \odot X_m$ and set $U^k \coloneqq \bigodot_{i=1}^{k} U$. Similarly, U^∞ denotes an infinite string obtained by concatenating infinitely many copies of U. At certain points, we may access such an infinite repetition of U also at negative positions; hence for an integer $j \in [\,0\,..\,|U|\,)$ and a (possibly negative) integer i, we formally set $U^\infty[\,i \cdot |U| + j\,] \coloneqq U[\,j\,]$.

A string T is *primitive* if it cannot be expressed as $T = U^k$ for any string U and any integer $k > 1$. A positive integer p is a *period* of a string T if $T[\,i\,] = T[\,i+p\,]$ for all $i \in [\,0\,..\,|T|-p\,)$. We refer to the smallest period as *the period* $\mathrm{per}(T)$ of the string.

For a string T of length n, we define the following *rotation* operations. The operation $\mathrm{rot}(\star)$ takes as input a string, and moves its last character to the front; that is, $\mathrm{rot}(T) \coloneqq T[\,n-1\,]T[\,0\,..\,n-2\,]$. The inverse operation $\mathrm{rot}^{-1}(\star)$ takes as input a string and moves its initial character to the end; that is, $\mathrm{rot}^{-1}(T) \coloneqq T[\,1\,..\,n-1\,]T[\,0\,]$. Observe that a primitive string T does not match any of its non-trivial rotations, that is, we have $T = \mathrm{rot}^j(T)$ if and only if $j \equiv 0 \pmod{n}$.

Alignment Graphs and (Weighted) Edit Distances

A *weight function* $w : \bar{\Sigma}^2 \to \mathbb{R}$ is a (not necessarily symmetric) function that assigns cost to character edits. A weight function w is *normalized* if $w(a \mapsto a) = 0$ and $w(a \mapsto b) \geq 1$ holds for all $a, b \in \bar{\Sigma}$; as in previous works, we work only with normalized weight functions. Moreover, we assume that $w : \bar{\Sigma}^2 \to [\,0\,,\,W\,]$, where W is some upper bound known to the algorithms.

Observe that w does not need to satisfy the triangle inequality nor does w need to be symmetric.

Definition 3.1 (The alignment graph $\mathsf{AG}^w(P,T)$ of a weight function w and strings P, T) [CKW23]**.** *For strings $P, T \in \Sigma^*$ and a weight function $w : \bar{\Sigma}^2 \to [\,0\,,\,W\,]$, we define the* alignment graph $\mathsf{AG}^w(P,T)$ *as a grid graph with vertices $[\,0\,..\,|P|\,] \times [\,0\,..\,|T|\,]$ and the following edges:*

- *vertical edges $(p,t) \to (p+1,t)$ of cost $w(P[\,p\,] \mapsto \varepsilon)$ for $(p,t) \in [\,0\,..\,|P|\,) \times [\,0\,..\,|T|\,]$,*
- *horizontal edges $(p,t) \to (p,t+1)$ of cost $w(\varepsilon \mapsto T[\,t\,])$ for $(p,t) \in [\,0\,..\,|P|\,] \times [\,0\,..\,|T|\,)$, and*
- *diagonal edges $(p,t) \to (p+1,t+1)$ of cost $w(P[\,p\,] \mapsto T[\,t\,])$ for $(p,t) \in [\,0\,..\,|P|\,) \times [\,0\,..\,|T|\,)$.*

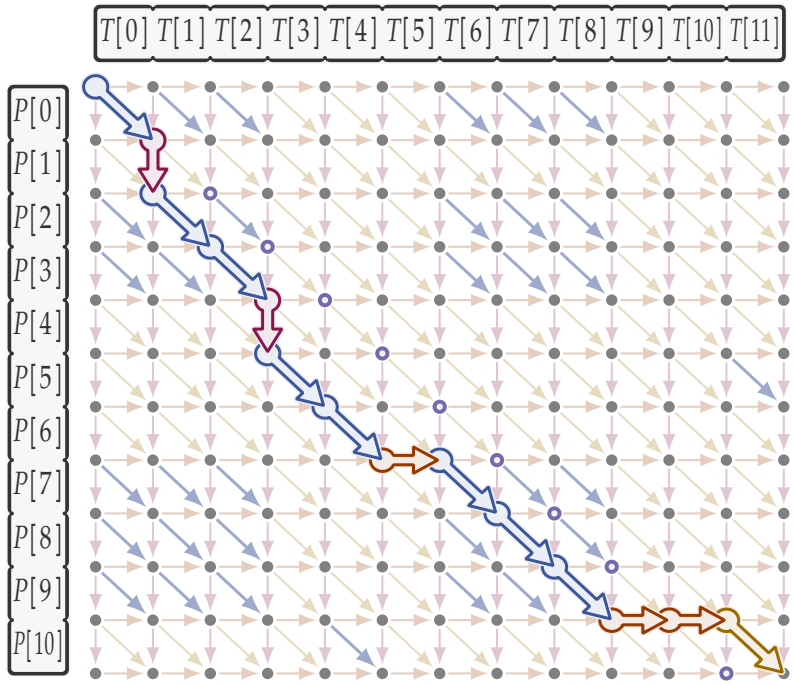
We visualize the alignment graph $\mathsf{AG}^w(P,T)$ as a grid graph with $|T|+1$ columns and $|P|+1$ rows that grows down and to the right, that is, the vertex $(0,0)$ is at the top-left, the vertex $(0,|T|)$ is at the top-right, and the vertex $(|P|,|T|)$ is at the bottom-right. Consult Figure 3 for a visualization.

A *diagonal* is a subgraph of the form $(i,j) \to (i+1,j+1) \to (i+2,j+2) \to \cdots$ that starts at the top-left boundary of $\mathsf{AG}^w(P,T)$ and extends to the bottom-right boundary of $\mathsf{AG}^w(P,T)$; we typically refer to this diagonal as the $(j-i)$-th diagonal.

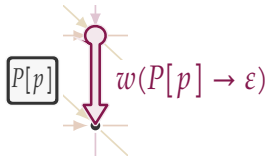
Example 3.2. The 0-th diagonal is the subgraph $(0,0) \to (1,1) \to (2,2) \to \cdots$, the 1-st diagonal is the subgraph $(0,1) \to (1,2) \to (2,3) \to \cdots$, and the (-1)-st diagonal is the subgraph $(1,0) \to (2,1) \to (3,2) \to \cdots$. Also consider Figure 3a for a visualization.

Observe that every vertex (p,t) of $\mathsf{AG}^w(P,T)$ lies on exactly one diagonal—the diagonal $t-p$; hence, we may refer to *the* diagonal passing through a vertex. We say that a path in $\mathsf{AG}^w(P,T)$ *intersects* a diagonal if the path shares a vertex with the diagonal.

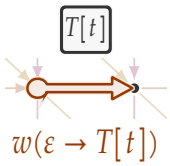
The alignment graph allows for a concise definition of an *alignment*.



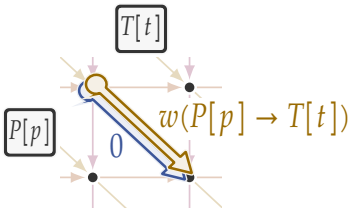
▪**a** An example of an alignment graph of strings P and T. Blue edges correspond to edges of weight 0, all other edges have a nonzero weight. We highlight an alignment $\mathcal{A} : P \rightsquigarrow T$ (a path from $(0,0)$ to $(|P|,|T|)$) that makes 6 edits to transform P into T. This alignment corresponds to the first part of the alignment in Figure 4a. The vertices of the 0-th diagonal are depicted purple (unless they are part of $\mathcal{A}$).



▪**b** A detailed view of a deletion and the associated cost.



▪**c** A detailed view of an insertion and the associated cost.



▪**d** A detailed view of a substitution (yellow; if $P[\,p\,] \neq T[\,t\,]$) and a match (blue; if $P[\,p\,] = T[\,t\,]$) and the associated costs.

Figure 3. The alignment graph of two strings P and T; as well as a detailed view of the edge costs. To obtain the corresponding augmented alignment graph, we add a back-edge of weight W for every depicted edge.

Definition 3.3. *For strings $P, T \in \Sigma^*$ and a weight function $w : \bar{\Sigma}^2 \to [\,0\,..\,W\,]$, an* alignment *of $P[\,p\,..\,p'\,)$ onto $T[\,t\,..\,t'\,)$, denoted by $\mathcal{A} : P[\,p\,..\,p'\,) \rightsquigarrow T[\,t\,..\,t'\,)$, is a path from (p,t) to (p',t') in $\mathsf{AG}^w(P,T)$, interpreted as a sequence of vertices. The $\mathsf{cost}\,\mathsf{ed}^w_{\mathcal{A}}(P[\,p\,..\,p'\,) \to T[\,t\,..\,t'\,))$ of an alignment $\mathcal{A}$ is the cost of the corresponding path in $\mathsf{AG}^w(P,T)$.*

We write $\mathbf{A}(P[\,p\,..\,p'\,), T[\,t\,..\,t'\,))$ for the set of all alignments of $P[\,p\,..\,p'\,)$ onto $T[\,t\,..\,t'\,)$.

For an alignment $\mathcal{A} = (p_j, t_j)_{j=0}^{r} \in \mathbf{A}(P[\,p\,..\,p'\,), T[\,t\,..\,t'\,))$ and an index $j \in [\,0\,..\,r\,)$, we say that

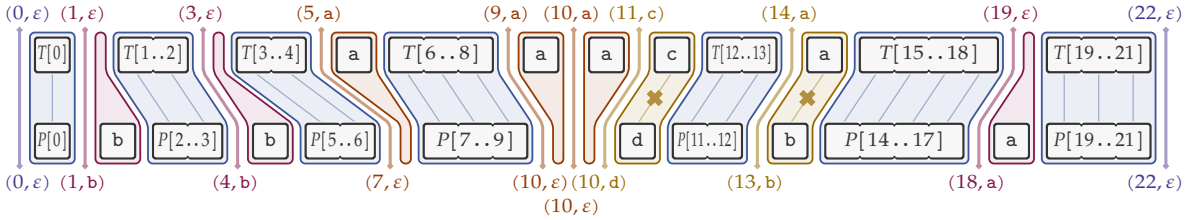
- $\mathcal{A}$ *deletes* $P[\,p_j\,]$, denoted by $P[\,p_j\,] \rightsquigarrow_{\mathcal{A}} \varepsilon$, if $(p_{j+1}, t_{j+1}) = (p_j + 1, t_j)$;
- $\mathcal{A}$ *inserts* $T[\,t_j\,]$, denoted by $\varepsilon \rightsquigarrow_{\mathcal{A}} T[\,t_i\,]$, if $(p_{j+1}, t_{j+1}) = (p_j, t_j + 1)$;
- $\mathcal{A}$ *aligns* $P[\,p_j\,]$ to $T[\,t_j\,]$, denoted by $P[\,p_j\,] \rightsquigarrow_{\mathcal{A}} T[\,t_j\,]$ if $(p_{j+1}, t_{j+1}) = (p_j + 1, t_j + 1)$;
- $\mathcal{A}$ *matches* $P[\,p_j\,]$ with $T[\,t_j\,]$ if $P[\,p_j\,] \rightsquigarrow_{\mathcal{A}} T[\,t_j\,]$ and $P[\,p_j\,] = T[\,t_j\,]$;
- $\mathcal{A}$ *substitutes* $P[\,p_j\,]$ with $T[\,t_j\,]$ if $P[\,p_j\,] \rightsquigarrow_{\mathcal{A}} T[\,t_j\,]$ but $P[\,p_j\,] \neq T[\,t_j\,]$.

Insertions, deletions, and substitutions are jointly called (character) *edits*.

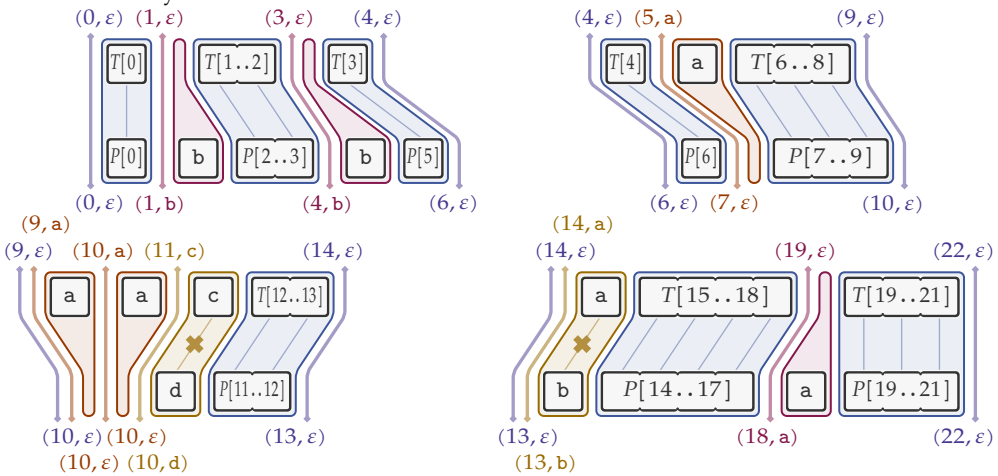
The *breakpoint representation* of an alignment

$$\mathcal{A} = ((0,\varepsilon),(0,\varepsilon)), ((p_j, P[\,p_j\,]), (t_j, T[\,t_j\,]))_{t=0}^{r} \in \mathbf{A}(P[\,p\,..\,p'\,), T[\,t\,..\,t'\,)), ((|P|,\varepsilon),(|T|,\varepsilon))$$

is the subsequence of $\mathcal{A}$ consisting of pairs $((p_j, P[\,p_j\,]), (t_j, T[\,t_j\,]))$ such that $\mathcal{A}$ does not match $P[\,p_j\,]$ with $T[\,t_j\,]$, padded with dummy entries at the front and at the back. Observe that the size of the breakpoint representation is $2 + \mathsf{ed}_{\mathcal{A}}(P,T)$ and that it can be used to retrieve the entire alignment: for any two consecutive elements $((x',\cdot),(y',\cdot)), ((x,\cdot),(y,\cdot))$ of the breakpoint representation, it suffices to add $(x-\delta, y-\delta)$ for $\delta \in (\,0\,..\,\max(x-x', y-y')\,)$. Consult Figure 4 for a visualization of an example. Unless stated otherwise, throughout this work, we represent alignments via their breakpoint representations without always explicitly specifying it.



▪**a** An alignment $\mathcal{A} : P \rightsquigarrow T$ of cost 8. Observe that each breakpoint (except for the first and last) corresponds to an edit operation that immediately follows.



▪**b** The alignment $\mathcal{A}$ translates a partition of $P = P[0..6)P[6..10)P[10..13)P[13..22)$ into a partition of $T = \mathcal{A}(P) = \mathcal{A}(P[0..6))\mathcal{A}(P[6..10))\mathcal{A}(P[10..13))\mathcal{A}(P[13..22)) = T[0..4)T[4..9)T[9..14)T[14..22)$.

■ **Figure 4.** The breakpoint representation of an alignment $\mathcal{A} : P \rightsquigarrow T$ and the corresponding strings, as well as an example how $\mathcal{A}$ translates fragments (and partitions) of P. The (edit operations of the) alignment are depicted as colored boxes; breakpoints are depicted as vertical lines separating the boxes of the alignment. For edit operations, we depict the characters involved; otherwise we depict the fragments of P and T that are matched. Consult Figure 3a for a visualization of the alignment graph of $P[0..11)$ and $T[0..12)$, as well as the corresponding part of $\mathcal{A}$.

Given $\mathcal{A} = (p_j, t_j)_{j=0}^{r} \in \mathbf{A}(P, T)$, we define the *inverse alignment* as $\mathcal{A}^{-1} := (t_j, p_j)_{j=0}^{r} \in \mathbf{A}(T, P)$.
We define the *weighted edit distance* of strings $P, T \in \Sigma^*$ with respect to a weight function w as

$$\mathsf{ed}^w(P \to T) := \min_{\mathcal{A} \in \mathbf{A}(P,T)} \mathsf{ed}^w_{\mathcal{A}}(P \to T);$$

that is, as the length of a shortest path from (p, t) to (p', t') in $\mathsf{AG}^w(P, T)$.

Fact 3.4 [DGH+23, Fact 2.5]. *If w is a metric on $\bar{\Sigma}$, then ed^w is a metric on $\bar{\Sigma}^*$.*

For an integer $k \geq 0$, we also define a capped version

$$\mathsf{ed}^w_{\leq k}(P \to T) := \begin{cases} \mathsf{ed}^w(P \to T) & \text{if } \mathsf{ed}^w(P \to T) \leq k, \\ \infty & \text{otherwise.} \end{cases}$$

We say that an alignment $\mathcal{A} \in \mathbf{A}(P, T)$ is w-optimal (or just optimal if there is no ambiguity) if $\mathsf{ed}^w_{\mathcal{A}}(P \to T) = \mathsf{ed}^w(P \to T)$.

For an alignment $\mathcal{A} : X[x..x') \rightsquigarrow Y[y..y')$ and a fragment $X[\bar{x}..\bar{x}')$ that is contained in $X[x..x')$, we write $\mathcal{A}(X[\bar{x}..\bar{x}'))$ for the fragment $Y[\bar{y}..\bar{y}')$ that is contained in $Y[y..y')$ which $\mathcal{A}$ aligns against $X[\bar{x}..\bar{x}')$. To avoid ambiguities, we formally set

$$\bar{y} := \min\{\hat{y} : (\bar{x}, \hat{y}) \in \mathcal{A}\} \quad \text{and} \quad \bar{y}' := \begin{cases} y' & \text{if } \bar{x}' = x', \\ \min\{\hat{y}' : (\bar{x}', \hat{y}') \in \mathcal{A}\} & \text{otherwise.} \end{cases}$$

This particular choice satisfies the following decomposition property.

Fact 3.5 (Alignments transfer decomposistions) [CKW22, CKW23]. *For any alignment $\mathcal{A}$ of X onto Y and a decomposition $X = X_1 \cdots X_t$ into t fragments, $Y = \mathcal{A}(X_1) \cdots \mathcal{A}(X_t)$ is a decomposition into t fragments with*

$$\mathsf{ed}^w_{\mathcal{A}}(X \to Y) \geq \sum_{i=1}^{t} \mathsf{ed}^w(X_i \to \mathcal{A}(X_i)).$$

Further, if $\mathcal{A}$ is an optimal alignment, then we have equality: $\mathsf{ed}^w_{\mathcal{A}}(X \to Y) = \sum_{i=1}^{t} \mathsf{ed}^w(X_i \to \mathcal{A}(X_i))$. ◢

For a text T, a pattern P, a weight function w, and an integer threshold k, a (k, w)-*error occurrence of P in T* is a position i such that for some $j \geq i$ we have $\mathsf{ed}^w(P \to T[\,i\,..\,j\,)) \leq k$.[9] We write

$$\mathrm{Occ}^w_k(P, T) \coloneqq \{i : \exists_{j \geq i}\ \mathsf{ed}^w(P \to T[\,i\,..\,j\,)) \leq k\}.$$

for the set of all (k, w)-error occurrence of P in T.

Remark 3.6. If for every $a, b \in \bar{\Sigma}$ we have that $w(a \mapsto b) = 1$ if $a \neq b$ and $w(a \mapsto b) = 0$ otherwise, then $\mathsf{ed}^w(X \to Y)$ corresponds to the standard *unweighted* edit distance (also known as Levenshtein distance [Lev65]). For this case, we drop the superscript w in $\mathsf{AG}^w(X, Y)$, ed^w, $\mathsf{ed}^w_{\mathcal{A}}$, $\mathsf{ed}^w_{\leq k}$, and $\mathrm{Occ}^w_k(P, T)$.

Additionally, since the unweighted edit distance is symmetric, we may write $\mathsf{ed}(X, Y)$ instead of $\mathsf{ed}(X \to Y)$, and, similarly for $\mathsf{ed}_{\mathcal{A}}$ and $\mathsf{ed}_{\leq k}$. ◢

For any two strings X, Y, any normalized weight function w, and any alignment $\mathcal{A} : X \rightsquigarrow\!\!\!\!\!\rightarrow Y$, we have $\mathsf{ed}^w_{\mathcal{A}}(X \to Y) \geq \mathsf{ed}_{\mathcal{A}}(X, Y)$. This simple observation yields the following fact.

Fact 3.7. *For any strings P, T, any integer threshold k, and any normalized weight function w, we have* $\mathrm{Occ}^w_k(P, T) \subseteq \mathrm{Occ}_k(P, T)$. ◢

We write

$$\mathsf{ed}(S, T^*) \coloneqq \min\{\mathsf{ed}(S, T^\infty[\,0\,..\,j\,)) : j \in \mathbb{Z}_{\geq 0}\}$$

for the minimum edit distance between a string S and any prefix of a string T^∞. Further, we write

$$\mathsf{ed}(S, {}^*T^*) \coloneqq \min\{\mathsf{ed}(S, T^\infty[\,i\,..\,j\,)) : i, j \in \mathbb{Z}_{\geq 0}, i \leq j\}$$

for the minimum edit distance between S and any substring of T^∞, and we set

$$\mathsf{ed}(S, {}^*T) \coloneqq \min\{\mathsf{ed}(S, T^\infty[\,i\,..\,j|T|\,)) : i, j \in \mathbb{Z}_{\geq 0}, i \leq j|T|\}.$$

Finally, we discuss a notion for the similarity of strings from a family of strings.

Definition 3.8. *The* median edit distance *of a family $\mathcal{S}$ of strings over an alphabet Σ is*

$$\mathsf{ed}(\mathcal{S}) \coloneqq \min_{\hat{S} \in \Sigma^*} \sum_{S \in \mathcal{S}} \mathsf{ed}(S, \hat{S}).$$ ◢

Remark 3.9 (Bounded median edit distance implies small size and small pairwise edit distance)**.** As the unweighted edit distance has a triangle inequality, a bound $\mathsf{ed}(\mathcal{S}) \leq d$ implies $||S| - |T|| \leq \mathsf{ed}(S, T) \leq d$ for any $S, T \in \mathcal{S}$. Further, as two non-equal strings have an edit distance of at least 1, we also have $|\mathcal{S}| = O(d)$. ◢

[9] We assume that k is integer as we often wish to define objects in terms of k and the usage of $\lceil k \rceil$ would create unnecessary clutter. Our algorithms may be adapted to handle any $k \in \mathbb{R}_{\geq 0}$.

Planar Graphs, the Monge Property, and (min, +)-Multiplication

Alignments graphs are *planar*. The *multiple-source shortest paths* (MSSP) data structure of Klein [Kle05] represents all shortest path trees rooted at the vertices of a single face f in a planar graph G of size n using a persistent dynamic tree.

Theorem 3.10 (Efficient MSSP on planar graphs) [Kle05]**.** *Given a directed planar graph G of size n with a distinguished face f and non-negative edge weights, we can construct in $O(n \log n)$ time an $O(n \log n)$-space data structure that, for any two vertices $u, v \in V(G)$, at least one of which is lies on f, computes the distance $\mathsf{dist}_G(u, v)$ in $O(\log n)$ time. Moreover, the shortest $u \rightsquigarrow v$ path P can be reported in $O(|P| \log\log \Delta(G))$ time, where $\Delta(G)$ is the maximum degree of G.*

Pairwise distances of vertices that lie on the outer face of a strongly connected planar graph satisfy the *Monge property*.

Fact 3.11 (Distance matrices of planar graphs are Monge) [FR06, Section 2.3]**.** *Consider a strongly connected directed planar graph G with non-negative edge weights. For vertices $u_0, \ldots, u_{p-1}, v_{q-1}, \ldots, v_0$ that (in this cyclic order) lie on the outer face of G, define a $p \times q$ matrix D with $D[\,i, j\,] = \mathsf{dist}_G(u_i, v_j)$.*

Then, D satisfies the Monge property*, that is,*

$$D[\,i, j\,] + D[\,i', j'\,] \leq D[\,i, j'\,] + D[\,i', j\,]$$

holds for all integers $0 \leq i \leq i' < p$ and $0 \leq j \leq j' < q$.

The composition of shortest paths in graphs can be interpreted as the (min, +)-multiplication of the corresponding distance matrices. Formally, given an $p \times q$ matrix A and an $q \times r$ matrix B, their (min, +)-product is the $p \times r$ matrix $C = A \oplus B$ is defined as

$$C[\,i, j\,] \coloneqq \min_{k \in [\,0\,..\,q\,)} \big(A[\,i, k\,] + B[\,k, j\,]\big).$$

By $A^{\oplus(t)}$, we denote the matrix equal to the (min, +)-product of t copies of a square matrix A.

In a seminal work [AKM+87], Aggarwal, Klawe, Moran, Shor, and Wilber presented an efficient algorithm to compute the (min, +)-product of Monge matrices given $O(1)$-time random access to their entries.

Theorem 3.12 [AKM+87]**.** *We can compute the* (min, +)*-product of an $p \times q$ Monge matrix and an $q \times r$ Monge matrix using $O(p \cdot r \cdot (1 + \log(q/\max(p, r))))$ queries to single positions of the input matrices.*

Furthermore, if matrices A and B are Monge, then their (min, +)-product $A \oplus B$ is also Monge.

Fact 3.13 [Tis15, Theorem 2]**.** *Let A, B, and C be matrices, such that $A \oplus B = C$. If A and B are Monge, then C is also Monge.*

Augmented Alignment Graphs

Although the alignment graph $\mathsf{AG}^w(P, T)$ is planar, it is not strongly connected, so we cannot apply Fact 3.11. To circumvent this issue, we augment it with back edges of sufficiently large weight.

Definition 3.14 (The augmented alignment graph $\overline{\mathsf{AG}}^w(P, T)$ for a weight function w and strings P and T) [GK25]**.** *For strings $P, T \in \Sigma^*$ and a weight function $w : \bar{\Sigma}^2 \to [\,0\,..\,W\,]$, we define the* augmented alignment graph $\overline{\mathsf{AG}}^w(P, T)$ *obtained from $\mathsf{AG}^w(X, Y)$ by adding, for every edge of $\mathsf{AG}^w(P, T)$, a back edge of weight $W + 1$.*

Remark 3.15. For our purpose of turning the alignment graph into a strongly connected graph, we technically do not need the diagonal back edges: insisting on both horizontal and vertical edges elegantly captures the special cases of either string being empty, but thereby also renders the diagonal edges superfluous. Nevertheless, we keep the diagonal back edges to stay in line with [GK25]; which in particular allows us to use their results in an opaque manner.

We proceed with a couple of useful properties of (shortest paths in) the augmented alignment graph.

Lemma 3.16 (Basic properties of the alignment graph) [GK25, Lemma 5.2]**.** *Consider strings $P, T \in \Sigma^*$ and a weight function $w : \bar{\Sigma}^2 \to [0, W]$. Any two vertices (p, t) and (p', t') of the graph $\overline{\mathsf{AG}}^w(P, T)$ satisfy the following properties.*

Monotonicity. *Shortest paths $(p, t) \rightsquigarrow (p', t')$ in $\overline{\mathsf{AG}}^w(P, T)$ are (non-strictly) monotone in both coordinates.*

Distance preservation. *If $p \le p'$ and $t \le t'$, then*

$$\mathsf{dist}_{\mathsf{AG}^w(P,T)}((p,t),(p',t')) = \mathsf{dist}_{\overline{\mathsf{AG}}^w(P,T)}((p,t),(p',t')).$$

Path irrelevance. *If ($p \le p'$ and $t \ge t'$) or ($p \ge p'$ and $t \le t'$), every path from (p, t) to (p', t') in $\overline{\mathsf{AG}}^w(P, T)$, that is monotone in both coordinates, is a shortest path between these two vertices.*

Remark 3.17. Consider a path $\mathcal{P}$ of cost at most k in the augmented alignment graph $\overline{\mathsf{AG}}^w(P, T)$ that originates at a vertex $(0, i)$ and terminates at a vertex $(|P|, j)$ for some $i, j \in [0\,..\,|T|]$.

Since w is normalized, non-diagonal edges of $\overline{\mathsf{AG}}^w(P, T)$ are of cost at least 1 each and connect adjacent diagonals. Consequently, $\mathcal{P}$ intersects at most $k + 1$ subsequent diagonals: it may only intersect the diagonals that are at most k to the left or to the right of both the i-th diagonal of the origin $(0, i)$ and the $(j - |P|)$-th diagonal of the target $(|P|, j)$. Thus, every vertex (p, t) of the path $\mathcal{P}$ satisfies $t - p \in [i - k\,..\,i + k] \cap [j - |P| - k\,..\,j - |P| + k]$,[10] and there is at least one such vertex for each $p \in [0\,..\,|P|]$.

We summarize Remark 3.17 in Lemma 3.18.

Lemma 3.18 (Paths of cost k that connect the top and bottom of the alignment graph may use at most k non-diagonal edges)**.** *Consider strings $P \in \Sigma^m$ and $T \in \Sigma^n$, and a threshold $k \in \mathbb{Z}_{\ge 0}$. Let $\mathcal{P}$ denote a path of cost at most k that connects the top boundary to the bottom boundary of $\overline{\mathsf{AG}}^w(P, T)$.*

For every vertex $(p, t) \in \mathcal{P}$, we have $-k \le t - p \le n - m + k$.

We may use Theorem 3.10 to compute boundary-to-boundary paths in the alignment graph.

Corollary 3.19. *Given a string P of length m, a string T of length n, and oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$, after an $O(nm \log(nm))$-time preprocessing, we can compute the distance between any two boundary vertices of $\overline{\mathsf{AG}}^w(X, Y)$ in $O(\log(nm))$ time.*

Proof. At preprocessing, we build an MSSP data structure over $\overline{\mathsf{AG}}^w(P, T)$, with the infinite face being the distinguished face.

[10] We could potentially restrict the interval for $t - p$ even further: transferring from diagonal i to diagonal $j - |P|$ incurs a cost of at least $|j - |P| - i|$; deviating to outside the diagonals between i and $j - |P|$ requires a return to the said diagonals. Hence, $\mathcal{P}$ may in fact deviate by only at most $\lfloor (k - |j - |P| - i|)/2 \rfloor \le k/2$ from the diagonals between i and $j - |P|$. However, to decrease the technical complexity a tiny little bit, we do not use this further refinement here.

Our Model of Computation and the PILLAR Model

Throughout this work, we consider the Real RAM model of computation that, besides typical word RAM operations, allows us to store, add, subtract, and compare real numbers in constant time (we do not use multiplication and other advanced operations)—this is a standard assumption for algorithms computing distances in graphs with real weights. If all weights admit an $O(\log n)$-bit fixed-point arithmetic representation, standard word RAM is sufficient to implement our algorithms.

We use the PILLAR model of [CKW20] to express our algorithms for the Verify problem and our algorithm for PMwWE in Section 7. In particular, we bound the running times of said algorithms in terms of the number of calls to a small set of very common operations (the primitive PILLAR operations) on strings and calls to standard operations supported constant time in the Real RAM model of computation.

Combining Main Theorem 2 with efficient implementations of said primitive operations, we then obtain (fast) algorithms for PMwWE in a plethora of settings: the standard setting, the compressed setting, the dynamic setting, the quantum setting, the packed setting, and the read-only setting. See Section 10 for details.

We continue with a brief summary of the PILLAR model. In the PILLAR model, we maintain a collection of strings $\mathcal{X}$; the operations in the PILLAR model work on fragments $X[\,\ell\,..\,r\,)$ of $X \in \mathcal{X}$, which are represented via a *handle*.[11] At the start of the computation, the PILLAR model provides a handle to each $X \in \mathcal{X}$, which represents $X[\,0\,..\,|X|\,)$. Using an Extract operation, we may obtain handles to other fragments of the strings in $\mathcal{X}$ [CKW20].

- $\texttt{Extract}(S, \ell, r)$: Given a fragment S and positions $0 \le \ell \le r \le |S|$, extract the (sub)fragment $S[\,\ell\,..\,r\,)$. If $S = X[\,\ell'\,..\,r'\,)$ for $X \in \mathcal{X}$, then $S[\,\ell\,..\,r\,)$ is defined as $X[\,\ell'+\ell\,..\,\ell'+r\,)$.

The other primitive PILLAR model operations read as follows [CKW20].

- $\texttt{LCP}(S, T)$: Compute the length of the longest common prefix of S and T.
- $\texttt{LCP}^R(S, T)$: Compute the length of the longest common suffix of S and T.
- $\texttt{IPM}(P, T)$: Assuming that $|T| \le 2|P|$, compute $\mathrm{Occ}(P, T)$ (represented as an arithmetic progression with difference $\mathrm{per}(P)$).
- $\texttt{Access}(S, i)$: Assuming $i \in [\,0\,..\,|S|\,)$, retrieve the character $S[\,i\,]$.
- $\texttt{Length}(S)$: Retrieve the length $|S|$ of the string S.

The Landau–Vishkin algorithm for computing the edit distance of two strings [LV88] can be reformulated as follows in the PILLAR model (see [CKW20, Lemma 6.1]).

Lemma 3.20 (Alignment (S, T)) [LV88]. *Given strings S and T and a threshold d, we can construct (the breakpoint representation of) an alignment $\mathcal{A} : S \rightsquigarrow T$ of optimal cost $\mathsf{ed}(S, T) \le d$ in $O(1 + d^2)$ time in the* PILLAR *model (or decide in the same time that no such alignment may exist).*

An efficient procedure in the PILLAR for computing an optimal alignment under a normalized weight function was presented in [CKW23].

Lemma 3.21 (WeightedED (S, T, w)) [CKW23, Theorem 4.1]. *Given strings S and T of length at most n and oracle access to a normalized weight function w, we can compute the value $k = \mathsf{ed}^w(S \to T)$ in time $O(1 + k^3 \log^2 n)$ in the* PILLAR *model.*

Finally, the adaptation of the Landau–Vishkin algorithm [LV89] already provided in [CKW22] lets us efficiently compute an optimal alignment of a string S onto a substring of Q^∞ starting at a given position x.

[11] The implementation details depend on the specific setting. For example, in the standard setting, a fragment $X[\,\ell\,..\,r\,)$ is represented by a reference to X and the endpoints ℓ, r.

Lemma 3.22 (`Alignment (S, Q, x)`) [CKW22, Lemma 2.10]. *Consider non-empty strings S, Q and an integer $x \in \mathbb{Z}$. We can construct (the breakpoint representation of) an alignment $\mathcal{A} : S \rightsquigarrow Q^{\infty}[\, x \,..\, y\,)$ of optimal cost $d := \mathsf{ed}(S, \mathrm{rot}^{-x}(Q)^*)$ in $O(1 + d^2)$ time in the* `PILLAR` *model.*

4 Pattern Matching with Weighted Edits in $\widetilde{O}(nk)$ Time

In this section, we prove Main Theorem 1. As a first simple step, we split the text T into a set $\mathcal{S}$ of overlapping fragments, such that it suffices to compute, for each $S \in \mathcal{S}$, the positions of S in an interval $[\, 0 \,..\, O(k)\,]$ where a (k, w)-error occurrence of P starts.

Remark 4.1. For a pattern P of length m, a text T of length n, and a threshold $k > 1$ write $\mathcal{S}$ for a set of $O(n/k)$ fragments $S_i := T[\, i \,..\, \max\{i + m + 2k, n\}\,)$ such that $i \equiv 0 \pmod{k}$.

We have $\mathrm{Occ}_k^w(P, T) = \bigcup_{S_i \in \mathcal{S}} i + \mathrm{Occ}_k^w(P, S_i)$, as each fragment U of T with $\mathsf{ed}^w(P \to U) \le k$ has a length in $[\, m - k \,..\, m + k\,]$ and hence is contained in at least one and at most four fragments $S \in \mathcal{S}$.

In a similarly standard fashion, we may filter out any $S \in \mathcal{S}$ that is too far from P under the unweighted edit distance to contain any (k, w)-error occurrences. For the remaining $S \in \mathcal{S}$, we obtain a cheap unweighted alignment $P \rightsquigarrow S$ as a useful byproduct.

Lemma 4.2. *Consider a threshold $k > 0$, a text T of length n, and a pattern P of length $m \le n + k$.*

- *For any fragment S of T of length at most $m + 2k$, if $\mathsf{ed}(P, S) > 4k$ then $\mathrm{Occ}_k^w(P, S) = \emptyset$.*
- *After preprocessing T for $O(n)$ time, given any fragment S of T of length at most $m + 2k$, we can in $O(k^2)$ time check whether $\mathsf{ed}(P, S) \le 4k$, and, if so, compute the breakpoint representation of an optimal unweighted alignment $\mathcal{A} : P \rightsquigarrow S$.*

Proof. We prove the contraposition of the first item. Fix a fragment S of T and suppose that $\mathrm{Occ}_k^w(P, S) \neq \emptyset$; that is, suppose that there are positions ℓ and r and an alignment $\mathcal{B} : P \rightsquigarrow S[\, \ell \,..\, r\,)$ of weighted cost at most k. We argue that $\mathcal{B}$ may be transformed into an unweighted alignment $\mathcal{A} : P \rightsquigarrow S$ of cost at most $4k$: consider the (unweighted) alignment $\mathcal{A} : P \rightsquigarrow S$ that is obtained as an extension from $\mathcal{B}$ by adding insertions for $S[\, 0 \,..\, \ell\,)$ and $S[\, r \,..\, |S|\,)$ appropriately.

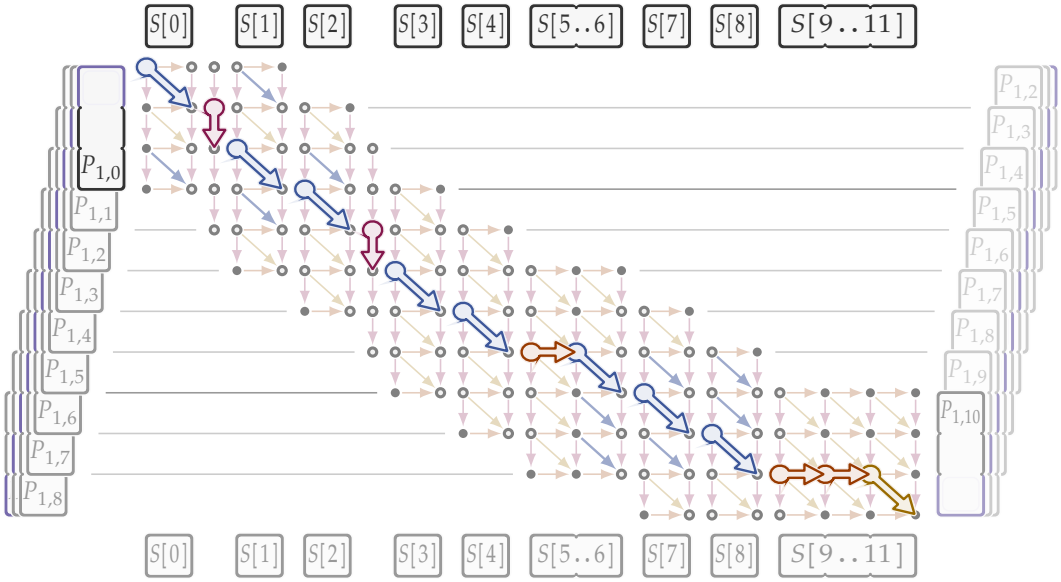
First, from $\mathsf{ed}(P, S[\, \ell \,..\, r\,)) \le \mathsf{ed}^w(P \to S[\, \ell \,..\, r\,)) \le k$ we conclude that $\mathcal{B}$ contributes a cost of at most k to $\mathcal{A}$. Next, $\mathsf{ed}^w(P \to S[\, \ell \,..\, r\,)) \le k$ additionally implies $|S[\, \ell \,..\, r\,)| \ge m - k$; which in turn implies $|S| - |S[\, \ell \,..\, r\,)| \le (m + 2k) - (m - k) = 3k$. Hence, the extra insertions incur a total cost of at most $3k$. In total, $\mathcal{A}$ has thus a cost of at most $4k$, completing the proof.

The second item follows directly from the combination of Lemma 3.20 and Theorem 10.1.

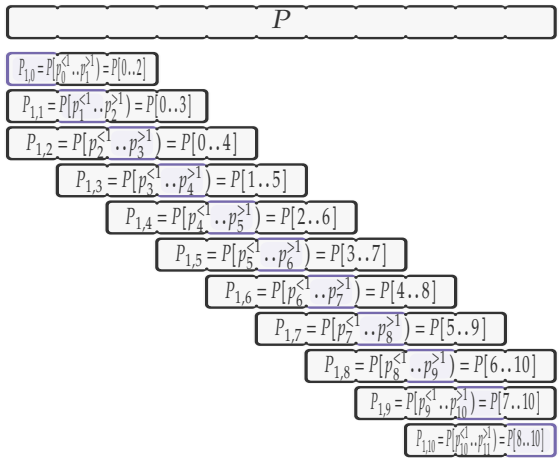
In a next step, we intend to use the cheap unweighted alignments $P \rightsquigarrow S$, and in particular that any such alignment matches large parts of P with S without any edits. As many of said matched parts are shared across the different $S \in \mathcal{S}$, we intend to save on computation time by essentially precomputing boundary-to-boundary shortest-path distances for the parts where fragments of P are matched to fragments of P. For a formal exposition, we start with naming parts of the corresponding alignment graphs. Consult Figures 5 and 6 for a visualization of an example.

Definition 4.3 (The i-th d-slice $P_{d,i}$ of a string P, the graphs $G^{P,S,d}_{\ell,r}$ (and $G^{P,d}_{\ell,r}$) as subgraphs of $\overline{\mathsf{AG}}^w(P, S)$ (and $\overline{\mathsf{AG}}^w(P, P)$) and the corresponding distance matrices $D^{P,S,d}_{\ell,r}$ (and $D^{P,d}_{\ell,r}$)). *Consider a string P of length m, a threshold $d > 0$, and an alignment $\mathcal{A} : P \rightsquigarrow S$ of cost at most d.*

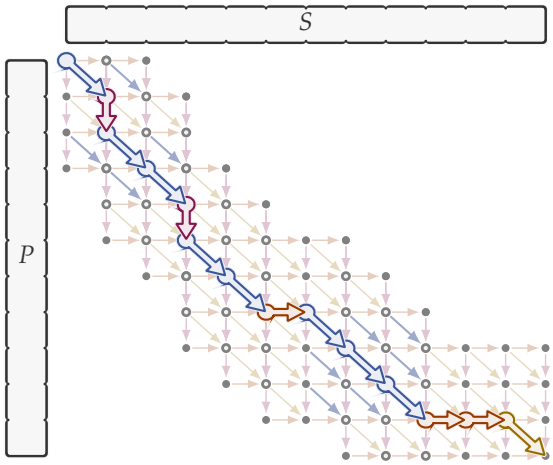
- *For $i \in [\, 0 \,..\, m\,)$, the i-th d-slice of P is the string $P_{d,i} := P[\, p_i^{<d} \,..\, p_{i+1}^{>d}\,)$, where $p_i^{<d} := \max(0, i - 2d)$ and $p_{i+1}^{>d} := \min(m, i + 1 + 2d)$. Further, we write $s_i := \min\{s : (i, s) \in \mathcal{A}\}$ for $i \in [\, 0 \,..\, m\,)$ and $s_m := |S|$.*



▪**a** For each $i \in [0..|P|]$, we depict the slice $G^{P,S,d}_{i,i}$ as a subgraph of $\overline{\mathsf{AG}}^w(P,S)$, the corresponding fragments of P and S, as well as the corresponding part of the alignment $\mathcal{A}$. For each slice, we highlight in gray its portal vertices. Observe that portal vertices of consecutive slices correspond to the same vertices in $\overline{\mathsf{AG}}^w(P,S)$.



▪**b** The d-slices of P in detail. For each slice $P_{d,i}$, we highlight the character $P[i]$.



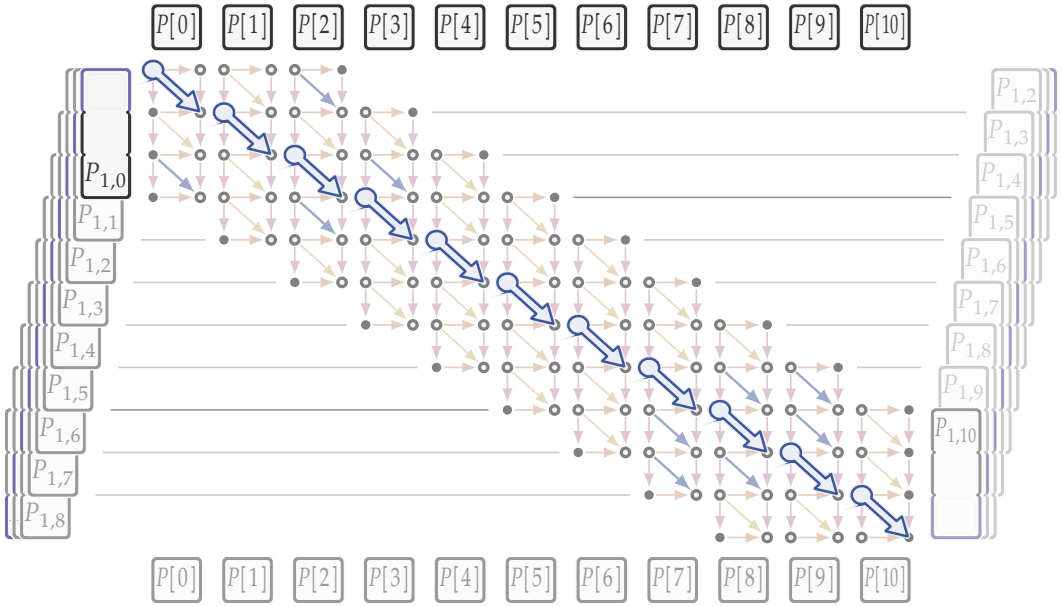
▪**c** The graph $G^{P,S,d}$, which is obtained by identifying the portal vertices of consecutive slices of Figure 5a. Again, we highlight portal vertices in gray.

■ **Figure 5.** The d-slices $P_{d,i}$ of P, the d-slices $G^{P,S,d}_{i,i}$ of $\overline{\mathsf{AG}}^w(P,S)$, and the graph $G^{P,S,d}$ with respect to alignment $\mathcal{A} : P \rightsquigarrow S$ from Figures 3 and 4. In order to conserve space but still have a meaningful example, we depict slices for $d = 1$, even though alignment alignment $\mathcal{A}$ is of cost 6. For the depicted example this does not cause any issues. In general, the alignment needs to be fully contained in the slices—which holds when the alignment is of cost at most d.

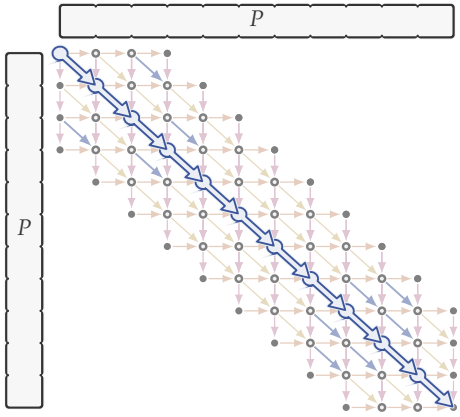
▬ *For $i \in [0..m)$, the i-th d-slice of $\overline{\mathsf{AG}}^w(P,S)$ is the subgraph $G^{P,S,d}_{i,i} := \overline{\mathsf{AG}}^w(P_{d,i}, \mathcal{A}(P[i]))$ (induced by $[p_i^{<d} .. p_{i+1}^{>d}] \times [s_i .. s_{i+1}]$).*

For $0 \le \ell < r < m$, we write $G^{P,S,d}_{\ell,r} := \bigcup_{\ell \le j \le r} G^{P,S,d}_{j,j}$ and we write $G^{P,S,d} := G^{P,S,d}_{0,m}$.

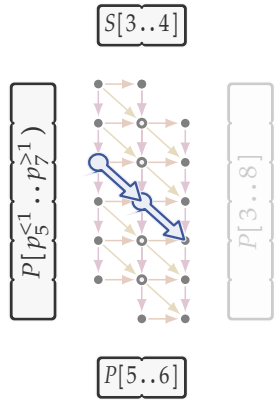
We also write $V^{P,S,d}_i := [p_i^{<d} .. p_i^{>d}] \times \{s_i\}$ for the i-th set of portal vertices of $G^{P,S,d}$; we index the vertices top-to-bottom, that is, $V^{P,S,d}_i = \{V^{P,S,d}_i[0], \ldots, V^{P,S,d}_i[p_i^{>d} - p_i^{<d}]\}$ with $V^{P,S,d}_i[j] := (p_i^{<d} + j, s_i)$.



▪ **a** For each $i \in [0\,..\,|P|]$, we depict the slice $G^{P,d}_{i,i}$ as a subgraph of $\overline{\mathsf{AG}}^w(P,P)$, the corresponding fragments of P, as well as the corresponding part of the alignment $\mathcal{A}$; compare Figure 5a. For each slice, we highlight in gray its portal vertices. Observe that portal vertices of consecutive slices correspond to the same vertices in $\overline{\mathsf{AG}}^w(P,S)$.



▪ **b** The graph $G^{P,S,d}$, which is obtained by identifying the portal vertices of consecutive slices of Figure 6a; compare Figure 5c Again, we highlight portal vertices in gray.



▪ **c** The subgraph $G^{P,S,d}_{5,6}$ in detail. As the alignment $\mathcal{A} : P \rightsquigarrow S$ matches $P[5\,..\,6] = S[3\,..\,4]$, we have $G^{P,S,d}_{5,6} = G^{P,d}_{5,6}$.

■ **Figure 6.** The d-slices $G^{P,d}_{i,i}$ of $\overline{\mathsf{AG}}^w(P,P)$, and the graph $G^{P,d}$ with respect to the trivial identity alignment; compare Figure 5. We also depict a detailed view of the subgraph $G^{P,S,d}_{5,6} = G^{P,d}_{5,6}$.

▬ *For $0 \le \ell \le r \le m$, we write $D^{P,S,d}_{\ell,r}$ for the matrix of distances from $V^{P,S,d}_{\ell}$ to $V^{P,S,d}_{r}$, that is,*

$$D^{P,S,d}_{\ell,r}[\,a,b\,] := \mathsf{dist}_{G^{P,S,d}}((p^{<d}_{\ell} - a, s_{\ell}), (p^{>d}_{r} - b, s_r))$$

for $a \in [\,0\,..\,p^{>d}_{\ell} - p^{<d}_{\ell}\,]$ and $b \in [\,0\,..\,p^{>d}_{r} - p^{<d}_{r}\,]$.
For the special case of $S = P$ and $\mathcal{A} = \mathrm{id}$, we omit the superscript S and write $G^{P,d}$, $V^{P,d}$, and $D^{P,d}$. ◢

Next, we formally define the data structure problem that enables us to efficiently compute $\mathrm{Occ}^w_k(P,S)$ for each $S \in \mathcal{S}$.

Problem Distance Matrix Vector Oracle, DMVO$(P,\ d,\ w)$.

Input. `DMVO-Init`$(P,\ d,\ w)$: Given a pattern P of length m, a positive integer d, and oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$, preprocess P and d.

Queries. `DMVO-Query`$(i,\ j,\ v)$: For integers $0 \le i \le j \le m$ and a vector v of dimension $(p_j^{>d} - p_j^{<d} + 1)$, return the vector $D^{P,d}_{i,j} \oplus v$.

Remark 4.4. One might wonder why a `DMVO-Query` computes a matrix-vector product instead of allowing for direct random access to $D^{P,d}_{i,j}$. Crucially, this allows us to exploit that $D^{P,d}_{i,j} \oplus v$ may be computed via $D^{P,d}_{i,j} \oplus v = D^{P,d}_{i,c} \oplus (D^{P,d}_{c,j} \oplus v)$ for $c \in [i..j]$—which ultimately allows us to save a logarithmic factor compared to an approach that first computes (single elements of) $D^{P,d}_{i,c} \oplus D^{P,d}_{c,j}$. Indeed, given $O(\log m)$-time random access to $D^{P,d}_{i,c}$ and $D^{P,d}_{c,j}$, one may obtain $O(\log^2 m)$-time random access to $D^{P,d}_{i,c} \oplus D^{P,d}_{c,j}$ by rewriting each such random access as a multiplication of a $1 \times d$ and a $d \times 1$ (Monge) matrix and employing Theorem 3.12 (that is, the SMAWK algorithm).

Lemma 4.5. *We can implement DMVO such that* `DMVO-Init` *takes time* $O(md \log^2 m)$ *and a single* `DMVO-Query` *takes time* $O(d \log m)$.

Proof. `DMVO-Init`. For each $s \in [0..\lceil \log m \rceil]$, we partition $[0..m)$ into subintervals of length 2^s,[12] and for each such subinterval $[\ell..r)$, we use Klein's algorithm (Theorem 3.10) on $G^{P,d}_{\ell,r}$ to compute a data structure with $O(\log m)$-time random access to $D^{P,d}_{\ell,r}$, and—crucially—$D^{P,d}_{\ell,i}$ and $D^{P,d}_{i,r}$ for every $i \in [\ell..r]$.

`DMVO-Query`. Suppose that we are given integers $0 \le i \le j \le m$ and a vector v. We first compute the largest integer s such that $[i..j)$ contains an integer $x := c \cdot 2^s$ (for an integer $c \ge 0$). Then, we consider two consecutive length-2^s intervals $[x - 2^s..x)$ and $[x..x + 2^s)$ for which we have precomputed (random access to) the corresponding distance matrices $D^{P,d}_{x-2^s,x}$ and $D^{P,d}_{x,x+2^s}$—and, crucially, $D^{P,d}_{i,x}$ and $D^{P,d}_{x,j}$. Using Theorem 3.12 (the SMAWK algorithm) twice, we compute and return $D^{P,d}_{i,x} \oplus (D^{P,d}_{x,j} \oplus v)$.

Correctness. First observe that every path from $V^{P,d}_i$ to $V^{P,d}_j$ in $G^{P,d}$ crosses $V^{P,d}_x$ for each $x \in (i..j)$. Thus, we know that $D^{P,d}_{i,j} = D^{P,d}_{i,x} \oplus D^{P,d}_{x,j}$. Further, by Fact 3.11, $D^{P,d}_{i,x}$ and $D^{P,d}_{x,j}$ are Monge.

Even though we compute distances in graphs $G^{P,d}_{\ell,r}$ for different pairs (ℓ, r), they are also distances in $G^{P,d}$; this is implied by Lemma 3.16 (path monotonicity in $\overline{\mathsf{AG}}^w(P,P)$) and the fact that $V^{P,d}_\ell$ and $V^{P,d}_r$ separate $G^{P,d}_{\ell,r}$ from the remainder of $G^{P,d}$. In particular, such a preprocessing of $G^{P,d}_{\ell,r}$ yields $O(\log m)$-time random access to $D^{P,d}_{\ell,i}$ and $D^{P,d}_{i,r}$ for every $i \in (\ell..r)$.

Recall that for an interval $[i..j)$ we choose s as the maximal integer for which an integer c exists with $i \le x = c \cdot 2^s < j$. Observe that this choice implies that $[i..j)$ contains neither $(c-1) \cdot 2^s$ nor $(c+1) \cdot 2^s$, that is, we have $x - 2^s = (c-1) \cdot 2^s < i$ and $j \le (c+1) \cdot 2^s = x + 2^s$. Hence, our precomputation of $D^{P,d}_{x-2^s,x}$ and $D^{P,d}_{x,x+2^s}$ indeed includes random access to $D^{P,d}_{i,x}$ and $D^{P,d}_{x,j}$.

Running time. For the running time of `DMVO-Init`, observe that for fixed ℓ and r, applying Theorem 3.10 (Klein's algorithm) to $G^{P,d}_{\ell,r}$ costs $O(d \cdot (r - \ell + 1) \cdot \log m)$ time; which yields $O(md \cdot \log m)$ total time for all intervals of a fixed length. Across the $O(\log m)$ different interval lengths, the preprocessing thus takes time $O(md \log^2 m)$ in total.

The running time of `DMVO-Query` is as claimed as it is dominated by a constant number of calls to Theorem 3.12 for $O(d) \times O(d)$ Monge matrices to which we have $O(\log m)$-time random access.

Next, we establish how to use DMVO to obtain a fast algorithm to compute (k, w)-error occurrences of P in a string $S \in \mathcal{S}$.

[12] For each s, the last subinterval might be shorter.

Algorithm 1. Using DMVO for a sped-up computation of (k, w)-error occurrences of a string P in a string S that is given via a breakpoint representation of an alignment of cost at most d. Observe that without using DMVO, the algorithm would amount to computing the top-to-bottom shortest-path distances in $G^{P,S,d}$ using Klein's algorithm (Theorem 3.10). We achieve a speed-up by using precomputed shortest-path distances for the subgraphs where $\mathcal{A}$ matches fragments exactly.

`BoostedPM`$(\mathcal{D} \leftarrow$ `DMVO`$(P,\ d,\ w), \mathcal{A} : P \rightsquigarrow S \leftarrow \{((0,\varepsilon),(0,\varepsilon)), \ldots, ((m,\varepsilon),(|S|,\varepsilon))\}, d, k, w)$

Using a left-to-right pass over the breakpoints in $\mathcal{A}$, compute
- the source vertices $V^{\top} := \{v_i^{\top} : i \in [\![0\,..\,s_{2d+1}]\!]\}$ with $v_i^{\top} := (i, 0)$;
- the left-center boundary portal-set $V^{P,S,d}_{2d+1}$;
- the set of important portal positions $I := \{i_0 = 2d+1, \ldots, i_q = m-2d-1\}$, where a portal position i_j is *marked* if it is to the right of a pure range, that is, if $\mathcal{A}(P[\![i_{j-1}\,..\,i_j]\!]) = P[\![i_{j-1}\,..\,i_j]\!]$;
- the center-right boundary portal-set $V^{P,S,d}_{m-2d-1}$;
- and the target vertices $V^{\perp} := \{v_i^{\perp} : i \in [\![0\,..\,|S| - s_{m-2d-1}]\!]\}$ with $v_i^{\perp} := (i + s_{m-2d-1}, m)$.

Use Klein's algorithm (Theorem 3.10) on $G^{P,S,d}_{m-2d-1,m}$ to obtain $D_{m-2d-1,\perp}$, where $D_{m-2d-1,\perp}[\![i,j]\!] := \mathsf{dist}_{G^{P,S,d}}(V^{P,S,d}_{m-2d-1}[\![i]\!], v_j^{\perp})$;

Use SMAWK (Theorem 3.12) for $v \leftarrow D_{m-2d-1,\perp} \oplus \vec{0}$;

foreach $j \leftarrow q, \ldots, 1$ **do**
- **if** *portal position* i_j *is marked* **then** $v \leftarrow$ `DMVO-Query`(i_{j-1}, i_j, v);
- **else** Use Klein's algorithm on $G^{P,S,d}_{i_{j-1},i_j}$ to obtain $D^{P,S,d}_{i_{j-1},i_j}$; then SMAWK for $v \leftarrow D^{P,S,d}_{i_{j-1},i_j} \oplus v$;

Use Klein's algo. on $G^{P,S,d}_{0,2d+1}$ to obtain $D_{\top,2d+1}$, where $D_{\top,2d+1}[\![i,j]\!] := \mathsf{dist}_{G^{P,S,d}}(v_i^{\top}, V^{P,S,d}_{2d+1}[\![j]\!])$; then SMAWK for $v \leftarrow D_{\top,2d+1} \oplus v$;

return $\{i : v[\![i]\!] \le k\}$;

Lemma 4.6. *Suppose that we are given an instance of a DMVO data structure for a positive integer threshold d, a pattern P of length $m > 4d+1$, and a normalized weight function $w : \bar{\Sigma}^2 \to [\![0, W]\!]$.*

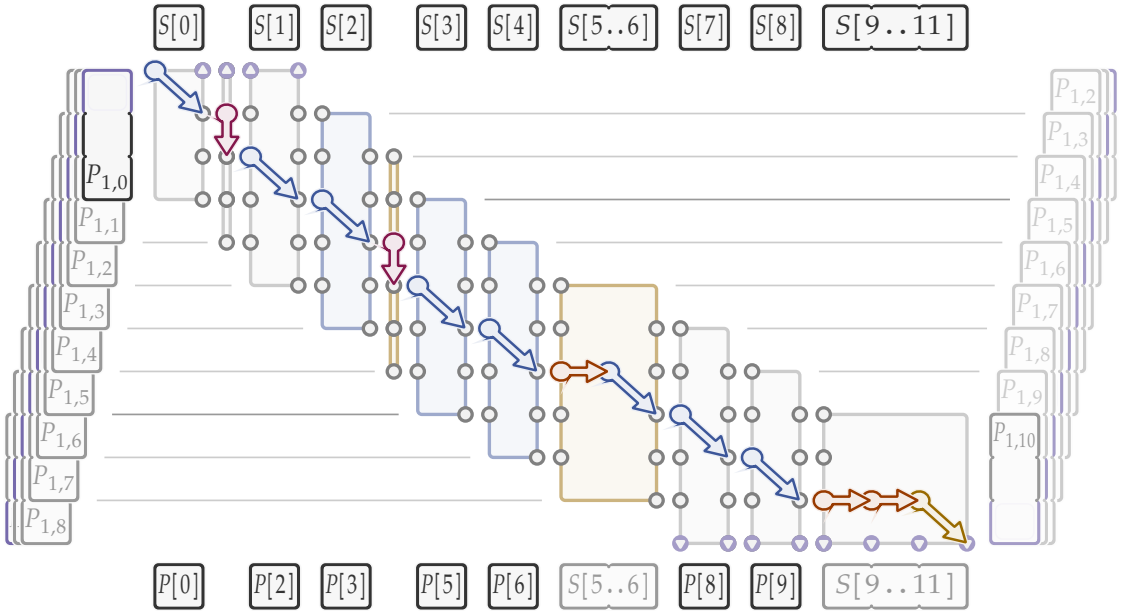
Given the breakpoint representation of an alignment $\mathcal{A} : P \rightsquigarrow S$ of unweighted cost at most d (implicitly defining a string $S \in \Sigma^*$) and a threshold $k \in [\![0\,..\,d]\!]$, we can compute $\mathrm{Occ}^w_k(P,S)$ using $O(d)$ calls to `DMVO-Query` *and $O(d^2 \log m)$ extra time.*

Proof. Notations. We say that a position (or slice) i is *left* if $p_i^{<d} = 0$ (or $i \le 2d$) and we say that a position (or slice) i is *right* if $p_{i+1}^{>d} = m$ (or $i \ge m-2d-1$). A slice is *center* if it is neither left nor right. Our assumption $m > 4d+1$ ensures that each position (and slice) is at most one of left, right, and center.

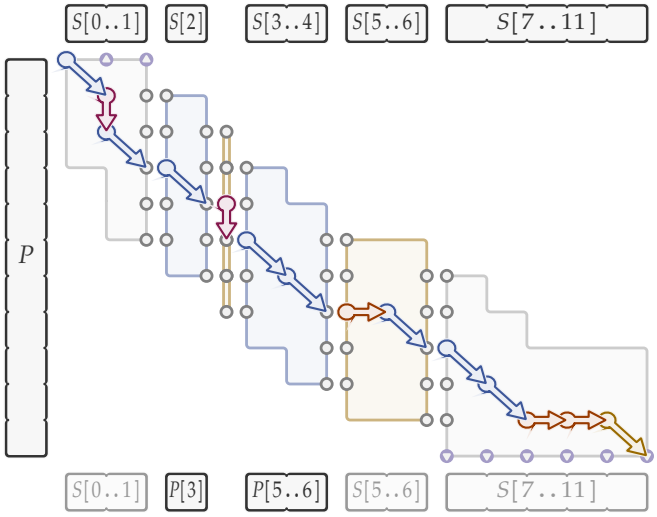
We say that a (center) position i with $\mathcal{A}(P[\![i]\!]) = P[\![i]\!]$ is *pure*. We group maximal stretches of pure (center) positions into *pure stretches*. We say that a position where a pure stretch starts is *important*, and we say that a position where a pure stretch ends is *important and marked*. We write $I := \{i_0 = 2d+1, \ldots, i_q = m-2d-1\}$ for the set of important portal positions, where we artificially add i_0 and i_q in case they are not present.

Next, we name important vertices of $G^{P,S,d}$. The leftmost center portal-set $V^{P,S,d}_{i_0}$ is the *left-center boundary*; the rightmost center portal-set $V^{P,S,d}_{i_q}$ is the *center-right boundary*. The *source* vertices are the topmost vertices of left slices, that is, $V^{P,S,d}_{\top} := \{0\} \times [\![s_0\,..\,s_{2d+1}]\!]$; the *target* vertices are the bottommost vertices of right slices, that is, $V^{P,S,d}_{\perp} := \{m\} \times [\![s_{m-2d-1}\,..\,s_m]\!]$. Observe that $V^{P,S,d}_{\perp}$ and $V^{P,S,d}_{\top}$ are the only vertices in $G^{P,S,d}$ with the first coordinate equal to m and 0, respectively.

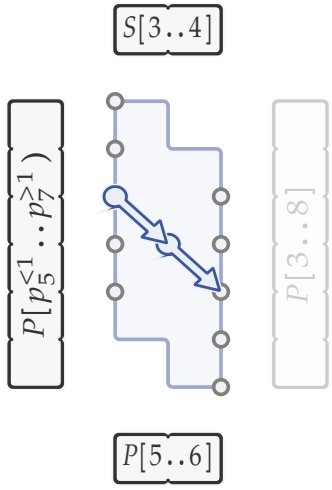
Consult Figure 7 for a visualization of an example.



▪**a** For each $i \in [0..|P|]$, we depict the slice $G^{P,S,d}_{i,i}$ in a stylized manner, the corresponding fragments of P and S, as well as the corresponding part of the alignment $\mathcal{A}$; compare Figure 5a. For each slice, we highlight in gray its portal vertices; we also highlight source vertices and target vertices in light purple. Left and right slices are colored gray, the (center) pure slices are colored blue, while the remaining slices are colored yellow.



▪**b** The graphs $G^{P,S,d}_{i_q,m}$ and $G^{P,S,d}_{0,i_0}$ (colored gray), as well as for each maximal interval $[i_{j-1}..i_j]$ of center positions that are pure (colored blue) and not pure (colored yellow) the graph $G^{P,S,d}_{i_{j-1},i_j}$ as a union of slices from Figure 7a. We again highlight portal vertices, as well as source and target vertices.



▪**c** The subgraph $G^{P,S,d}_{5,6}$ in a stylized manner; compare Figure 6c. As the alignment $\mathcal{A} : P \rightsquigarrow S$ matches $P[5..6] = S[3..4]$, we have $G^{P,S,d}_{5,6} = G^{P,d}_{5,6}$.

■ **Figure 7.** Stylized variants of the d-slices $G^{P,S,d}_{i,i}$ of $\overline{\mathsf{AG}}^w(P,S)$ and the subgraphs considered by the algorithm underlying Lemma 4.6. The source vertices in the top left and the target vertices in the bottom right are marked in light purple. Consecutive slices with the same color are merged. The portals are exactly those vertices that belong to subgraphs with different colors.

Algorithm description. We first use a single pass over the breakpoint description of $\mathcal{A}$ to identify the source vertices, the left-center boundary, the boundaries of (center) pure stretches (and the corresponding portal-sets $V^{P,S,d}_{i_j}$ for $j \in (0..q)$), the center-right boundary, and the target vertices. Additionally, using (the breakpoint representation of) $\mathcal{A}$, we explicitly construct $G^{P,S,d}_{i_q,m}$, $G^{P,S,d}_{0,i_0}$, as well as $G^{P,S,d}_{i_{j-1},i_j}$ for each maximal interval $[i_{j-1}..i_j]$ of center positions that are not pure in a straightforward manner in time linear in the total size of these graphs.

Using Klein's algorithm (Theorem 3.10), we compute (random access to) the matrix of distances from any center-right boundary vertex to any target vertex in $G^{P,S,d}_{i_q,m}$; we then min-plus multiply the obtained matrix with a suitably-sized zero vector using Theorem 3.12 (SMAWK algorithm) to obtain a vector v.

Next, we process the center positions in I from right to left. For a pure stretch (i_{j-1}, i_j), that is, when i_j is marked, we use a `DMVO-Query` to compute the min-plus product $D^{P,d}_{i_{j-1},i_j} \oplus v$. For a non-pure stretch (i_{j-1}, i_j), that is, when i_j is not marked, we use Klein's algorithm (Theorem 3.10) on $G^{P,S,d}_{i_{j-1},i_j}$ to obtain (random access to) $D^{P,S,d}_{i_{j-1},i_j}$ and use Theorem 3.12 to compute $D^{P,S,d}_{i_{j-1},i_j} \oplus v$.

Finally, again using Klein's algorithm (Theorem 3.10), we compute (random access to) the matrix of distances from any source vertex to any left-center boundary vertex in $G^{P,S,d}_{0,i_0}$; we then min-plus multiply the obtained matrix with v using Theorem 3.12 (SMAWK algorithm) and return any position of the resulting vector that has value at most k.

Consult Algorithm 1 for a detailed description as pseudo-code.

Correctness. We argue about the correctness in two steps. First, we show that each shortest $(0, \ell)$-to-(m, r) path of length at most k in $G^{P,S,d}$ indeed corresponds to an alignment of P onto $S[\,\ell\,..\,r\,)$ of the same cost. We then argue that portal-sets act as separators of $G^{P,S,d}$ and thus allow us to split the distance computations and reuse the precomputed distances via DMVO whenever the corresponding subgraph of $G^{P,S,d}$ is isomorphic to a subgraph of $G^{P,d}$ (encapsulated in DMVO).

Claim 4.7. *For each fragment $S[\,\ell\,..\,r\,)$ of S and each $k \in [\,0\,,\,d\,]$, the following two statements are equivalent:*

1. $\mathsf{ed}^w(P \to S[\,\ell\,..\,r\,)) \le k$; *and*
2. $(0, \ell)$ *is a source,* (m, r) *is a target, and* $\mathsf{dist}_{G^{P,S,d}}((0, \ell), (m, r)) \le k$.

Proof. The (1) $\Leftarrow$ (2) implication follows directly from Lemma 3.16 (distance preservation between $\mathsf{AG}^w(P, S)$ and $\overline{\mathsf{AG}}^w(P, S)$) and the fact that $G^{P,S,d}$ is a subgraph of $\overline{\mathsf{AG}}^w(P, S)$.

For the converse implication, suppose that there is an alignment $O : P \rightsquigarrow S[\,\ell\,..\,r\,)$ of weighted cost at most $k \le d$. It suffices to prove that O (interpreted as a path from $(0, \ell)$ to (m, r) in $\overline{\mathsf{AG}}^w(P, S)$) is contained in $G^{P,S,d}$. By Lemma 3.18, every vertex $(x, y) \in O$ satisfies $y \in [\,x - d\,..\,x + |S| - m + d\,]$. Let us focus on a single edge $(x, y) \to (x', y')$ of O with $x' \in \{x, x + 1\}$ and $y' \in \{y, y + 1\}$. Let us fix an index $i \in [\,0\,..\,m\,)$ such that $[\,y\,..\,y'\,] \subseteq [\,s_i\,..\,s_{i+1}\,]$.

Since the path corresponding to $\mathcal{A}$ intersects at most d diagonals, we have $(i - s_i) - (m - |S|) \le d$. Additionally, $s_i \le y \le x + |S| - m + d$. Combining these two inequalities, we get $i \le x + 2d$, that is, $x \ge i - 2d$. Since $x \ge 0$, we conclude that $x \ge \max(0, i - 2d) = p^{<d}_i$.

Similarly, we have $(s_{i+1} - (i + 1)) - (0 - 0) \le d$ and $s_{i+1} \ge y' \ge x' - d$. Combining these two inequalities, we get $x' - d - (i + 1) \le d$, that is, $x' \le (i + 1) + 2d$. Since $x \le m$, we conclude that $x \le \min(m, i + 1 + 2d) = p^{>d}_{i+1}$. Thus, the edge $(x, y) \to (x', y')$ is contained in $\overline{\mathsf{AG}}^w(P'_i, S_i)$ and hence in $G^{P,S,d}$. ◿

Next, observe that for pure positions, the corresponding d-slices of $\overline{\mathsf{AG}}^w(P, S)$ and $\overline{\mathsf{AG}}^w(P, P)$ are indeed isomorphic, that is, we have $G^{P,S,d}_{i,s_i} = G^{P,d}_{i,i}$—naturally, this observation extends to pure stretches. Hence, our calls to `DMVO-Query` for a pure stretch (i_{j-1}, i_j) compute in fact $D^{P,S,d}_{i_{j-1},i_j} \oplus v = D^{P,d}_{i_{j-1},i_j} \oplus v$.

Finally, observe that each portal-set $V^{P,S,d}_{i_j}$ is indeed a separator of $G^{P,S,d}$—which allows us to split shortest path computations along this set and then combine the results using a min-plus product.

In total, we indeed compute, for each source vertex, its minimum shortest-path distance to a target vertex.

Running time. First observe that the subgraphs corresponding to left and right positions (of which there are $O(d)$ in total) each have a size of $O(d^2)$. Next, as the alignment $\mathcal{A}$ makes at most d edits, the total size of subgraphs between pure stretches is again $O(d^2)$ (as they span $O(d)$ positions of S). Similarly, we may bound the number of pure stretches by $O(d)$—which yields the number of calls to `DMVO-Query`. Finally, we perform $O(d)$ min-plus matrix-vector multiplications using SMAWK (Theorem 3.12).

In total, our calls to Klein's algorithm (Theorem 3.10) take time $O(d^2 \log m)$; all performed min-plus multiplications take the same time in total. ◢

Finally, we combine the various intermediate results of this section to obtain our first main result.

Main Theorem 1. *PMwWE can be solved in $O((n \log m + m \log^2 m) \cdot k)$ time.*

Proof. We first proceed as in Remark 4.1 to obtain a set $\mathcal{S}$ of $O(n/k)$ strings, each of length at most $m + 2k$. It suffices to compute $\mathrm{Occ}_k^w(P, S)$ for all $S \in \mathcal{S}$ and merge the results in $O(n)$ time via bucket sort.

Next, we preprocess T for Lemma 4.2 and we initialize Lemma 4.5 on P, $d := 4k$, and w.

For each $S \in \mathcal{S}$, we proceed as follows. First, if $m \leq 4d + 1$, we use the classical dynamic programming algorithm to compute $\mathrm{Occ}_k^w(P, S)$. Otherwise, we use Lemma 4.2 to obtain a weighted alignment $\mathcal{A}_S : P \rightsquigarrow S$ of cost at most $d := 4k$ or conclude that $\mathrm{Occ}_k^w(P, S) = \emptyset$. Next, we use Lemma 4.6 on $\mathcal{A}_S$, with the initialized DMVO data structure, $d := 4k$ and k.

We briefly argue about the running time. Our global preprocessing takes time $O(km \log^2 m)$. For a fixed $S \in \mathcal{S}$, Lemmas 4.2, 4.5 and 4.6 combined run in time $O(k^2 \log m)$; in the case when we run the classic DP algorithm, we have $|S| \leq m + 2k < m + d = O(d)$ and thus the DP algorithm takes time $O((|P|+1)\cdot(|S|+1)) = O(d^2) = O(k^2)$. Over all $O(n/k)$ elements of $\mathcal{S}$, this takes $O((n/k)(k^2 + k^2 \log m)) = O(nk \log m)$ time. In total, we obtain the claimed running time. ◢

5 Toolbox for Monge Matrices and Ferns

In this section, we recall useful results on the representation, the structure, and the multiplication of Monge matrices. Further, we introduce the notion of *fern matrices*. A fern matrix represents some pairwise distances from a set of leftmost nodes in the top boundary of an alignment graph to a set of rightmost nodes in the bottom boundary of this alignment graph. Specifically, a fern matrix and the corresponding actual boundary-to-boundary distance matrix must agree on all values that are at most k, that is, the two matrices are k-equivalent. Fern matrices are crucial for the following reason. While we may not be able to efficiently compute the actual boundary-to-boundary distance matrix, we show that we can efficiently compute a (k-equivalent) fern matrix that is Monge.

5.1 Toolbox for Monge Matrices

Explicitly storing Monge matrices is in general too expensive for our purposes. Fortunately, their structure often allows us to represent them using much less space.

Definition 5.1. *of a matrix A; the condensed representation of A) For a matrix A of size $p \times q$, the* density matrix *of A, denoted $A^\square$, is a matrix of size $(p-1) \times (q-1)$ such that*

$$A^\square[\,i, j\,] = A[\,i, j+1\,] + A[\,i+1, j\,] - A[\,i, j\,] - A[\,i+1, j+1\,]$$

for $i \in [\,0\,..\,p-1\,)$ and $j \in [\,0\,..\,q-1\,)$.

The core *of a matrix A is the set*

$$\mathrm{core}(A) := \{(i, j, A^\square[\,i, j\,]) : i \in [\,0\,..\,p-1\,), j \in [\,0\,..\,q-1\,), A^\square[\,i, j\,] \neq 0\}.$$

We denote the size of the core *of A by $\delta(A) := |\mathrm{core}(A)|$.*

For a matrix A, the condensed representation of A *comprises in the values in the topmost row and the leftmost column of A and the core of A.* ◢

Fact 5.2. *The core of any contiguous submatrix of a matrix A is at most $\delta(A)$.*

Monge matrices with small core admit small representations that allow fast access to their entries.

Definition 5.3 [GK25, Lemma 4.4]**.** *The* Core-based Matrix Oracle data structure $\mathsf{CMO}(A)$ *of a matrix A of size $p \times q$ with $N := \max(p, q)$ provides the following interface.*
Random access: *given $i \in [0\,..\,p)$ and $j \in [0\,..\,q)$, in time $O(\log N)$ returns $A[i, j]$.*
Full core listing: *in time $O(1 + \delta(A))$ returns* $\mathrm{core}(A)$.

Lemma 5.4 [GK25, Lemma 4.4]**.** *Given a matrix A of size $p \times q$, we can construct a core-based matrix oracle for A in time $O(pq + \delta(A)\log(p + q))$.*

Proof. Compared to [GK25, Lemma 4.4], we pay $O(pq)$ to first compute the core of A which (together with the first row and column of A) use as the input to the original algorithm.

Given the core-based matrix oracle data structure of two matrices, one can efficiently obtain the core-based matrix oracle data structure of their min-plus product.

Lemma 5.5 [GK25, Lemma 4.6]**.** *There is an algorithm that, given $\mathsf{CMO}(A)$ and $\mathsf{CMO}(B)$ for Monge matrices $A \in \mathbb{Z}_{\geq 0}^{p\times q}$ and $B \in \mathbb{Z}_{\geq 0}^{q\times r}$ with $N := \max(p, q, r)$, builds $\mathsf{CMO}(A \oplus B)$ in time $O((N + \delta(A) + \delta(B))\log N)$.*

The following operation enables us to efficiently retrieve representations of contiguous submatrices of Monge matrices with small core.

Lemma 5.6 [GK25, Lemma 4.7]**.** *There is an algorithm that, given $\mathsf{CMO}(A)$ for a Monge matrix A of maximum dimension N, builds $\mathsf{CMO}(C)$ in time $O((N+\delta(A))\log N)$ for any contiguous submatrix C of A.*

For our purposes, in the interest of efficiency, instead of exactly computing a desired distance matrix D, it suffices to compute a different matrix D' that agrees with D on all values that do not exceed some threshold k.

Definition 5.7. *For a real threshold $k \in \mathbb{R}$, we say that real numbers $a, b \in \mathbb{R}$ are* k-equivalent*, denoted $a \overset{k}{=} b$, if $a = b$ or $k \leq \min\{a, b\}$.*
Two matrices $A, B \in \mathbb{R}^{p\times q}$ are k-equivalent, denoted $A \overset{k}{=} B$, if all their entries are k-equivalent, that is, $A[i, j] \overset{k}{=} B[i, j]$ holds for each $i \in [0\,..\,p)$ and $j \in [0\,..\,q)$.

Remark 5.8. Equivalently, two numbers a and b are k-equivalent if they are equal whenever at least one of them is at most k.
The $\overset{k}{=}$ relation is a *congruence* with respect to the $(\min, +)$ algebra on $\mathbb{R}_{\geq 0}$: it is an equivalence relation such that, for every $a, a', b, b' \in \mathbb{R}_{\geq 0}$, if $a \overset{k}{=} b$ and $a' \overset{k}{=} b'$, then $a + a' \overset{k}{=} b + b'$ and $\min\{a, a'\} \overset{k}{=} \min\{b, b'\}$. Thus, $\overset{k}{=}$ is also a congruence with respect to the $(\min, +)$-product.

Consider a scenario where we have to compute a matrix that is k-equivalent to the product of a (long) size-ℓ sequence of integer Monge matrices of dimensions at most N with cores at most t. In this scenario, we can use the following lemma to ensure that the core of all computed matrices does not exceed $\widetilde{O}(\sqrt{k})$ and that the output can be produced in time $\widetilde{O}(\ell \cdot N \cdot (t + \sqrt{k}))$.

Lemma 5.9 [GK25, Lemma 4.14]**.** *There is an algorithm that, given $\mathsf{CMO}(A)$ and $\mathsf{CMO}(B)$ for Monge matrices $A \in \mathbb{Z}_{\geq 0}^{p\times q}$ and $B \in \mathbb{Z}_{\geq 0}^{q\times r}$ with $N := \max(p, q, r)$ and a $k \in \mathbb{Z}_{>0}$, in time $O((N + \delta(A) + \delta(B))\log N)$ builds $\mathsf{CMO}(C')$, where $C' \in \mathbb{Z}_{\geq 0}^{p\times r}$ is a Monge matrix that satisfies $\delta(C') \leq O(N\sqrt{k})$ and $C' \overset{k}{=} A \oplus B$.*

In the regime of small integer edit weights, we often operate on bounded-difference matrices.

Definition 5.10 [GK25, Definition 4.8]. *Let $\beta \in \mathbb{Z}_{>0}$ be a positive integer and let $A \in \mathbb{Z}^{p\times q}$ be an integer matrix. We say that A is β-bounded-difference if*
- $|A[\,i,j\,] - A[\,i,j-1\,]| \le \beta$ *holds for each* $i \in [\,0\,..\,p\,)$ *and* $j \in [\,1\,..\,q\,)$, *and*
- $|A[\,i,j\,] - A[\,i-1,j\,]| \le \beta$ *holds for each* $i \in [\,1\,..\,p\,)$ *and* $j \in [\,0\,..\,q\,)$.

Fact 5.11. *Any contiguous submatrix of an integer-valued β-bounded-difference matrix is also an integer-valued β-bounded-difference matrix.*

Crucially, integer-valued bounded-difference Monge matrices have small cores.

Lemma 5.12 [GK25, Lemma 4.9]. *If a Monge matrix $A \in \mathbb{Z}^{p\times q}$ is β-bounded-difference, then $\delta(A) \le 2\beta(\min\{p,q\}-1)$.*

Moreover, the bounded-difference property is preserved under the min-plus product.

Lemma 5.13 [GK25, Lemma 4.10]. *Let $A \in \mathbb{Z}^{p\times q}$ and $B \in \mathbb{Z}^{q\times r}$ be β-bounded-difference matrices. Then, $C \coloneqq A \oplus B$ is also a β-bounded-difference matrix.*

5.2 Ferns and Distance Matrices

Definition 5.14 [GK25, Definitions 5.5 and 5.6]. *For two strings $X, Y \in \Sigma^*$, the* boundary distance matrix *$D^w_{X,Y}$ stores the distances from every* input *vertex on the top-left boundary to every* output *vertex on the bottom-right boundary of the alignment graph $\mathsf{AG}^w(X,Y)$. Formally, consider the sequence of points $(\lambda_i)_{i\in[\,0\,..\,|X|+|Y|\,]}$ with*

$$\lambda_i \coloneqq \begin{cases} (|X|-i,0) & \text{for } i \in [\,0\,..\,|X|\,], \\ (0,i-|X|) & \text{for } i \in [\,|X|\,..\,|X|+|Y|\,], \end{cases}.$$

and the sequence of points $(\rho_i)_{i\in[\,0\,..\,|X|+|Y|\,]}$ with

$$\rho_i \coloneqq \begin{cases} (|X|,i) & \text{for } i \in [\,0\,..\,|Y|\,], \\ (|X|+|Y|-i,|Y|) & \text{for } i \in [\,|Y|\,..\,|X|+|Y|\,]. \end{cases}$$

Then, $D^w_{X,Y}$ is an $(|X|+|Y|+1)\times(|X|+|Y|+1)$ distance matrix with

$$D^w_{X,Y}[\,a,b\,] \coloneqq \mathsf{dist}_{\mathsf{AG}^w(X,Y)}(\lambda_a,\rho_b).$$

The following observation allows us to utilize the toolbox described in Section 5.

Fact 5.15 [GK25, Observation 5.7]. *For two strings $X, Y \in \Sigma^*$ and a weight function $w : \bar{\Sigma}^2 \to [\,0\,..\,W\,]$, the matrix $D^w_{X,Y}$ is $(W+1)$-bounded-difference and hence $\delta(D^w_{X,Y}) = O(W\cdot(|X|+|Y|))$.*

Recall from the technical overview that a *puzzle piece* is a pair of strings; to keep in line with our intuition that the said strings are substrings of a pattern P and a text T, we typically denote puzzle pieces as $(\dot{P},\dot{T})$.

Definition 5.16. *For a puzzle piece $(\dot{P},\dot{T})$, write $G \coloneqq \overline{\mathsf{AG}}^w(\dot{P},\dot{T})$ for the corresponding augmented alignment graph and dist_G for the shortest path distances in G. For $(\dot{P},\dot{T})$, and integer $\Delta,\nabla \in [\,0\,..\,|\dot{T}|\,]$, the $(\Delta\to\nabla)$-*restricted distance matrix *of $(\dot{P},\dot{T})$ with respect to w is a $(\Delta+1)\times(\nabla+1)$ matrix $D^w_{\dot{P},\dot{T}}|_{\Delta\to\nabla}$ where, for each $i \in [\,0\,..\,\Delta\,]$ and $j \in [\,0\,..\,\nabla\,]$, we set*

$$D^w_{\dot{P},\dot{T}}|_{\Delta\to\nabla}[\,i,j\,] \coloneqq \mathsf{dist}_G((0,i),(|\dot{P}|,|\dot{T}|-\nabla+j)).$$

If $\Delta=\nabla$, we simplify our notation and write Δ instead of $\Delta\to\nabla$.

Remark 5.17. The $(\Delta \to \nabla)$-restricted distance matrix of Definition 5.16 stores selected costs to traverse the alignment graph from the top boundary to the bottom boundary.

In particular, if $i \le |\dot{T}| - \nabla + j$, then $D^{w}_{\dot{P},\dot{T}}\big|_{\Delta\to\nabla}[\,i, j\,] = \mathsf{ed}^{w}(\dot{P} \to \dot{T}[\,i\,..\,|\dot{T}| - \nabla + j\,))$. Consequently, $D^{w}_{\dot{P},\dot{T}}\big|_{\Delta\to\nabla}$ encodes the edit distances from $\dot{P}$ to all fragments of $\dot{T}$ that start within the $\Delta+1$ leftmost positions of $\dot{T}$ and end within the $\nabla+1$ rightmost positions of $\dot{T}$. As long as $k \le |\dot{P}| - |\dot{T}| + \min\{\Delta, \nabla\}$, this includes all fragments of length at least $|\dot{P}| - k$, and hence all (k, w)-error occurrences of $\dot{P}$ in $\dot{T}$.

From Fact 3.11, we also learn that $D^{w}_{\dot{P},\dot{T}}\big|_{\Delta\to\nabla}$ is indeed a Monge matrix, as it is a submatrix of the boundary distance matrix. ◢

Unfortunately, we are not able to directly compute the matrix $D^{w}_{\dot{P},\dot{T}}\big|_{\Delta}$ in an efficient manner. Nevertheless, for our purposes, it suffices to compute a suitable approximation—which we can do efficiently. We formalize the required concepts next.

Definition 5.18. *For a puzzle piece $(\dot{P}, \dot{T})$, integer $\Delta, \nabla \in [\,0\,..\,|\dot{T}|\,]$, and a threshold $k \in \mathbb{R}_{\ge 0}$, a $(\Delta \to \nabla, k)$-*fern[13] *of strings $\dot{P}$ and $\dot{T}$ with respect to w is a matrix that is k-equivalent to $D^{w}_{\dot{P},\dot{T}}\big|_{\Delta\to\nabla}$. We write $\mathcal{F}^{w}_{\dot{P},\dot{T}}\big|^{k}_{\Delta\to\nabla}$ for the set of all $(\Delta \to \nabla, k)$-ferns of $\dot{P}$ and $\dot{T}$ with respect to w.*

Whenever $\Delta = \nabla$, we simplify our notation and write Δ instead of $\Delta \to \nabla$. ◢

Remark 5.19. Ferns are not uniquely defined; that is, typically we have $|\mathcal{F}^{w}_{\dot{P},\dot{T}}\big|^{k}_{\Delta\to\nabla}| > 1$. This is intentional to allow for efficient algorithms that compute and multiply ferns. ◢

Lemma 5.20. *Consider a puzzle piece $(\dot{P}, \dot{T})$, integers $\Delta', \Delta, \nabla', \nabla \in [\,0\,..\,|\dot{T}|\,]$ with $\Delta' \le \Delta$ and $\nabla' \le \nabla$, and thresholds $k', k \in \mathbb{R}_{\ge 0}$ with $k' \le k$. For any $F \in \mathcal{F}^{w}_{\dot{P},\dot{T}}\big|^{k}_{\Delta\to\nabla'}$, we have*

$$F' \coloneqq F[\,[\,0\,..\,\Delta'\,], [\,\nabla - \nabla'\,..\,\nabla\,]\,] \in \mathcal{F}^{w}_{\dot{P},\dot{T}}\big|^{k'}_{\Delta'\to\nabla'}.$$

Further the core of F' is at most $\delta(F)$ and, if F is integer-valued and β-bounded difference, then so is F'.

Proof. For all $i \in [\,0\,..\,\Delta'\,]$ and $j \in [\,0\,..\,\nabla'\,]$, we have

$$\begin{aligned} F[\,i, \nabla - \nabla' + j\,] &\overset{k}{=} \mathsf{dist}_G((0, i), (|\dot{P}|, |\dot{T}| - \nabla + \nabla - \nabla' + j)) \\ &= \mathsf{dist}_G((0, i), (|\dot{P}|, |\dot{T}| - \nabla' + j)). \end{aligned}$$

This implies that F' is k-equivalent (and hence k'-equivalent) to $D^{w}_{\dot{P},\dot{T}}\big|_{\Delta'\to\nabla'}$.

The claimed upper bounds on the core and the difference of F' follow instantly from Fact 5.2 and Fact 5.11, respectively. ◢

For a puzzle piece $(\dot{P}, \dot{T})$, we define its *shift* as $\mathsf{shift}(\dot{P}, \dot{T}) \coloneqq |\dot{T}| - |\dot{P}|$ and its *torsion* as $\mathsf{tor}(\dot{P}, \dot{T}) \coloneqq |\mathsf{shift}(\dot{P}, \dot{T})|$. The torsion of a sequence of puzzle pieces is the sum of the torsions of the individual pieces.

We ultimately intend to fit together multiple puzzle pieces into longer strings, so-called Δ-puzzles. We formalize related concepts next.

Definition 5.21. *For an integer $\Delta > 0$, we say that strings $S, S' \in \Sigma^{\ge\Delta}$ fit (with respect to Δ) if they have an overlap of size Δ, that is, if*

$$S[\,|S| - \Delta\,..\,|S|\,) = S'[\,0\,..\,\Delta\,).$$

In this case, we denote their Δ-concatenation *as*

$$S \odot_{\Delta} S' \coloneqq S \odot S'[\,\Delta\,..\,|S'|\,) = S[\,0\,..\,|S| - \Delta\,) \odot S'.$$

[13] In the unweighted setting, a similar structure called *seaweed* fulfills a similar purpose when paired with its corresponding *seaweed product*. Hence, we chose the name *fern*, as common knowledge tells us that (liquid) seaweed (extract) is a good fertilizer to cultivate many species of ferns.

Two puzzle pieces $(\dot{P}, \dot{T})$ *and* $(\ddot{P}, \ddot{T})$ fit *(with respect to* Δ*), denoted by* $(\dot{P}, \dot{T}) \mapsto_\Delta (\ddot{P}, \ddot{T})$*, if*

$$\dot{T}[\,|\dot{T}| - \Delta\,..\,|\dot{T}|\,) = \ddot{T}[\,0\,..\,\Delta\,) \quad \text{and} \quad \dot{P}[\,|\dot{P}| - \Delta\,..\,|\dot{P}|\,) = \ddot{P}[\,0\,..\,\Delta\,).$$

For two fitting puzzle pieces $(\dot{P}, \dot{T}) \mapsto_\Delta (\ddot{P}, \ddot{T})$*, we define the* stitched piece *as* $(\dot{P} \odot_\Delta \ddot{P}, \dot{T} \odot_\Delta \ddot{T})$. ⌟

Remark 5.22. For a sequence of puzzle pieces $\mathcal{P} := (\dot{P}_1, \dot{T}_1), \ldots, (\dot{P}_z, \dot{T}_z)$ where consecutive pieces Δ-fit, we also say that the sequences $\dot{P}_1, \ldots, \dot{P}_z$ and $\dot{T}_1, \ldots, \dot{T}_z$ form Δ-puzzles; we write $P := \mathsf{val}_\Delta(\dot{P}_1, \ldots, \dot{P}_z)$ and $T := \mathsf{val}_\Delta(\dot{T}_1, \ldots, \dot{T}_z)$ for the resulting stitched strings. For simplicity, we also say that the sequence $\mathcal{P}$ has the value (P, T). ⌟

For a puzzle piece $(\dot{P}, \dot{T})$ and a fixed Δ, the first Δ and the last Δ characters of either string may end up overlapping corresponding parts of a neighboring puzzle piece. Hence, when dealing with the alignment graphs of pieces (and when computing shortest paths therein), we have to ensure that we avoid duplicate vertices along the paths that we consider: if we were to naively compute top-to-bottom distances for each piece, we would be unable to combine such distances for consecutive pieces. This motivates the following definition.

Definition 5.23. *For a puzzle piece* $(\dot{P}, \dot{T})$ *and an integer* Δ*, besides* $D^w_{\dot{P},\dot{T}}|_\Delta$*, we define*

- *the* internal *distance matrix* $\overset{\circ}{\circ}D^w_{\dot{P},\dot{T}}|_\Delta := D^w_{\dot{P}[\,\lfloor\Delta/2\rfloor\,..\,|\dot{P}|-\lceil\Delta/2\rceil\,),\dot{T}}|_\Delta$;
- *the* leading *distance matrix* $\circ D^w_{\dot{P},\dot{T}}|_\Delta := D^w_{\dot{P}[\,0\,..\,|\dot{P}|-\lceil\Delta/2\rceil\,),\dot{T}}|_\Delta$;
- *the* trailing *distance matrix* $^\circ D^w_{\dot{P},\dot{T}}|_\Delta := D^w_{\dot{P}[\,\lfloor\Delta/2\rfloor\,..\,|\dot{P}|\,),\dot{T}}|_\Delta$.

For a threshold $k \in \mathbb{R}_{\geq 0}$*,*

- *an internal* (Δ, k)*-fern is a matrix* k*-equivalent to* $\overset{\circ}{\circ}D^w_{\dot{P},\dot{T}}|_\Delta$*; we use* $\overset{\circ}{\circ}\mathcal{F}^w_{\dot{P},\dot{T}}|^k_\Delta$ *for the set of all such ferns.*
- *a leading* (Δ, k)*-fern is a matrix* k*-equivalent to* $\circ D^w_{\dot{P},\dot{T}}|_\Delta$*; we use* $\circ\mathcal{F}^w_{\dot{P},\dot{T}}|^k_\Delta$ *for the set of all such ferns.*
- *a trailing* (Δ, k)*-fern is a matrix* k*-equivalent to* $^\circ D^w_{\dot{P},\dot{T}}|_\Delta$*; we use* $^\circ\mathcal{F}^w_{\dot{P},\dot{T}}|^k_\Delta$ *for the set of all such ferns.* ⌟

Remark 5.24. For our purposes, it suffices to define internal, leading, and trailing distance matrices only for $\Delta = \nabla$, allowing for a slightly simplified definition compared to Definitions 5.16 and 5.18. ⌟

We conclude with discussing the main property of ferns: they decompose under the (min, +)-product. We argue via the corresponding distance matrices.

Theorem 5.25. *Consider a sequence of puzzle pieces* $\mathcal{P} := (\dot{P}_0, \dot{T}_0), \ldots, (\dot{P}_{z-1}, \dot{T}_{z-1})$ *that forms a pair of* Δ*-puzzles of size* $z \geq 2$ *with values* $P := \dot{P}_0 \odot_\Delta \cdots \odot_\Delta \dot{P}_{z-1}$ *and* $T := \dot{T}_0 \odot_\Delta \cdots \odot_\Delta \dot{T}_{z-1}$*.*

For every integer $k \leq \lfloor\Delta/2\rfloor - \mathsf{tor}(\mathcal{P})$*, we have*

$$D^w_{P,T}|_\Delta \overset{k}{=} \circ D^w_{\dot{P}_0,\dot{T}_0}|_\Delta \oplus \Big(\bigoplus_{s=1}^{z-2} \overset{\circ}{\circ}D^w_{\dot{P}_s,\dot{T}_s}|_\Delta\Big) \oplus {}^\circ D^w_{\dot{P}_{z-1},\dot{T}_{z-1}}|_\Delta.$$

Proof. It is instructive to unfold the definitions of the leading, internal, and trailing distance matrices; for notational simplicity, we do so by defining a "cut" version of the sequence $(\dot{P}_i)$ that removes the overlaps of consecutive strings. Formally, we set

$$\begin{aligned} P_0 &:= \dot{P}_0[\,0\,..\,|\dot{P}_0| - \lceil\Delta/2\rceil\,), \\ P_s &:= \dot{P}_s[\,\lfloor\Delta/2\rfloor\,..\,|\dot{P}_s| - \lceil\Delta/2\rceil\,) \quad \text{for every } s \in (\,0\,..\,z-1\,), \text{ and} \\ P_{z-1} &:= \dot{P}_{z-1}[\,\lfloor\Delta/2\rfloor\,..\,|\dot{P}_{z-1}|\,). \end{aligned}$$

Now, we start by analyzing how the torsion of $\mathcal{P}$ evolves in the span of the sequence. To that end, define sequences $(p_s)_{s=0}^z$ with $p_s := |P_0| + \cdots + |P_{s-1}| = |P_0 \odot \cdots \odot P_{s-1}|$ and $(t_s)_{s=0}^z$ with $t_s := |\dot{T}_0| + \cdots + |\dot{T}_{s-1}| - s \cdot \Delta$, which equals $|\dot{T}_0 \odot_\Delta \cdots \odot_\Delta \dot{T}_{s-1}| - \Delta$ for $s \in (\,0\,..\,z\,]$.

Claim 5.26. *For each $s \in (0\,..\,z)$, we have*

$$k \le \min\{p_s - t_s, (|P| - p_s) - (|T| - t_s) + \Delta\}.$$

Proof. Observe that, indeed, we have

$$P = \dot{P}_0 \odot_\Delta \cdots \odot_\Delta \dot{P}_{z-1} = P_0 \odot \cdots \odot P_{z-1}.$$

In particular, for every $s \in (0\,..\,z)$, we have

$$p_s = |\dot{P}_0 \odot \cdots \odot \dot{P}_{s-1}| - \lceil \Delta/2 \rceil \quad \text{and} \quad |P| - p_s = |\dot{P}_s \odot \cdots \odot \dot{P}_{z-1}| - \lfloor \Delta/2 \rfloor.$$

Consequently, we obtain

$$\begin{aligned} p_s - t_s + \Delta &= |\dot{P}_0 \odot \cdots \odot \dot{P}_{s-1}| - |\dot{T}_0 \odot_\Delta \cdots \odot_\Delta \dot{T}_{s-1}| + \lfloor \Delta/2 \rfloor \\ &= \lfloor \Delta/2 \rfloor - \sum_{t=0}^{s-1} \left(|\dot{T}_t| - |\dot{P}_t| \right) \ge \lfloor \Delta/2 \rfloor - \mathsf{tor}(\mathcal{P}) \ge k \end{aligned}$$

and

$$\begin{aligned} (|P| - p_s) - (|T| - t_s) + \Delta &= |\dot{P}_s \odot \cdots \odot \dot{P}_{z-1}| - |\dot{T}_s \odot_\Delta \cdots \odot_\Delta \dot{T}_{z-1}| + \lceil \Delta/2 \rceil \\ &= \lceil \Delta/2 \rceil - \sum_{t=s}^{z-1} \left(|\dot{T}_t| - |\dot{P}_t| \right) \ge \lfloor \Delta/2 \rfloor - \mathsf{tor}(\mathcal{P}) \ge k; \end{aligned}$$

which completes the proof of the claim. ⌟

Next, consider the alignment graph $G \coloneqq \overline{\mathsf{AG}}^w(P, T)$ and observe that, for each $s \in [0\,..\,z)$, the subgraph induced by $[p_s\,..\,p_{s+1}] \times [t_s\,..\,t_{s+1} + \Delta]$ is isomorphic to $G_s \coloneqq \overline{\mathsf{AG}}^w(P_s, \dot{T}_s)$.

Claim 5.27. *For any sequence $(i_s)_{s=0}^z$ with $i_s \in [0\,..\,\Delta]$ for $s \in [0\,..\,z]$, we have $D^w_{P,T}|_\Delta[i_0, i_z] \le \sum_{s=0}^{z-1} D^w_{P_s,\dot{T}_s}|_\Delta[i_s, i_{s+1}]$.*

Proof. By triangle inequality, we have

$$\begin{aligned} D^w_{P,T}|_\Delta[i_0, i_z] &= \mathsf{dist}_G((0, i_0), (|P|, |T| - \Delta + i_z)) = \mathsf{dist}_G((p_0, t_0 + i_0), (p_z, t_z + i_z)) \\ &\le \sum_{s=0}^{z-1} \mathsf{dist}_G((p_s, t_s + i_s), (p_{s+1}, t_{s+1} + i_{s+1})). \end{aligned}$$

Moreover, the aforementioned embedding of G_s into G yields

$$\begin{aligned} &\mathsf{dist}_G((p_s, t_s + i_s), (p_{s+1}, t_{s+1} + i_{s+1})) \\ &\quad \le \mathsf{dist}_{G_s}((0, i_s), (p_{s+1} - p_s, t_{s+1} - t_s + i_{s+1})) = D^w_{P_s,\dot{T}_s}|_\Delta[i_s, i_{s+1}]. \end{aligned}$$

Combining the inequalities yields the claim. ⌟

As the sequence (i_s) is arbitrary, Claim 5.27 implies

$$D^w_{P,T}|_\Delta[i_0, i_z] \le \Big(\bigoplus_{s=0}^{z-1} D^w_{P_s,\dot{T}_s}|_\Delta \Big)[i_0, i_z];$$

that is, the natural statement that forcing a path to contain certain vertices may only increase its length. We conclude with a proof that we may indeed describe also a *shortest* path of length at most k in a similar manner.

Claim 5.28. *For any $i_0, i_z \in [0\,..\,\Delta]$, if $D^w_{P,T}|_\Delta[i_0, i_z] \le k$, then we also have*

$$\Big(\bigoplus_{s=0}^{z-1} D^w_{P_s,\dot{T}_s}|_\Delta \Big)[i_0, i_z] \le D^w_{P,T}|_\Delta[i_0, i_z].$$

Proof. Consider a shortest path $\mathcal{A}$ from $(0, i_0) = (p_0, t_0 + i_0)$ to $(|P|, |T| - \Delta + i_z) = (p_z, t_z + i_z)$ and assume that its length does not exceed k. For each $s \in (0\,..\,z)$, let $(p_s, \hat{t}_s) \in \mathcal{A}$ be an arbitrary vertex on $\mathcal{A}$ whose first coordinate is equal to p_s.

By Lemma 3.18, we have $\hat{t}_s \in [\,p_s - k\,..\,p_s + |T| - |P| + k\,]$. In particular, we have

$$\hat{t}_s \geq p_s - k \geq p_s - (p_s - t_s) = t_s \quad \text{and}$$
$$\hat{t}_s \leq p_s + |T| - |P| + k \leq p_s + |T| - |P| + (|P| - p_s) - (|T| - t_s) + \Delta = t_s + \Delta.$$

Hence, Claim 5.26 yields $\hat{t}_s = t_s + i_s$ for some $i_s \in [\,0\,..\,\Delta\,]$. By Lemma 3.16, we have

$$\begin{aligned}&\mathsf{dist}_G((p_s, t_s + i_s), (p_{s+1}, t_{s+1} + i_{s+1}))\\ &\quad = \mathsf{dist}_{G_s}((0, i_s), (p_{s+1} - p_s, t_{s+1} - t_s + i_{s+1}))\\ &\quad = D^w_{P_s, \dot{T}_s}\big|_\Delta[\,i_s, i_{s+1}\,].\end{aligned}$$

Thus, $D^w_{P,T}\big|_\Delta[\,i_0, i_z\,] \leq \sum_{s=0}^{z-1} D^w_{P_s,\dot{T}_s}\big|_\Delta[\,i_s, i_{s+1}\,]$ holds for some $i_1, \ldots, i_{z-1} \in [\,0\,..\,\Delta\,]$, which in turn implies the claim. ⌟

In total, we obtain $D^w_{P,T}\big|_\Delta \overset{k}{=} \bigoplus_{s=0}^{z-1} D^w_{P_s,\dot{T}_s}\big|_\Delta$ and thus the claim. ⌟

Remark 5.29. Since $\overset{k}{=}$ is a congruence relation, in the context of Theorem 5.25, a (Δ, k)-fern of (P, T) can be obtained as the (min, +)-product of a leading (Δ, k)-fern of $(\dot{P}_0, \dot{T}_0)$, internal (Δ, k)-ferns of $(\dot{P}_s, \dot{T}_s)$ for $s \in [\,1\,..\,z-1\,)$, and a trailing (Δ, k)-fern of $(\dot{P}_{z-1}, \dot{T}_{z-1})$. ⌟

Remark 5.30. We may intuitively understand the proof of Theorem 5.25 as follows: a shortest path of length at most k may use at most k horizontal or vertical edges of the alignment graph. Thus, all shortest path that we care about are contained in a diagonal band that allows for a "safety margin" of at least k to the left and to the right.

Now, when cutting the original fern into small ferns, we just have to ensure that the inputs and outputs of every small fern fully contains the safety margin—for this, we need the bound on the torsion of the sequence of puzzle pieces. ⌟

5.3 Algorithms for Multiplying Matrices

Definition 5.31. *We say that P is a* matrix property *if the following holds for every dimension $q \in \mathbb{Z}_{>0}$: There is a family $P_q \subseteq \mathbb{R}_{\geq 0}^{q \times q}$ of $q \times q$ matrices that* satisfy *P, and this family is compatible with the* (min, +)*-product, that is, for every two matrices $A, B \in P_q$, we have $A \oplus B \in P_q$.*

*We say that P admits $T^P_M(q)$-*time matrix multiplication *if there is an algorithm that, given two matrices $A, B \in P_q$, in time $T^P_M(q)$ computes $A \oplus B$.*

*We say that P admits $T^P_V(q)$-*time vector multiplication *if there is an algorithm that, given a matrix $A \in P_q$ and a vector $v \in \mathbb{R}_{\geq 0}^{q \times 1}$, in time $T^P_V(q)$ computes a vector $A \oplus v \in \mathbb{R}_{\geq 0}^{q \times 1}$.* ⌟

Remark 5.32. In this work, we typically study the matrix properties of being Monge or bounded difference; they are matrix properties due to [Tis15, Theorem 2] and Lemma 5.13. ⌟

Lemma 5.33. *The Monge property is a matrix property that admits matrix multiplication in $O(q^2)$ time and vector multiplication in $O(q)$ time.*

Proof. We use [AKM+87] (see Theorem 3.12); the (min, +)-product of Monge matrices is Monge again by [Tis15, Theorem 2]. ⌟

Remark 5.34. Observe that we may use [Kle05] (see Theorem 3.10) to compute Monge ferns for a pair of strings (see Lemma 6.14 which follows Corollary 3.19). Unfortunately, this is fast enough only in very specific scenarios. Hence, we later devise more sophisticated algorithms to efficiently construct Monge ferns.

Lemma 5.35. *For every threshold $\beta \in \mathbb{Z}_{>0}$, the β-bounded difference (integer) Monge matrices define a matrix property. Represented as core-based matrix oracles, they admit $O(\beta q \log q)$-time matrix multiplication and $O(q \log q)$-time vector multiplication.*

Proof. By Lemma 5.12, our input matrices have a core of $O(\beta q)$; hence, Lemma 5.9 allows us to multiply them in time $O(\beta q \log q)$. Finally, the resulting matrix is again β-bounded difference by Lemma 5.13 and Monge by [Tis15, Theorem 2].

For the vector multiplication, we proceed as for general weights; however, we afterward list the resulting vector explicitly using $O(q)$ queries to the core-based matrix oracle.

Finally, it is instructive to restate (a simplified) [GK25, Theorem 5.10] in our new terminology.

Lemma 5.36 [GK25, Theorem 5.10, simplified]**.** *There is a data structure that supports all of the following operations.*

Initialization. *Given non-empty strings $X, Y \in \Sigma^+$ of total length n and oracle access to an integer weight function $w : \bar{\Sigma}^2 \to [0\,..\,W]$, initialize the data structure in time $O(Wn^2 \log n)$.*

Fern retrieval. *Given integer Δ and k return a $(W+1)$-bounded difference Monge (Δ, k)-fern (represented as a core-based matrix oracle of size and core $O(W\Delta)$) in time $O(W\Delta \log n)$.*

Character edits. *Apply a single character edit to either X or Y in time $O(Wn \log n)$.*

Proof. Compared to [GK25], we output a fern instead of a Core-based matrix oracle of the whole distance matrix. To that end, we use Lemma 5.6 to extract the core corresponding to the fern (submatrix) of the distance matrix that is returned by the original data structure.

Now, in particular, we argue about the core of the output fern; the running time guarantees then follow from Definition 5.3. First, as a contiguous submatrix of a $(W+1)$-bounded difference Monge fern, the output fern has the same properties. Next, by Lemma 5.12, the fern thus has a core of at most $O(\Delta W)$; completing the proof.

6 Efficient Solutions for Growing Ferns, Verifying Occurrences, and Listing Fragments

As mentioned in the introduction, the Landau–Vishkin algorithm for pattern matching under unweighted edit distance [LV89] can be viewed as a repeated application of a subroutine that, given an $O(k)$-size interval I of positions in T, computes $I \cap \mathrm{Occ}_k(P, T)$. A solution for this problem, called Verify, that takes $O(k^2)$ time in the PILLAR model can be found in a work [CH02] of Cole and Hariharan, who used it as a subroutine in their $O(n + k^4 \cdot n/m)$ algorithm for PMwE. Unfortunately, under weighted edit distance, one cannot hope to obtain a solution with the same complexity: an algorithm for Verify under weighted edit distance taking $O(k^2)$ time in the PILLAR model would imply an $O(n + k^2)$-time algorithm for deciding whether the weighted edit distance of two strings of length at most n is at most k, thus refuting the lower bound of Cassis, Kociumaka, and Wellnitz [CKW23] for this problem—see Theorem 10.2. In this section, we show that Verify can be solved in $\widetilde{O}(k^3)$ time in the PILLAR model in the case of weighted edit distance. Further, if the weights are integers upper bounded by some threshold W, we show that Verify can be solved in $\widetilde{O}(\min(k^{2.5}, W \cdot k^2))$ time in the PILLAR model.

Our techniques also lead to an efficient algorithm for the problem of computing all fragments U of T with $\mathrm{ed}^w(P \to U) \le k$, as well as $\mathrm{ed}^w(P \to U)$. This problem is formally defined below.

Problem ListAllOccs(P, T, k, w).
Input. A text T of length n, a pattern P of length m, an integer threshold $k > 0$, and oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$.
Output. The set $\{(i, j, \mathsf{ed}^w(P \to T[i..j))) : 0 \le i \le j \le n \text{ and } \mathsf{ed}^w(P \to T[i..j)) \le k\}$.

On the way to our solutions for the Verify problem and the ListAllOccs problem, we derive an efficient procedure for constructing ferns of P and T (recall Definition 5.18), which plays a critical role in Section 9.

Problem GrowFern(P, T, k, w).
Input. A text T of length n, a pattern P of length m, an integer threshold $k > 0$, and oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$.
Output. A $(\min(n, k), k)$-fern of P and T with respect to w.

Instances of the GrowFern problem where the involved strings are of length $O(k)$ can be solved efficiently using an MSSP data structure Corollary 3.19.

Lemma 6.1. *Any instance of GrowFern with $|P| + |T| = O(k)$ can be solved in time $O(k^2 \log k)$.*

Proof. We first build the data structure of Corollary 3.19 for $\overline{\mathsf{AG}}^w(P, T)$ in $O(k^2 \log k)$ time. Then, we compute $\mathsf{ed}^w(P \to T[i..j))$ for all $(i, j) \in [0..\min(n, k)] \times [n - \min(n, k)..n]$ using said data structure in $O(k^2 \log k)$ time—we issue queries asking for the distance from $(0, i)$ to $(m, n - j)$ for every $(i, j) \in [0..\min(n, k)]^2$.

6.1 Growing Ferns: the Case of Small Edit and Self-edit Distance

Let us recall the definition of *self-edit distance*, a notion introduced in [CKW23]. We say that an alignment $\mathcal{A} : X \rightsquigarrow\!\!\!\!\!\rightarrow X$ is a *self-alignment* if $\mathcal{A}$ does not align any character $X[x]$ to itself. We define the *self-edit distance* of X as $\mathsf{self\text{-}ed}(X) := \min_{\mathcal{A}} \mathsf{ed}_{\mathcal{A}}(X, X)$, where the minimization ranges over self-alignments $\mathcal{A} : X \rightsquigarrow\!\!\!\!\!\rightarrow X$. In words, $\mathsf{self\text{-}ed}(X)$ is the minimum (unweighted) cost of a self-alignment. We can interpret a self-alignment as a $(0, 0) \rightsquigarrow (|X|, |X|)$ path in the alignment graph $\mathsf{AG}(X, X)$ that does not contain any edge of the main diagonal. Self-edit distance enjoys the following useful properties.

Lemma 6.2 (Properties of self-ed) [CKW23, Lemma 4.2] and [GK25, Lemma 6.2]**.** *Let $X \in \Sigma^*$ denote a string. Then, all of the following hold:*
Monotonicity. *For any $\ell' \le \ell \le r \le r' \in [0..|X|]$, we have*

$$\mathsf{self\text{-}ed}(X[\ell..r)) \le \mathsf{self\text{-}ed}(X[\ell'..r')).$$

Sub-additivity. *For any $\mu \in [0..|X|]$, we have*

$$\mathsf{self\text{-}ed}(X) \le \mathsf{self\text{-}ed}(X[0..\mu)) + \mathsf{self\text{-}ed}(X[\mu..|X|)).$$

Triangle inequality. *For any $Y \in \Sigma^*$, we have $\mathsf{self\text{-}ed}(Y) \le \mathsf{self\text{-}ed}(X) + 2\mathsf{ed}(X, Y)$.*
Continuity. *For any $i \in (0..|X|)$, we have $\mathsf{self\text{-}ed}(X[0..i]) - \mathsf{self\text{-}ed}(X[0..i)) \in \{0, 1\}$.*

The following lemma gives a structural characterization of a pair of strings that are at small edit distance and have small self-edit distance.

Lemma 6.3 [GK25, Lemma 7.1]; see also [CKW23, Lemma 4.6 and Claim 4.11]**.** *There is a* PILLAR *algorithm that, given strings X, Y and a positive integer k satisfying $\mathsf{self\text{-}ed}(X) \le k \le |X|$ and $\mathsf{ed}(X, Y) \le k$, in $O(k^2)$ time builds a decomposition $X = \bigodot_{i=0}^{z-1} X_i$ and a sequence of fragments $(Y_i)_{i=0}^{z-1}$ of Y that satisfies all of the following.*

- *Each phrase* $X_i = X[\,x_i\,..\,x_{i+1}\,)$ *is of length* $x_{i+1} - x_i \in [\,k\,..\,2k\,)$ *and each fragment* $Y_i = Y[\,y_i\,..\,y'_{i+1}\,)$ *satisfies* $y_i = \max(0, x_i - k)$ *and* $y'_{i+1} = \min(x_{i+1} + 3k, |Y|)$.
- *There is a set* $F \subseteq [\,0\,..\,z\,)$ *of size* $|F| = O(k)$ *such that* $X[\,x_i\,..\,x_{i+1}\,) = X[\,x_{i-1}\,..\,x_i\,)$ *and* $Y[\,y_i\,..\,y'_{i+1}\,) = Y[\,y_{i-1}\,..\,y'_i\,)$ *hold for each* $i \in [\,0\,..\,z\,) \setminus F$ *(in particular,* $0 \in F$*).*

The algorithm returns the set F *and, for each* $i \in F$*, the endpoints of* $X_i = X[\,x_i\,..\,x_{i+1}\,)$.[14] ◢

In the case when the weights are small integers, we utilize the following lemma from [GK25] that allows us to efficiently compute, for each i, a representation of $D^w_{X_i,Y_i}$.[15]

Lemma 6.4 [GK25, see Lemma 7.2]**.** *Consider a weight function* $w : \bar{\Sigma}^2 \to [\,0\,..\,W\,]$*, a positive integer* k*, and two strings* $X, Y \in \Sigma^{\le n}$ *such that* $\mathsf{self\text{-}ed}(X) \le k \le |X|$ *and* $\mathsf{ed}(X, Y) \le k$*. Moreover, suppose that* $X = \bigodot_{i=0}^{m-1} X_i$*,* $(Y_i)_{i=0}^{m-1}$*, and* $F \subseteq [\,0\,..\,m\,)$ *satisfy the conditions in the statement of Lemma 6.3.*

There is an algorithm that, given oracle access to w*, the integer* k*, the strings* X, Y*, the set* F*, and the endpoints of* X_i *for each* $i \in F$*, takes* $O(W \cdot k^2 \log^2 n)$ *time plus* $O(k^2)$ `PILLAR` *operations and returns a reference to* $\mathsf{CMO}(D^w_{X_i,Y_i})$ *for each* $i \in F$*.*

A variant of this algorithm that takes $O(k^{2.5} \log^2 n)$ *time plus* $O(k^2)$ `PILLAR` *operations returns, for each* $i \in F$*, a reference to* $\mathsf{CMO}(M_i)$*, where* M_i *is a Monge matrix* M_i *such that* $M_i \overset{k}{=} D^w_{X_i,Y_i}$ *and* $\delta(M_i) = O(k^{1.5})$. ◢

In the next lemma, we show an efficient solution for ListAllOccs in the case when $\mathsf{self\text{-}ed}(P) \le k$ and $\mathsf{ed}(P, T) \le k$. The proof of said lemma closely follows the proof of [GK25, Lemma 7.3].

Lemma 6.5 (`GrowFern-SmallSED(`P`,` T`,` k`,` w`)`) [GK25, adapted from Lemma 7.3]**.** *Any instance of the* GrowFern *problem where* $\mathsf{self\text{-}ed}(P) \le k$ *and* $\mathsf{ed}(P, T) \le k$ *can be solved in*

1. $O(k^3 \log(nk))$ *time in the* `PILLAR` *model, with the returned fern being Monge;*
2. $O(W \cdot k^2 \log^2(nk))$ *time in the* `PILLAR` *model if all the weights are integers, with the returned fern being Monge and* $(W+1)$*-bounded-difference;*
3. $O(k^{2.5} \log^2(nk))$ *time in the* `PILLAR` *model if all the weights are integers, with the returned fern* Φ *being Monge and satisfying* $\delta(\Phi) = O(k^{1.5})$.

Proof. First, observe that $|n - m| \le \mathsf{ed}(P, T) \le k$.

If $m < 20k$, we have $n < 19k$ and it hence suffices to use Lemma 6.1.

We henceforth assume that $m \ge 20k$, which implies that $n \ge 19k$. We first run the algorithm of Lemma 6.3 for P, T, and k, arriving at a decomposition $P = \bigodot_{i=0}^{z-1} P_i$ and a sequence of fragments $(T_i)_{i=0}^{z-1}$, such that $P_i = P[\,p_i\,..\,p_{i+1}\,)$ and $T_i = T[\,t_i\,..\,t'_{i+1}\,)$ for $t_i = \max\{p_i - k, 0\}$ and $t'_{i+1} = \min\{p_{i+1} + 3k, n\}$ for all $i \in [\,0\,..\,z\,)$. The decomposition is represented using a set F of size $O(k)$ such that $P_i = P_{i-1}$ and $T_i = T_{i-1}$ holds for each $i \in [\,0\,..\,z\,) \setminus F$ and the endpoints of P_i for $i \in F$. To easily handle corner cases, we set $t'_0 := t'_1$ and $t_z := t_{z-1}$ (so that the sequences $(t_i)_{i=0}^{z}$ and $(t'_i)_{i=0}^{z}$ remain monotone), and we assume that $[\,0\,..\,z\,) \setminus F \subseteq [\,2\,..\,z-5\,]$; if the original set F does not satisfy the latter condition, we insert to it the missing $O(1)$ elements.

For each $i \in [\,0\,..\,z\,)$, consider a subgraph G_i of $\overline{\mathsf{AG}}^w(P, T)$ induced by $[\,p_i\,..\,p_{i+1}\,] \times [\,t_i\,..\,t'_{i+1}\,]$. Denote by G the union of all subgraphs G_i. For each $i \in [\,0\,..\,z\,]$, denote $V_i := \{p_i\} \times [\,t_i\,..\,t'_i\,]$. Note that $V_0 \supseteq \{0\} \times [\,0\,..\,2k\,]$. Additionally, $V_z \supseteq \{m\} \times [\,n-k\,..\,n\,]$ since $p_z - p_{z-1} \ge k$ implies $p_{z-1} \le m - k$, $t_{z-1} = p_{z-1} - k \le m - 2k$, and $n \ge m - k$. Further, $V_i = V(G_i) \cap V(G_{i-1})$ for $i \in (\,0\,..\,z\,)$. Moreover, for integers $0 \le i \le j \le z$, let $D_{i,j}$ denote the matrix of pairwise distances from V_i to V_j in G, where rows of $D_{i,j}$ represent vertices of V_i in the increasing order of the second coordinate, and columns of $D_{i,j}$ represent vertices of V_j in the increasing order of the second coordinate. Note that $D_{i,j}$ is a Monge matrix by Fact 3.11.

---

[14] This determines the whole decomposition because $(x_i)_{i=\ell}^{r}$ is an arithmetic progression for every $(\,\ell\,..\,r\,] \subseteq (\,0\,..\,z\,] \setminus F$.

[15] [GK25, Lemma 7.2] returns an intricate data structure that is outlined in [GK25, Theorem 5.10]. In Lemma 6.4, instead of describing this data structure in detail, we specify as output only the part of said data structure that is useful to us.

Claim 6.6. (1) *We can compute $D_{i,i+1}$ for all $i \in F$ in $\mathcal{O}(k^3 \log k)$ time in total in the* PILLAR *model.*
(2) *If all the weights are integers, we can compute $\mathsf{CMO}(D_{i,i+1})$, for all $i \in F$, in $\mathcal{O}(W \cdot k^2 \log^2 n)$ time in the* PILLAR *model;*
(3) *If all the weights are integers, we can compute $\mathsf{CMO}(Z_{i,i+1})$, for all $i \in F$, where $Z_{i,i+1}$ is a Monge matrix such that $Z_{i,i+1} \overset{k}{=} D_{i,i+1}$ and $\delta(Z_{i,i+1}) = \mathcal{O}(k^{1.5})$, in $\mathcal{O}(k^{2.5} \log^2 n)$ time in the* PILLAR *model.*

Proof. By Lemma 3.16 (path monotonicity), the shortest paths (in G) between vertices of G_i stay within G_i. Hence, $D_{i,i+1}$ can be computed in $\mathcal{O}(k^2 \log k)$ time using an MSSP data structure (Theorem 3.10) for each $i \in F$, for a total time of $\mathcal{O}(k^3 \log k)$.

In the case of integer weights, we can instead use either of the following approaches.

- We can use Lemma 6.4 to construct $\mathsf{CMO}(D^w_{P_i,T_i})$ for all $i \in F$ using $\mathcal{O}(W \cdot k^2 \log^2 n)$ total time in the PILLAR model. Then, for each $i \in F$, we can extract $\mathsf{CMO}(D_{i,i+1})$ in $\mathcal{O}(W \cdot k \log k)$, time using Lemma 5.6, for a total of $\mathcal{O}(W \cdot k^2 \log k)$ time, noting that the core of each considered matrix is of size $\mathcal{O}(W \cdot k)$ due to Fact 5.15.
- We can use Lemma 6.4 to construct, for each $i \in F$, $\mathsf{CMO}(M_i)$ for a Monge matrix M_i such that $M_i \overset{k}{=} D^w_{P_i,T_i}$ and $\delta(M_i) = \mathcal{O}(k^{1.5})$ in $\mathcal{O}(k^{2.5} \log^2 n)$ total time in the PILLAR model. Then, for each $i \in F$, we can extract $\mathsf{CMO}(Z_{i,i+1})$ in $\mathcal{O}(k^{1.5} \log k)$, for a total of $\mathcal{O}(k^{2.5} \log k)$ time, using Lemma 5.6. ⌟

As shown in the proof of [GK25, Claim 7.4], our definition of F ensures that $D_{i,i+1} = D_{i-1,i}$ for each $i \in [0\,..\,z) \setminus F$. We rely on the following fact to prove Claim 6.8.

Claim 6.7 [GK25, Claim 7.4]. *Let $0 = j_0 < j_1 < \ldots < j_{|F|-1} < j_{|F|} = z$ be the elements of $F \cup \{z\}$. We have*

$$D_{0,z} = \bigoplus_{i=0}^{|F|-1} D^{\oplus(j_{i+1}-j_i)}_{j_i,j_i+1}.$$ ⌟

Claim 6.8. (1) *We can compute $D_{0,z}$ in $\mathcal{O}(k^3 \log n)$ time in the* PILLAR *model.*
(2) *If all weights are integers, $D_{0,z}$ is $(W+1)$-bounded difference and we can compute it in time $\mathcal{O}(W \cdot k^2 \log^2 n)$ time in the* PILLAR *model.*
(3) *If all weights are integers, in $\mathcal{O}(k^{2.5} \log^2 n)$ time in the* PILLAR *model, we can compute a Monge matrix $Z_{0,z}$ such that $Z_{0,z} \overset{k}{=} D_{0,z}$ and $\delta(Z_{0,z}) = \mathcal{O}(k^{1.5})$.*

Proof. In the case of arbitrary weights, we first employ Claim 6.6(1) to compute, for each $i \in [0\,..\,|F|)$, the matrix D_{j_i,j_i+1}, in $\mathcal{O}(k^3 \log k)$ time in the PILLAR model in total. Given $D_{a,b}$ and $D_{b,c}$ for any integers $0 \le a \le b \le c \le z$, we can calculate $D_{a,c}$ using Theorem 3.12 in time $\mathcal{O}(k^2)$. Therefore, using binary exponentiation, we can calculate M_i where $M_i := D^{\oplus(j_{i+1}-j_i)}_{j_i,j_i+1}$ in time $\mathcal{O}(k^2 \log n)$. Doing this for all $i \in [0\,..\,|F|)$ requires $\mathcal{O}(k^3 \log n)$ time. We then calculate $D_{0,z}$ using the fact that $D_{0,z} = M_0 \oplus M_1 \oplus \cdots \oplus M_{|F|-1}$ in time $\mathcal{O}(|F| \cdot k^2) = \mathcal{O}(k^3)$.

In the case of integer weights, we present two algorithms.

- *An $\mathcal{O}(W \cdot k^2 \log^2 n)$-*PILLAR*-time algorithm.* We employ Claim 6.6(2) to compute $\mathsf{CMO}(D_{j_i,j_i+1})$ for all $i \in [0\,..\,|F|)$, in $\mathcal{O}(W \cdot k^2 \log^2 n)$ total time in the PILLAR model. We then use Lemma 5.5 instead of Theorem 3.12 and binary exponentiation to compute $\mathsf{CMO}(D_{0,z})$. This requires $\mathcal{O}(W \cdot k^2 \log^2 n)$ time in total as each matrix D_{j_i,j_i+1} is $(W+1)$-bounded-difference due to Fact 5.15, each computed matrix is $(W+1)$-bounded-difference due to Lemma 5.13, and thus the core of each considered matrix is $\mathcal{O}(W \cdot k)$ due to Lemma 5.12. In the end, we extract the $(W+1)$-bounded-difference matrix $D_{0,z}$ from $\mathsf{CMO}(D_{0,z})$ in $\mathcal{O}(k^2 \log k)$ time (see Definition 5.3).

▭ *An $O(k^{2.5}\log^2 n)$-*PILLAR*-time algorithm.* We employ Claim 6.6(3) to compute, for each $i \in [0..|F|)$, $\mathsf{CMO}(Z_{j_i,j_i+1})$ for a Monge matrix Z_{j_i,j_i+1} that is k-equivalent to D_{j_i,j_i+1} and its core has size $O(k^{1.5})$ in total time $O(k^{2.5}\log^2 n)$ in the PILLAR model. We then use Lemma 5.9 instead of Theorem 3.12 and binary exponentiation to compute a matrix $Z_{0,z}$ that is k-equivalent to $D_{0,z}$. Lemma 5.9 guarantees that the size of the core of each computed matrix is $O(k^{1.5})$ and hence the total running time for all $O(k\log n)$ min-plus multiplications is $O(k^{2.5}\log^2 n)$. In the end, we again extract the matrix from its CMO representation in $O(k^2\log k)$ time, noting that its core is of size $O(k^{1.5})$. ◿

◸ Claim 6.9 (See also Claim 4.7). *For every fragment $T[\ell..r)$ of T and threshold $d \in [0,k]$, the following are equivalent:*

(a) $\mathsf{ed}^w(P \to T[\ell..r)) \le d$;

(b) $\ell \le t'_{0'}$, $r \ge t_z$, *and* $\mathsf{dist}_G((0,\ell),(m,r)) \le d$.

Proof. The (a) $\Leftarrow$ (b) implication follows directly from Lemma 3.16 (distance preservation between $\mathsf{AG}^w(P,T)$ and $\overline{\mathsf{AG}}^w(P,T)$) and the fact that G is a subgraph of $\overline{\mathsf{AG}}^w(P,T)$.

For the converse implication, suppose that there is an alignment $\mathcal{O} : P \rightsquigarrow T[\ell..r)$ of weighted cost at most $d \le k$. It suffices to prove that $\mathcal{O}$ (interpreted as a path from $(0,\ell)$ to (m,r) in $\overline{\mathsf{AG}}^w(P,T)$) is contained in G. For this, let us focus on a single edge $(p,t) \to (p',t')$ of $\mathcal{O}$ with $p' \in \{p,p+1\}$ and $t' \in \{t,t+1\}$. Let us fix an index $i \in [0..z)$ such that $[p..p'] \subseteq [p_i..p_{i+1}]$. Note that $(t'-p')-(r-m) \le k$ implies $t' \le p'+k+r-m \le p'+k+n-m \le p'+2k \le p_{i+1}+2k$. Since $t' \le n$, we conclude that $t' \le t'_{i+1}$. Further, note that $(p-t)-(0-\ell) \le k$ implies $t \ge p-k-\ell \ge p-k \ge p_i-k$. Since $t \ge 0$, we conclude that $t \ge t_i$. Thus, since $t,t' \in [t_i..t'_{i+1}]$, the edge $(p,t) \to (p',t')$ is contained in G_i and hence in G. ◿

The above claim guarantees that the matrix returned by each of the algorithms encapsulated in Claim 6.8 is an $O(k)\times O(k)$ Monge matrix whose top-right contiguous $(k+1)\times(k+1)$ submatrix Φ is in $\mathcal{F}^w_{P,T}|^k_k$ due to the fact that $V_0 \supseteq \{0\}\times[0..k]$ and $V_z \supseteq \{m\}\times[n-k..n]$. We extract Φ in $O(k^2\log k)$ time (see Definition 5.3) and return it. In the case when $m \ge 20k$, the time complexity of the algorithm is dominated by the running time of the procedure encapsulated in Claim 6.8. This concludes the proof of the lemma. ◢

6.2 Growing Ferns: The General Case

Before proving the main lemma of this section, let us state some useful lemmas from [CKW23, GK25].

◤ **Lemma 6.10** [CKW23, Lemma 4.5]. *There is an $O(k^2)$-time* PILLAR *algorithm that, given a string $X \in \Sigma^n$ and an integer $k \in \mathbb{Z}_{\ge 0}$ determines whether $\mathsf{self\text{-}ed}(X) \le k$ and, if so, retrieves (the breakpoint representation of) an optimal self-alignment $\mathcal{A} : X \rightsquigarrow X$.* ◢

◤ **Lemma 6.11** [CKW23, part of Lemma 4.9]. *There is an algorithm that, given two strings $X,Y \in \Sigma^{\le n}$ and integers $1 \le d \le k \le n$ such that $\mathsf{self\text{-}ed}(X) \le k$ and $\big||X|-|Y|\big| \le 2d$, as well as (oracle access to) a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$, computes $\mathsf{ed}^w_{\le d}(X \to Y)$ as well as the breakpoint representation of the underlying alignment (if this value is not ∞) in $O(k^2 d\log n)$ time in the* PILLAR *model.* ◢

◤ **Lemma 6.12** [CKW23, Lemma 4.4]. *For two strings $X,Y \in \Sigma^*$ and positions $i \le j, i' \le j' \in [0..|Y|]$, write $\mathcal{A} \in \mathbf{A}(X, Y[i..j))$ and $\mathcal{A}' \in \mathbf{A}(X, Y[i'..j'))$. If the alignments $\mathcal{A}$ and $\mathcal{A}'$, interpreted as paths in $\mathsf{AG}(X,Y)$, do not contain any common diagonal edge, then*

$$\mathsf{self\text{-}ed}(X) \le |i-i'| + \mathsf{ed}_{\mathcal{A}}(X, Y[i..j)) + \mathsf{ed}_{\mathcal{A}'}(X, Y[i'..j')) + |j-j'|.$$ ◢

◤ **Lemma 6.13** [GK25, Lemma 7.6]. *There is a* PILLAR *algorithm that, given two strings $X,Y \in \Sigma^{\le n}$ as well as oracle access to a weight function $w : \bar{\Sigma}^2 \to [0..W]$, finds $\mathsf{ed}^w(X \to Y)$ as well as the breakpoint representation of a w-optimal alignment $\mathcal{A} : X \rightsquigarrow Y$. The running time of the algorithm is $O(\min\{k^2 \cdot W\log^2 n, k^{2.5}\log^3 n\})$ time plus $O(k^2 \log\min\{n, W+1\})$* PILLAR *operations, where $k = \mathsf{ed}^w(X \to Y)$.* ◢

Lemma 6.14 (`GrowFern(`P`, `T`, `k`, `w`)`). *Any instance of the* GrowFern *problem can be solved in*

1. $O(k^3 \log(nk) \log n)$ *time in the* PILLAR *model with the returned fern being Monge;*
2. $O(W \cdot k^2 \log^2(nk))$ *time in the* PILLAR *model if all weights are integers, with the returned fern being Monge and* $(W+1)$*-bounded-difference;*
3. $O(k^{2.5} \log^3(nk))$ *time in the* PILLAR *model if all weights are integers, with the returned fern* Φ *being Monge and satisfying* $\delta(\Phi) = O(k^{1.5})$.

Proof. We first compute $\mathsf{ed}(P,T)$ in $O(k^2)$ time in the PILLAR model using Lemma 3.20.

If $\mathsf{ed}(P,T) > 3k$, then we can output a trivial fern all of whose entries are set to $k+1$. This follows from the fact that if there exist $i \le k$ and $j \ge n-k$ such that the $(0,i)$-to-(m,j) shortest path in $\overline{\mathsf{AG}}^w(P,T)$ has weight at most k, then $\mathsf{ed}(P,T) \le |T[\,0\,..\,i\,)| + \mathsf{ed}^w(P \to T[\,i\,..\,j\,)) + |T[\,j\,..\,n\,)| \le 3k$.

If $\mathsf{ed}(P,T) \in (\,k\,..\,3k\,]$, we simply replace k with $\mathsf{ed}(P,T)$, thus reducing to the case when $k \ge \mathsf{ed}(P,T)$. In the end, we simply trim the returned fern in $O(k^2)$ time according to Lemma 5.20.

We can thus henceforth assume that $\mathsf{ed}(P,T) \le k$ and distinguish between two cases depending on whether $\mathsf{self\text{-}ed}(P) \le 57k$; this can be determined in $O(k^2)$ time in the PILLAR model using Lemma 6.10.

Case I: $\mathsf{self\text{-}ed}(P) \le 57k$. We call `GrowFern-SmallSED(`$P$`, `$T$`, `$57k$`, `$w$`)` from Lemma 6.5, which, within the stated bounds, returns a Monge fern $\Phi \in \mathcal{F}^w_{P,T}|^{57k}_{\min(n,57k)}$. It suffices to trim this fern in $O(k^2)$ time according to Lemma 5.20 thus obtaining a Monge $(\min(n,k),k)$-fern of P and T with respect to w whose difference bounds and core size are inherited from the corresponding constraints on Φ.

Case II: $\mathsf{self\text{-}ed}(P) > 57k$. The monotonicity and continuity properties of self-edit distance (Lemma 6.2) imply that, using $O(\log m)$ invocations of Lemma 6.10 in a binary search fashion, we can compute the following fragments in $O(k^2 \log m)$ time in the PILLAR model:

- a prefix P_p of P with self-edit distance $28k$;
- a prefix P'_p of P with self-edit distance $14k$;
- a suffix P_s of P with self-edit distance $28k$;
- a suffix P'_s of P with self-edit distance $14k$.

Due to the monotonicity, sub-additivity, and triangle inequality properties of self-edit distance (Lemma 6.2), the fragments P_p and P_s do not overlap, that is, we have $P = P_p P_\mu P_s$ for some string P_μ of self-edit distance at least $57k - 28k - 28k = k$.

We then intend to compute $\min_{0 \le u \le v \le |P_p|+n-m+k} \mathsf{ed}^w_{\le k}(P_p \to T[\,u\,..\,v\,))$ along with a witness fragment $T[\,p_1\,..\,p_3\,)$. Note that $|P_p|+n-m+k \le |P_p|+2k$ and hence, if this minimum value is at most k, we have $p_1 \in [\,0\,..\,3k\,]$ and $p_3 \in [\,|P_p|+n-m-2k\,..\,|P_p|+n-m+k\,)$. It thus suffices to call `GrowFern-SmallSED(`P_p`, `$T[\,0\,..\,|P_p|+n-m+k\,)$`, `$28k$`, `$w$`)` using Lemma 6.5 and iterate over its entries. If the computed value is ∞, we output a trivial fern all of whose entries are set to $k+1$. Otherwise, that is, if the computed value is not ∞, we then compute the breakpoint representation of a witness alignment $\mathcal{A}_p$, and we set $T_p := T[\,p_1\,..\,p_3\,) = \mathcal{A}_p(P_p)$. To see that this is correct, suppose that there exists an alignment $\mathcal{A} : P \rightsquigarrow T[\,\ell\,..\,r\,)$ of weighted cost at most k. Let $T[\,\ell\,..\,v\,) := \mathcal{A}(P_p)$. By Lemma 3.18, we have $v \le |P_p|+n-m+k$, so we have $\mathsf{ed}^w(P_p \to T[\,\ell\,..\,v\,)) \le k$ for $0 \le i \le v \le |P_p|+n-m+k$. It remains to explain how we compute the breakpoint representation of $\mathcal{A}_p$. In the case of general weights, we do so using Lemma 6.11 in $O(k^3 \log m)$ time in the PILLAR model. In the case of integer weights, we employ Lemma 6.13 which requires $O(\min\{k^2 \cdot W \log^2 n, k^{2.5} \log^3 n\})$ time plus $O(k^2 \log \min\{n, W+1\})$ PILLAR operations.

Symmetrically, we compute $\min_{m-k-|P_s| \le u \le v \le n} \mathsf{ed}^w_{\le k}(P_s \to T[\,u\,..\,v\,))$. As above, if this value is ∞, we output a trivial fern all of whose entries are set to $k+1$. Otherwise, that is, if this value is not ∞, we also obtain the breakpoint representation of a witness alignment $\mathcal{A}_s$, and we set $T_s := T[\,s_1\,..\,s_3\,) := \mathcal{A}_s(P_s)$. The time complexity and the correctness of this procedure follow by arguments symmetric to the one made in the previous paragraph.

We henceforth assume that we have alignments $\mathcal{A}_p$ and $\mathcal{A}_s$ at hand and set $T'_p \coloneqq T[\,p_1\,..\,p_2\,) \coloneqq \mathcal{A}_p(P'_p)$ and $T'_s \coloneqq T[\,s_1\,..\,s_2\,) \coloneqq \mathcal{A}_s(P'_s)$.

Claim 6.15. *For any two positions $\beta, \gamma \in [\,0\,..\,n\,]$ such that $\mathsf{ed}^w(P \to T[\,\beta\,..\,\gamma\,)) \le k$,*

$$\begin{aligned}\mathsf{ed}^w(P \to T[\,\beta\,..\,\gamma\,)) = \mathsf{ed}^w(P[\,0\,..\,|P'_p|\,) \to T[\,\beta\,..\,p_2\,)) \\ + \mathsf{ed}^w(P[\,|P'_p|\,..\,m-|P'_s|\,) \to T[\,p_2\,..\,s_2\,)) \\ + \mathsf{ed}^w(P[\,m-|P'_s|\,..\,m\,) \to T[\,s_2\,..\,\gamma\,)).\end{aligned}$$

Proof. We say that a vertex (r_1, c_1) dominates a vertex (r_2, c_2) if $r_1 \ge r_2$ and $c_1 \ge c_2$.

Let us consider an optimal alignment $\mathcal{A} : P \rightsquigarrow T[\,\beta\,..\,\gamma\,)$. Our goal is to show that, when we interpret alignments $\mathcal{A}$, $\mathcal{A}_p$, and $\mathcal{A}_s$ as paths in $\mathsf{AG}^w(P, T)$, each of the following holds.

1. $\mathcal{A}$ and $\mathcal{A}_p$ share a vertex that is dominated by $(|P'_p|, p_2)$.
2. $\mathcal{A}$ and $\mathcal{A}_p$ share a vertex that dominates $(|P'_p|, p_2)$ and is dominated by $(|P_p|, p_3)$.
3. $\mathcal{A}$ and $\mathcal{A}_s$ share a vertex that dominates $(m - |P_s|, s_1)$ and is dominated by $(m - |P'_s|, s_2)$.
4. $\mathcal{A}$ and $\mathcal{A}_s$ share a vertex that dominates $(m - |P'_s|, s_2)$.

The setting is illustrated in Figure 8. We prove the existence of the two vertices specified in Items 1 and 2—the existence of the remaining ones follows by symmetry.

Proof of Item 1: Toward a contradiction, suppose that $\mathcal{A}$ and $\mathcal{A}_p$ do not share any vertex dominated by $(|P'_p|, p_2)$. Then, in particular, $\mathcal{A}$ and $\mathcal{A}_p$ do not share any diagonal edge with an endpoint dominated by $(|P'_p|, p_2)$. The condition of Lemma 6.12 is thus satisfied and we have that

$$\mathsf{self\text{-}ed}(P'_p) \le |\beta - p_1| + \mathsf{ed}_{\mathcal{A}}(P'_p, \mathcal{A}(P'_p)) + \mathsf{ed}_{\mathcal{A}_p}(P'_p, T'_p) + |\beta + |\mathcal{A}(P'_p)| - p_2| \le 3k + k + k + 5k = 10k,$$

since $p_1, \beta \in [\,0\,..\,3k\,]$ and the lengths of $\mathcal{A}(P'_p)$ and T'_p differ by at most k. This implies that $\mathsf{self\text{-}ed}(P'_p) \le 10k < 14k$, which contradicts the definition of P'_p.

Proof of Item 2: Toward a contradiction, suppose that $\mathcal{A}$ and $\mathcal{A}_p$ do not share any vertex that dominates $(|P'_p|, p_2)$ and is dominated by $(|P_p|, p_3)$. Then, in particular, these two alignments do not share any diagonal edge with such an endpoint, and we can again use Lemma 6.12. Let us denote $P[\,|P'_p|\,..\,|P_p|\,)$ by P''_p. We have

$$\begin{aligned}\mathsf{self\text{-}ed}(P''_p) \le |\beta + |\mathcal{A}(P'_p)| - p_2| + \mathsf{ed}_{\mathcal{A}}(P''_p, \mathcal{A}(P''_p)) + \mathsf{ed}_{\mathcal{A}_p}(P''_p, \mathcal{A}_p(P''_p)) + |\beta + |\mathcal{A}(P_p)| - p_3| \\ \le 5k + k + k + 5k = 12k,\end{aligned}$$

similarly to before. By the sub-additivity and triangle inequality properties of self-edit distance from Lemma 6.2, we then have that $\mathsf{self\text{-}ed}(P_p) \le \mathsf{self\text{-}ed}(P'_p) + \mathsf{self\text{-}ed}(P''_p) \le 14k + 12k < 28k$, which contradicts the definition of P_p.

We are now ready to conclude the proof of the claim. We tweak alignment $\mathcal{A}$ by making it behave like alignment $\mathcal{A}_p$ between a pair of vertices u_1 and u_2 that satisfy the conditions specified in Item 1 and Item 2, respectively, and like alignment $\mathcal{A}_s$ between a pair of vertices u_3 and u_4 that satisfy the conditions specified in Item 3 and Item 4. Since global alignments are also locally optimal, the cost of the obtained alignment $\mathcal{A}'$ remains unchanged. As $\mathcal{A}'$ is an optimal alignment and contains $(|P'_p|, p_2)$ and $(m - |P'_s|, s_2)$, the statement of the claim follows. ⌟

We are now ready to compute the output. We first call `GrowFern-SmallSED`$(P'_p,\ T[\,0\,..\,p_2\,),\ 14k,\ w)$ using the appropriate variant of Lemma 6.5, noting that its conditions are satisfied: $\mathsf{self\text{-}ed}(P'_p) = 14k$ and $\mathsf{ed}(P'_p, T[\,0\,..\,p_2\,)) \le p_1 + k \le 4k$. Let us denote the returned fern by Φ_p. We also call `GrowFern-SmallSED`$(P'_s,\ T[\,s_2\,..\,n\,),\ 14k,\ w)$, obtaining a fern Φ_s. We also compute $\psi \coloneqq \mathsf{ed}^w(P[\,|P'_p|\,..\,|P|-|P'_s|\,) \to T[\,p_2\,..\,s_2\,))$ in $O(k^3 \log^2(nk))$ time in the `PILLAR` model using Lemma 3.21 in the case of general weights and in $O(\min\{k^2 \cdot W \log^2 n, k^{2.5} \log^3 n\})$ time plus $O(k^2 \log \min\{n, W+1\})$

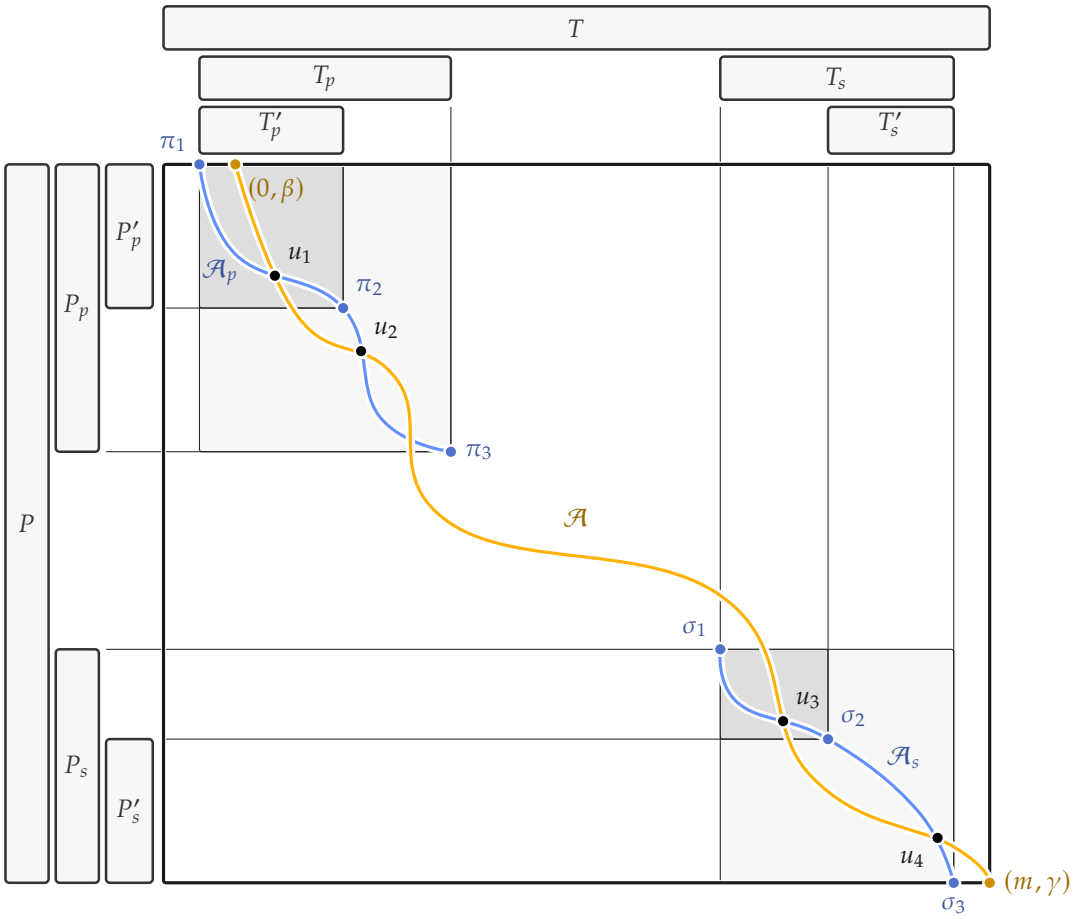


Figure 8. An illustration of the setting in the proof of Claim 6.15 with $\pi_1 = (0, p_1)$, $\pi_2 = (|P'_p|, p_2)$, $\pi_3 = (|P_p|, p_3)$, $\sigma_1 = (m - |P_s|, s_1)$, $\sigma_2 = (m - |P'_s|, s_2)$, and $\sigma_3 = (m, s_3)$. Vertices u_1, u_2, u_3, and u_4 satisfy the requirements of Item 1, Item 2, Item 3, and Item 4, respectively; observe that there are vertices that satisfy the requirements of Item 2. An alignment that contains $(|P'_p|, p_2)$ and $(m - |P'_s|, s_2)$ and hence witnesses the equality at the statement of the lemma can be obtained as follows: follow $\mathcal{A}$ from $(0, \beta)$ to u_1, $\mathcal{A}_p$ from u_1 to u_2, $\mathcal{A}$ from u_2 to u_3, $\mathcal{A}_s$ from u_3 to u_4, and $\mathcal{A}$ from u_4 to (m, γ).

PILLAR operations using Lemma 6.13 in the case of integer weights. Finally, we compute a Monge matrix that is k-equivalent to the min-plus product of the last column $\Phi_p[\,[\,0\,..\,k\,],14k\,]$ of Φ_P with the first row $\Phi_s[\,0,[\,13k\,..\,14k\,]\,]$ of Φ_s and increment all entries of their min-plus product by ψ, thus obtaining a matrix Ψ. Due to Claim 6.15, $\Psi \in \mathcal{F}^{w}_{P,T}|^{k}_{k}$. The time required for computing Ψ (using one of Theorem 3.12 and Lemmas 5.5 and 5.9 depending on the case) is dominated by the time required by the calls to the appropriate variant of `GrowFern-SmallSED`.

6.3 Efficient Solutions for the ListAllOccs and Verify Problems

Corollary 6.16 (`ListAllOccs(P, T, k, w)`). *ListAllOccs can be solved in* $O(\min(\max(k, n-m) \cdot k^2 \log^2(mk), n(m+k)\log(nm)))$ *time in the* PILLAR *model.*

In the case of integer weights, ListAllOccs can also be solved in $O(\max(k, n-m) \cdot k \log^2(mk) \cdot \min(\sqrt{k}\log(mk), W))$ *time in the* PILLAR *model.*

Proof. If $n < m - k$, then P has no (k, w)-error occurrences in T, so we henceforth assume $n \geq m - k$.

First, we observe that Corollary 3.19 directly yields an algorithm with running time $O(n(m+k)\log(nm))$. It suffices to preprocess the graph $\overline{\mathsf{AG}}^w(P,T)$ according to Corollary 3.19 in $O(nm\log(nm))$ time and ask $O(nk)$ queries in $O(\log(nm))$ time each: For each $i \in [\,0\,..\,n\,]$, we ask $O(k)$ queries to retrieve $\mathsf{ed}^w(P \to T[\,i\,..\,j\,))$ for all $j \in [\,i\,..\,n\,] \cap [\,i+m-k\,..\,i+m+k\,]$.

Let us now describe efficient algorithms for the case when $n \le m+2k$. Observe that if there are positions $i,j \in [\,0\,..\,n\,)$ such that $\mathsf{ed}^w(P \to T[\,i\,..\,j\,)) \le k$, we have $\mathsf{ed}(P,T) \le |T[\,0\,..\,i\,)| + \mathsf{ed}^w(P \to T[\,i\,..\,j\,)) + |T[\,j\,..\,n\,)| \le 4k$. We can check whether $\mathsf{ed}(P,T) \le 4k$ in $O(k^2)$ time in the PILLAR model using Lemma 3.20, and, if this is not the case return the empty set. In the case when $\mathsf{ed}(P,T) \le 4k$, we call $\texttt{GrowFern}(P,T,4k,w)$ from Lemma 6.5 thus obtaining a fern $\Phi \in \mathcal{F}^w_{P,T}|^{4k}_{\Delta}$ for $\Delta = \min(n,4k)$. In order to compute the desired set of occurrences, it then suffices to iterate over the entries of Φ and, for each entry $\Phi[\,\ell,r\,] \le k$ such that $\ell \le n-\Delta+r$, report the fragment $T[\,\ell\,..\,n-\Delta+r\,)$ along with the value d. This post-processing takes $O(k^2)$ time.

Next, we present efficient algorithms for the case when $n \ge m+2k$. We consider fragments $T_t := T[\,tk\,..\,\min\{m+(t+2)k,n\}\,)$ for all $t \in [\,0\,..\,\lfloor (n-m+k)/k \rfloor\,]$. Each of them is of length at most $m+2k$, so we can apply the procedure from the previous case to list all (k,w)-error occurrences of P in T_t, along with the underlying costs. For each (k,w)-error occurrence $T_t[\,i'\,..\,j'\,)$, if $i' \le k$, we report a (k,w)-error occurrence $T[\,i'+tk\,..\,j'+tk\,)$ of the same cost.

Every (k,w)-error occurrence $T_t[\,i'\,..\,j'\,)$ corresponds to a (k,w)-error occurrence $T[\,i'+tk\,..\,j'+tk\,)$ of the same cost, and thus the reported occurrences are correct. Moreover, $i' \in [\,0\,..\,k\,)$ implies $i'+tk \in [\,tk\,..\,(t+1)k\,)$, so the reported occurrences are distinct. It remains to prove that all (k,w)-error occurrences are indeed reported. For this, consider a (k,w)-error occurrence $T[\,i\,..\,j\,)$. The condition $j-i \ge m-k$ implies $i \le n-m+k$. Hence, $t := \lfloor i/k \rfloor \in [\,0\,..\,\lfloor (n-m+k)/k \rfloor\,]$. Furthermore, the condition $j-i \le m+k$ implies $j \le i+m+k < (t+1)k+m+k = m+(t+2)k$. Thus, $T[\,i\,..\,j\,) = T_t[\,i'\,..\,j'\,)$ for $i' := i-tk \in [\,0\,..\,k\,)$ and $j' := j-tk$. Consequently, our subroutine for T_t reports $T_t[\,i'\,..\,j'\,)$ as a (k,w)-occurrence of P, and thus we also report $T[\,i\,..\,j\,)$.

The running time is dominated by listing the (k,w)-error occurrences of P in the fragments T_t. Due to the assumption $n \ge m+2k$, the number of such fragments can be written as $1+\lfloor (n-m+k)/k \rfloor \le (n-m+2k)/k \le 2(n-m)/k$. For each of them, the procedure takes $O(k^3\log^2(mk))$ time in the PILLAR model in the general case and $O(k^2\log^2(mk)\cdot\min(\sqrt{k}\log(mk),W))$ time in the PILLAR model in the case of integer weights. The stated bounds follow. ◢

Corollary 6.17 ($\texttt{Verify}(P,\ T,\ k,\ I,\ w)$). *The* Verify *problem can be solved in the* PILLAR *model in time* $O(k^2(k+|I|)\log^2(mk))$.

In the case of integer weights, The Verify *problem can be solved in the* PILLAR *model in time* $O((k+|I|)\cdot k\log^2(mk)\cdot\min(\sqrt{k}\log(mk),W))$.

Proof. We assume $I \subseteq [\,0\,..\,n\,]$ without loss of generality (otherwise, we set $I := I \cap [\,0\,..\,n\,]$). We define $T' := T[\,\min I\,..\,\min\{n,\max I+m+k\}\,)$ and use Corollary 6.16 to list all (k,w)-error occurrences of P in T' along with their costs. While processing these (k,w)-error occurrences $T'[\,i'\,..\,j'\,)$, we skip those with $i' \ge |I|$, and group the remaining ones by the starting position i'. For each starting position $i' \in [\,0\,..\,|I|\,)$ with at least one (k,w)-error occurrence, we report a pair (i,d), where $i := i' + \min I$ and d is the minimum cost among the (k,w)-error occurrences $T'[\,i'\,..\,j'\,)$.

Clearly, whenever we report a pair (i,d), the underlying (k,w)-error occurrence $T'[\,i'\,..\,j'\,)$ corresponds to a (k,w)-error occurrence $T[\,i\,..\,j\,)$ of cost d, where $i := i' + \min I$ and $j := j' + \min I$. Conversely, if $i \in I$ and $T[\,i\,..\,j\,)$ is a (k,w)-error occurrence, then $j-i \le m+k$ implies $j \le i+m+k \le \max I+m+k$, and thus $T'[\,i'\,..\,j'\,)$, where $j' := j-\min I$, is a (k,w)-error occurrence. It is listed by the subroutine of Corollary 6.16, so our algorithm reports a pair (i,d) for $d \le \mathsf{ed}^w(P \to T[\,i\,..\,j\,)) = \mathsf{ed}^w(P \to T'[\,i'\,..\,j'\,))$.

As far as the running time is concerned, observe that $n' := |T'| \leq |I| + m + k$, and thus the algorithm of Corollary 6.16 takes $O(\max(k, |I|+k) \cdot k^2 \log(mk)) = O((k+|I|)k^2 \log^2(mk))$ time. In the case of integer weights, the algorithm of Corollary 6.16 takes $O((k+|I|) \cdot k \log^2(mk) \cdot \min(\sqrt{k}\log(mk), W))$ time. ◢

7 An $\widetilde{O}(n + k^4 \cdot n/m)$-time Algorithm

In line with several works on pattern matching, we focus on a bounded-ratio version of PMwWE, that is, we assume that the length of the text is smaller than $3/2\, m + k$

Remark 7.1 (The standard trick to reduce the text-length to be roughly the pattern-length)**.** Given a text of arbitrary length n, we can reduce the problem to $\lfloor 2n/m \rfloor$ instances of the bounded-ratio version of the problem using the so-called *standard trick* [Abr87]. That is, we consider the overlapping fragments

$$T_i := T[\, \lfloor i \cdot m/2 \rfloor \,..\, \min\{n, \lfloor (i+3) \cdot m/2 \rfloor + k - 1\}\,),$$

for $i \in [\, 0 \,..\, \lfloor 2n/m \rfloor - 1\,)$, and compute the (k, w)-error occurrences of P in each of $T_0, \ldots, T_{\lfloor 2n/m \rfloor - 1}$. We then merge the obtained partial results. Thus, an algorithm that solves the bounded-ratio version of PMwWE in time $f(m, k, W)$ in the PILLAR model (where W is an upper-bound on the weights), directly yields an algorithm that solves an arbitrary instance of PMwWE in time $O(n/m) \cdot f(m, k, W)$ in the PILLAR model. ◢

In Section 7.1, we show that we can focus on a variant of PMwWE where the pattern is almost periodic, that is, the pattern is at small edit distance from a string with small period. In particular, we show how an algorithm from [CKW20, CKW22] for PMwE can be used to reduce PMwWE to several instances of (weighted) Verify (which we can solve using Corollary 6.17) and an instance of the following problem.[16]

Problem AlignedPeriodicMatches$(P,\ T,\ k,\ Q,\ \mathcal{A}_P,\ \mathcal{A}_T,\ w)$**.**

Input. A pattern P of length m,
a positive integer k,
a text T of length $n \in [\, m - k \,..\, \lceil 3/2\, m \rceil + k\,)$,
a primitive string Q of length $q := |Q| \leq m/128k$,
an edit-distance alignment $\mathcal{A}_P : P \rightsquigarrow Q^\infty[\, 0 \,..\, y_P\,)$ of cost $d_P := \mathsf{ed}(P, {}^*Q^*) = \mathsf{ed}(P, Q^*) \leq 16k$,
and an edit-distance alignment $\mathcal{A}_T : T \rightsquigarrow Q^\infty[\, x_T \,..\, y_T\,)$ of cost $d_T := \mathsf{ed}(T, {}^*Q^*) \leq 48k$, where $x_T \in [\, 0 \,..\, q\,)$.
Additionally, oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [\, 0\,,\, W\,]$.

Output. The set $\mathrm{Occ}_k^w(P, T)$.

We formalize our reduction as follows.

Lemma 7.13 (Reduction of PMwWE to AlignedPeriodicMatches and Verify)**.** *An instance of PMwWE with $n < 3/2\, m + k$ can be reduced in $O(k^{3.5}\sqrt{\log m \log k})$ time in the* PILLAR *model to one of the following:*

- *an instance* AlignedPeriodicMatches$(P,\ T',\ k,\ Q,\ \mathcal{A}_P,\ \mathcal{A}_{T'},\ w)$*, where T' is a fragment of T;*
- *$O(k)$ instances* Verify *with the same P, T, k, and w, each with an interval of size $O(k)$.* ◢

Our goal is to then solve AlignedPeriodicMatches by reducing it to an instance of (a variant of) the so-called dynamic puzzle matching problem, in a similar fashion to [CKW22]. To this end, let us recall the notion of a puzzle from [CKW22].

---

[16] The unweighted analogue of AlignedPeriodicMatches from [CKW22] is called NewPeriodicMatches, is formally stated below, and has an extra parameter d which we have set to $16k$ in this work.

Definition 7.2 (A Δ-puzzle of strings and the value of a Δ-puzzle) [CKW22, Definition 1.4]. *For a $\Delta \in \mathbb{Z}_{\geq 0}$, we say that $z \geq 2$ strings $S_1, \ldots, S_z$ form a* Δ-puzzle *if*
- $|S_i| \geq \Delta$ *for each* $i \in [1..z]$, *and*
- $S_i[|S_i| - \Delta..|S_i|) = S_{i+1}[0..\Delta)$ *for each* $i \in [1..z)$.

The value *of the puzzle is* $\mathsf{val}_\Delta(S_1, \ldots, S_z) \coloneqq S_1 \odot S_2[\Delta..|S_2|) \odot S_3[\Delta..|S_3|) \cdots S_z[\Delta..|S_z|)$.

Below, we provide a weighted variant of the dynamic puzzle matching problem from [CKW22]. While we use a more intricate variant of this problem, this simpler version allows us to introduce useful notions in its (simpler) context and help the reader develop intuition.

Problem DynamicPuzzleMatching$(k, \Delta, w, \mathcal{S}_\beta, \mathcal{S}_\mu, \mathcal{S}_\varphi)$.

Input. Positive integers k and Δ, oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$, as well as string families $\mathcal{S}_\beta$, $\mathcal{S}_\mu$, and $\mathcal{S}_\varphi$ of *leading*, *internal*, and *trailing* pieces, respectively.

Maintained Object. A sequence $\mathcal{I} = (U_1, V_1)(U_2, V_2) \cdots (U_z, V_z)$ of ordered pairs of strings (a *DPM-sequence*), that additionally satisfies the following two conditions at initialization time and after each update.
1. $U_1, V_1 \in \mathcal{S}_\beta$, $U_z, V_z \in \mathcal{S}_\varphi$, and, for all $i \in (1..z)$, $U_i, V_i \in \mathcal{S}_\mu$,
2. the torsion $\mathsf{tor}(\mathcal{I}) \coloneqq \sum_{i=1}^{z} \big||U_i| - |V_i|\big|$ satisfies $\mathsf{tor}(\mathcal{I}) \leq \Delta/2 - k$.

Initialization. `DPM-Init(`$\mathcal{I}'$`)`: Initialize $\mathcal{I}$ as $\mathcal{I}'$.

Update Operations. `DPM-Delete(`i`)`: Delete the i-th pair of strings.
`DPM-Insert(`$(U', V'), i$`)`: Insert the pair of strings (U', V') after the i-th pair of strings.
`DPM-Substitute(`$(U', V'), i$`)`: Substitute the i-th pair of strings with the pair of strings (U', V').

Query Operations. `DPM-Query`: Return $\mathrm{Occ}_k^w(\mathcal{I}) \coloneqq \mathrm{Occ}_k^w(\mathsf{val}_\Delta(U_1, \ldots, U_z), \mathsf{val}_\Delta(V_1, \ldots, V_z))$ under a promise that $U_1, \ldots, U_z$ and $V_1, \ldots, V_z$ are Δ-puzzles.

Remark 7.3. A rather straightforward adaptation of a reduction from [CKW22] would yield an instance of DynamicPuzzleMatching with $O(k^3)$ update operations. While each of these operations can be handled in $\widetilde{O}(k)$ time in the unweighted case, in the weighted case we only know how to perform each of them in $O(k^2)$ time. This way, we could obtain an algorithm for AlignedPeriodicMatches and, in turn, for PMwWE, running in time $\widetilde{O}(k^5)$ in the PILLAR model.

The overhead for processing the introduced DPM update operations in the weighted case essentially boils down to the fact that the distance matrices we consider are not unit-Monge (in particular, constant-bounded-difference) as they are in the unweighted case: while the (min, +)-multiplication of $k \times k$ *unit-Monge matrices* can be performed in $\widetilde{O}(k)$ time [Tis15], the (min, +)-multiplication of arbitrary $k \times k$ Monge matrices requires $\widetilde{O}(k^2)$ time (see Theorem 3.12). We address the challenge posed by the absence of the unit-Monge property—and thereby devise an algorithm that runs in $\widetilde{O}(k^4)$ time in the PILLAR model—as follows.
- We refine DynamicPuzzleMatching to BleachCommitDPM, allowing for several extra operations.
- We reduce the AlignedPeriodicMatches problem to BleachCommitDPM.
- While the total number of DPM operations in our instance of BleachCommitDPM is still $O(k^3)$, we perform them in total time $\widetilde{O}(k^4)$ by employing our efficient implementation of BleachCommitDPM from Section 9 (see Theorem 9.10) which reduces them to $O(k^2)$ matrix-matrix multiplications and $O(k^3)$ (cheaper) matrix-vector multiplications.

We obtain the following solution for the AlignedPeriodicMatches problem.

Lemma 7.40. *AlignedPeriodicMatches can be solved in $O(k^4 \log^2(mk))$ time in the* PILLAR *model, with the output set $\mathrm{Occ}_k^w(P,T)$ represented as $O(k^4)$ disjoint arithmetic progressions.*

Further, in Section 8, we obtain the following result for integer metric weight functions.

Lemma 8.44. *AlignedPeriodicMatches can be solved in $\widetilde{O}(k^{3.5}W^4)$ time in the* PILLAR *model if w is an integer metric weight function, with the output set $\mathrm{Occ}_k^w(P,T)$ represented as a collection of disjoint arithmetic progressions.*

All in all, a combination of Lemma 7.13 with Lemmas 7.40 and 8.44 and our efficient algorithms for the Verify problem (Corollary 6.17) yields Main Theorem 2.

Main Theorem 2. *PMwWE on strings with $n < 3/2\, m + k$ can be solved in the* PILLAR *model*
- *in time $O(k^4 \log^2(mk))$ for general weights; and*
- *in time $\widetilde{O}(k^{3.5} \cdot W^4)$ if w is a metric weight function with values in $[0\,..\,W]$.*

The output set $\mathrm{Occ}_k^w(P,T)$ is represented as a collection of disjoint arithmetic progressions.

Proof. Consider an instance of PMwWE with $n \le 3/2\, m + k$. Using Lemma 7.13, we reduce this instance, in $O(k^{3.5}\sqrt{\log m \log k})$ time in the PILLAR model to
- an instance AlignedPeriodicMatches$(P,\ T',\ k\ Q,\ \mathcal{A}_P,\ \mathcal{A}_{T'},\ w)$, where T' is a fragment of T,
- $O(k)$ instances of Verify with the same P, T, k, and w, each with an interval of size $O(k)$.

In the case of arbitrary weights, the obtained instance of AlignedPeriodicMatches can be solved in $O(k^4 \log^2(mk))$ time in the PILLAR model using Lemma 7.40, while the obtained instances of Verify can be solved in $O(k^4 \log^2(mk))$ time in total in the PILLAR model due to Corollary 6.17. PMwWE can thus be solved in $O(k^4 \log^2(mk))$ time in the PILLAR model in the case of arbitrary weights.

If w is an integer metric weight function, the obtained instance of AlignedPeriodicMatches can be solved in $O(k^{3.5}W \log^2(mk))$ time in the PILLAR model using Lemma 8.44, while the obtained instances of Verify can be solved in $O(k^{3.5}W \log^2(mk))$ time in total in the PILLAR model due to Corollary 6.17. PMwWE can thus be solved in $O(k^{3.5}W \log^2(mk))$ time in the PILLAR model if w is an integer metric weight function.

7.1 Reduction to Almost Periodic Patterns

As a first step in solving the bounded-ratio version of PMwWE, we analyze the pattern according to [CKW20, Lemma 6.4].

Lemma 7.4 (`Analyze(P, k)`) **[CKW20, Lemma 6.4].** *Let P denote a string of length m and let $k \le m$ denote a positive integer. Then, there is an algorithm that computes one of the following.*
1. *A set of $2k$ disjoint fragments $B_1, \ldots, B_{2k}$ in P, called* breaks, *each having period* $\mathrm{per}(B_i) > m/128k$ *and length $|B_i| = \lfloor m/8k \rfloor$.*
2. *Disjoint fragments $H_1, \ldots, H_r$ in P, called* repetitive regions, *of total length $\sum_{i=1}^{r} |H_i| \ge 3/8 \cdot m$ such that each region H_i satisfies $|H_i| \ge m/8k$ and is constructed along with a primitive approximate period Q_i such that $|Q_i| \le m/128k$ and $\mathsf{ed}(H_i, {}^*Q_i^*) = \lceil 8k/m \cdot |H_i| \rceil$.*[17]
3. *A primitive approximate period Q of P with $|Q| \le m/128k$ and $\mathsf{ed}(P, {}^*Q^*) < 8k$.*

The algorithm runs in $O(k^2)$ time in the PILLAR *model.*

If the analysis of the pattern using Lemma 7.4 yields breaks we employ the following lemma.

[17] Observe that we have renamed the repetitive regions to $H_\star$, as we use $R_\star$ later with a different meaning.

Fact 7.5 ($O(k^3)$-time PILLAR model algorithm for computing candidate positions in the case of breaks) [CKW20, (proofs of) Lemmas 5.21 and 6.12]**.** *Suppose that we are given a threshold k, a pattern P of length m, disjoint breaks $B_1, \ldots, B_{2k}$ in P, each satisfying $\mathrm{per}(B_i) \geq m/128k$, and a text T of length $n < 3/2\, m + k$. Then, we can compute a set of $O(k)$ intervals, each of length k, whose union is a superset of $\mathrm{Occ}_k(P,T)$ in $O(k^3)$ time in the* PILLAR *model.*

Corollary 7.6 ($O(k^3)$-time PILLAR model reduction from PMwWE to Verify in the case of breaks)**.** *Suppose that we are given a threshold k, a pattern P of length m, disjoint breaks $B_1, \ldots, B_{2k}$ in P, each satisfying $\mathrm{per}(B_i) \geq m/128k$, a text T of length $n < 3/2\, m + k$, and oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$.*

Then, computing $\mathrm{Occ}_k^w(P,T)$ can be reduced in $O(k^3)$ time in the PILLAR *model to solving $O(k)$ instances of Verify with the same P, T, k, and w, each with an interval of size $O(k)$.*

Proof. We first apply Fact 7.5 to compute $O(k)$ k-length intervals $I_1, \ldots, I_t$ whose union is a superset of $\mathrm{Occ}_k(P,T)$ in $O(k^3)$ time in the PILLAR model. Then, as $\mathrm{Occ}_k^w(P,T) \subseteq \mathrm{Occ}_k(P,T)$ (see also Fact 3.7), it suffices to solve $O(k)$ instances of Verify with the same P, T, k, and w, each with an interval of size $O(k)$, and merge the results.

Let us now consider the case when the analysis of the pattern using Lemma 7.4 returns disjoint repetitive regions. To complete the reduction in this case, we need the following result for the NewPeriodicMatches problem from [CKW22, Section 1.3] that is formally defined below and is an unweighted analogue of the AlignedPeriodicMatches problem.

Problem NewPeriodicMatches$(P,\ T,\ k,\ d,\ Q,\ \mathcal{A}_P,\ \mathcal{A}_T)$**.**

Input. A pattern P of length m, an integer threshold $k \in [0\,..\,m]$, a positive integer $d \geq 2k$, a text T of length $n \in [m-k\,..\,\lceil 3/2\, m \rceil + k)$, a primitive string Q of length $q := |Q| \leq m/8d$, an edit-distance alignment $\mathcal{A}_P : P \rightsquigarrow Q^\infty[0\,..\,y_P)$ of cost $d_P := \mathsf{ed}(P, {}^*Q^*) = \mathsf{ed}(P, Q^*) \leq d$, and an edit-distance alignment $\mathcal{A}_T : T \rightsquigarrow Q^\infty[x_T\,..\,y_T)$ of cost $d_T := \mathsf{ed}(T, {}^*Q^*) \leq 3d$, where $x_T \in [0\,..\,q)$.

Output. The set $\mathrm{Occ}_k(P,T)$ represented as $O(d^3)$ disjoint arithmetic progressions with difference q.

Fact 7.7 [CKW22, Lemma 1.3]**.** *We can solve the NewPeriodicMatches problem in $O(d^{3.5}\sqrt{\log n \log d})$ time in the* PILLAR *model.*

The following result is shown in [CKW22, Sections 2.3 and 3.1]; here, we provide a sketch of its proof.

Lemma 7.8 ($\widetilde{O}(k^{3.5})$-time PILLAR model algorithm for computing candidate positions in the case of repetitive regions) [CKW22, Sections 2.3 and 3.1]**.** *Suppose that we are given a threshold k, a pattern P of length m, and a text T of length $n < 3/2\, m + k$. Suppose that we are also given disjoint repetitive regions $H_1, \ldots, H_r$ in P of total length at least $\sum_{i=1}^r |H_i| \geq 3/8\, m$ such that each region H_i satisfies $|H_i| \geq m/8k$ and has a primitive approximate period Q_i with $|Q_i| \leq m/128k$ and $\mathsf{ed}(H_i, Q_i^*) = \lceil 8k/m \cdot |H_i| \rceil$.*

Then, we can compute a set of $O(k)$ intervals, each of length k, whose union is a superset of $\mathrm{Occ}_k(P,T)$ in $O(k^{3.5}\sqrt{\log m \log k})$ time in the PILLAR *model.*

Proof sketch. For each repetitive region H_i, set $k_i := \lfloor 4 \cdot k/m \cdot |H_i| \rfloor$ and $d_i := \lceil 8 \cdot k/m \cdot |H_i| \rceil$. We can compute $\mathrm{Occ}_{k_i}(H_i, T)$ by making several calls to the procedure encapsulated Fact 7.7; consult the discussion following [CKW22, Fact 2.14] as well as [CKW22, Section 3.1] for a rigorous description of these instances. The total time taken by all these calls is $O(k^{3.5}\sqrt{\log m \log k})$ due to Fact 7.7.

Next, using the sets $\mathrm{Occ}_{k_i}(H_i, T)$ we can identify in $O(k^2 \log\log k)$ time $O(k)$ length-k intervals whose union is a superset of $\mathrm{Occ}_k(P,T)$ using a marking scheme.

Corollary 7.9 ($\widetilde{O}(k^{3.5})$-time PILLAR model reduction from PMwWE to Verify in the case of repetitive regions)**.** *Suppose that we are given a threshold k, a pattern P of length m, a text T of length $n < 3/2\, m + k$, and oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$. Suppose that we are also given disjoint repetitive regions $H_1, \ldots, H_r$ in P of total length at least $\sum_{i=1}^{r} |H_i| \geq 3/8\, m$ such that each region H_i satisfies $|H_i| \geq m/8k$ and has a primitive approximate period Q_i with $|Q_i| \leq m/128k$ and $\mathsf{ed}(H_i, Q_i^*) = \lceil 8k/m \cdot |H_i| \rceil$.*

Then, computing $\mathrm{Occ}_k^w(P,T)$ can be reduced in $O(k^{3.5}\sqrt{\log m \log k})$ time in the PILLAR model to solving $O(k)$ instances of Verify with the same P, T, k, and w, each with an interval of size $O(k)$.

Proof. We apply Lemma 7.8 to compute $O(k)$ k-length intervals $I_1, \ldots, I_t$ whose union is a superset of $\mathrm{Occ}_k(P,T)$ in $O(k^{3.5}\sqrt{\log m \log k})$ time in the PILLAR model. Then, as $\mathrm{Occ}_k^w(P,T) \subseteq \mathrm{Occ}_k(P,T)$ (see also Fact 3.7), it suffices to solve $O(k)$ instances of Verify with the same P, T, k, and w, each with an interval of size $O(k)$, and merge the results. ◢

Finally, we consider the case when the analysis of the pattern using Lemma 7.4 returns a primitive approximate period Q of P with $|Q| \leq m/128k$ and $\mathsf{ed}(P, {}^*Q^*) < 8k$. We show that, in this case, PMwWE can be reduced, in $O(k^2)$ time in the PILLAR model, to an instance AlignedPeriodicMatches $(P, T', k, Q, \mathcal{A}_P, \mathcal{A}_T, w)$, where T' is a fragment of T. We use the following two facts. The first one is a simplified version of [CKW20, Lemma 6.8], while the second one is a simplified version of [CKW20, Lemma 6.5]

Fact 7.10 (`FindRelevantFragment(`P`,` T`,` k`,` d`,` Q`)`, Trimming T to an approximately periodic fragment with all approximate occurrences) [CKW20, compare Lemma 6.8]**.** *Let P denote a pattern of length m, let T denote a text of length n, and let $0 \leq k \leq m$ denote a threshold such that $n < 3/2\, m + k$. Further, let $d \geq 2k$ denote a positive integer and let Q denote a primitive string that satisfies $|Q| \leq m/8d$ and $\mathsf{ed}(P, {}^*Q^*) \leq d$.*

Then, there is an algorithm that computes a fragment T' of T such that $\mathsf{ed}(T', {}^*Q^*) \leq 3d$ and $|\mathrm{Occ}_k(P,T)| = |\mathrm{Occ}_k(P,T')|$. The algorithm runs in $O(d^2)$ time in the PILLAR model. ◢

Fact 7.11 (`FindAWitness(`k`,` Q`,` S`)`, $O(k^2)$-time PILLAR algorithm that can be used to rotate Q such that $\mathsf{ed}(P, {}^*Q^*) = \mathsf{ed}(P, Q^*)$) [CKW20, compare Lemma 6.5]**.** *Let k denote a positive integer, let S denote a string, and let Q denote a primitive string that satisfies $|S| \geq (2k+1)|Q|$.*

Then, we can compute a witness *$x \in \mathbb{Z}_{\geq 0}$ such that $\mathsf{ed}(S, \mathrm{rot}^{-x}(Q)^*) = \mathsf{ed}(S, {}^*Q^*) \leq k$, or report that $\mathsf{ed}(S, {}^*Q^*) > k$. The algorithm takes $O(k^2)$ time in the PILLAR model.* ◢

Lemma 7.12 ($O(k^2)$-time PILLAR model reduction from PMwWE to AlignedPeriodicMatches in the case of approximate periodicity)**.** *Suppose that we are given a threshold k, a pattern P of length m, a primitive approximate period Q of P with $|Q| \leq m/128k$ and $\mathsf{ed}(P, {}^*Q^*) < 8k$, a text T of length $n < 3/2\, m + k$, and oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$.*

Then, computing $\mathrm{Occ}_k^w(P,T)$ can be reduced in $O(k^2)$ time in the PILLAR model to solving an instance AlignedPeriodicMatches$(P, T', k, Q, \mathcal{A}_P, \mathcal{A}_{T'}, w)$, where T' is a fragment of T.

Proof. We first use Fact 7.10 with $d = 16k$ in order to replace T by its fragment T' which contains all k-error occurrences of P (and hence all (k,w)-error occurrences by Fact 3.7) and is at small edit distance from a fragment of Q^∞; it is readily verified that all conditions of Fact 7.10 are satisfied in this call. We can assume that the obtained text is of length at least $m - k$ as otherwise $\mathrm{Occ}_k^w(P,T) = \emptyset$.

We then use Fact 7.11 to rotate Q so that it satisfies $\mathsf{ed}(P, {}^*Q^*) = \mathsf{ed}(P, Q^*)$ as follows (similarly to [CKW22]). We make a call `FindAWitness(`$16k$`,` Q`,` P`)` to derive $x \in \mathbb{Z}_{\geq 0}$ such that $\mathsf{ed}(P, \mathrm{rot}^{-x}(Q)^*) = \mathsf{ed}(P, {}^*Q^*)$ in $O(k^2)$ time in the PILLAR model, noting that $|P| \geq 128k|Q| \geq (2 \cdot 16k + 1)|Q|$. Then, we make a call `Alignment(`P`,` Q`,` x`)` to the function of Lemma 3.22, which yields an optimal (unweighted) alignment $\mathcal{A} : P \rightsquigarrow Q^\infty[x \,..\, y)$ of cost $\mathsf{ed}(P, {}^*Q^*)$ in $O(k^2)$ time in the PILLAR model. Next, we extract a fragment Q' of P that $\mathcal{A}$ matches without any edits against $Q^\infty[x' \,..\, x' + |Q|)$ for some $x' \equiv x \bmod |Q|$. Finally, we replace Q with Q', and set $q := |Q|$.

In order to complete the reduction, it suffices to obtain optimal (unweighted) alignments from P and T to fragments of Q^∞, since we have already shown that all the assumptions of AlignedPeriodicMatches are satisfied. A call $\texttt{Alignment}(P, Q, x)$ returns an optimal alignment $\mathcal{A}_P : P \rightsquigarrow Q^\infty[0\,..\,y_P)$ of cost $d_P = \mathsf{ed}(P, {}^*Q^*)$ (for some $y_P \in \mathbb{Z}_{\geq 0}$) in $\mathcal{O}(k^2)$ time in the PILLAR model. Then, we perform a call $\texttt{FindAWitness}(48k, Q, T)$ to compute $x \in \mathbb{Z}_{\geq 0}$ such that $\mathsf{ed}(P, \mathrm{rot}^{-x}(Q)^*) = \mathsf{ed}(P, {}^*Q^*)$, noting that $|T| \geq m - k \geq 128k|Q| - k \geq (2 \cdot 48k + 1)|Q|$. This call takes $\mathcal{O}(k^2)$ time in the PILLAR model. We set x_T to be the unique integer in $[0\,..\,q)$ that is equivalent to x modulo q. Then, a call $\texttt{Alignment}(T, Q, x_T)$ returns an optimal alignment $\mathcal{A}_T : T \rightsquigarrow Q^\infty[x_T\,..\,y_T)$ of cost $d_T = \mathsf{ed}(T, {}^*Q^*)$ (for some $y_T \in \mathbb{Z}_{\geq 0}$) in $\mathcal{O}(k^2)$ time in the PILLAR model. ◢

We are now ready to complete the reduction.

Lemma 7.13 (Reduction of PMwWE to AlignedPeriodicMatches and Verify)**.** *An instance of PMwWE with $n < 3/2\, m + k$ can be reduced in $\mathcal{O}(k^{3.5}\sqrt{\log m \log k})$ time in the PILLAR model to one of the following:*
- *an instance AlignedPeriodicMatches$(P, T', k, Q, \mathcal{A}_P, \mathcal{A}_{T'}, w)$, where T' is a fragment of T;*
- *$\mathcal{O}(k)$ instances Verify with the same P, T, k, and w, each with an interval of size $\mathcal{O}(k)$.*

Proof. We analyze the pattern according to Lemma 7.4. Depending on whether this analysis returns breaks, repetitive regions, or an approximate period, we employ Corollary 7.6, Corollary 7.9, or Lemma 7.12, respectively. ◢

7.2 Puzzle Pieces for P and T

For the rest of Section 7, we fix an instance AlignedPeriodicMatches$(P, T, k, Q, \mathcal{A}_P, \mathcal{A}_T, w)$ and set $\kappa := k + d_P + d_T$ and $\tau := q\left\lceil \kappa/2q \right\rceil$.

After providing intuition, we encapsulate the main ideas of [CKW22, Section 3.2] in Lemma 7.14. For the sake of an intuitive explanation, let us assume here that $q \gg k$. Due to the primitivity of Q, the fact that d_P and d_T are $\mathcal{O}(k)$, and the large number of Q's exact occurrences in both P and any fragment of T of length roughly m, each k-error occurrence of the pattern much start $\mathcal{O}(k)$ positions away from a position of T that $\mathcal{A}_T$ aligns with a position of Q^∞ equivalent to $0 \pmod q$. This allows us to create $\mathcal{O}(m/q)$ substrings R_j of T, each of length $m + \mathcal{O}(k)$, that contain all k-error occurrences of P in T. This intuition is formalized below (for arbitrary values of q and k).

Lemma 7.14 ($\mathcal{O}(d_T + 1)$-time algorithm for computing a collection of short substrings R_j of T that capture all k-edit occurrences and there are only a few distinct ones) [CKW22, Section 3.2]**.** *Consider an instance of the AlignedPeriodicMatches problem. Let $\kappa := k + d_P + d_T$, $\tau := q\left\lceil \kappa/2q \right\rceil$, and*

$$J := \left[\left\lceil (x_T - \kappa)/\tau \right\rceil\,..\,\left\lfloor (y_T - y_P + \kappa)/\tau \right\rfloor\right] = \{j \in \mathbb{Z} : x_T \leq j\tau + \kappa \text{ and } j\tau + y_P - \kappa \leq y_T\}.$$

There exists a family $(R_j)_{j\in J}$ of substrings of T, where $R_j = T[r_j\,..\,r'_j)$ for all $j \in J$, such that:
- *$|R_j| \leq m + 3\kappa - k$ for all $j \in J$, and*
- *$\mathrm{Occ}_k(P, T) = \bigcup_{j\in J}\left(\mathrm{Occ}_k(P, R_j) + r_j\right) \subseteq \bigcup_{j\in\mathbb{Z}}[jq - x_t - \kappa - d_T\,..\,jq - x_T + \kappa + d_T]$.*

Given the alignment $\mathcal{A}_T$, in $\mathcal{O}(d_T + 1)$ time, we can construct the sequences $(r_j)_{j\in J}$ and $(r'_j)_{j\in J}$, represented as concatenations of $\mathcal{O}(d_T + 1)$ arithmetic progressions with difference τ.[18] ◢

Finally, let us present a first algorithm for AlignedPeriodicMatches, which is particularly useful for small sets J, that is, when $q \gg k$.

[18] Parameters κ and τ are defined in the first sentence of [CKW22, Section 3.2], J and the R_js are defined in [CKW22, Definition 3.6], the bound on the lengths of the R_js is from [CKW22, Remark 3.9], while $\mathrm{Occ}_k(P, T) = \bigcup_{j\in J}\left(\mathrm{Occ}_k(P, R_j) + r_j\right)$ is from [CKW22, Corollary 3.11]. The efficient computation of sequences $(r_j)_{j\in J}$ and $(r'_j)_{j\in J}$ is from [CKW22, Lemma 3.12].

Lemma 7.15 (Efficient algorithm for the AlignedPeriodicMatches problem for the case $q \gg k$)**.** *For each $j \in J$, we can compute the set $\mathrm{Occ}_k^w(P, R_j)$ in $O(k^3 \log^2(mk))$ time in the* PILLAR *model. In particular, we can solve the AlignedPeriodicMatches problem in $O(k^3|J| \log^2(mk))$ time in the* PILLAR *model.*

In the case of integer weights, for each $j \in J$, we can compute the set $\mathrm{Occ}_k^w(P, R_j)$ in $O(k^2 W \log^2(mk))$ time in the PILLAR *model and hence solve the AlignedPeriodicMatches problem in $O(k^2 W|J| \log^2(mk))$ time in the* PILLAR *model.*

In either case, we explicitly return the output set $\mathrm{Occ}_k^w(P, T)$, which is of size $O(k|J|)$.

Proof. A combination of Lemma 7.14 with Fact 3.7 yields that $\mathrm{Occ}_k^w(P, T) = \bigcup_{j \in J} \left(\mathrm{Occ}_k^w(P, R_j) + r_j\right)$. Now, recall from Lemma 7.14 that, for each $j \in J$, we have $|R_j| \leq m + 3\kappa - k$. Thus, for each $j \in J$, we can compute $\mathrm{Occ}_k^w(P, R_j)$ using `Verify` from Corollary 6.17.

For general weights, each call to `Verify` takes $O(k^3 \log^2(mk))$ time in the PILLAR model and hence, by merging the partial results, we obtain a PILLAR algorithm with running time $O(k^3|J| \log^2(mk))$.

In the case of integer weights, each call to `Verify` takes $O(k^2 W \log^2(mk))$ time in the PILLAR model and hence, by merging the partial results, we obtain a PILLAR algorithm with running time $O(k^2 W|J| \log^2(mk))$. ◢

We next recall the definitions (from [CKW22]) of suitable puzzle pieces for the strings P and T toward a reduction from AlignedPeriodicMatches to DynamicPuzzleMatching, as well as several results about their properties and efficient computation.

We define partitions of P and T into *tiles* from which appropriate puzzle pieces can be then constructed.

Definition 7.16 (τ-tile partition of a string with respect to a primitive string) [CKW22, Definition 4.1]**.** *Consider a string S, a primitive string Q of length q, an integer $\tau \in \mathbb{Z}_{>0}$ divisible by q, and an alignment $\mathcal{A}_S : S \rightsquigarrow Q^\infty[\, x_S \,..\, y_S \,)$, where $x_S \in [\, 0 \,..\, q \,)$ and $y_S \geq x_S$.*

*Partition Q^∞ into blocks of length τ and number them starting from 1. For the j-th block $Q_j := Q^\infty[\, \max\{x_S, (j-1)\tau\} \,..\, \min\{y_S, j\tau\} \,)$, we define the j-*th tile of S (with respect to $\mathcal{A}_S$) *as*

$$S[\, s_{i-1} \,..\, s_i \,) := \mathcal{A}_S^{-1}(Q_j).$$

*Further, we define the τ-*tile partition of S with respect to $\mathcal{A}_S$ *as the partition*

$$S = \bigodot_{i=1}^{\beta_S} S[\, s_{i-1} \,..\, s_i \,),$$

where $\beta_S := \lceil y_S/\tau \rceil$ is the index of the last non-empty tile of S. ◢

Fact 7.17 (Upper bound on the number of substrings constructed by Lemma 7.14 with respect to the number of tiles in P) [CKW22, proof of Lemma 4.5]**.** *We have $|J| = O(\beta_P)$.* ◢

A combination of Fact 7.17 and Lemma 7.15, yields that, if $\beta_P = \lceil y_P/\tau \rceil$ is very small (that is if the tiles are very long), then we can already efficiently solve the AlignedPeriodicMatches problem.

Lemma 7.18 (Efficient solution for the AlignedPeriodicMatches problem parameterized by the number of tiles in P)**.** *We can solve AlignedPeriodicMatches in time $O(k^3 \beta_P \log^2(mk))$ in the* PILLAR *model returning explicitly the output set $\mathrm{Occ}_k^w(P, T)$, which is of size $O(k\beta_P)$.*

In the case of integer weights, we can solve AlignedPeriodicMatches in time $O(k^2 W \beta_P \log^2(mk))$ in the PILLAR *model returning explicitly the output set $\mathrm{Occ}_k^w(P, T)$, which is of size $O(k\beta_P)$.* ◢

In particular, Lemma 7.18 allows us to assume (without loss of generality) that $\beta_P \geq 20$ as, otherwise, Lemma 7.18 implies Lemma 7.40. We make this assumption in the remainder of Section 7.2 and in Section 7.3.

Now, let $P = \bigodot_{i=1}^{\beta_P} P[\, p_{i-1} .. p_i \,)$ denote the τ-tile partition of P with respect to $\mathcal{A}_P$, and let $T = \bigodot_{i=1}^{\beta_T} T[\, t_{i-1} .. t_i \,)$ denote the τ-tile partition of T with respect to $\mathcal{A}_T$.

Let us set $\Delta \coloneqq 6\kappa$ and $z \coloneqq \beta_P - 17$. By (essentially) extending the tiles from the τ-tile partition of P by an additional Δ characters, we obtain a Δ-puzzle with value P.

Lemma 7.19 (Δ-puzzle for P from its tile partition) [CKW22, Lemma 4.6]**.** *The following sequence $P_1, \ldots, P_z$ forms a Δ-puzzle with value P:*
- $P_1 \coloneqq P[\, p_0 .. p_2 + \Delta \,)$;
- $P_i \coloneqq P[\, p_i .. p_{i+1} + \Delta \,)$ *for* $i \in (\, 1 .. z \,)$;
- $P_z \coloneqq P[\, p_z .. |P| \,)$.

Similarly, we obtain Δ-puzzles for T. In particular, we use the τ-tile partition of T in order to represent the fragments R_j (from Lemma 7.14) as Δ-puzzles.

Lemma 7.20 (Δ-puzzle for T from its tile partition) [CKW22, Lemma 4.7]**.** *For each $j \in J$, the following sequence $T_{j,1}, \ldots, T_{j,z}$ forms a Δ-puzzle with value R_j:*
- $T_{j,1} \coloneqq T[\, r_j .. t_{j+2} + \Delta \,)$;
- $T_{j,i} \coloneqq T[\, t_{j+i} .. t_{j+i+1} + \Delta \,)$ *for* $i \in (\, 1 .. z \,)$;
- $T_{j,z} \coloneqq T[\, t_{j+z} .. r'_j \,)$.

Taken together, we obtain the families of puzzle pieces that we use in the remainder of this work.

Definition 7.21 [CKW22, Definition 4.8]**.**
- *We write $\mathcal{S}_\beta \coloneqq \{P_1\} \cup \{T_{j,1} : j \in J\}$ for the family of* leading *puzzle pieces.*
- *We write $\mathcal{S}_\mu \coloneqq \{P_i : i \in (\, 1 .. z \,)\} \cup \{T_i : i \in (\, \min J + 1 .. \max J + z \,)\}$ for the family of* internal *puzzle pieces.*
- *We write $\mathcal{S}_\varphi \coloneqq \{P_z\} \cup \{T_{j,z} : j \in J\}$ for the family of* trailing *puzzle pieces.*

Remark 7.22 [CKW22, Remark 4.9]**.** For convenience, for $i \in (\, \min J + 1 .. \max J + z \,)$, we write $T_i \coloneqq T[\, t_i .. t_{i+1} + \Delta \,)$. Observe that by construction, we have $T_i = T_{j,i'}$ for all $j + i' = i$ with $i' \in (\, 1 .. z \,)$; that is, overlapping parts of different R_j's share their internal pieces. Observe further that this is an essential property for our approach to work: when moving from R_j to R_{j+1}, we exploit that we need to only shift the pieces T_i, and not recompute them altogether.

Finally, the following lemma establishes that the pairs of pieces P_i and $T_{j,i}$ from Definition 7.21 are roughly of the same length on average.

Lemma 7.23 [CKW22, Lemma 4.10]**.** *For each $j \in J$, we have $\sum_{i=1}^{z} \big||T_{j,i}| - |P_i|\big| \leq 3\kappa - k$.*

Special Puzzle Pieces

In order for puzzle pieces to be useful to us, we need to be able to efficiently compute the families $\mathcal{S}_\beta$, $\mathcal{S}_\mu$, and $\mathcal{S}_\varphi$ specified in Definition 7.21. As it turns out, when considering instances of AlignedPeriodicMatches, most puzzle pieces are substrings of Q^∞—this means that it suffices to locate and compute the *special* pieces, that is, the pieces that are different from some specific substrings of Q^∞. We start with a formal definition of special puzzle pieces.

Definition 7.24 (Special puzzle pieces) [CKW22, Definition 4.11]**.** *We say that an internal piece is* special *if and only if it is different from* $Q^\infty[\,0\,..\,\tau+\Delta\,)$*. We write* $\mathit{Special}(P)$ *for the set of* special internal pieces *of* P *and* $\mathit{Special}(T)$ *for the set of* special internal pieces *of* T*; that is, we set*

$$\mathit{Special}(P) := \{P_i : i \in (\,1\,..\,z\,) \text{ and } P_i \neq Q^\infty[\,0\,..\,\tau+\Delta\,)\} \quad \text{and}$$
$$\mathit{Special}(T) := \{T_i : i \in (\,1+\min J\,..\,z+\max J\,) \text{ and } T_i \neq Q^\infty[\,0\,..\,\tau+\Delta\,)\}.$$

Further, we say that a leading piece $T_{j,1}$ *is* special *if and only if it is different from* $Q^\infty[\,-\kappa\,..\,2\tau+\Delta\,)$ *and that a trailing piece* $T_{j,z}$ *is* special *if and only if it is different from* $Q^\infty[\,z\tau\,..\,y_P+\kappa\,)$*. Similar to before, we write* $\mathit{Special}_{\beta\varphi}(T)$ *for the set of special leading and trailing pieces of* T*.*

Indeed, there are only very few special pieces. We compute them in $O(k)$ time in the PILLAR model using the following lemma.

Lemma 7.25 [CKW22, Lemma 1.6]**.** *The median edit distance of each of the families* $\mathcal{S}_\beta$*,* $\mathcal{S}_\mu$*, and* $\mathcal{S}_\varphi$ *is* $O(k)$*. Further, each of the multisets* $\mathit{Special}(P)$*,* $\mathit{Special}(T)$*, and* $\mathit{Special}_{\beta\varphi}(T)$ *is of size* $O(k)$ *and can be computed in* $O(k)$ *time in the* PILLAR *model.*

7.3 Reduction to (Bleach-Commit) Dynamic Puzzle Matching

Let us sketch a warm-up reduction from AlignedPeriodicMatches to DynamicPuzzleMatching. In an instance of DynamicPuzzleMatching, we iterate over sequences $\mathcal{I}_j$ of puzzle pieces, such that the value of the Δ-puzzle consisting of first components is P and that of the second components is R_j, for each $j \in J$. Then, it would suffice to ask a `DPM-Query` for each $\mathcal{I}_j$ and merge the results. A formal definition of the $\mathcal{I}_j$s is provided below.

Definition 7.26 (Sequence of pairs of puzzle pieces for T_j and P) [CKW22, Definition 4.21]**.** *For each* $j \in J$*, set* $\mathcal{I}_j := (P_1, T_{j,1})(P_2, T_{j,2}) \cdots (P_z, T_{j,z})$*.*

Note that, due to Lemmas 7.19 and 7.20, we have $\mathsf{val}_\Delta(P_1, \ldots, P_z) = P$ and $\mathsf{val}_\Delta(T_{j,1}, \ldots, T_{j,z}) = R_j$.

In order to achieve a more efficient solution, we distinguish so-called *canonical pairs* in a DPM-sequence $\mathcal{I}$, with respect to our fixed instance of the AlignedPeriodicMatches problem; recall that $\Delta := 6(k + d_P + d_T)$ and $\tau := q\lceil \kappa/2q \rceil$.

Definition 7.27 (Canonical pairs of pieces) [CKW22, Definition 4.23]**.** *We call a pair of pieces* canonical *if it equals*

$$Q := (Q^\infty[\,0\,..\,\tau+\Delta\,), Q^\infty[\,0\,..\,\tau+\Delta\,)).$$

A crucial combinatorial property exploited in [CKW22], is the fact that, for unweighted edit distance, one can "trim" long runs of canonical pairs without affecting the output of a `DPM-Query`. Here, we extend this observation to weighted edit distance. To this end, let us recall the definition of the Trim operator for DPM-sequences. We call a maximal contiguous subsequence of internal pairs that satisfy some property an *internal run* of pairs that satisfy said property.

Definition 7.28 (Trimmed sequences of pairs of pieces according to plain pairs and a threshold) [CKW22, Definition 4.27]**.** *Consider a DPM-sequence* $\mathcal{I}$*, where the elements of some subset* $\mathit{Plain}(\mathcal{I})$ *of the internal canonical DPM-pairs are labeled as* plain*. (We call all other pieces* non-plain*.)*

For a positive integer α*, we write* $\mathit{Tm}(\mathcal{I}, \mathit{Plain}(\mathcal{I}), \alpha)$ *for the DPM-sequence obtained from* $\mathcal{I}$ *by removing exactly one plain DPM-pair from any internal run of plain DPM-pairs of length at least* $\alpha+1$*; if* $\mathcal{I}$ *does not contain any such run, we set* $\mathit{Tm}(\mathcal{I}, \mathit{Plain}(\mathcal{I}), \alpha) := \mathcal{I}$*.*

Further, we write $\mathit{Trim}(\mathcal{I}, \mathit{Plain}(\mathcal{I}), \alpha) := \mathit{Tm}^\star(\mathcal{I}, \mathit{Plain}(\mathcal{I}), \alpha)$ *for an iterated application of* Tm *until the DPM-sequence remains unchanged. (That is, in* $\mathit{Trim}(\mathcal{I}, \mathit{Plain}(\mathcal{I}), \alpha)$ *every internal run of plain DPM-pairs is of length at most* α*.)*

For technical reasons, we may not want to include all canonical pairs Q in the set $\mathsf{Plain}(\mathcal{I})$.

Remark 7.29 [CKW22, Remark 4.28]**.** Write $\mathsf{Red}(P) \supseteq \mathsf{Special}(P)$ for a set of *red* pieces of P and write $\mathsf{Red}(T) \supseteq \mathsf{Special}(T)$ for a set of red pieces of T. Now, for a $j \in J$, we set

$$\mathsf{Plain}(\mathcal{I}_j) \coloneqq \{(P_i, T_{j,i}) : i \in (1\,..\,z) \text{ and } P_i \notin \mathsf{Red}(P) \text{ and } T_{j,i} \notin \mathsf{Red}(T).$$

Abusing notation, we write $\mathsf{Trim}(\mathcal{I}_j, \mathsf{Red}(P), \mathsf{Red}(T), \alpha) \coloneqq \mathsf{Trim}(\mathcal{I}_j, \mathsf{Plain}(\mathcal{I}_j), \alpha)$.

The following lemma, whose proof is similar to the proof of [CKW22, Lemma 4.32], implies that we can cap the exponent of internal runs of canonical pieces to $k+2$ without altering the set of (k,w)-error occurrences. The proof of [CKW22, Lemma 4.32] relied on the greedy nature of optimal unweighted alignments to show an analogue of the inclusion $\mathrm{Occ}_k^w(\mathcal{I}) \subseteq \mathrm{Occ}_k^w(\mathcal{I}^+)$ shown in Lemma 7.30. In the presence of weights, we have to take a different approach; this is the main difference between the proof of the following lemma and the proof of [CKW22, Lemma 4.32].

Lemma 7.30. *Fix a string $\hat{Q}$, integers $k \geq 0$ and $\Delta > 0$, and a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$. Further, consider a DPM-sequence $\mathcal{I} = (U_1, V_1)(U_2, V_2)\cdots(U_z, V_z)$ whose pieces form Δ-puzzles and that has a torsion of $\mathsf{tor}(\mathcal{I}) \leq \Delta/2 - k$, and a set $\mathsf{Plain}(\mathcal{I})$ of DPM-pairs labeled as plain, where $\mathsf{Plain}(\mathcal{I}) \subseteq \{(U_i, V_i) \in \mathcal{I} : i \in (1\,..\,z)$ and $U_i = V_i = \hat{Q}\}$.*

Suppose that $\mathcal{I}$ contains an internal run $\mathcal{X} \coloneqq (U_x, V_x)\cdots(U_{x+k'}, V_{x+k'})$ such that all DPM-pairs in $\mathcal{X}$ are in $\mathsf{Plain}(\mathcal{I})$ and $k' \geq k+1$.

Let $\mathcal{I}^-$ be the DPM-sequence obtained by replacing $\mathcal{X}$ with $\mathcal{X}^- \coloneqq (U_x, V_x)\cdots(U_{x+k'-1}, V_{x+k'-1})$ and $\mathcal{I}^+$ be the DPM-sequence obtained by replacing $\mathcal{X}$ with $\mathcal{X}^+ = \mathcal{X} \odot (\hat{Q}, \hat{Q})$. We have $\mathrm{Occ}_k^w(\mathcal{I}) \subseteq \mathrm{Occ}_k^w(\mathcal{I}^-)$ and $\mathrm{Occ}_k^w(\mathcal{I}) \subseteq \mathrm{Occ}_k^w(\mathcal{I}^+)$.

Proof. The following claim states that the deletion of an internal piece from a Δ-puzzle yields a Δ-puzzle.

Claim 7.31 [CKW22, see Claim 4.33]. *Given a Δ-puzzle $\mathcal{X} \coloneqq X_1, \ldots, X_i, X_{i+1}, \ldots, X_z$ with $X_i = X_{i+1}$ and value $X \coloneqq \mathsf{val}_\Delta(\mathcal{X})$, the sequence*

$$\mathcal{X}' \coloneqq X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_z = X_1, \ldots, X_i, X_{i+2}, \ldots, X_z$$

forms a Δ-puzzle with value

$$\mathsf{val}_\Delta(\mathcal{X}') = X[\,0\,..\,\xi_i + \lfloor\Delta/2\rfloor\,)X[\,\xi_i + |X_i| - \lceil\Delta/2\rceil\,..\,|X|\,) = X[\,0\,..\,\xi_i + p\,)X[\,\xi_{i+1} + p\,..\,|X|\,),$$

where $\xi_i \coloneqq \sum_{t=1}^{i-1} |X_t| - \Delta$ and $p \in [\,0\,..\,|X_i| - \Delta\,]$.

Let $U \coloneqq \mathsf{val}_\Delta(U_1, \ldots, U_z)$ and $V \coloneqq \mathsf{val}_\Delta(V_1, \ldots, V_z)$. For each $j \in (1\,..\,z)$, write $U[\,u_j\,..\,u'_j\,) = U_j$ and $V[\,v_j\,..\,v'_j\,) = V_j$; that is $u_j = \sum_{t=1}^{j-1}(|U_t| - \Delta)$ and $v_j = \sum_{t=1}^{j-1}(|V_t| - \Delta)$. With Claim 7.31 in mind, we also write $\hat{U}_j \coloneqq U[\,\hat{u}_j\,..\,\hat{u}'_j\,)$, where

$$\hat{u}_j \coloneqq u_j + \lfloor\Delta/2\rfloor = \sum_{t=1}^{j-1}(|U_t| - \Delta) + \lfloor\Delta/2\rfloor \qquad \text{and}$$

$$\hat{u}'_j \coloneqq \hat{u}_j + |U_j| - \Delta = \sum_{t=1}^{j-1}(|U_t| - \Delta) + |U_j| - \lceil\Delta/2\rceil = u'_j - \lceil\Delta/2\rceil.$$

For convenience, we define $\hat{u}'_1$ and $\hat{u}_z$ analogously. Observe that we have $\hat{u}'_j = \hat{u}_{j+1}$, that is, we obtain a partition $U = U[\,0\,..\,\hat{u}'_1\,)\hat{U}_2\cdots\hat{U}_{z-1}U[\,\hat{u}_z\,..\,|U|\,)$.

Consider an alignment $\mathcal{A} : U \rightsquigarrow V[\,a\,..\,b\,)$ that satisfies $\mathsf{ed}^w_{\mathcal{A}}(U \to V[\,a\,..\,b\,)) = \mathsf{ed}^w_{\leq k}(U \to V[\,a\,..\,b\,))$.

Claim 7.32 [CKW22, see Claim 4.34]. *For any $j \in (1\,..\,z)$, the fragment $\hat{V}_j := V[\,\hat{v}_j\,..\,\hat{v}'_j\,) := \mathcal{A}(\hat{U}_j)$ is contained in V_j.*

Proof. Observe that, by construction, we have

$$\begin{aligned} v_j &= u_j + \sum_{t=1}^{j-1}(|V_t| - |U_t|) \\ &\le u_j + \sum_{t=1}^{z}\big||U_t| - |V_t|\big| \\ &= u_j + \mathsf{tor}(\mathcal{I}) \\ &\le u_j + \lfloor \Delta/2 \rfloor - k \\ &\le a + \hat{u}_j - k \le \hat{v}_j. \end{aligned}$$

Symmetrically, we obtain

$$\begin{aligned} \hat{v}'_j \le b + \hat{u}'_j - |U| + k &\le |V| - |U| + u'_j - \lceil \Delta/2 \rceil + k \\ &\le |V| - |U| + u'_j - \mathsf{tor}(\mathcal{I}) = |V| - |U| + u'_j - \sum_{t=1}^{z}\big||U_t| - |V_t|\big|) \\ &\le |V| - |U| + u'_j - \sum_{t=j+1}^{z}(|V_t| - |U_t|) = v'_j, \end{aligned}$$

thus completing the proof. ⌟

As the alignment $\mathcal{A}$ can make at most k edits, at least two of the first components of the (at least $k+2$) DPM-pairs of $\mathcal{X}$ get aligned without any edits. We may thus assume without loss of generality that $\hat{V}_i = \hat{U}_i$ for some $i \in [\,x\,..\,x+k'\,)$. Claim 7.32 ensures that $\hat{V}_i$ is contained in V_i.

We first prove that $\mathrm{Occ}^w_k(\mathcal{I}) \subseteq \mathrm{Occ}^w_k(\mathcal{I}^-)$. Let us remove the pair (U_i, V_i) from $\mathcal{I}$, thus obtaining $\mathcal{I}^-$. Let $U^- := \mathsf{val}_\Delta(U_1, \ldots, U_{i-1}, U_{i+1}, \ldots, U_z)$ and $V^- := \mathsf{val}_\Delta(V_1, \ldots, V_{i-1}, V_{i+1}, \ldots, V_z)$. Since $\mathsf{val}_\Delta(V_x, \ldots, V_{x+k'})$ has a period $|U_i| - \Delta$, and $\hat{V}_i = \hat{U}_i$ is of length $|U_i| - \Delta$, we conclude that

$$\begin{aligned} \mathsf{ed}^w(U \to V[\,a\,..\,b\,)) &= \mathsf{ed}^w((U[\,0\,..\,\hat{u}_i\,) \to V[\,a\,..\,\hat{v}_i\,)) \\ &\qquad + \mathsf{ed}^w(U[\,\hat{u}_i + |U_i| - \Delta\,..\,|U|\,) \to V[\,\hat{v}_i + |U_i| - \Delta\,..\,b\,)) \\ &= \mathsf{ed}^w(U^-[\,0\,..\,\hat{u}_i\,) \to V^-[\,a\,..\,\hat{v}_i\,)) \\ &\qquad + \mathsf{ed}^w(U^-[\,\hat{u}_i\,..\,|U^-|\,) \to V^-[\,\hat{v}_i\,..\,b - |U_i| + \Delta\,)) \\ &\ge \mathsf{ed}^w(U^- \to V^-[\,a\,..\,b - |U_i| + \Delta\,)). \end{aligned}$$

We can thus obtain an alignment $\mathcal{A}^- : U^- \rightsquigarrow V^-[\,a\,..\,b - |U_i| + \Delta\,)$ of weighted cost at most k, completing the proof of $\mathrm{Occ}^w_k(\mathcal{I}) \subseteq \mathrm{Occ}^w_k(\mathcal{I}^-)$.

We now prove $\mathrm{Occ}^w_k(\mathcal{I}) \subseteq \mathrm{Occ}^w_k(\mathcal{I}^+)$. Let us replace $\mathcal{X}$ by $\mathcal{X}^+$, thus obtaining $\mathcal{I}^+$, by inserting $(\hat{Q}, \hat{Q})$ just after (U_i, V_i). Set

$$U^+ := \mathsf{val}_\Delta(U_1, \ldots, U_i, \hat{Q}, U_{i+1}, \ldots, U_z) \quad \text{and} \quad V^+ := \mathsf{val}_\Delta(V_1, \ldots, V_i, \hat{Q}, V_{i+1}, \ldots, V_z).$$

Similarly to before, due to the periodicity of $\mathsf{val}_\Delta(V_x, \ldots, V_{x+k'})$, we have

$$\begin{aligned}
\mathsf{ed}^w(U \to V[\,a\,..\,b\,)) &= \mathsf{ed}^w(U[\,0\,..\,\hat{u}'_i\,) \to V[\,a\,..\,\hat{v}'_i\,)) + \mathsf{ed}^w(U[\,\hat{u}'_i\,..\,|U|\,) \to V[\,\hat{v}'_i\,..\,b\,)) \\
&= \mathsf{ed}^w(U[\,0\,..\,\hat{u}'_i\,) \to V[\,a\,..\,\hat{v}'_i\,)) \\
&\quad + \mathsf{ed}^w(\hat{Q}[\,0\,..\,|\hat{Q}| - \Delta\,) \to \hat{Q}[\,0\,..\,|\hat{Q}| - \Delta\,)) \\
&\quad + \mathsf{ed}^w(U[\,\hat{u}'_i\,..\,|U|\,) \to V[\,\hat{v}'_i\,..\,b\,)) \\
&= \mathsf{ed}^w(U^+[\,0\,..\,\hat{u}'_i\,) \to V^+[\,a\,..\,\hat{v}'_i\,)) \\
&\quad + \mathsf{ed}^w(U^+[\,\hat{u}'_i\,..\,\hat{u}'_i + |\hat{Q}| - \Delta\,) \to V^+[\,\hat{v}'_i\,..\,\hat{v}'_i + |\hat{Q}| - \Delta\,)) \\
&\quad + \mathsf{ed}^w(U^+[\,\hat{u}'_i + |\hat{Q}| - \Delta\,..\,|U^+|\,) \to V^+[\,\hat{v}'_i + |\hat{Q}| - \Delta\,..\,b + |\hat{Q}| - \Delta\,)) \\
&\geq \mathsf{ed}^w(U^+ \to V^+[\,a\,..\,b + |\hat{Q}| - \Delta\,)).
\end{aligned}$$

We can thus obtain an alignment $\mathcal{A}^+ : U^+ \rightsquigarrow V^+[\,a\,..\,b + |\hat{Q}| - \Delta\,)$ of weighted cost at most k, completing the proof of $\mathrm{Occ}^w_k(\mathcal{I}) \subseteq \mathrm{Occ}^w_k(\mathcal{I}^+)$. ◢

Noting that $\Delta = 6\kappa$ and hence $\mathsf{tor}(\mathcal{I}_j) \leq 3\kappa - k = \Delta/2 - k$, a combination of Lemmas 7.23 and 7.30 yields the following result, which is an analogue of [CKW22, Lemma 4.30] for weighted edit distance.

Corollary 7.33. *For any $j \in J$, we have*

$$\mathrm{Occ}^w_k(P, R_j) = \mathrm{Occ}^w_k(\mathcal{I}_j) = \mathrm{Occ}^w_k(\mathit{Trim}(\mathcal{I}_j, \mathit{Special}(P), \mathit{Special}(T), k+2)).$$ ◢

Remark 7.34. The $k+2$ in Corollary 7.33 can be optimized to $k+1$, which is a more intuitive parameter; we opted to not implement this optimization as the proof of an analogously refined variant of Lemma 7.30 would be lengthier that that of Lemma 7.30. ◢

We next argue that, even with Corollary 7.33 at hand, we cannot iterate over all distinct DPM-sequences $\mathsf{Trim}(\mathcal{I}_j, \mathsf{Special}(P), \mathsf{Special}(T), k+2)$, for $j \in J$, with $o(k^3)$ update operations in the worst case. Let us fix a pair $(P_x, T_y) \in \mathsf{Special}(P) \times \mathsf{Special}(T)$ and consider the maintenance of a DPM-sequence $\mathcal{I}$ that iterates over all specified sequences. Suppose that at some point, the length-$(k+4)$ sequence

$$(P_x, Q^\infty[\,0\,..\,\tau + \Delta\,))\boldsymbol{Q}^{k+2}(Q^\infty[\,0\,..\,\tau + \Delta\,), T_y)$$

is a contiguous subsequence of $\mathcal{I}$. As we, intuitively, shift P over T, the exponent of the internal run of canonical pairs enclosed by the DPM-pairs of strings containing P_x and T_y might change $\Omega(k)$ times. It is actually not hard to construct an instance of AlignedPeriodicMatches where $\mathsf{Special}(P) \times \mathsf{Special}(T)$ is of size $\Omega(k^2)$ and each pair in $\mathsf{Special}(P) \times \mathsf{Special}(T)$ is "responsible" for $\Omega(k)$ updates as in the considered example for P_x and T_y. In order to circumvent this issue, we refine DynamicPuzzleMatching by introducing some further update operations that admit faster implementations than those in the original problem (while offloading some computation to the query procedure). A refined reduction allows us to only use $O(k^2)$ *expensive* update operations and $O(k^3)$ *cheap* update operations.

Definition 7.35 (Blank pieces for representing runs of canonical pairs)**.** *For a positive integer j, a* blank piece of order j *is a compact representation of a sequence of j copies of $\boldsymbol{Q}$ as $(\boldsymbol{Q}, j)$.* ◢

Definition 7.36. *Operation Unpack takes as input a sequence $\mathcal{J}$ of ordered pairs of strings and blank pieces and replaces each blank piece of the form $(\boldsymbol{Q}, j)$ with j copies of $\boldsymbol{Q}$. The output is denoted by $\mathsf{Unpack}(\mathcal{J})$.*

We say that a sequence $\mathcal{J}$ is a Bleached-representation of a sequence $\mathcal{I}$ of ordered pairs of strings if $\mathcal{I} = \mathsf{Unpack}(\mathcal{J})$. ◢

Problem BleachCommitDPM.

Maintained Object. A sequence $\mathcal{Y}$ of puzzle pieces and blank pieces (a *BCDPM-sequence*).

Initialization. `BCDPM-Init`$(\mathcal{Y}', \alpha, k, \Delta, \mathcal{S}_\beta, \mathcal{S}_\mu, \mathcal{S}_\varphi, Q, w)$: Given
- a BCDPM-sequence $\mathcal{Y}'$,
- positive integers k, $\Delta = O(k)$, and $\alpha = O(k)$,
- string families $\mathcal{S}_\beta, \mathcal{S}_\mu$, and $\mathcal{S}_\varphi$ of *leading*, *internal*, and *trailing* pieces, respectively, that satisfy $\mathsf{ed}(\mathcal{S}_\beta) + \mathsf{ed}(\mathcal{S}_\mu) + \mathsf{ed}(\mathcal{S}_\varphi) = O(k)$,
- a canonical pair of strings Q,
- and oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$,

initialize $\mathcal{Y}$ as $\mathcal{Y}'$.

Invariants. At initialization time and after each update, all blank pieces in $\mathcal{Y}$ are of order at most α and the DPM-sequence

$$\mathcal{I} := \mathsf{Unpack}(\mathcal{Y}) = (U_1, V_1)(U_2, V_2)\cdots(U_z, V_z)$$

satisfies $U_1, V_1 \in \mathcal{S}_\beta$, $U_z, V_z \in \mathcal{S}_\varphi$, and, for all $i \in (1..z)$, $U_i, V_i \in \mathcal{S}_\mu$.

Update Operations. `BCDPM-Delete`(i): Delete the i-th element of $\mathcal{Y}$.

`BCDPM-Insert`$((U', V'), i)$: Insert the pair (U', V') of strings after the i-th element of $\mathcal{Y}$.

`BCDPM-Substitute`$((U', V'), i)$: Substitute the i-th element of $\mathcal{Y}$, which is a pair of strings, with the pair of strings (U', V').

`BCDPM-Commit`(i, ζ): Replace the i-th element of $\mathcal{Y}$, which is a blank piece, with ζ copies of Q.

`BCDPM-Bleach`(i, ζ): Replace the ζ pieces, that follow the i-th element of $\mathcal{Y}$ and are all equal to Q, with a single blank piece (Q, ζ).

`BCDPM-Set`(i, ζ): Substitute the i-th element of $\mathcal{Y}$, which is a blank piece, with blank piece (Q, ζ).

Query Operations. `BCDPM-Query`: Return

$$\mathrm{Occ}_k^w(\mathcal{Y}) := \mathrm{Occ}_k^w(\mathsf{val}_\Delta(U_1, \ldots, U_z), \mathsf{val}_\Delta(V_1, \ldots, V_z))$$

under a promise that $U_1, \ldots, U_z$ and $V_1, \ldots, V_z$ are Δ-puzzles and that $\mathsf{tor}(\mathcal{I}) \le \Delta/2 - k$.

The following theorem is analogous to [CKW22, Lemma 4.36]. The authors of [CKW22] showed that one can iterate over $\mathcal{I}'_j := \mathsf{Trim}(\mathcal{I}_j, \mathsf{Red}(P), \mathsf{Red}(T), \alpha)$, for all $j \in J \setminus \{\min J\}$ satisfying $\mathcal{I}'_{j-1} \neq \mathcal{I}'_j$, in an instance of DynamicPuzzleMatching using a small number of updates. They showed that if we skip over some $\mathcal{I}'_j$ that is equal to $\mathcal{I}'_{j+1}$ can be handled by extending, by one element, some maintained arithmetic progressions (with difference τ) of (k, w)-error occurrences. The crucial novelties in the proof of the following theorem are the definition of BCDPM-sequences $\mathcal{Y}'_j$ that satisfy $\mathsf{Unpack}(\mathcal{Y}'_j) = \mathcal{I}'_j$, the replacement of all DPM update operations apart from $O(k^2)$ of them with (cheap) updates of the type `BCDPM-Set` and (the proof of) Claim 7.39. To achieve the latter, we crucially observe that most operations either extend or shrink an internal run of plain pairs: we maintain such runs using blank pieces, and spend extra time during the query to treat them.

Theorem 7.37 (`TrimmedPM`$(T, P, k, w, Q, \mathcal{A}_P, \mathcal{A}_T, \mathit{Red}(P), \mathit{Red}(T)\ \alpha)$, Reduction from the AlignedPeriodicMatches problem to the BleachCommitDPM problem)**.** *Suppose that we are given an instance* AlignedPeriodicMatches$(P, T, k, Q, \mathcal{A}_P, \mathcal{A}_T, w)$ *with*

$$\beta_P := \left\lceil y_P/q \left\lceil (k + d_P + d_T)/2q \right\rceil \right\rceil \ge 20,$$

and a positive integer $\alpha = O(k)$ as well as sets of puzzle pieces $\mathit{Red}(P) \supseteq \mathit{Special}(P)$ and $\mathit{Red}(T) \supseteq \mathit{Special}(T)$ of size at most ρ each.

In time $O(\rho^2\alpha(k + \log n))$ in the PILLAR *model, we can reduce the computation of a representation of*

$$\mathcal{G} := \bigcup_{j\in J}(r_j + \mathrm{Occ}_k^w(\mathit{Trim}(\mathcal{I}_j, \mathit{Red}(P), \mathit{Red}(T), \alpha)))$$

that consists in $O(k\rho^2\alpha)$ arithmetic progressions with difference $\tau = O(\max\{q, k\})$ to an instance of the BleachCommitDPM *problem with the maintained BCDPM-sequence initialized with a call* `BCDPM-Init(`$\mathcal{Y}'$`,` α`,` k`,` Δ`,` $\mathcal{S}_\beta$`,` $\mathcal{S}_\mu$`,` $\mathcal{S}_\varphi$`,` $\mathcal{Q}$`,` w`)`*, where $\mathcal{Y}'$ is an BCDPM-sequence of length $O(\rho\alpha)$, under*

- *$O(\rho^2)$ calls to each of* `BCDPM-Delete`, `BCDPM-Insert`, `BCDPM-Substitute`, `BCDPM-Commit`, *and* `BCDPM-Bleach`, *and*
- *$O(\rho^2\alpha)$ calls to each of* `BCDPM-Set` *and* `BCDPM-Query`, *such that the total number of blank pieces over all calls to* `BCDPM-Query` *is $O(\rho^2\alpha)$.*

Proof. We set $\mathcal{S}_\beta, \mathcal{S}_\mu, \mathcal{S}_\varphi$ according to Definition 7.21, choose $\Delta := 6\kappa = 6(k + d_P + d_T) = O(k)$, and set $\mathcal{Q} := (Q^\infty[0\,..\,\tau + \Delta), Q^\infty[0\,..\,\tau + \Delta))$. We compute sets $\mathsf{Special}(P)$ and $\mathsf{Special}(T)$ in $O(k)$ time in the PILLAR model using Lemma 7.25. Note that Lemma 7.25 ensures that $\mathsf{ed}(\mathcal{S}_\beta) + \mathsf{ed}(\mathcal{S}_\mu) + \mathsf{ed}(\mathcal{S}_\varphi) = O(k)$.

We initialize the BleachCommitDPM data structure with a call to the operation `BCDPM-Init(`$\mathcal{Y}'_{\min J}$`,` α`,` k`,` Δ`,` $\mathcal{S}_\beta$`,` $\mathcal{S}_\mu$`,` $\mathcal{S}_\varphi$`,` $\mathcal{Q}$`,` w`)`. We denote the maintained BCDPM-sequence by $\mathcal{Y}$. Then, we wish to transform $\mathcal{Y}$ to be equal to $\mathcal{Y}'_j$, for each $j \in J$, so that we can call a `BCDPM-Query` to compute $\mathrm{Occ}_k^w(\mathcal{Y}'_j)$ and construct arithmetic progressions of (k, w)-occurrences, akin to [CKW22]. In the next claim, we generate sequences update_j of BleachCommitDPM updates operations (BCDPM-updates) that allow us to do exactly that and satisfy the conditions of the statement of the theorem.

Claim 7.38. *In $O(\rho^2\alpha \log n)$ time in total, we can compute, for all $j \in J \setminus \{\min J\}$ with $\mathcal{Y}'_{j-1} \neq \mathcal{Y}'_j$, a sequence update_j of BleachCommitDPM update operations that transforms $\mathcal{Y}'_{j-1}$ to $\mathcal{Y}'_j$, such that the required invariants are maintained at all times. Further, these sequences of updates are returned in increasing order with respect to j and contain in total*

- *$O(\rho^2)$ calls to each of* `BCDPM-Delete`, `BCDPM-Insert`, `BCDPM-Substitute`, `BCDPM-Commit`, *and* `BCDPM-Bleach`, *and*
- *$O(\rho^2\alpha)$ calls to* `BCDPM-Set`.

Proof. In order to develop some intuition, let us examine how the length of the run of plain pairs that succeeds the head of $\mathcal{I}$ changes in the course of the algorithm. This length may decrease as a DPM pair A containing some $T_t \in \mathsf{Red}(T)$ approaches the head of $\mathcal{I}$ while we slide P on T. When such a DPM-pair disappears in the process of transforming $\mathcal{I}_{t-1}$ to $\mathcal{I}_t$, the length of the considered run might increase depending on where the leftmost pair with a red piece in $\mathcal{I}_t$ is; roughly speaking, the run of plain pairs after A in $\mathcal{I}_{t-1}$ becomes the run of plain pairs that succeeds the head of $\mathcal{I}_t$.

We say that a run of plain pairs is *enclosed* by the two DPM-pairs that are adjacent to it. Suppose that in each of $\mathcal{I}_{j-1}$ and $\mathcal{I}_j$ there is a run of plain pairs that is enclosed by a pair that contains a fragment $P_i \in \mathsf{Red}(P) \cup \{P_1, P_z\}$ and a pair that contains a fragment $T_t \in \mathsf{Red}(T)$. The lengths of these two runs differ by at most one; details are provided below. Observe that any run of plain pairs in $\mathcal{I}_{j-1}$ that is not enclosed by a pair that contains a fragment $P_i \in \mathsf{Red}(P) \cup \{P_1, P_z\}$ and a pair that contains a fragment $T_t \in \mathsf{Red}(T)$ either remains intact in $\mathcal{I}_j$ or is shifted to the left by one position. In particular, each change to the length of a run of plain pairs as we transform $\mathcal{I}'_{j-1}$ to $\mathcal{I}'_j$ can be attributed to a single pair of fragments $P_i \in \mathsf{Red}(P) \cup \{P_1, P_z\}$ and $T_t \in \mathsf{Red}(T)$ getting closer to (or farther from) each other.

Let us call an internal run of plain pairs of a DPM-sequence *active* if it is of size smaller than α and either of the following holds:

- it is enclosed by the leading pair and a pair containing a piece of $\mathsf{Red}(T)$,

- it is enclosed by the trailing pair and a pair containing a piece of $\mathsf{Red}(T)$,
- it is enclosed by distinct DPM-pairs such that one contains a piece of $\mathsf{Red}(P)$ and no piece from $\mathsf{Red}(T)$ and the other contains a piece of $\mathsf{Red}(T)$ and no piece from $\mathsf{Red}(P)$.

Internal runs of plain pairs that are not active are called *inactive*. For a DPM-sequence $\mathcal{S}$ with some canonical pairs marked as plain, let us denote by $\mathsf{BleachAct}(\mathcal{S})$ the Bleached-representation of $\mathcal{S}$, obtained by replacing each active internal run of r plain pairs by a blank piece (Q, r). For each $j \in J$, let $\mathcal{Y}'_j := \mathsf{BleachAct}(\mathcal{I}'_j)$. Note that $\mathcal{Y}'_{j-1} \neq \mathcal{Y}'_j$ if and only if $\mathcal{I}'_{j-1} \neq \mathcal{I}'_j$ for each $j \in J \setminus \{\min J\}$.

For convenience, apart from $\mathcal{Y}$, we also explicitly maintain the sequence $\mathcal{I} = \mathsf{Unpack}(\mathcal{Y})$, initialized as $\mathcal{I}'_{\min J}$, that we transform to be equal to each $\mathcal{I}'_j$. In particular, we first compute updates for $\mathcal{I}$ and then transform them to updates for $\mathcal{Y}$, ensuring that, at all times, $\mathcal{Y} := \mathsf{BleachAct}(\mathcal{I})$. We store $\mathcal{Y}$ and $\mathcal{I}$ as doubly-linked lists Y and I, respectively, and maintain bidirectional pointers between pairs in Y and I that contain a common red piece, labelled with said red piece(s). (We do not explicitly mention when such pointers need to be updated.)

We compute sequences of updates for I. Each of them is of one of the following forms:

- $\mathsf{list\text{-}sub}(\mathsf{pointer}, (P_i, T_{j,i}))$: given a handle pointer to an element of I, substitute $(P_i, T_{j,i})$ for this element;
- $\mathsf{list\text{-}ins}(\mathsf{pointer}, (P_i, T_{j,i}))$: given a handle pointer to an element of I, insert $(P_i, T_{j,i})$ before this element;
- $\mathsf{list\text{-}del}(\mathsf{pointer})$: given a handle pointer to an element of I, delete its successor in I.

Further, each update is of the following types, and is annotated with its type:

- Type 1: A substitution of the head/tail of the sequence or an update that only involve pairs that contain a red piece.
- Type 2: An insertion or a deletion of a plain pair that results in the creation or the destruction of a run of plain pairs.
- Type 3: An insertion or a deletion of a plain pair that changes whether a run of plain pairs is active.
- Type 4: An insertions or a deletion of a plain pair that shrinks or expand an active internal run that remains active after the update.

In total, our sequences will contain $O(\rho^2)$ updates of Types 1-3 and $O(\rho^2\alpha)$ updates of Type 4. We show how to maintain $\mathcal{Y}$ subject to each update on I, such that $\mathcal{Y} := \mathsf{BleachAct}(\mathcal{I})$ by transforming, in $O(\log \rho)$ time, each update for I to

- $O(1)$ BCDPM-updates for $\mathcal{Y}$ if the update is of Type 1, 2, or 3, or
- to a single `BCDPM-Set` if the update is of Type 4.

We maintain Y analogously.

In order to facilitate the efficient transformation of updates for I to updates for $\mathcal{Y}$, we use a balanced binary search tree to support the following operation in $O(\log \rho)$ time at the cost of an $O(\log \rho)$-time additive overhead for each update operation on Y: given a handle to an element of Y, return its rank, that is, the number of elements that precede it in Y. Additionally, in $O(\rho \log \rho)$ time, we insert the elements of $\mathsf{Red}(P) \cup \{P_1, P_z\}$ in a doubly linked list L_P and the elements of $\mathsf{Red}(T)$ in a doubly linked list L_T, sorted with respect to their starting positions. At all times, we store a bidirectional pointer between each element of each of L_P and L_T and the DPM-pair that contains it in I. (We do not explicitly mention when such pointers need to be updated.) For both P and T, we write $\mathsf{pointer}(P_i)$ (or $\mathsf{pointer}(T_i)$) for the pointer from the element of L_P (or L_T) corresponding to P_i (or T_i) to the element of I that contains P_i (or T_i).

Throughout the course of the algorithm, to ensure the correct and efficient maintenance of our pointers, we only update pairs that contain a piece in $\mathsf{Red}(P)$ in each of I and Y using substitution operations.

We now explain how to map each computed update for I to a BCDPM-update for $\mathcal{Y}$. Observe that using our pointers and rank structure, given a handle to an element of I that contains a red piece, we can compute in $O(\log \rho)$ time the rank of the pair that contains the same red piece in Y and, hence, in $\mathcal{Y}$. We transform an update for I to an update for Y and $\mathcal{Y}$ as follows:

- If the update is of Type 1, we simply change the index of the update based on the computed rank.
- If the update is of Type 2, we consider two cases:
  - if the update results in the creation of a run, we issue a `BCDPM-Insert` update, followed by a `BCDPM-Bleach` update;
  - if the update results in the destruction of a run, we issue a `BCDPM-Delete` update.
- For a Type 4 update, we also consider two cases:
  - if the update is of the form list-del and changes the status of an inactive run to active, we issue a `BCDPM-Bleach` update followed by a `BCDPM-Set` update;
  - if the update is of the form list-ins and changes the status of an active run to inactive, we issue a `BCDPM-Commit` update.
- If the update is of Type 5, we transform it to a `BCDPM-Set` update.

We now explain how to compute the desired updates for I, which we insert in a global min-heap with keys in $(\min J \mathinner{..} \max J]$. When we are done generating updates, we pop them from the heap, appending each update with key j to the sequence $\mathtt{update}_j$. This process indeed returns sequences $\mathtt{update}_j$ in increasing order with respect to j.

We first issue updates for the head and the tail of the sequence.[19] To this end, we first compute the union HeadTail of the sets

$$\{j \in J : T_{j,1} \neq Q^{\infty}[-\kappa \mathinner{..} 2\tau + \Delta)\} \quad \text{and} \quad \{j \in J : T_{j,z} \neq Q^{\infty}[z\tau \mathinner{..} y_P + \kappa)\},$$

in $O(k)$ time using Lemma 7.25. For each $j \in \mathsf{HeadTail} \setminus \min J$, for each $e \in \{1, z\}$, we issue an update $\mathsf{list\text{-}sub}(\mathsf{pointer}(P_e), (P_e, T_{j,e}))$ with key j. Further, for each $j \in \mathsf{HeadTail} \setminus \max J$, for each $e \in \{1, z\}$, we issue update $\mathsf{list\text{-}sub}(\mathsf{pointer}(P_e), (P_e, T_{j+1,e}))$ with key $j+1$. The number of such issued updates is $O(k) = O(\rho)$ and their all of Type 1.

Let us now issue all updates that involve some red piece. For each pair of fragments $P_i \in \mathsf{Red}(P)$ and $T_t \in \mathsf{Red}(T)$ we do the following.

- If $t - i \in (\min J \mathinner{..} \max J]$, we issue updates:
  - $\mathsf{list\text{-}sub}(\mathsf{pointer}(P_i), (P_i, T_t))$ with key $t - i$, substituting (P_i, T_{t-1}) with (P_i, T_t) and
  - if $i + 1 < z$ and $P_{i+1} \notin \mathsf{Red}(P)$, $\mathsf{list\text{-}del}(\mathsf{pointer}(P_i))$ with key $t - i$, deleting the pair (P_{i+1}, T_t).
- If $t - i + 1 \in (\min J \mathinner{..} \max J]$, we issue updates:
  - if $i > 2$ and $P_{i-1} \notin \mathsf{Red}(P)$, $\mathsf{list\text{-}ins}(\mathsf{pointer}(P_i), (P_{i-1}, T_t))$ with key $t - i + 1$, inserting (P_{i-1}, T_t) just before the pair containing P_i, and
  - if $T_{t+1} \notin \mathsf{Red}(T)$, $\mathsf{list\text{-}sub}(\mathsf{pointer}(P_i), (P_i, T_{t+1}))$ with key $t - i + 1$, replacing (P_i, T_t) with (P_i, T_{t+1}).

Over all pairs of red pieces, we thus issue $O(\rho^2)$ such updates in total, and each of these updates only involves pairs that contain a red piece. We thus annotate these updates as Type 1.

Next, we compute updates that enable the efficient maintenance of the (trimmed) run of plain pairs that is enclosed by a pair containing some $P_i \in \mathsf{Red}(P) \cup \{P_1, P_z\}$ and a pair containing some $T_t \in \mathsf{Red}(T)$, when such a run exists.

First, we consider the case when the pair containing P_i precedes the pair containing $T_t = T_{j-1,t-(j-1)}$ in some $\mathcal{I}'_{j-1}$ and these two pairs enclose a non-empty run of plain pairs. In this case, the run would either shrink in $\mathcal{I}'_j$ or retain its length α. Let the successor of P_i in L_P be P_v. Further, if T_t is not the first element in L_T let its predecessor in L_T be $T_u = T_{j-1,u-(j-1)}$; otherwise, set $u := -\infty$. The following conditions must be satisfied:

[19] Some of the issued updates might be redundant or duplicate, but this is not a problem.

(i) $t-(j-1) \le v$, which is equivalent to $j > t-v$, so that P_v is not in a pair strictly between the two pairs and hence none of the elements of $\mathsf{Red}(P)$ is,
(ii) $u-(j-1) \le i$, which is equivalent to $j > u-i$, so that there is no non-plain pair containing some element of $\mathsf{Red}(T)$ between the two pairs, and
(iii) $t-(j-1)-i > 1$, which is equivalent to $j \le t-i-1$, so that the precedence condition is satisfied and the two pairs are not adjacent.

Further, the run should shrink in $\mathcal{I}'_j$ only if the two pairs are close enough, that is, if $t-(j-1)-i-1 \le \alpha$, which is equivalent to $j > t-i-\alpha-1$. Hence, for each

$$j \in (\max\{\min J, t-v, u-i, t-i-\alpha-1\} .. \min\{\max J, t-i-1\}],$$

we issue an update $\mathsf{list\text{-}del}(\mathsf{pointer}(P_i))$ with key j, deleting a plain pair from the run succeeding the pair that contains P_i with key j. Observe that only the last update may result in a destruction of the run, while only the first update may change its status by turning it from inactive to active. We can compute whether this is the case for these two updates in $O(\log \rho)$ time, and hence annotate all updates appropriately (each of them is of Type 2, 3, or 4). Observe that the total number of issued updates is at most $(t-i-1)-(t-i-\alpha-1) = \alpha$ and at most two of them are not of Type 4.

We now consider the complementary case when in each of $\mathcal{I}'_{j-1}$ and $\mathcal{I}'_j$, the pair containing P_i succeeds the pair containing T_t and these two pairs either enclose a non-empty run of plain pairs or are adjacent.[20] In this case, the run would either be expanded by one plain pair in $\mathcal{I}'_j$ or retain its length α. Let the predecessor of P_i in L_P be P_y. Further, if T_t is not the last element in L_T let its successor in L_T be $T_x = T_{j-1,x-(j-1)}$; otherwise, set $x := \infty$. The following conditions must be satisfied:

(i) $t-(j-1) > y$, which is equivalent to $j \le t-y$, so that P_x is strictly to the left of both pairs in $\mathcal{I}'_{j-1}$, as if this is not the case then the condition on $\mathcal{I}'_j$ would not be satisfied,
(ii) $x-(j-1) > i$, which is equivalent to $j \le x-i$, so that T_y (if it exists) is strictly to the right of both pairs in $\mathcal{I}'_{j-1}$, as if this is not the case then the condition on $\mathcal{I}'_j$ would not be satisfied,
(iii) $t-(j-1) < i$, which is equivalent to $j > t-i+1$, so that the precedence condition is satisfied.

Further, the run should be expanded in $\mathcal{I}'_j$ only if the two pairs are close enough, that is, if $i-(t-(j-1))-1 < \alpha$, which is equivalent to $j \le \alpha+t-i+1$. Hence, for each

$$j \in (\max\{\min J, t-i+1\} .. \min\{\max J, t-y, x-i, \alpha+t-i+1\}],$$

we issue an update $\mathsf{list\text{-}ins}(\mathsf{pointer}(P_i), (Q^\infty[0..\tau+\Delta), Q^\infty[0..\tau+\Delta)))$, inserting a plain pair in the (potentially empty) run preceding the pair that contains P_i with key j. Observe that only the first update may result in the creation of a run, while only the first update may change its status from active to inactive. We can compute whether this is the case for these two updates in $O(\log \rho)$ time, and hence annotate all updates appropriately (each of them is of Type 2, 3, or 4). Observe that the total number of such issued updates is at most $(\alpha+t-i+1)-(t-i+1) = \alpha$ and at most two of them are not of Type 4.

We can compute all updates for a given pair (in the intermediate interface) in $O(\alpha)$ time using lists L_P and L_T.

The invariants of BleachCommitDPM are clearly satisfied at initialization. By construction, throughout the course of the algorithm each blank piece in $\mathcal{Y}$ is of order at most α. Additionally the second invariant is satisfied at all times, since all the updates involving the head or tail of $\mathcal{Y}$ only involve pairs in $\mathcal{S}_\beta \times \mathcal{S}_\beta$ and $\mathcal{S}_\varphi \times \mathcal{S}_\varphi$, respectively, while all remaining updates involve pairs in $\mathcal{S}_\mu \times \mathcal{S}_\mu$. ⌟

We are now ready to complete the reduction to BleachCommitDPM; see Algorithm 2 for a pseudocode implementation.

Using Claim 7.38, we initialize the set

$$\mathcal{U} = \{\mathsf{update}_j : j \in (\min J .. \max J] \text{ and } \mathcal{Y}'_{j-1} \neq \mathcal{Y}'_j\}.$$

[20] Observe that the pairs can only be adjacent in $\mathcal{I}'_{j-1}$.

Algorithm 2. Reduction of computing $\mathcal{G}$ to an instance of BleachCommitDPM.

```
1  TrimmedPM(T, P, k, w, Q, A_P, A_T, Red(P), Red(T), α)
2      G ← ∅;
3      Construct set U of sequences update_j specified in Claim 7.38;        // O(ρ²α log n) time
4      foreach j ∈ (min J .. max J] such that r_j − r_{j−1} ≠ τ and Y'_{j−1} = Y'_j do
5          update_j ← {BCDPM-Substitute((P_1, T_{j,1}),1)};
6          U ← U ∪ {update_j};
7      BCDPM-Init(Y'_{min J}, α, k, Δ, S_β, S_μ, S_φ, Q, w);                 // O(ρα + ρ log k) time
8      t ← min J;
9      foreach j ∈ {i ∈ (min J .. max J + 1] : update_i ≠ ∅ or i = max J + 1} in incr. order do
10         Occ_k^w(Y'_j) ← BCDPM-Query;
11         foreach a ∈ Occ_k^w(Y'_j) do
12             G ← G ∪ {r_t + a + i·τ : i ∈ [0 .. j − t)};
13         if j ≤ max J then
14             Apply the sequence of updates update_j on Y;
15             t ← j;
16     return G;
```

To simplify the construction of certain arithmetic progressions later in the proof, we insert to $\mathcal{U}$ the following single-element (void) sequences of updates. For each $j \in (\min J \,..\, \max J]$ such that $r_j - r_{j-1} \neq \tau$ and $\mathcal{Y}'_{j-1} = \mathcal{Y}'_j$, we insert to $\mathcal{U}$ the sequence $\mathtt{update}_j$ that consists of a single element: $\mathtt{BCDPM\text{-}Substitute}((P_1, T_{j,1}),1)$, that is, the substitution of the first pair of the maintained sequence with $(P_1, T_{j,1})$. By Lemma 7.14, we only generate $O(d_T + 1)$ new updates.

Now, we initialize the sequence $\mathcal{Y}$ with $\mathcal{Y}'_{\min J}$. We consider the sequences of updates $\mathtt{update}_j$ in $\mathcal{U}$ in increasing order with respect to j and apply them to the maintained sequence $\mathcal{Y}$. Prior to performing the sequence of updates $\mathtt{update}_j$ we do the following. Suppose that the maintained sequence $\mathcal{Y}$ corresponds to $\mathcal{Y}'_t$, that is, either $\mathtt{update}_j$ is the first element of $\mathcal{U}$ and $t = \min J$ or the previously applied sequence of updates was $\mathtt{update}_t$. First, we compute $\mathrm{Occ}_k^w(\mathcal{Y}'_t)$ using a $\mathtt{BCDPM\text{-}Query}$, noting that $\mathrm{tor}(\mathcal{I}'_t) \le \mathrm{tor}(\mathcal{I}_t) \le \Delta/2 - k$ is satisfied at all times due to the fact that $\mathcal{I}'_j$ is a subsequence of $\mathcal{I}_j$ and Lemma 7.23. Now, observe that we have $\mathcal{Y}'_t = \cdots = \mathcal{Y}'_{j-1}$ and hence $\mathrm{Occ}_k^w(\mathcal{Y}'_t) = \cdots = \mathrm{Occ}_k^w(\mathcal{Y}'_{j-1})$. Further, for each $i \in (t \,..\, j)$, we have $r_i - r_{i-1} = \tau$ and hence $r_i + \mathrm{Occ}_k^w(\mathcal{Y}'_i) = r_t + \mathrm{Occ}_k^w(\mathcal{Y}'_t) + (i-t)\tau$. We can thus efficiently return $\bigcup_{i=t}^{j-1}(r_i + \mathrm{Occ}_k^w(\mathcal{Y}'_t))$ as the union of $O(|\mathrm{Occ}_k^w(\mathcal{Y}'_i)|)$ arithmetic progressions

$$\bigcup_{a \in \mathrm{Occ}_k^w(\mathcal{Y}'_t)} \{r_t + a + i \cdot \tau : i \in [0 \,..\, j - t)\}.$$

(Observe that in the case when $t = j - 1$, each of the constructed arithmetic progressions consists of a single element.) Finally, we apply the sequence of updates $\mathtt{update}_j$. The case when there are no more updates to be processed is treated analogously (with j set to $\max J + 1$). The upper bound on the number of returned arithmetic progressions is embedded in the analysis of the time complexity of the algorithm.

We now proceed to analyze the time required by Algorithm 2.

First, recall that the elements of $\mathcal{U}$ as returned by Claim 7.38 in $O(k^2\alpha \log n)$ time are sorted in increasing order with respect to j and a representation of $(r_j)_{j \in J}$ as the concatenation of $O(d_T + 1)$ arithmetic progressions with difference τ can be computed in $O(d_T + 1)$ time in the PILLAR model due to Lemma 7.14. Thus, Lines 4–6 can be implemented in $O(k^2\alpha)$ time in a parallel left-to-right scan of $\mathcal{U}$ and the aforementioned representation of $(r_j)_{j \in J}$, maintaining the sortedness of $\mathcal{U}$.

Observe that each $\mathcal{I}'_j$, for $j \in J$ is of length $O(k\alpha)$ as it consists of a head, a tail, $O(\rho)$ non-plain pairs and $O(\rho)$ internal runs of plain pairs, each of length at most α. Sequence $\mathcal{Y}'_{\min J}$ can be thus computed in $O(\rho\alpha + \rho \log \rho)$ time in the PILLAR model: the head and the tail are computed in $O(k)$ time in the PILLAR model due to Lemma 7.25, while the pairs consisting of internal pieces can be computed by processing the elements of $\mathsf{Red}(P)$ and $\mathsf{Red}(T)$ in a left-to-right manner, after sorting them with respect to their starting positions in $O(\rho \log \rho)$ time. The BCDPM sequence in the call to BCDPM-Init is thus a sequence of length $O(\rho\alpha)$.

The for-loop of Lines 9–15 is executed $O(\rho^2\alpha)$ times as this is an upper bound on the number of sequences of updates that are processed and hence on the calls to BCDPM-Query. For each $j \in J$, due to Lemma 7.23, we have that the difference of the two Δ-puzzles represented by the first and second components of the elements of each $\mathcal{I}'_j$ is at most $3\kappa - k$ and hence $|\mathrm{Occ}^w_k(\mathcal{Y}'_j)| = O(k)$. Thus, each query in Line 2 returns a set of size $O(k)$. Hence, apart from the time required for BleachCommitDPM updates and queries, each iteration of the for-loop takes $O(k)$ time, for a total of $O(k\rho^2\alpha)$ time. It readily follows that the algorithm outputs $O(k\rho^2\alpha)$ arithmetic progressions.

In total, in $O(k\rho^2\alpha + \rho^2\alpha \log n)$ time in the PILLAR model, the problem in scope reduces to an instance of BleachCommitDPM with $\mathcal{S}_\beta + \mathcal{S}_\mu + \mathcal{S}_\varphi = O(k)$ (due to Lemma 7.25), $\Delta = 6\kappa = O(k)$, $\mathcal{Y}$ initialized as a sequence of length $O(\rho\alpha)$, and updates satisfying the condition in the lemma's statement due to Claim 7.38.

It only remains to bound the total number of blank pieces over all calls to DPM-Query.

Claim 7.39. *The total number of blank pieces over all calls to* DPM-Query *is* $O(\rho^2\alpha)$.

Proof. In a sequence $\mathcal{Y}'_j$, a blank piece is adjacent either to the leading element of $\mathcal{Y}'_j$, or to the trailing element of $\mathcal{Y}'_j$, or to a pair of strings containing a piece from $\mathsf{Red}(P)$ and a pair of strings containing a piece from $\mathsf{Red}(T)$.

We have already established that the number of calls to BCDPM-Query are $O(\rho^2\alpha)$. Thus, we get a trivial bound of $O(\rho^2\alpha)$ for the number of blank pieces adjacent to the leading/trailing element of the BCDPM-sequences involved in all calls to BCDPM-Query. It thus remains to bound, over all calls to BCDPM-Query, the number of blank pieces adjacent to two internal elements of the BCDPM-sequence such that one contains a piece from $\mathsf{Red}(P)$ and the other contains a piece from $\mathsf{Red}(T)$. For each of the $O(\rho^2)$ pairs $(P_x, T_y) \in \mathsf{Red}(P) \times \mathsf{Red}(T)$, a pair containing P_x and a pair containing T_y can be adjacent at the same time to an internal run of canonical pieces of size smaller than α in $\mathcal{I}'_j$ for $O(\alpha)$ values of j. As only internal runs of $\mathcal{I}'_j$ of size smaller than α are represented by blank pieces, the bound follows. ◿

This concludes the proof of the theorem. ◿

7.4 Wrap-up: Solution for Almost Periodic Patterns

In order to complete the description of our efficient algorithm for AlignedPeriodicMatches, we use the data structure encapsulated in the following theorem, which is proved in Section 9.

Theorem 9.10 (Efficient implementation for BleachCommitDPM)**.** *There is an implementation for BleachCommitDPM that satisfies all of the following.*

- BCDPM-Init($\mathcal{Y}'_{\min J}$, α, k, Δ, $\mathcal{S}_\beta$, $\mathcal{S}_\mu$, $\mathcal{S}_\varphi$, Q, w) *can be performed in time*
  - $O(k^4 \log^2(k\lambda) + |\mathcal{Y}'|k^2)$ *for general weights and in time*
  - $O(k^3 W \log^2(k\lambda) + |\mathcal{Y}'|kW \log k)$ *for integer weights,*

  where λ is the length of the longest string in $\mathcal{S}_\beta \cup \mathcal{S}_\mu \cup \mathcal{S}_\varphi$;
- *any call to* BCDPM-Set *can be performed in constant time;*
- *any call to* BCDPM-Query *can be performed in $O(k\rho)$ time for general weights and in $O(k\rho \log \Delta)$ time for integer weights, where ρ is the number of blank pieces at the time of the query;*

- *any call to any other operation can be performed in time*
  - $O(k^2 \log(k + |\mathcal{Y}|))$ *for general weights and in time*
  - $O(kW \log^2(k + |\mathcal{Y}|))$ *for integer weights,*

  where $\mathcal{Y}$ is the BCDPM sequence to which the operation is applied.

We are now ready to prove Lemma 7.40.

Lemma 7.40. *AlignedPeriodicMatches can be solved in $O(k^4 \log^2(mk))$ time in the* PILLAR *model, with the output set $\mathrm{Occ}_k^w(P,T)$ represented as $O(k^4)$ disjoint arithmetic progressions.*

Proof. Let us set $\tau := q\lceil k + d_P + d_T/2q\rceil$, and $\Delta := 6\kappa$.

If $\beta_P := \lceil y_P/\tau\rceil < 20$, we are done, as we can solve the AlignedPeriodicMatches problem in $O(k^3\beta_P \log^2(mk))$ time in the PILLAR model, returning an $O(k)$-size explicit representation of $\mathrm{Occ}_k^w(P,T)$, due to Lemma 7.18. We henceforth suppose that $\beta_P > 20$, in which case all the statements of Sections 7.2 and 7.3 hold.

Lemma 7.14 states that $\mathrm{Occ}_k(P,T) = \bigcup_{j\in J}\left(\mathrm{Occ}_k(P,R_j) + r_j\right)$. Due to Fact 3.7, we also have $\mathrm{Occ}_k^w(P,T) \subseteq \bigcup_{j\in J}\left(\mathrm{Occ}_k^w(P,R_j) + r_j\right)$. Then, Lemmas 7.19 and 7.20 state that $\mathsf{val}_\Delta(P_1,\ldots,P_z) = P$ and $\mathsf{val}_\Delta(T_{j,1},\ldots,T_{j,z}) = R_j$, respectively. Further, due to Corollary 7.33, we have that, for all $j \in J$, $\mathrm{Occ}_k^w(P,R_j) = \mathrm{Occ}_k^w(\mathcal{I}_j) = \mathrm{Occ}_k^w(\mathsf{Trim}(\mathcal{I}_j, \mathsf{Special}(P), \mathsf{Special}(T), k+2)$. It thus suffices to compute

$$\mathcal{G} := \bigcup_{j\in J}(r_j + \mathrm{Occ}_k^w(\mathsf{Trim}(\mathcal{I}_j, \mathsf{Special}(P), \mathsf{Special}(T), k+2))).$$

To this end, we call procedure `TrimmedPM`$(T, P, k, w, Q, \mathcal{A}_P, \mathcal{A}_T, \mathit{Special}(P), \mathit{Special}(T), k+2)$ from Theorem 7.37, which computes $\mathcal{G}$ represented as $O(k^4)$ arithmetic progressions with difference τ since we have $\mathsf{Special}(P) + \mathsf{Special}(T) = O(k)$ from Lemma 7.25. In what follows, we use the implementation of BleachCommitDPM from Theorem 9.10 for arbitrary weight functions. Let us analyze the time complexity of this call. The reduction takes time $O(k^3(k + \log n))$ in the PILLAR model. We next analyze the time required by our calls to BCDPM operations:

- We call `BCDPM-Init`$(\mathcal{Y}', k+2, k, \Delta, \mathcal{S}_\beta, \mathcal{S}_\mu, \mathcal{S}_\varphi, Q, w)$ with $\mathcal{Y}'$ being an BCDPM sequence of size $O(k^2)$. This requires $O(k^4 \log^2(mk) + k^2 \cdot k^2) = O(k^4 \log^2(mk))$ time, as the lengths of all strings in $\mathcal{S}_\beta \cup \mathcal{S}_\mu \cup \mathcal{S}_\varphi$ are $O(m)$.
- We make $O(k^2)$ calls to each of `BCDPM-Delete`, `BCDPM-Insert`, `BCDPM-Substitute`, `BCDPM-Commit`, and `BCDPM-Bleach`. These calls are processed in total time $O(k^2 \cdot k^2 \log k) = O(k^4 \log k)$ time, since the maintained sequence $\mathcal{Y}$ is of length polynomial in k throughout the course of the algorithm.
- We make $O(k^3)$ calls to `BCDPM-Set` and `BCDPM-Query`, with the total number of blank pieces over all calls to `BCDPM-Query` being $O(k^3)$. Let us denote the number number of calls to `BCDPM-Query` by ν, and the number of blank pieces in the i-th call to the `BCDPM-Query` by b_i. The time required for the $O(k^3)$ calls to `BCDPM-Set` and `BCDPM-Query` is then $O(k \cdot \left(k^3 + \sum_{i\in[1..\nu]} b_i\right)) = O(k^4)$.

In total, our call to `TrimmedPM` thus takes $O(k^4 \log^2(mk))$ time in the PILLAR model.

8 An $\widetilde{O}(n + k^{3.5} \cdot W^4 \cdot n/m)$-time Algorithm for Metric Integer Weights

In this section, we incorporate the results of Sections 6 and 9 into the approach of [CKW22] in order to prove the following result, which is the crucial step on the way to proving Main Theorem 2 for integer metric weights.

Lemma 8.44. *AlignedPeriodicMatches can be solved in $\widetilde{O}(k^{3.5}W^4)$ time in the* PILLAR *model if w is an integer metric weight function, with the output set $\mathrm{Occ}_k^w(P,T)$ represented as a collection of disjoint arithmetic progressions.*

Most statements in this section have direct unweighted counterparts in [CKW20, CKW22].

8.1 Toolbox

Lemma 8.1 (`FindAWitness(`k`, `Q`, `S`, `w`)`) [CKW20, see Lemma 6.5]. *Let k denote a positive integer, let S denote a string, let Q denote a primitive string that satisfies $|S| \geq (2k+1)|Q|$ or $|Q| \leq 3k+1$, and let $w : \bar{\Sigma}^2 \to [0\,..\,W]$ be a weight function.*

Then, we can compute a witness $Q^\infty[\,x\,..\,y\,)$ *such that* $\mathsf{ed}^w(S \to Q^\infty[\,x\,..\,y\,)) = \mathsf{ed}^w(S \to {}^*Q^*) \leq k$, *or report that* $\mathsf{ed}^w(S \to {}^*Q^*) > k$. *In the former case, the algorithm returns the breakpoint representation of a w-optimal alignment $\mathcal{A} : S \rightsquigarrow Q^\infty[\,x\,..\,y\,)$. The algorithm takes $\widetilde{O}(k^2 \cdot W)$ time in the* `PILLAR` *model.*

Proof. For a set $A \subseteq \mathbb{Z}$ and an integer $p > 0$, we define $A \bmod p := \{a \bmod p : a \in A\}$. We first compute a (short) interval J such that $\mathrm{Occ}_k(S, Q^\infty) \bmod |Q| \subseteq J \bmod |Q|$. If $|Q| \leq 3k+1$, then we simply set $J = [\,0\,..\,|Q|\,)$. Otherwise, we decompose the prefix $S[\,0\,..\,(2k+1)|Q|\,)$ into $2k+1$ fragments S_i of length $|Q|$. For each fragment S_i, we construct $\texttt{Rotations}(S_i, Q) = \{j \in \mathbb{Z} : Q = \mathrm{rot}^j(S_i)\}$. Next, we define an auxiliary set I as the union of intervals $[\,p\,..\,p+k\,]$ such that for at least $k+1$ fragments S_i of S, we have $[\,p\,..\,p+k\,] \cap \texttt{Rotations}(S_i, Q) \neq \emptyset$. Finally, we set $J \subseteq [\,0\,..\,2|Q|\,)$ to be a shortest interval satisfying $I \bmod |Q| \subseteq J \bmod |Q|$. Here, $J \bmod |Q|$ can be interpreted as a shortest cyclic interval (modulo $|Q|$) containing $I \bmod |Q|$.

Having computed the set J, we use Corollary 6.16 to determine $\mathsf{ed}^w_{\leq k}(S \to Q^\infty[\,x\,..\,y\,))$ for all positions $x \in J$ and $y \in J + [\,|S|-k\,..\,|S|+k\,]$. If all the values are ∞, we report that $\mathsf{ed}^w(S \to {}^*Q^*) > k$. Otherwise, we pick a fragment $Q^\infty[\,x\,..\,y\,)$ minimizing $\mathsf{ed}^w_{\leq k}(S \to Q^\infty[\,x\,..\,y\,))$, and we find an w-optimal alignment $\mathcal{A} : S \rightsquigarrow Q^\infty[\,x\,..\,y\,)$ using Lemma 6.13.

The correctness is based on the aforementioned characterization of J:

Claim 8.2. *The interval J satisfies* $\mathrm{Occ}^w_k(S, Q^\infty) \bmod |Q| \subseteq J \bmod |Q|$.

Proof. The claim trivially holds if $|Q| \leq 3k+1$, so we assume that $|Q| > 3k+1$. For every $i \in [\,0\,..\,2k\,]$, define $S_i := S[\,i\,|Q|\,..\,(i+1)\,|Q|\,)$. Consider an optimum alignment between S and its (k,w)-error occurrence $Q^\infty[\,x\,..\,y\,)$, and let $Q_i = Q^\infty[\,x_i\,..\,x_{i+1}\,)$ denote the fragment aligned to S_i. Consider the multi-set $R := \bigcup_i \texttt{Rotations}(S_i, Q)$. Next, consider the values $\delta_i := x_i - i|Q|$ for $i \in [\,0\,..\,2k\,]$. We have $\delta_0 = x$, and $\delta_{i+1} = \delta_i + |Q_i| - |S_i|$ for $i > 0$. Since

$$\sum_{i=0}^{2k} \big||Q_i| - |S_i|\big| \leq \sum_{i=0}^{2k} \mathsf{ed}^w(Q_i \to S_i) \leq k,$$

all values δ_i belong to an interval of the form $[\,p\,..\,p+k\,]$ for some integer p. Moreover, note that $Q_i = S_i$ holds for at least $k+1$ fragments Q_i; these fragments satisfy $Q = \mathrm{rot}^{\delta_i}(S_i)$ and thus contribute δ_i to R. We conclude that there is an interval $[\,p\,..\,p+k\,]$ containing x and at least $k+1$ elements of R. Consequently, we have $x \in I$. By definition of J, this yields $x \bmod |Q| \in J \bmod |Q|$. ◿

Now, let $Q^\infty[\,x\,..\,y\,)$ denote a witness that satisfies $\mathsf{ed}^w(S \to {}^*Q^*) = \mathsf{ed}^w(S \to Q^\infty[\,x\,..\,y\,))$. By Claim 8.2, there is a matching fragment $Q^\infty[\,x'\,..\,y'\,) = Q^\infty[\,x\,..\,y\,)$ starting at $x' \in J$. Thus, we may assume without loss of generality that $x \in J$. As we verify all possible starting positions in J using Corollary 6.16, we correctly compute the starting position x and the ending position y of the minimum-cost occurrence of S in Q^∞.

As for the running time, we prove the following characterization of J:

Claim 8.3. *The interval J satisfies $|J| \leq 3k+1$.*

Proof. The claim trivially holds if $|Q| \le 3k+1$, so we assume that $|Q| > 3k+1$. Recall that the multiset $R := \bigcup_{i=0}^{2k} \texttt{Rotations}(S_i, Q)$. As the string Q is primitive, R is the union of at most $2k+1$ infinite arithmetic progressions with difference $|Q|$. In particular, if $[\,p\,..\,p+k\,]$ and $[\,p'\,..\,p'+k\,]$ contain at least $k+1$ elements of R each, then $([\,p\,..\,p+k\,] \bmod |Q|) \cap ([\,p'\,..\,p'+k\,] \bmod |Q|) \neq \emptyset$, and thus $[\,p'\,..\,p'+k\,] \bmod |Q| \subseteq [\,p-k\,..\,p+2k\,] \bmod |Q|$. Since J is the union of such intervals $[\,p'\,..\,p'+k\,]$, we have $J \bmod |Q| \subseteq [\,p-k\,..\,p+2k\,] \bmod |Q|$. By definition of I, we conclude that $|I| \le |[\,p-k\,..\,p+2k\,]| = 3k+1$. ⌟

Now, observe that computing the multiset R (represented as the union of infinite arithmetic progressions modulo $|Q|$) takes $O(k)$ time in the PILLAR model; computing the sets I and J can be done in $O(k \log\log k)$ time by sorting R (restricted to $[\,0\,..\,|Q|\,)$) and a subsequent cyclic scan over R. Further, by Claim 8.3, we call Corollary 6.16 for a substring of Q^∞ of length $|S| + O(k)$. This costs $\widetilde{O}(k^2 W)$ time, just like the application of Lemma 6.13. ⌟

8.2 Locked Fragments and their Properties

For strings S and Q and a weight function w, we write $\mathsf{ed}^w(S \to Q^*) := \min\{\mathsf{ed}^w(S \to Q^\infty[\,0\,..\,j\,)) : j \in \mathbb{Z}_{\ge 0}\}$ to denote the minimum edit distance between a string S and any prefix of a string Q^∞. Further, we write $\mathsf{ed}^w(S \to {}^*Q^*) := \min\{\mathsf{ed}^w(S \to Q^\infty[\,i\,..\,j\,)) : i, j \in \mathbb{Z}_{\ge 0}, i \le j\}$ to denote the minimum edit distance between S and any substring of Q^∞, and we set $\mathsf{ed}^w(S \to {}^*Q) := \min\{\mathsf{ed}^w(S \to Q^\infty[\,i\,..\,j|Q|\,)) : i, j \in \mathbb{Z}_{\ge 0}, i \le j|Q|\}$.

Definition 8.4 [CKW20, see Definition 5.5]**.** *Let S denote a string, let Q denote a primitive string, and let w be a weight function. We say that a fragment L of S is* locked *(with respect to Q and w) if at least one of the following holds:*

- *For some integer α, we have $\mathsf{ed}^w(L \to {}^*Q^*) = \mathsf{ed}^w(L \to Q^\alpha)$.*
- *The fragment L is a suffix of S and $\mathsf{ed}^w(L \to {}^*Q^*) = \mathsf{ed}^w(L \to Q^*)$.*
- *The fragment L is a prefix of S and $\mathsf{ed}^w(L \to {}^*Q^*) = \mathsf{ed}^w(L \to {}^*Q)$.*
- *We have $L = S$.* ⌟

Lemma 8.5 [CKW20, see Lemma 5.6]**.** *Let S denote a string, let Q denote a primitive string, and let w be an integer weight function. There exist disjoint locked fragments $L_1, \ldots, L_\ell$ of S with $\mathsf{ed}^w(L_i \to {}^*Q^*) > 0$ such that*

$$\mathsf{ed}^w(S \to {}^*Q^*) = \sum_{i=1}^{\ell} \mathsf{ed}^w(L_i \to {}^*Q^*) \quad \text{and} \quad \sum_{i=1}^{\ell} |L_i| \le (5|Q|+1)\mathsf{ed}^w(S \to {}^*Q^*).$$

Proof. Let us choose integers $x \le y$ so that $\mathsf{ed}^w(S \to {}^*Q^*) = \mathsf{ed}^w(S \to Q^\infty[\,x\,..\,y\,))$. Without loss of generality, we may assume that $x \in [\,0\,..\,|Q|\,)$. If $y \le |Q|$, then $|S| \le |Q| + \mathsf{ed}^w(S \to {}^*Q^*)$; thus setting the whole string S as the only locked fragment satisfies the claimed conditions. Hence, we may assume that $y > |Q|$.

An arbitrary w-optimum alignment of S and $Q^\infty[\,x\,..\,y\,)$ yields a partition $S = S_0^{(0)} \cdots S_{s^{(0)}}^{(0)}$ with $s^{(0)} = \lfloor (y-1)/|Q| \rfloor$ such that $\mathsf{ed}^w(S \to Q^\infty[\,x\,..\,y\,)) = \sum_{i=0}^{s^{(0)}} \Delta_i^{(0)}$, where

$$\Delta_i^{(0)} = \begin{cases} \mathsf{ed}^w(S_0^{(0)} \to Q[\,x\,..\,|Q|\,)) & \text{if } i = 0, \\ \mathsf{ed}^w(S_i^{(0)} \to Q) & \text{if } 0 < i < s^{(0)}, \\ \mathsf{ed}^w(S_{s^{(0)}}^{(0)} \to Q[\,0\,..\,y - s^{(0)}|Q|\,)) & \text{if } i = s^{(0)}. \end{cases}$$

We start with this partition and then coarsen it by exhaustively applying the merging rules specified below, where each rule is applied only if the previous rules cannot be applied. In each case, we re-index the unchanged fragments $S_i^{(t)}$ to obtain a new partition $S = S_0^{(t+1)} \cdots S_{s^{(t+1)}}^{(t+1)}$ and re-index the corresponding values $\Delta_i^{(t)}$ accordingly. We say that a fragment $S_i^{(t)}$ is *interesting* if $i = 0$, $i = s^{(t)}$, $S_i^{(t)} \neq Q$, or $\Delta_i^{(t)} > 0$.

1. If subsequent fragments $S_i^{(t)}$ and $S_{i+1}^{(t)}$ are both interesting, then merge $S_i^{(t)}$ and $S_{i+1}^{(t)}$, obtaining $S_i^{(t+1)} := S_i^{(t)} S_{i+1}^{(t)}$ and $\Delta_i^{(t+1)} := \Delta_i^{(t)} + \Delta_{i+1}^{(t)}$.
2. If $0 < i < s^{(t)}$ and $\Delta_i^{(t)} > 0$, then merge the subsequent fragments $S_{i-1}^{(t)} = Q$, $S_i^{(t)}$, and $S_{i+1}^{(t)} = Q$, obtaining $S_{i-1}^{(t+1)} := S_{i-1}^{(t)} S_i^{(t)} S_{i+1}^{(t)}$, and set $\Delta_{i-1}^{(t+1)} := \Delta_i^{(t)} - 1$.
3. If $0 < i = s^{(t)}$ and $\Delta_i^{(t)} > 0$, then merge the subsequent fragments $S_{i-1}^{(t)} = Q$ and $S_i^{(t)}$, obtaining $S_{i-1}^{(t+1)} := S_{i-1}^{(t)} S_i^{(t)}$, and set $\Delta_{i-1}^{(t+1)} := \Delta_i^{(t)} - 1$.
4. If $0 = i < s^{(t)}$ and $\Delta_i^{(t)} > 0$, then merge the subsequent fragments $S_i^{(t)}$ and $S_{i+1}^{(t)} = Q$, obtaining $S_i^{(t+1)} := S_i^{(t)} S_{i+1}^{(t)}$, and set $\Delta_i^{(t+1)} := \Delta_i^{(t)} - 1$.

Let $S = S_0 \cdots S_s$ denote the obtained final partition. We select as locked fragments all the fragments S_i with $\mathsf{ed}^w(S_i \to {}^*Q^*) > 0$. Below, we show that this selection satisfies the desired properties. We start by proving that we indeed picked locked fragments.

Claim 8.6. *Each fragment $S_i^{(t)}$ of each partition $S = S_0^{(t)} \cdots S_{s^{(t)}}^{(t)}$ satisfies at least one of the following:*

- $\mathsf{ed}^w(S_i^{(t)} \to Q^\alpha) \le \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + \Delta_i^{(t)}$ *for some integer α;*
- $i = s^{(t)}$ *and* $\mathsf{ed}^w(S_i^{(t)} \to Q^*) \le \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + \Delta_i^{(t)}$;
- $i = 0$ *and* $\mathsf{ed}^w(S_i^{(t)} \to {}^*Q) \le \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + \Delta_i^{(t)}$;
- $i = 0 = s^{(t)}$.

Proof. We proceed by induction on t. The base case follows from the definition of the values $\Delta_i^{(0)}$.

As for the inductive step, we assume that the claim holds for all fragments $S_i^{(t)}$ and we prove that it holds for all fragments $S_i^{(t+1)}$. We consider several cases based on the merge rule applied.

1. For a type-1 merge of interesting fragments $S_i^{(t)}$ and $S_{i+1}^{(t)}$ into $S_i^{(t+1)}$, it suffices to prove that $S_i^{(t+1)}$ satisfies the claim.
   - If $0 < i < s^{(t+1)}$, then $\mathsf{ed}^w(S_i^{(t)} \to Q^\alpha) \le \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + \Delta_i^{(t)}$ and $\mathsf{ed}^w(S_{i+1}^{(t)} \to Q^{\alpha'}) \le \mathsf{ed}^w(S_{i+1}^{(t)} \to {}^*Q^*) + \Delta_{i+1}^{(t)}$ hold by the inductive assumption for some integers α, α'. Consequently,
   $$\begin{aligned}\mathsf{ed}^w(S_i^{(t+1)} \to Q^{\alpha+\alpha'}) &= \mathsf{ed}^w(S_i^{(t)} S_{i+1}^{(t)} \to Q^\alpha Q^{\alpha'}) \le \mathsf{ed}^w(S_i^{(t)} \to Q^\alpha) + \mathsf{ed}^w(S_{i+1}^{(t)} \to Q^{\alpha'}) \\ &\le \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + \Delta_i^{(t)} + \mathsf{ed}^w(S_{i+1}^{(t)} \to {}^*Q^*) + \Delta_{i+1}^{(t)} \le \mathsf{ed}^w(S_i^{(t+1)} \to {}^*Q^*) + \Delta_i^{(t+1)}.\end{aligned}$$
   - If $0 < i = s^{(t+1)}$, then $\mathsf{ed}^w(S_i^{(t)} \to Q^\alpha) \le \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + \Delta_i^{(t)}$ and $\mathsf{ed}^w(S_{i+1}^{(t)} \to Q^*) \le \mathsf{ed}^w(S_{i+1}^{(t)} \to {}^*Q^*) + \Delta_{i+1}^{(t)}$ hold by the inductive assumption for some integer α. Consequently,
   $$\begin{aligned}\mathsf{ed}^w(S_i^{(t+1)} \to Q^*) &= \mathsf{ed}^w(S_i^{(t)} S_{i+1}^{(t)} \to Q^*) \le \mathsf{ed}^w(S_i^{(t)} \to Q^\alpha) + \mathsf{ed}^w(S_{i+1}^{(t)} \to Q^*) \\ &\le \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + \Delta_i^{(t)} + \mathsf{ed}^w(S_{i+1}^{(t)} \to {}^*Q^*) + \Delta_{i+1}^{(t)} \le \mathsf{ed}^w(S_i^{(t+1)} \to {}^*Q^*) + \Delta_i^{(t+1)}.\end{aligned}$$
   - The analysis of the case that $0 = i < s^{(t+1)}$ is symmetric to that of the above case—this can be seen by reversing all strings in scope.
   - If $0 = i = s^{(t+1)}$, then the claim holds trivially.
2. For a type-2 merge of $S_{i-1}^{(t)}$, $S_i^{(t)}$, and $S_{i+1}^{(t)}$ into $S_{i-1}^{(t+1)}$, it suffices to prove that $S_{i-1}^{(t+1)}$ satisfies the claim.
   - If $\mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) \ge \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + 1$, we observe that $\mathsf{ed}^w(S_i^{(t)} \to Q^\alpha) \le \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + \Delta_i^{(t)}$ holds by the inductive assumption for some integer α. Consequently,
   $$\begin{aligned}\mathsf{ed}^w(S_{i-1}^{(t+1)} \to Q^{\alpha+2}) &= \mathsf{ed}^w(Q S_i^{(t)} Q \to Q Q^\alpha Q) \le \mathsf{ed}^w(S_i^{(t)} \to Q^\alpha) \\ &\le \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + \Delta_i^{(t)} \le \mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) - 1 + \Delta_i^{(t)} = \mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) + \Delta_{i-1}^{(t+1)}.\end{aligned}$$

- If $\mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) < \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*)+1$, then let $x' \le y'$ denote integers that satisfy $\mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) = \mathsf{ed}^w(S_{i-1}^{(t+1)} \to Q^\infty[\,x'\,..\,y'\,))$. This also yields integers x'', y'' with $x' \le x'' \le y'' \le y'$ such that

$$\begin{aligned}&\mathsf{ed}^w(S_{i-1}^{(t+1)} \to Q^\infty[\,x'\,..\,y'\,))\\ &\qquad= \mathsf{ed}^w(Q \to Q^\infty[\,x'\,..\,x''\,)) + \mathsf{ed}^w(S_i^{(t)} \to Q^\infty[\,x''\,..\,y''\,)) + \mathsf{ed}^w(Q \to Q^\infty[\,y''\,..\,y'\,)).\end{aligned}$$

  Due to

$$\begin{aligned}\mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) &\le \mathsf{ed}^w(S_i^{(t)} \to Q^\infty[\,x''\,..\,y''\,))\\ &\le \mathsf{ed}^w(S_{i-1}^{(t+1)} \to Q^\infty[\,x'\,..\,y'\,)) = \mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) < \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + 1,\end{aligned}$$

  we have $\mathsf{ed}^w(Q \to Q^\infty[\,x'\,..\,x''\,)) + \mathsf{ed}^w(Q \to Q^\infty[\,y''\,..\,y'\,)) < 1$. As the string Q is primitive, this means that x', x'', y'', y' are all multiples of $|Q|$. Consequently,

$$\begin{aligned}\mathsf{ed}^w(S_{i-1}^{(t+1)} \to Q^{(y'-x')/|Q|}) &= \mathsf{ed}^w(S_{i-1}^{(t+1)} \to Q^\infty[\,x'\,..\,y'\,))\\ &= \mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) \le \mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) + \Delta_{i-1}^{(t-1)}.\end{aligned}$$

- For a type-3 merge of $S_{i-1}^{(t)}$ and $S_i^{(t)}$ into $S_{i-1}^{(t+1)}$, it suffices to prove that $S_{i-1}^{(t+1)}$ satisfies the claim.
  - If $\mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) \ge \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*)+1$, we observe that $\mathsf{ed}^w(S_i^{(t)} \to Q^*) \le \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*)+\Delta_i^{(t)}$ holds by the inductive assumption. Consequently,

$$\begin{aligned}\mathsf{ed}^w(S_{i-1}^{(t+1)} \to Q^*) &= \mathsf{ed}^w(QS_i^{(t)} \to Q^*) \le \mathsf{ed}^w(S_i^{(t)} \to Q^*)\\ &\le \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + \Delta_i^{(t)} \le \mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) - 1 + \Delta_i^{(t)} = \mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) + \Delta_{i-1}^{(t+1)}.\end{aligned}$$

  - If $\mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) < \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*)+1$, then let $x' \le y'$ denote integers that satisfy $\mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) = \mathsf{ed}^w(S_{i-1}^{(t+1)} \to Q^\infty[\,x'\,..\,y'\,))$. This also yields an integer x'' with $x' \le x'' \le y'$ such that

$$\mathsf{ed}^w(S_{i-1}^{(t+1)} \to Q^\infty[\,x'\,..\,y'\,)) = \mathsf{ed}^w(Q \to Q^\infty[\,x'\,..\,x''\,)) + \mathsf{ed}^w(S_i^{(t)} \to Q^\infty[\,x''\,..\,y'\,)).$$

    Due to

$$\begin{aligned}\mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) \le \mathsf{ed}^w(S_i^{(t)} \to Q^\infty[\,x''\,..\,y'\,)) &\le \mathsf{ed}^w(S_{i-1}^{(t+1)} \to Q^\infty[\,x'\,..\,y'\,))\\ &= \mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) < \mathsf{ed}^w(S_i^{(t)} \to {}^*Q^*) + 1,\end{aligned}$$

    we have $\mathsf{ed}^w(Q \to Q^\infty[\,x'\,..\,x''\,)) < 1$. As the string Q is primitive, this means that x', x'' are both multiples of $|Q|$. Consequently,

$$\begin{aligned}\mathsf{ed}^w(S_{i-1}^{(t+1)} \to Q^*) &= \mathsf{ed}^w(S_{i-1}^{(t+1)} \to Q^\infty[\,x'\,..\,y'\,)) = \mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*)\\ &\le \mathsf{ed}^w(S_{i-1}^{(t+1)} \to {}^*Q^*) + \Delta_{i-1}^{(t-1)}.\end{aligned}$$

- The analysis of type-4 merges is symmetrical to that of type-3 merges—this can be seen by reversing all strings in scope.

  This completes the proof of the inductive step. ⌟

Observe that if no merge rule can be applied to a partition $S = S_0^{(t)} \cdots S_{s^{(t)}}^{(t)}$, then $s^{(t)} = 0$ or $\Delta_0^{(t)} = \cdots = \Delta_{s^{(t)}}^{(t)} = 0$. Consequently, Claim 8.6 implies that all fragments S_i in the final partition $S = S_0 \cdots S_s$ are locked.

⌜ Claim 8.7. *For each partition $S = S_0^{(t)} \cdots S_{s^{(t)}}^{(t)}$, the total length $\lambda^{(t)}$ of interesting fragments satisfies*

$$\lambda^{(t)} + 2|Q| \sum_{i=0}^{s^{(t)}} \Delta_i^{(t)} \le (5|Q|+1)\mathsf{ed}^w(S \to {}^*Q^*).$$

Proof. We proceed by induction on t. In the base case of $t = 0$, each interesting fragment other than $S_0^{(0)}$ and $S_{s^{(0)}}^{(0)}$ satisfies $\Delta_i^{(0)} > 0$. Hence, the number of interesting fragments is at most $2 + \sum_{i=0}^{s^{(0)}} \Delta_i^{(0)} = 2 + \mathsf{ed}^w(S \to {}^*Q^*)$. Moreover, the length of each fragment $S_i^{(0)}$ does not exceed $|Q| + \Delta_i^{(0)}$. Consequently,

$$\lambda^{(0)} + 2|Q| \sum_{i=0}^{s^{(0)}} \Delta_i^{(0)} \le (2 + \mathsf{ed}^w(S \to {}^*Q^*))|Q| + (2|Q| + 1) \sum_{i=0}^{s^{(0)}} \Delta_i^{(0)} \le (5|Q| + 1)\, \mathsf{ed}^w(S \to {}^*Q^*).$$

This completes the proof in the base case.

As for the inductive step, it suffices to prove that $\lambda^{(t+1)} + 2|Q| \sum_{i=0}^{s^{(t+1)}} \Delta_i^{(t+1)} \le \lambda^{(t)} + 2|Q| \sum_{i=0}^{s^{(t)}} \Delta_i^{(t)}$:

- For a type-1 merge (where we merge two interesting fragments), we have

$$\lambda^{(t+1)} + 2|Q| \sum_{i=0}^{s^{(t+1)}} \Delta_i^{(t+1)} = \lambda^{(t)} + 2|Q| \sum_{i=0}^{s^{(t)}} \Delta_i^{(t)}.$$

- For a type-2, type-3, or type-4 merge (where we merge a fragment with its one or two non-interesting neighbors), we have

$$\lambda^{(t+1)} + 2|Q| \sum_{i=0}^{s^{(t+1)}} \Delta_i^{(t+1)} \le \lambda^{(t)} + 2|Q| + 2|Q| \sum_{i=0}^{s^{(t+1)}} \Delta_i^{(t+1)} = \lambda^{(t)} + 2|Q| \sum_{i=0}^{s^{(t)}} \Delta_i^{(t)}.$$

Overall, we obtain the claimed bound. ⌟

We conclude that the total length of interesting fragments S_i does not exceed $(5|Q| + 1)\mathsf{ed}^w(S \to {}^*Q^*)$.

Claim 8.8. *We have* $\mathsf{ed}^w(S \to {}^*Q^*) = \sum_{i=0}^{s} \mathsf{ed}^w(S_i \to {}^*Q^*)$.

Proof. The claim is immediate if $s = 0$; hence, assume that $s \ge 1$. Observe that the inequality $\sum_{i=0}^{s} \mathsf{ed}^w(S_i \to {}^*Q^*) \le \mathsf{ed}^w(S \to {}^*Q^*)$ easily follows from disjointness of fragments S_i; thus, we focus on proving $\mathsf{ed}^w(S \to {}^*Q^*) \le \sum_{i=0}^{s} \mathsf{ed}^w(S_i \to {}^*Q^*)$.

For $0 \le i \le s$, let Q_i denote a substring of Q^∞ that satisfies $\mathsf{ed}^w(S_i \to {}^*Q^*) = \mathsf{ed}^w(S_i \to Q_i)$. Since each S_i is locked (by Claim 8.6), we may assume that for $0 < i < s$ the substring Q_i is a power of Q, the substring Q_s is a prefix of a power of Q, and the substring Q_0 is a suffix of a power of Q. Consequently, $Q_0 \cdots Q_s$ is a substring of Q^∞, and we have

$$\mathsf{ed}^w(S \to {}^*Q^*) \le \mathsf{ed}^w(S_0 \cdots S_s \to Q_0 \cdots Q_s) \le \sum_{i=0}^{s} \mathsf{ed}^w(S_i \to Q_i) = \sum_{i=0}^{s} \mathsf{ed}^w(S_i \to {}^*Q^*),$$

thus completing the proof. ⌟

The locked fragments created satisfy $\mathsf{ed}^w(S \to {}^*Q^*) = \sum_{i=1}^{\ell} \mathsf{ed}^w(L_i \to {}^*Q^*)$ due to Claim 8.8. Moreover, since $\mathsf{ed}^w(S_i \to {}^*Q^*) > 0$ holds only for interesting fragments, Claim 8.7 yields $\sum_{i=1}^{\ell} |L_i| \le (5|Q| + 1)\, \mathsf{ed}^w(S \to {}^*Q^*)$, completing the proof. ⌟

Definition 8.9 [CKW20, see Definition 5.10]**.** *Let S denote a string, let Q denote a primitive string, let $h \ge 0$ denote an integer, and let w be a weight function. We say that a prefix L of S is* h-locked *(with respect to Q) if at least one of the following holds:*

- *For every $p \in [0\,..\,|Q|)$, if $\mathsf{ed}^w(L \to \mathrm{rot}^p(Q)^*) \le h$, then $\mathsf{ed}^w(L \to \mathrm{rot}^p(Q)^*) = \mathsf{ed}^w(L \to Q^\infty[-p\,..\,j|Q|))$ for some $j \in \mathbb{Z}$.*
- *We have $L = S$.* ⌟

Lemma 8.10 [CKW20, see Lemma 5.11]. *Let S denote a string, let Q denote a primitive string, let $h \geq 0$ be an integer, and w be an integer weight function. There are disjoint locked fragments $L_1, \ldots, L_\ell \leq S$, such that L_1 is a k-locked prefix of S, L_ℓ is a suffix of S, $\mathsf{ed}^w(L_i \to {}^*Q^*) > 0$ for $1 < i < \ell$,*

$$\mathsf{ed}^w(S \to {}^*Q^*) = \sum_{i=1}^{\ell} \mathsf{ed}^w(L_i \to {}^*Q^*), \quad \text{and} \quad \sum_{i=1}^{\ell} |L_i| \leq (5|Q|+1)\mathsf{ed}^w(S \to {}^*Q^*) + 2(h+1)|Q|.$$

Proof. We proceed as in the proof of Lemma 8.5 except that $\Delta_0^{(0)}$ is artificially increased by $h+1$, the prefix S_0 in the final partition is included as L_1 among the locked fragments even if $\mathsf{ed}^w(S_0 \to {}^*Q^*) = 0$, and the suffix S_s is included as L_ℓ among the locked fragments even if $\mathsf{ed}^w(S_s \to {}^*Q^*) = 0$.

It is easy to see that Claims 8.6 and 8.8 remain satisfied, whereas the upper bound in Claim 8.7 is increased by $2(h+1)|Q|$. We only need to prove that S_0 is a h-locked prefix of S. For this, we prove the following claim using induction.

Claim 8.11. *For each partition $S = S_0^{(t)} \cdots S_{s^{(t)}}^{(t)}$, at least one of the following holds.*

- *For every $p \in [0\,..\,|Q|)$, if $\mathsf{ed}^w(S_0^{(t)} \to \mathrm{rot}^p(Q)^*) \leq h - \Delta_0^{(t)}$, then $\mathsf{ed}^w(S_0^{(t)} \to Q^\infty[\,|Q|-p\,..\,j|Q|\,)) \leq \mathsf{ed}^w(S_0^{(t)} \to \mathrm{rot}^p(Q)^*) + \Delta_0^{(t)}$ holds for some integer j.*
- *We have $S_0^{(t)} = S$.*

Proof. We proceed by induction on t. In the base case of $t = 0$, the claim holds trivially since $\mathsf{ed}^w(S_0^{(0)} \to \mathrm{rot}^p(Q)^*) \geq 0 > h - \Delta_0^{(0)}$ holds for every p due to $\Delta_0^{(0)} \geq h+1$.

As for the induction step, we assume that the claim holds for $S_0^{(t)}$ and we prove that it holds for $S_0^{(t+1)}$. The claim holds trivially if the merge rule applied did not affect $S_0^{(t)}$. Given that $S_0^{(t)}$ is interesting by definition, the merges that might affect $S_0^{(t)}$ are of type 1 (if $S_1^{(t)}$ is interesting) or 4 (otherwise).

1 Consider a type-1 merge of $S_0^{(t)}$ and $S_1^{(t)}$. If $s^{(t)} = 1$, then $S_0^{(t+1)} = S$ satisfies the claim trivially. Hence, we may assume that $1 < s^{(t)}$ so that Claim 8.6 yields $\mathsf{ed}^w(S_1^{(t)} \to Q^\alpha) \leq \mathsf{ed}^w(S_1^{(t)} \to {}^*Q^*) + \Delta_1^{(t)}$ for some integer α. Let us fix $p \in [0\,..\,|Q|)$ with $\mathsf{ed}^w(S_0^{(t+1)} \to \mathrm{rot}^p(Q)^*) \leq h - \Delta_0^{(t+1)}$. Due to $\Delta_0^{(t+1)} \geq \Delta_0^{(t)}$, this yields $\mathsf{ed}^w(S_0^{(t)} \to \mathrm{rot}^p(Q)^*) \leq h - \Delta_0^{(t)}$, so the inductive assumption implies $\mathsf{ed}^w(S_0^{(t)} \to Q^\infty[\,|Q|-p\,..\,j|Q|\,)) \leq \mathsf{ed}^w(S_0^{(t)} \to \mathrm{rot}^p(Q)^*) + \Delta_0^{(t)}$ for some integer j. Consequently,

$$\begin{aligned}\mathsf{ed}^w(S_0^{(t+1)} \to Q^\infty[\,|Q|-p\,..\,(j+\alpha)|Q|\,)) &= \mathsf{ed}^w(S_0^{(t)}S_1^{(t)} \to Q^\infty[\,|Q|-p\,..\,j|Q|\,)Q^\alpha)\\ &\leq \mathsf{ed}^w(S_0^{(t)} \to Q^\infty[\,|Q|-p\,..\,j|Q|\,)) + \mathsf{ed}^w(S_1^{(t)} \to Q^\alpha)\\ &\leq \mathsf{ed}^w(S_0^{(t)} \to \mathrm{rot}^p(Q)^*) + \Delta_0^{(t)} + \mathsf{ed}^w(S_1^{(t)} \to {}^*Q^*) + \Delta_1^{(t)} \leq \mathsf{ed}^w(S_0^{(t+1)} \to \mathrm{rot}^p(Q)^*) + \Delta_0^{(t+1)}.\end{aligned}$$

2 Consider a type-4 merge of $S_0^{(t)}$ and $S_1^{(t)}$. Let us fix $p \in [0\,..\,|Q|)$ with $\mathsf{ed}^w(S_0^{(t+1)} \to \mathrm{rot}^p(Q)^*) \leq h - \Delta_0^{(t+1)}$.

- If $\mathsf{ed}^w(S_0^{(t+1)} \to \mathrm{rot}^p(Q)^*) \geq \mathsf{ed}^w(S_0^{(t)} \to \mathrm{rot}^p(Q)^*) + 1$, then

$$\mathsf{ed}^w(S_0^{(t)} \to \mathrm{rot}^p(Q)^*) \leq \mathsf{ed}^w(S_0^{(t+1)} \to \mathrm{rot}^p(Q)^*) - 1 \leq h - \Delta_0^{(t+1)} - 1 = h - \Delta_0^{(t)}.$$

Hence, the inductive assumption implies $\mathsf{ed}^w(S_0^{(t)} \to Q^\infty[\,|Q|-p\,..\,j|Q|\,)) \leq \mathsf{ed}^w(S_0^{(t)} \to \mathrm{rot}^p(Q)^*) + \Delta_0^{(t)}$ for some integer j. Consequently,

$$\begin{aligned}\mathsf{ed}^w(S_0^{(t+1)} \to Q^\infty[\,|Q|-p\,..\,(j+1)|Q|\,)) &= \mathsf{ed}^w(S_0^{(t)}Q \to Q^\infty[\,|Q|-p\,..\,j|Q|\,)Q)\\ &\leq \mathsf{ed}^w(S_0^{(t)} \to Q^\infty[\,|Q|-p\,..\,j|Q|\,)) \leq \mathsf{ed}^w(S_0^{(t)} \to \mathrm{rot}^p(Q)^*) + \Delta_0^{(t)}\\ &\leq \mathsf{ed}^w(S_0^{(t+1)} \to \mathrm{rot}^p(Q)^*) - 1 + \Delta_0^{(t)} = \mathsf{ed}^w(S_0^{(t+1)} \to \mathrm{rot}^p(Q)^*) + \Delta_0^{(t+1)}.\end{aligned}$$

- If $\mathsf{ed}^w(S_0^{(t+1)} \to \mathrm{rot}^p(Q)^*) < \mathsf{ed}^w(S_0^{(t)} \to \mathrm{rot}^p(Q)^*) + 1$, then let y' denote an arbitrary integer that satisfies $\mathsf{ed}^w(S_0^{(t+1)} \to \mathrm{rot}^p(Q)^*) = \mathsf{ed}^w(S_0^{(t+1)} \to Q^\infty[\,|Q| - p\,..\,y'\,))$. This also yields an integer y'' with $|Q| - p \le y'' \le y'$ such that

$$\mathsf{ed}^w(S_0^{(t+1)} \to Q^\infty[\,|Q| - p\,..\,y'\,)) = \mathsf{ed}^w(S_0^{(t)} \to Q^\infty[\,|Q| - p\,..\,y''\,)) + \mathsf{ed}^w(Q \to Q^\infty[\,y''\,..\,y'\,)).$$

Due to

$$\begin{aligned}\mathsf{ed}^w(S_0^{(t)} \to \mathrm{rot}^p(Q)^*) \le \mathsf{ed}^w(S_0^{(t)} \to Q^\infty[\,|Q| - p\,..\,y''\,)) &\le \mathsf{ed}^w(S_0^{(t+1)} \to Q^\infty[\,|Q| - p\,..\,y'\,)) \\ &= \mathsf{ed}^w(S_0^{(t+1)} \to \mathrm{rot}^p(Q)^*) < \mathsf{ed}^w(S_0^{(t)} \to \mathrm{rot}^p(Q)^*) + 1,\end{aligned}$$

we have $\mathsf{ed}^w(Q \to Q^\infty[\,y''\,..\,y''\,)) < 1$. As the string Q is primitive, this means that y'', y' are both multiples of $|Q|$. Consequently,

$$\mathsf{ed}^w(S_0^{(t+1)} \to Q^\infty[\,|Q| - p\,..\,y'\,)) = \mathsf{ed}^w(S_0^{(t+1)} \to \mathrm{rot}^p(Q)^*) \le \mathsf{ed}^w(S_0^{(t+1)} \to \mathrm{rot}^p(Q)^*) + \Delta_0^{(t+1)}.$$

This completes the proof of the inductive step. ⌟

Given that the final partition $S = S_0^{(t)} \cdots S_{s^{(t)}}^{(t)}$ satisfies $s^{(t)} = 0$ or $\Delta_0^{(t)} = 0$, we conclude that S_0 is indeed h-locked. ⌟

Lemma 8.12 (`Locked(S, Q, d, h,w)`, see [CKW20, Lemma 6.9])**.** *Let S denote a string, let Q denote a primitive string, let $w : \bar{\Sigma}^2 \to [\,0\,..\,W\,]$ be an integer weight function, let d denote a positive integer such that $\mathsf{ed}^w(S \to {}^*Q^*) \le d$ and $|S| \ge (2d+1)|Q|$, and let $h \in \mathbb{Z}_{\ge 0}$.*

Then, there is an algorithm that computes disjoint locked fragments $L_1, \ldots, L_\ell \le S$ such that

- *$S = L_1 \cdot \bigodot_{i=1}^{\ell-1}(Q^{\alpha_i} L_{i+1})$ for positive integers $\alpha_1, \ldots, \alpha_{\ell-1}$;*
- *L_1 is an h-locked prefix of S and L_ℓ is a suffix of S;*
- *$\mathsf{ed}^w(S \to {}^*Q^*) = \sum_{i=1}^{\ell} \mathsf{ed}^w(L_i \to {}^*Q^*)$ and $\mathsf{ed}^w(L_i \to {}^*Q^*) > 0$ for $i \in (\,1\,..\,\ell\,)$; and*
- *$\displaystyle\sum_{i=1}^{\ell} |L_i| \le (5|Q| + 1)\mathsf{ed}^w(S \to {}^*Q^*) + 2(h+1)|Q|$.*

The algorithm takes $\widetilde{O}(d^2 \cdot W + h)$ time in the `PILLAR` *model.*

Proof. We implement the process from the proof of Lemma 8.10.

First, we use Fact 7.11 to construct a w-optimal alignment $\mathcal{A} : S \rightsquigarrow Q^\infty[\,x\,..\,y\,)$ between S and a substring of Q^∞. Then, based on the alignment $\mathcal{A}$, we construct a decomposition $S = S_0^{(0)} \cdots S_{s^{(0)}}^{(0)}$ such that $S_i^{(0)}$ is aligned against

$$Q_i^{(0)} := Q^\infty[\,\max(x, |Q|(\lceil x/|Q|\rceil + i - 1))\,..\,\min(|Q|(\lceil x/|Q|\rceil + i), y)\,)$$

by alignment $\mathcal{A}$, and a sequence $\Delta_i^{(0)}$ such that we have $\Delta_i^{(0)} = \mathsf{ed}^w(S_i^{(0)} \to Q_i^{(0)})$ for $i > 0$ and $\Delta_0^{(0)} = \mathsf{ed}^w(S_0^{(0)} \to Q_0^{(0)}) + h + 1$. Since this sequence might be long, we only generate *interesting* fragments $S_i^{(0)}$ and store them, along with the values $\Delta_i^{(0)}$, in a queue F in left-to-right order. (Recall that $S_i^{(t)}$ is interesting if $i = 0$, $i = s^{(t)}$, $S_i^{(t)} \ne Q$, or $\Delta_i^{(t)} > 0$.)

The process of constructing the interesting fragments $S_i^{(0)}$ is somewhat tedious. We maintain a fragment $Q^\infty[\,\ell_Q\,..\,r_Q\,)$, interpreted as $Q_i^{(0)}$ for increasing values of i, a fragment $S[\,\ell_S\,..\,r_S\,)$, interpreted as a candidate for $S_i^{(0)}$, and an integer Δ, interpreted as $\Delta_i^{(0)}$. They are initialized to $Q^\infty[\,x\,..\,|Q|\lceil x/|Q|\rceil\,)$, $S[\,0\,..\,Q|\lceil x/|Q|\rceil - x\,)$, and $k+1$, respectively.

Next, we process pairs (s, q) corresponding to subsequent errors in the alignment A. The interpretation of the j-th pair (s, q) is that $S[\,0\,..\,s\,)$ is aligned with $Q^\infty[\,x\,..\,x+q\,)$ with j errors so that the j-th error is a substitution of $S[\,s\,]$ into $Q^\infty[\,x+q\,]$, and insertion of $Q^\infty[\,x+q\,]$, or a deletion of $S[\,s\,]$.

The first step of processing (s, q) is only performed if $Q^\infty[\, x \,..\, x+q \,)$ is not (yet) contained in $Q^\infty[\, \ell_Q \,..\, r_Q \,)$. If this is not the case, then we push $S[\, \ell_S \,..\, r_S \,)$ with budget Δ to the queue F of interesting fragments, and we update the maintained data: The fragment $Q^\infty[\, \ell_Q \,..\, r_Q \,)$ is set to be the fragment of Q^∞ matching Q and containing $Q^\infty[\, x+q \,]$; between the previous and the current value of $Q^\infty[\, \ell_Q \,..\, r_Q \,)$, there are zero or more copies of Q aligned in A without error. Hence, we skip the same number of copies of Q in S (these are the uninteresting fragments $S_i^{(0)}$) and set $S[\, \ell_S \,..\, r_S \,)$ to be the subsequent fragment of length $|Q|$. Finally, the budget Δ is reset to 0.

In the second step, we update r_S according to the type of the currently processed error: We increment r_S in case of deletion of $S[\, s \,]$ and we decrement r_S in case of insertion of $Q^\infty[\, x+q \,]$. This way, we guarantee that $|S(\, s \,..\, r_S \,)| = |Q^\infty(\, x+q \,..\, r_Q \,)|$, and that A aligns $S[\, \ell_S \,..\, r_S \,)$ with $Q^\infty[\, \ell_Q \,..\, r_Q \,)$ provided that we have already processed all errors involving $Q^\infty[\, \ell_Q \,..\, r_Q \,)$. Additionally, we increase Δ to acknowledge the currently processed error between $S[\, \ell_S \,..\, r_S \,)$ and $Q^\infty[\, \ell_Q \,..\, r_Q \,)$.

In a similar way, we process $(s, q) = (|S|, y)$, interpreting it as extra substitution. This time, however, we do not increase Δ (because this is a not a real error). Finally, we push $S[\, \ell_S \,..\, |S| \,) = S^{(0)}_{s^{(0)}}$ with budget Δ to the queue F.

In the second phase of the algorithm, we transform the decomposition $S = S^{(0)}_0 \cdots S^{(0)}_{s^{(0)}}$ and the sequence $\Delta^{(0)}_0 \cdots \Delta^{(0)}_{s^{(0)}}$ using the four types of merge operations described in the proof of Lemma 8.5.

We maintain an invariant that a stack L contains already processed interesting fragments, all with budget equal to 0, in left-to-right order (so that the top of L represents the rightmost one), while F contains fragments that have not been processed yet (and may have positive budgets) also in the left-to-right order (so that the front of F represents the leftmost one). Additionally, the currently processed fragment $S[\, \ell \,..\, r \,)$ is guaranteed to be to the right of all fragments in L and to the left of all fragments in F. The fragments in L, the fragment $S[\, \ell \,..\, r \,)$, and the fragments in F form the sequence of all interesting fragments in the current decomposition $S = S^{(t)}_0 \cdots S^{(t)}_{s^{(t)}}$.

In each iteration of the main loop, we pop the front fragment $S[\, \ell \,..\, r \,)$ with budget Δ from the queue F and exhaustively perform merge operations involving it: We first try applying a type-1 merge with the fragment to the left (which must be on the top of L). If this is not possible, we type applying a type-1 merge with the fragment to the right (which must be on the front of F). If also this is not possible, then $S[\, \ell \,..\, r \,)$ is surrounded by uninteresting fragments. In this case, we perform a type-2, type-3, or 4 merge provided that $\Delta > 0$. Otherwise, we push $S[\, \ell \,..\, r \,)$ to L and proceed to the next iteration.

Finally, the algorithm returns the sequence of (locked) fragments represented in the stack L.

The correctness of the algorithm follows from Lemma 8.12; no deep insight is needed to prove that our implementation indeed follows the procedure described in the proof of Lemma 8.5 and extended in the proof of Lemma 8.12.

As the alignment $\mathcal{A}$ is of size $|\mathcal{A}| \le d$, initially, the are $O(d)$ interesting fragments with total budget $O(d+h)$. Each subsequent iteration decreases the number of interesting locked fragments or their total budget, so there are at most $O(d+k)$ iterations in total. Overall the algorithm runs in $O(d+k)$ time in the PILLAR model, on top of the cost of constructing $\mathcal{A}$, which is $\widetilde{O}(d^2 W)$ due to Fact 7.11. ◢

Corollary 8.13. *Consider an instance of* AlignedPeriodicMatches*, where w is an integer metric weight function. The values $d_P^w := \mathsf{ed}^w(P \to {}^*Q^*)$ and $d_T^w := \mathsf{ed}^w(T \to {}^*Q^*)$ satisfy $d_w^P + d_w^T = O(kW)$ and can be computed in $\widetilde{O}(k^2W^3)$ time. Moreover, for any parameter $\partial_P \in \mathbb{Z}_{>0}$, the sets $\mathcal{L}^P := \mathtt{Locked}(P, Q, d_P^w, \partial_P, w)$ and $\mathcal{L}^T := \mathtt{Locked}(T, Q, d_T^w, 0, w)$ can be constructed in $\widetilde{O}(k^2W^3 + \partial_P)$ time.*

Proof. Since the weights are upper-bounded by w, we have $d_P^w = \mathsf{ed}^w(P \to {}^*Q^*) \le W \cdot \mathsf{ed}^w(P \to {}^*Q^*) = W \cdot d_P = O(kW)$ and analogously for d_T^w. The values d_P^w and d_T^w can be computed in $\widetilde{O}(k^2W^3)$ time using Corollary 6.17. The cost of using Lemma 8.12 is $\widetilde{O}(k^2W^3 + \partial_P)$. ◢

8.3 Analyzing the Text Using Locked Fragments

In what follows, we adapt the marking scheme of [CKW22] to AlignedPeriodicMatches.

Definition 8.14 [CKW22, see Definition 5.5]. *For a weight function $w : \bar{\Sigma}^2 \to [0\,..\,W]$, a text T, a pattern P, a primitive string Q with $\mathsf{ed}^w(P \to {}^*Q^*) = d_P^w$ and $\mathsf{ed}^w(T \to {}^*Q^*) = d_T^w$, and corresponding sets of locked fragments $\mathcal{L}^P := \mathtt{Locked}(P, Q, d_P^w, \partial_P, w)$ and $\mathcal{L}^T := \mathtt{Locked}(T, Q, d_T^w, 0, w)$, write* mk *for the function that maps an integer v to a (weighted) number of locked fragments in $\mathcal{L}^P$ that (almost) overlap locked fragments in $\mathcal{L}^T$ when aligning P to position v.*

Formally, we first define a function $\mathrm{mk} : \mathbb{Z} \times \mathbb{Z}_{\geq 0} \times \mathcal{L}^P \times \mathcal{L}^T \to \mathbb{Z}_{\geq 0}$ *by*

$$\mathrm{mk}(v, \hat{\kappa}, L^P = P[\,\ell_P\,..\,r_P\,), L^T = T[\,\ell_T\,..\,r_T\,)) := \begin{cases} \mathsf{ed}^w(L^T \to {}^*Q^*), & \text{if } \ell_P = 0 \text{ and } v \in (\,\ell_T - \hat{\kappa} - r_P\,..\,r_T + \hat{\kappa} - \ell_P\,); \\ \min\{\mathsf{ed}^w(L^T \to {}^*Q^*), \mathsf{ed}^w(L^P \to {}^*Q^*)\}, & \text{if } \ell_P \neq 0 \text{ and } v \in (\,\ell_T - \hat{\kappa} - r_P\,..\,r_T + \hat{\kappa} - \ell_P\,); \\ 0, & \text{otherwise.} \end{cases}$$

Now, set

$$\mathrm{mk}(v, \hat{\kappa}, \mathcal{L}^P, \mathcal{L}^T) := \sum_{L^P \in \mathcal{L}^P} \sum_{L^T \in \mathcal{L}^T} \mathrm{mk}(v, \hat{\kappa}, L^P, L^T).$$

When $\hat{\kappa}$, $\mathcal{L}^P$, and $\mathcal{L}^T$ are clear from context we may just write $\mathrm{mk}(v)$ for $\mathrm{mk}(v, \hat{\kappa}, \mathcal{L}^P, \mathcal{L}^T)$.

We continue with a set of useful observations about our marking scheme. Fix an instance of AlignedPeriodicMatches and sets of locked fragments $\mathcal{L}^P := \mathtt{Locked}(P, Q, d_P^w, \partial_P, w)$ and $\mathcal{L}^T := \mathtt{Locked}(T, Q, d_T^w, 0, w)$, with $\partial_P = 2kW$. Let $\mu := \max(d_P^w, d_T^w, \partial_P, 1) = O(kW)$.

Lemma 8.15 [CKW22, see Lemma 5.6]. *For every $\hat{\kappa} \in \mathbb{Z}_{\geq 0}$, we have*

$$\sum_{v \in \mathbb{Z}} \mathrm{mk}(v, \hat{\kappa}, \mathcal{L}^P, \mathcal{L}^T) = O(W^2 \cdot k^2 \cdot (\hat{\kappa} + |Q|)).$$

Proof. Fix a locked fragment $L^P = P[\,\ell_P\,..\,r_P\,) \in \mathcal{L}^P$ and a locked fragment $L^T = P[\,\ell_T\,..\,r_T\,) \in \mathcal{L}^T$. If $\ell_P \neq 0$, then the definition of mk yields

$$\begin{aligned} \sum_{v \in \mathbb{Z}} \mathrm{mk}(v, \hat{\kappa}, L^P, L^T) &\leq \sum_{v \in (\,\ell_T - \hat{\kappa} - r_P\,..\,r_T + \hat{\kappa} - \ell_P\,)} \min\{\mathsf{ed}^w(L^T \to {}^*Q^*), \mathsf{ed}^w(L^P \to {}^*Q^*)\} \\ &\leq \min\{\mathsf{ed}^w(L^T \to {}^*Q^*), \mathsf{ed}^w(L^P \to {}^*Q^*)\}(|L^T| + |L^P| + 2\hat{\kappa}) \\ &\leq \mathsf{ed}^w(L^T \to {}^*Q^*)(|L^P| + 2\hat{\kappa}) + \mathsf{ed}^w(L^P \to {}^*Q^*)|L^T|. \end{aligned}$$

Similarly, if $\ell_P = 0$, then the definition of mk yields

$$\begin{aligned} \sum_{v \in \mathbb{Z}} \mathrm{mk}(v, \hat{\kappa}, L^P, L^T) &\leq \sum_{v \in (\,\ell_T - \hat{\kappa} - r_P\,..\,r_T + \hat{\kappa} - \ell_P\,)} \mathsf{ed}^w(L^T \to {}^*Q^*) \\ &\leq \mathsf{ed}^w(L^T \to {}^*Q^*)(|L^T| + |L^P| + 2\hat{\kappa}) \\ &\leq \mathsf{ed}^w(L^T \to {}^*Q^*)(|L^P| + 2\hat{\kappa}) + d_T^w|L^T|. \end{aligned}$$

Overall, we have

$$\begin{aligned}\sum_{v\in\mathbb{Z}} \mathrm{mk}(v,\hat\kappa,\mathcal{L}^P,\mathcal{L}^T) &\le \sum_{L^P\in\mathcal{L}^P}\sum_{L^T\in\mathcal{L}^T} (\mathsf{ed}^w(L^T\to {}^*Q^*)(|L^P|+2\hat\kappa)+\mathsf{ed}^w(L^P\to {}^*Q^*)|L^T|) + d_T^w\sum_{L^T\in\mathcal{L}^T}|L^T|\\ &\le d_T^w\sum_{L^P\in\mathcal{L}^P}(|L^P|+2\hat\kappa)+(d_P^w+d_T^w)\sum_{L^T\in\mathcal{L}^T}|L^T|\\ &\le d_T^w(d_P^w+2)2\hat\kappa + d_T^w\sum_{L^P\in\mathcal{L}^P}|L^P|+(d_P^w+d_T^w)\sum_{L^T\in\mathcal{L}^T}|L^T|\\ &= O(\mu^2\hat\kappa + d_T^w(d_T^w+\partial_P)|Q| + (d_P^w+d_T^w)d_T^w|Q|)\\ &= O(\mu^2(\hat\kappa+|Q|))\\ &= O(k^2W^2(\hat\kappa+|Q|)).\end{aligned}$$

Definition 8.16. *For a set $\mathcal{U}$ of fragments of a string S and an interval $I\subseteq\mathbb{Z}$, we write $\mathcal{U}_I := \{S[\,x\,..\,y\,)\in\mathcal{U} : [\,x\,..\,y\,)\cap I\neq\emptyset\}$.*

Definition 8.17 [CKW22, Definition 5.8]**.** *For any fixed thresholds $\hat\kappa\in\mathbb{Z}_{\ge0}$ and $\eta\in\mathbb{Z}_{>0}$, we say that a position $v\in[\,0\,..\,n-m+k\,]$ is* light *if the following conditions are simultaneously satisfied:*

- $\mathrm{mk}(v,\hat\kappa,\mathcal{L}^P,\mathcal{L}^T)<\eta$,
- $\mathcal{L}^T_{[\,v-\hat\kappa\,..\,v+\hat\kappa\,)}=\emptyset$,
- $\mathcal{L}^T_{[\,v+m-\hat\kappa\,..\,v+m+\hat\kappa\,)}=\emptyset$, *and*
- $\mathcal{L}^T_{[\,v+|L_1^P|-\hat\kappa\,..\,v+|L_1^P|+\hat\kappa\,)}=\emptyset$.

Otherwise, the position $v\in[\,0\,..\,n-m+k\,]$ is called heavy*. We denote the sets of heavy and light positions by $\mathbb{H}$ and $\mathbb{L}$, respectively.*

Lemma 8.18 ($\mathtt{Heavy}(P,T,k,w,Q,\mathcal{L}^P,\mathcal{L}^T,\hat\kappa,\eta)$) [CKW22, see Lemma 5.9]**.** *Consider an instance of AlignedPeriodicMatches with an integer metric weight function, families $\mathcal{L}^P=\mathtt{Locked}(P,Q,d_P^w,\partial_P,w)$ and $\mathcal{L}^T=\mathtt{Locked}(T,Q,d_T^w,0,w)$, as well as thresholds $\kappa\in\mathbb{Z}_{\ge0}$ and $\eta\in\mathbb{Z}_{>0}$,*

The set $\mathbb{H}$ of heavy positions, represented as the union of $O(k^2W^2)$ disjoint integer ranges, can be computed in $\widetilde{O}(k^2W^3)$ time.

Proof. We implement the marking process according to Definition 8.14, assigning η extra marks to all positions made heavy due to the last three items in Definition 8.17. Formally, we produce the following $O(\mu^2)=O(k^2W^2)$ weighted intervals:

- $(\,\ell_T-\hat\kappa-r_P\,..\,r_T+\hat\kappa-\ell_P\,)$ of weight $\min\{\mathsf{ed}^w(L^T\to{}^*Q^*),\mathsf{ed}^w(L^P\to{}^*Q^*)\}$ for each locked fragment $L^P=P[\,\ell_P\,..\,r_P\,)\in\mathcal{L}^P$ with $\ell_P\neq0$ and $L_T=T[\,\ell_T\,..\,r_T\,)\in\mathcal{L}^T$,
- $(\,\ell_T-\hat\kappa-r_P\,..\,r_T+\hat\kappa-\ell_P\,)$ of weight $\mathsf{ed}^w(L^T\to{}^*Q^*)$ for each locked fragment $L^P=P[\,\ell_P\,..\,r_P\,)\in\mathcal{L}^P$ with $\ell_P=0$ and $L_T=T[\,\ell_T\,..\,r_T\,)\in\mathcal{L}^T$,
- $(\,\ell_T-\hat\kappa\,..\,r_T+\hat\kappa\,)$ of weight η for each locked fragment $L_T=T[\,\ell_T\,..\,r_T\,)\in\mathcal{L}^T$,
- $(\,\ell_T-\hat\kappa-m\,..\,r_T+\hat\kappa-m\,)$ of weight η for each locked fragment $L_T=T[\,\ell_T\,..\,r_T\,)\in\mathcal{L}^T$,
- $(\,\ell_T-\hat\kappa-|L_1^P|\,..\,r_T+\hat\kappa-|L_1^P|\,)$ of weight η for each locked fragment $L_T=T[\,\ell_T\,..\,r_T\,)\in\mathcal{L}^T$.

The heavy positions are exactly those positions in $[\,0\,..\,n-m+k\,]$ that are contained in intervals of total weight at least η; they can be computed using a sweep-line procedure with events corresponding to interval endpoints. The number of events is $O(k^2W^2)$, so the output consists of $O(k^2W^2)$ disjoint ranges. In terms of the running time, the bottleneck is computing the locked costs; each such cost c can be computed in $\widetilde{O}(c^2W)$ time using Fact 7.11, and this sums up to $\widetilde{O}(k^2W^3)$ because $\sum c=O(\mu)=O(kW)$.

Lemma 8.19 [CKW22, see Lemma 5.10]. *The total number of heavy positions does not exceed*

$$|\mathbb{H}| = O((kW/\eta + 1) \cdot kW \cdot (\hat{\kappa} + |Q|)).$$

Moreover, for any integer $b \in \mathbb{Z}_{>0}$,

$$\left|\bigcup_{v\in\mathbb{H}} [\, v - b \,..\, v + b \,]\right| = O((kW/\eta + 1) \cdot kW \cdot (\hat{\kappa} + b + |Q|)).$$

Proof. By Lemma 8.15, the number of positions violating the first condition of Definition 8.17 does not exceed $O((kW)^2 \cdot (\hat{\kappa} + |Q|)/\eta)$. As for the remaining conditions, each locked fragment $L^T \in \mathcal{L}^T$ may yield at most $3(|L^T| + 2\hat{\kappa})$ heavy positions, for a total of:

$$\sum_{L^T\in\mathcal{L}^T} 3(|L^T| + 2\hat{\kappa}) = O((|Q| + \hat{\kappa})d_T^w) = O(kW(|Q| + \hat{\kappa})).$$

As for the second claim, observe that if v is heavy, then all positions in $[\, v-b \,..\, v+b \,] \cap [\, 0 \,..\, n-m+k \,]$ would be heavy if we increased the threshold $\hat{\kappa}$ to $\hat{\kappa} + b$. This is because the left endpoint of all intervals considered in the proof of Lemma 8.18 contains a $-\hat{\kappa}$ term whereas the right endpoint contains a $+\hat{\kappa}$ term (and there is no other dependency on $\hat{\kappa}$). Further, $|\bigcup_{v\in\mathbb{H}} [\, v - b \,..\, v + b \,] \setminus [\, 0 \,..\, n - m + k \,]| \le 2b$, since $\mathbb{H} \subseteq [\, 0 \,..\, n - m + k \,]$. ◢

Further, consider a partition of the positions of T in $[\, 0 \,..\, n - m + k \,]$ into heavy and light using Lemma 8.18. First, we show how to compute (k, w)-error occurrences that start at heavy positions; this is a straightforward application of Lemmas 7.14 and 7.15. Then, we reduce the problem of computing (k, w)-error occurrences that start at light positions to an instance $\texttt{TrimmedPM}(T, P, k, w, Q, \mathcal{A}_T, \mathcal{A}_P, \mathsf{Red}(P), \mathsf{Red}(T), \alpha)$, where $\mathsf{Red}(P)$ and $\mathsf{Red}(T)$ are both of size $O(kW)$ and contain all the internal pieces that (almost) overlap locked fragments, and $\eta \in [\, 1 \,..\, kW \,]$.

8.4 Computing Occurrences Starting at Heavy Positions

Lemma 8.20 ($\texttt{HeavyMatches}(P,\ T,\ k,\ w,\ Q,\ \mathcal{A}_P,\ \mathcal{A}_T,\ \mathbb{H})$) [CKW22, see Lemma 6.1]. *Given an instance of the* AlignedPeriodicMatches *problem and the set $\mathbb{H}$ of heavy positions constructed using Lemma 8.18, $\mathrm{Occ}_k^w(P, T) \cap \mathbb{H}$ can be computed in $\widetilde{O}((kW/\eta + 1)k^3W^3)$ time in the* PILLAR *model.*

Proof. Recall that the set $\mathbb{H}$ is represented as the union of $O(k^2W^2)$ disjoint heavy ranges (listed in the left-to-right order).

Our algorithm starts with an application Lemma 7.14 to construct the sequence $(r_j)_{j\in J}$, represented as a concatenation of $O(k)$ arithmetic progressions. Our first goal is to enumerate elements of the set $J' := \{j \in J : [\, r_j \,..\, r_{j+1} \,) \cap \mathbb{H} \neq \emptyset\}$. For this, we simultaneously traverse the sequence $(r_j)_{j\in J}$ along with the heavy ranges constituting $\mathbb{H}$. For each heavy range $H \subseteq \mathbb{H}$, we list all $j \in J$ such that $[\, r_j \,..\, r_{j+1} \,) \cap H \neq \emptyset$ (only the smallest such j might have already been listed for an earlier heavy range). In the second phase, we compute $O := \bigcup_{j\in J'}(r_j + \mathrm{Occ}_k^w(P, R_j)) \cap [\, r_j \,..\, r_{j+1} \,)$ using Corollary 6.17, making sure that the positions are listed in the left-to-right order. Finally, we simultaneously scan O and the heavy ranges constituting $\mathbb{H}$, reporting all positions $v \in O \cap \mathbb{H}$.

As for correctness, observe that $R_j = T[\, r_j \,..\, r'_j \,)$, so $O \subseteq \mathrm{Occ}_k^w(P, T)$ and, due to the final filtering step, we output a subset of $\mathrm{Occ}_k^w(P, T) \cap \mathbb{H}$. To prove the converse inclusion, consider a position $v \in \mathrm{Occ}_k^w(P, T) \cap \mathbb{H}$. A combination of Lemma 7.14 with Fact 3.7 yields that there exists $j \in J$ such that $v \in (r_j + \mathrm{Occ}_k^w(P, R_j)) \cap [\, r_j \,..\, r_{j+1} \,)$ and, by the definition of J', we also have $j \in J'$. Consequently, v is indeed reported.

As for the complexity analysis, let us first compute the running time in terms of $|J'|$. Constructing the sequence $(r_j)_{j\in J}$ costs $O(k)$ time. The set J' can be computed in $O(|J'| + k + k^2W^2) = O(|J'| + k^2W^2)$ time. The applications of Corollary 6.17 cost $\widetilde{O}(k^2W)$ time each (in the PILLAR model), for a total of $\widetilde{O}(|J'| \cdot k^2W^2)$ time in the PILLAR model. A (crude) upper bound on the output size is $O(|J'|k^2W^2)$, so the final filtering step works in $O((1 + |J'|)k^2W^2)$ time. Overall, the running time in the PILLAR model is $O((1 + |J'|)k^2W^2)$.

It remains to bound $|J'|$. Observe that each interval $[\, r_j \,..\, r_{j+1} \,)$ (possibly except for the last one with $j = \max J$) has length at most $\tau + d_T$. Moreover, the total length of any s intervals $[\, r_j \,..\, r_{j+1} \,)$ with $0 < r_j < r_{j+1} < n$ is at least $s\tau - d_T$. On top of that, there might be one non-empty interval with $r_j = 0$ and one non-empty interval with $r_{j+1} = n$. We conclude that

$$|J'| \le 2 + \frac{\sum_{j\in J'\setminus\{\max J\}} |[\, r_j \,..\, r_{j+1} \,)| + d_T}{\tau} \le 2 + \frac{|\bigcup_{v\in\mathbb{H}}[\, v - \tau - d_T \,..\, v + \tau + d_T \,]| + d_T}{\tau}.$$

Since $\tau = \Theta(\max(|Q|, k))$ and $d_T, \hat{\kappa} = O(kW)$, the bound of Lemma 8.19 yields that $|J'| = O((kW/\eta + 1)kW^2)$. Hence, the total running time is $\widetilde{O}((kW/\eta + 1)k^3W^3)$ just as claimed. ◢

8.5 Computing Occurrences Starting at Light Positions

Combinatorial Insights

The following fact follows directly from Definition 8.17.

Fact 8.21 [CKW22, see Fact 6.2]. *For any light position v in T and for any $x \in (\, v + m - \hat{\kappa} \,..\, v + m + \hat{\kappa} \,)$, we have $\mathcal{L}^T_{[\, v \,..\, x \,)} = \{T[\, \ell \,..\, r \,) \in \mathcal{L}^T : [\, \ell \,..\, r \,) \subseteq [\, v \,..\, v + m \,)\}$.* ◢

Let $\mathcal{A}^w_T : T \rightsquigarrow Q^\infty[\, x^w_T \,..\, y^w_T \,)$ be a w-optimal alignment of cost $\mathsf{ed}^w(T \to {}^*Q^*)$. For each position v of T, set $\rho(v) \in [\, x^w_T \,..\, y^w_T \,]$ such that $\mathcal{A}^w_T(T[\, v \,..\, n \,)) = Q^\infty[\, \rho(v) \,..\, y^w_T \,)$. Observe that

$$\begin{aligned}\mathsf{ed}^w(T \to {}^*Q^*) &= \mathsf{ed}^w(T[\, 0 \,..\, v \,) \to Q^\infty[\, x^w_T \,..\, \rho(v) \,)) + \mathsf{ed}^w(T[\, v \,..\, n \,) \to Q^\infty[\, \rho(v) \,..\, y^w_T \,)) \\ &= \mathsf{ed}^w(T[\, 0 \,..\, v \,) \to {}^*\mathrm{rot}^{-\rho(v)}(Q)) + \mathsf{ed}^w(T[\, v \,..\, n \,) \to \mathrm{rot}^{-\rho(v)}(Q)^*).\end{aligned}$$

Lemma 8.22 [CKW22, see Lemma 6.3]. *For any two positions $v < x$ of T such that $\mathcal{L}^T_{[\, v \,]} = \mathcal{L}^T_{[\, x \,]} = \emptyset$, we have*

$$\mathsf{ed}^w(T[\, v \,..\, x \,) \to {}^*Q^*) = \mathsf{ed}^w(T[\, v \,..\, x \,) \to Q^\infty[\, \rho(v) \,..\, \rho(x) \,)) = \sum_{L\in\mathcal{L}^T_{[\, v \,..\, x \,)}} \mathsf{ed}^w(L \to {}^*Q^*).$$

Proof. Set $\mathcal{L}^T_{[\, v \,..\, x \,)} = \{L^T_j \in \mathcal{L}^T : j \in [\, j_1 \,..\, j_2 \,]\}$, and observe that all elements of this set are fragments of $T[\, v \,..\, x \,)$. Now, we have

$$\begin{aligned}&\mathsf{ed}^w(T[\, v \,..\, x \,) \to Q^\infty[\, \rho(v) \,..\, \rho(x) \,)) \\ &\quad= \mathsf{ed}^w(T \to Q^\infty[\, x^w_T \,..\, y^w_T \,)) - \mathsf{ed}^w(T[\, 0 \,..\, v \,) \to Q^\infty[\, x^w_T \,..\, \rho(v) \,)) \\ &\qquad\qquad - \mathsf{ed}^w(T[\, x \,..\, n \,) \to Q^\infty[\, \rho(x) \,..\, y^w_T \,)) \\ &\quad\le d_T - \sum_{j=1}^{j_1-1} \mathsf{ed}^w(L^T_j \to {}^*Q^*) - \sum_{j=j_2+1}^{\ell^T} \mathsf{ed}^w(L^T_j \to {}^*Q^*) \\ &\quad= \sum_{j=j_1}^{j_2} \mathsf{ed}^w(L^T_j \to {}^*Q^*) \\ &\quad\le \mathsf{ed}^w(T[\, v \,..\, x \,) \to {}^*Q^*) \\ &\quad\le \mathsf{ed}^w(T[\, v \,..\, x \,) \to Q^\infty[\, \rho(v) \,..\, \rho(x) \,)).\end{aligned}$$

◢

Lemma 8.23 [CKW22, see Lemma 6.4]. *Consider a light position v of T. If $\mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(v)}(Q)^*) < \partial_P$, then there is an $x \in [\, v \,..\, n \,]$ such that*

$$\mathsf{ed}^w(P \to T[\, v \,..\, x \,)) \le \mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(v)}(Q)^*) + \sum_{i=2}^{\ell^P} \mathsf{ed}^w(L_i^P \to {}^*Q^*) + \sum_{L \in \mathcal{L}^T_{[\, v \,..\, v+m \,)}} \mathsf{ed}^w(L \to {}^*Q^*).$$

Proof. We first prove two auxiliary claims.

Claim 8.24. *We have $\mathsf{ed}^w(P \to \mathrm{rot}^{-\rho(v)}(Q)^*) = \mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(v)}(Q)^*) + \sum_{i=2}^{\ell^P} \mathsf{ed}^w(L_i^P \to {}^*Q^*) < \partial_P + d_P^w$.*

Proof. The inequality $\mathsf{ed}^w(P \to \mathrm{rot}^{-\rho(v)}(Q)^*) \ge \mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(v)}(Q)^*) + \sum_{i=2}^{\ell^P} \mathsf{ed}^w(L_i^P \to {}^*Q^*)$ holds trivially.

By Lemma 8.12, we have $P = L_1^P Q^{\alpha_1} L_2^P Q^{\alpha_2} \cdots Q^{\alpha_{\ell^P-1}} L_{\ell^P}^P$ for some non-negative integers α_i. Since L_1^P is ∂_P-locked, we have $\mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(v)}(Q)^*) = \mathsf{ed}^w(L_1^P \to Q^\infty[\, \rho(v) \,..\, jq \,))$ for some non-negative integer j. Further, for each $i \in (\, 1 \,..\, \ell^P \,)$, we have $\mathsf{ed}^w(L_i^P \to {}^*Q^*) = \mathsf{ed}^w(L_i^P \to Q^{\beta_i})$ for some non-negative integer β_i. Finally, we have $\mathsf{ed}^w(L_{\ell^P}^P \to {}^*Q^*) = \mathsf{ed}^w(L_{\ell^P}^P \to Q^\infty[\, 0 \,..\, y \,))$ for some non-negative integer y.

Set $\gamma = \alpha_1 + \sum_{i=2}^{\ell^P-1}(\alpha_i + \beta_i)$. The above discussion implies that there is an alignment of P with the prefix $Q[\, \rho(v) \,..\, jq \,)Q^\gamma Q^\infty[\, 0 \,..\, y \,)$ of $Q[\, \rho(v) \,..\, jq \,)Q^\infty$ that costs $\mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(v)}(Q)^*) + \sum_{i=2}^{\ell^P} \mathsf{ed}^w(L_i^P \to {}^*Q^*)$, thus proving that $\mathsf{ed}^w(P \to \mathrm{rot}^{-\rho(v)}(Q)^*) \le \mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(v)}(Q)^*) + \sum_{i=2}^{\ell^P} \mathsf{ed}^w(L_i^P \to {}^*Q^*)$. This concludes the proof of the claimed equality.

The claimed inequality holds since $\mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(a)}(Q)^*) < \partial_P$ and $\sum_{i=2}^{\ell^P} \mathsf{ed}^w(L_i^P \to {}^*Q^*) \le d_P^w$. ◿

Claim 8.25. *There is an $x \in (\, v + m - \hat\kappa \,..\, v + m + \hat\kappa \,) \cap [\, 0 \,..\, n \,)$ such that $\mathsf{ed}^w(P \to \mathrm{rot}^{-\rho(v)}(Q)^*) = \mathsf{ed}^w(P \to Q^\infty[\, \rho(v) \,..\, \rho(x) \,))$.*

Proof. Recall that $\mathcal{L}^T_{[\, v+m-\hat\kappa \,..\, v+m+\hat\kappa \,)} = \varnothing$ holds because v is a light position of T. Since $\mathcal{L}^T$ contains a suffix of T, we conclude that either $n \le v + m - \hat\kappa$ or $v + m + \kappa \le n$. The former case contradicts $v \le n - m + k < n - m + \hat\kappa$, so $T[\, v + m - \hat\kappa \,..\, v + m + \hat\kappa \,)$ must be a fragment of T disjoint with all locked fragments in $\mathcal{L}^T$. This means that $\mathcal{A}_T^w$ matches $T[\, v + m - \hat\kappa \,..\, v + m + \hat\kappa \,)$ without any edits to $\mathcal{A}_T^w(T[\, v + m - \hat\kappa \,..\, v + m + \hat\kappa \,))$. Since $(v, \rho(v)) \in \mathcal{A}_T^w$ and the cost of $\mathcal{A}_T^w$ is d_T^w, the fragment $\mathcal{A}_T^w(T[\, v + m - \hat\kappa \,..\, v + m + \hat\kappa \,))$ must contain $Q^\infty[\, \rho(v) + m - \hat\kappa + d_T^w \,..\, \rho(v) + m + \hat\kappa - d_T^w \,)$. Consequently, for every $y \in (\, \rho(v) + m - \hat\kappa + d_T^w \,..\, \rho(v) + m + \hat\kappa - d_T^w \,)$, there exists $x \in (\, v + m - \hat\kappa \,..\, v + m + \hat\kappa \,) \cap [\, 0 \,..\, n \,)$ such that $y = \rho(x)$. Since $\mathsf{ed}^w(P \to \mathrm{rot}^{-\rho(v)}(Q)^*) \le \partial_P + d_P^w < \hat\kappa - d_T^w$ holds by Claim 8.24, such x exists in particular for y chosen so that $\mathsf{ed}^w(P \to \mathrm{rot}^{-\rho(v)}(Q)^*) = \mathsf{ed}^w(P \to Q^\infty[\, \rho(v) \,..\, y \,))$. ◿

We get the desired bound by combining Claims 8.24 and 8.25, Fact 8.21, and Lemma 8.22 via the triangle inequality; observe that Lemma 8.22 is applicable because $\mathcal{L}^T_{[x]} \subseteq \mathcal{L}^T_{[v+m-\hat{\kappa}\,..\,v+m+\hat{\kappa})} = \varnothing$.

$$\begin{aligned}
&\mathsf{ed}^w(P \to T[v\,..\,x)) \\
&\quad\le \mathsf{ed}^w(P \to Q^\infty[\rho(v)\,..\,\rho(x))) + \mathsf{ed}^w(T[v\,..\,x) \to Q^\infty[\rho(v)\,..\,\rho(x))) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{(symmetry, triangle inequality)} \\
&\quad= \mathsf{ed}^w(P \to \mathrm{rot}^{-\rho(v)}(Q)^*) + \mathsf{ed}^w(T[v\,..\,x) \to Q^\infty[\rho(v)\,..\,\rho(x))) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{(Claim 8.25)} \\
&\quad= \mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(v)}(Q)^*) + \sum_{i=2}^{\ell^P} \mathsf{ed}^w(L_i^P \to {}^*Q^*) + \sum_{L \in \mathcal{L}^T_{[v\,..\,x)}} \mathsf{ed}^w(L \to {}^*Q^*) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{(Claim 8.24 and Lemma 8.22)} \\
&\quad= \mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(v)}(Q)^*) + \sum_{i=2}^{\ell^P} \mathsf{ed}^w(L_i^P \to {}^*Q^*) + \sum_{L \in \mathcal{L}^T_{[v\,..\,v+m)}} \mathsf{ed}^w(L \to {}^*Q^*). \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\text{(Fact 8.21)}
\end{aligned}$$

This completes the proof of the lemma. ◾

For a position v and a locked fragment $L \in \mathcal{L}^P \cup \mathcal{L}^T$, we write $\mathrm{mk}(v, L)$ for the number of marks placed in v due to pairs of locked fragments that contain L; formally:

$$\mathrm{mk}(v, L) = \begin{cases} \sum_{L^T \in \mathcal{L}^T} \mathrm{mk}(v, \hat{\kappa}, L, L^T) & \text{if } L \in \mathcal{L}^P; \\ \sum_{L^P \in \mathcal{L}^P} \mathrm{mk}(v, \hat{\kappa}, L^P, L) & \text{if } L \in \mathcal{L}^T. \end{cases}$$

Definition 8.26 [CKW22, see Definition 6.7]. *For a light position v of T, set*

$$\mathcal{D}(v) := \{L_1^P\} \cup \{L \in \mathcal{L}^P \cup \mathcal{L}^T_{[v\,..\,v+m)} : \mathrm{mk}(v, L) < \mathsf{ed}^w(L \to {}^*Q^*)\}.$$ ◾

Let us now provide some intuition on what follows. Consider an alignment $\mathcal{A} : P \rightsquigarrow T[v\,..\,w)$, where v is a light position of T and the cost of $\mathcal{A}$ with respect to w is not larger than k. In the next lemma, we essentially lower-bound the cost of the restriction of such an alignment $\mathcal{A}$ to each locked fragment. For instance, we lower-bound $\mathsf{ed}^w(L \to \mathcal{A}(L))$ for each $L \in \mathcal{L}^P$. Our lower bound is positive only for elements of $\mathcal{D}(v)$. Consider some locked fragment $L \in \mathcal{D}(v) \cap \mathcal{L}^P$ other than L_1^P. Roughly speaking, at most $\mathrm{mk}(v, L)$ errors of L with a fragment Q' of Q^∞ cancel out with errors between $\mathcal{A}(L)$ and Q', yielding a lower bound $\mathsf{ed}^w(L \to \mathcal{A}(L)) \ge \mathsf{ed}^w(L \to {}^*Q^*) - \mathrm{mk}(v, L)$. Then, the definition of $\mathcal{D}(v)$ guarantees that, for any $L \in \mathcal{D}(v) \cap \mathcal{L}^P$, $\mathcal{A}(L)$ is disjoint from all $L' \in \mathcal{D}(v) \cap \mathcal{L}^T$. One can exploit this property to obtain a lower bound for the cost of $\mathcal{A}$ by showing that we can sum over the lower bounds for individual locked fragments in $\mathcal{D}(v)$. In fact, we use this reasoning to lower bound the cost of an alignment between two other strings, obtained from P and T, respectively, via the deletion of some fragments; this happens in the proof of Lemma 8.37 in Section 8.5.

Lemma 8.27 [CKW22, see Lemma 6.8]. *Consider a light position v of T and a locked fragment $L \in \mathcal{L}^P \cup \mathcal{L}^T_{[v\,..\,v+m)}$.*

1 *If $L = T[v+\ell\,..\,v+r) \in \mathcal{L}^T$, then every $U \in \{P[i\,..\,j) : i \ge \ell - k \text{ and } j \le r + k\}$ satisfies*

$$\mathsf{ed}^w(L \to U) \ge \mathsf{ed}^w(L \to {}^*Q^*) - \mathrm{mk}(v, L).$$

2 *If $L = P[\ell\,..\,r) \in \mathcal{L}^P \setminus \{L_1^P\}$, then every $U \in \{T[i\,..\,j) : i \ge v + \ell - k \text{ and } j \le v + r + k\}$ satisfies*

$$\mathsf{ed}^w(L \to U) \ge \mathsf{ed}^w(L \to {}^*Q^*) - \mathrm{mk}(v, L).$$

▪3 *If L is L_1^P, then for every $U \in \{T[\,v\,..\,v+j\,) : j \in [\,|L|-k\,..\,|L|+k\,]\}$ satisfies*

$$\mathsf{ed}^w(L \to U) \geq \mathsf{ed}^w(L \to \mathrm{rot}^{-\rho(v)}(Q)^*) - \mathrm{mk}(v, L).$$

Proof. Consider some $L \in \mathcal{L}^P \cup \mathcal{L}^T_{[\,v\,..\,v+m\,)}$. If $L \in \mathcal{L}^T$, let $Y = P$; otherwise, let $Y = T$. Further, let $U \coloneqq Y[\,u_1\,..\,u_2\,)$ denote any of the fragments specified in the statement of the lemma, and let

$$\zeta \coloneqq \begin{cases} \mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(v)}(Q)^*) & \text{if } L = L_1^P, \\ \mathsf{ed}^w(L \to {}^*Q^*) & \text{otherwise.} \end{cases}$$

Let x and y denote integers that satisfy $\mathsf{ed}^w(U \to Q^\infty[\,x\,..\,y\,)) = \mathsf{ed}^w(U \to {}^*Q^*)$. In the case where $L = L_1^P$, we choose $x = \rho(v)$ and $y = \rho(u_2)$ so that $Q^\infty[\,x\,..\,y\,)$ is a prefix of $\mathrm{rot}^{-\rho(v)}(Q)^\infty$. This is allowed by Lemma 8.22 because $\mathcal{L}^T_{[\,v-\hat\kappa\,..\,v+\hat\kappa\,)} = \mathcal{L}^T_{[\,v+|L_1^P|-\hat\kappa\,..\,v+|L_1^P|+\hat\kappa\,)} = \varnothing$, and it ensures that the following inequality holds in all three cases (the inequality holds trivially if $L \neq L_1^P$):

$$\mathsf{ed}^w(L \to Q^\infty[\,x\,..\,y\,)) \geq \zeta. \tag{1}$$

Further, our marking scheme implies

$$\mathrm{mk}(v, L) \geq \sum_{K \in \mathcal{L}^Y_{[u_1\,..\,u_2\,)}} \mathsf{ed}^w(K \to {}^*Q^*). \tag{2}$$

Let $Y[\,u_1'\,..\,u_2'\,)$ be the fragment that is covered by $Y[\,u\,..\,w\,)$ and the elements of $\mathcal{L}^Y_{[u_1\,..\,u_2\,)}$. We have

$$\mathsf{ed}^w(U \to Q^\infty[\,x\,..\,y\,)) = \mathsf{ed}^w(U \to {}^*Q^*) \leq \mathsf{ed}^w(Y[\,u_1'\,..\,u_2'\,) \to {}^*Q^*) = \sum_{K \in \mathcal{L}^Y_{[u_1\,..\,u_2\,)}} \mathsf{ed}^w(K \to {}^*Q^*), \tag{3}$$

where the last equality follows from the properties of locked fragments as computed by Lemma 8.12.

We are now ready to prove the claimed inequality.

$$\begin{aligned}
\mathsf{ed}^w(L \to U) &\geq |\mathsf{ed}^w(L \to Q^\infty[\,x\,..\,y\,)) - \mathsf{ed}^w(U \to Q^\infty[\,x\,..\,y\,))| && \text{(symmetry, triangle inequality)} \\
&\geq \mathsf{ed}^w(L \to Q^\infty[\,x\,..\,y\,)) - \mathsf{ed}^w(U \to Q^\infty[\,x\,..\,y\,)) \\
&\geq \zeta - \mathsf{ed}^w(U \to Q^\infty[\,x\,..\,y\,)) && \text{(due to (1))} \\
&\geq \zeta - \sum_{K \in \mathcal{L}^Y_{[i\,..\,j\,)}} \mathsf{ed}^w(K \to {}^*Q^*) && \text{(due to (3))} \\
&\geq \zeta - \mathrm{mk}(v, L). && \text{(due to (2))}
\end{aligned}$$

This completes the proof of the lemma. ◢

Shrinking Runs of Plain Pairs

We set $\mathsf{Red}(P)$ to be equal to

$$\mathsf{Special}(P) \cup \{P_i : i \in (\,1\,..\,z\,) \text{ and } \exists_{P[\,\ell\,..\,r\,) \in \mathcal{L}^P}\, [\,\ell - 13\hat\kappa\,..\,r + 13\hat\kappa\,) \cap [\,p_i\,..\,p_{i+1} + \Delta\,) \neq \varnothing\}.$$

Similarly, we set $\mathsf{Red}(T)$ to be equal to

$$\mathsf{Special}(T) \cup \{T_i : i \in (\,\min J + 1\,..\,\max J + z\,) \text{ and } \exists_{T[\,\ell\,..\,r\,) \in \mathcal{L}^T}\, [\,\ell - 13\hat\kappa\,..\,r + 13\hat\kappa\,) \cap [\,t_i\,..\,t_{i+1} + \Delta\,) \neq \varnothing\}.$$

Next, we upper-bound the sizes of these two sets and show how to construct them efficiently.

Lemma 8.28 [CKW22, see Lemma 6.9]. *Given P, T, $\mathcal{A}_P$, $\mathcal{A}_T$, $\mathcal{L}^P$, and $\mathcal{L}^T$, the sets $\mathsf{Red}(P)$ and $\mathsf{Red}(T)$ are of size $O(kW^2)$ and can be constructed in time $\widetilde{O}(kW^2)$ in the* PILLAR *model.*

Proof. We start with upper-bounding the size of each of the sets $\mathsf{Red}(P)$ and $\mathsf{Red}(T)$. $\mathsf{Special}(P)$ and $\mathsf{Special}(T)$ are of size $O(d) = O(k)$ due to Lemma 7.25. The task is therefore to bound, for each of P and T, the number of internal pieces that are within $13\hat{\kappa} = O(kW)$ positions of a locked fragment.

Let (S, β_S) denote either of (P, β_P) or (T, β_T) and $\mathcal{L}^S = \{L[\,\ell_j\,..\,h_j\,) : j \in [\,1\,..\,\ell^S\,]\}$. It suffices to upper bound the number of tiles $S[\,s_{i-1}\,..\,s_i\,)$ with $i \in (\,1\,..\,\beta_S\,)$ in the τ-tile partition of S (with respect to $\mathcal{A}_S$) such that $[\,s_{i-1}\,..\,s_i\,)$ overlaps $[\,\ell_j - \Delta - 13\hat{\kappa}\,..\,h_j + 13\hat{\kappa}\,)$ for some $j \in [\,1\,..\,\ell^S\,]$; that is,

$$|\{i \in (\,1\,..\,\beta_S\,) : \exists_{j\in[\,1\,..\,\ell^S\,]}[\,\ell_j - \Delta - 13\hat{\kappa}\,..\,h_j + 13\hat{\kappa}\,) \cap [\,s_{i-1}\,..\,s_i\,) \neq \varnothing\}|.$$

Intuitively, we extend each locked fragment in $\mathcal{L}^S$ by $\Delta + 13\hat{\kappa}$ characters to the left and by $13\hat{\kappa}$ characters to the right, thus covering at most $\|\mathcal{L}^S\| + \ell^S \cdot (2 \cdot 13 + 6)\hat{\kappa} = \|\mathcal{L}^S\| + 32\ell^S\hat{\kappa}$ positions of S, since $\Delta = 6\kappa \le 6\hat{\kappa}$. Let us first upper bound the size ϕ of the set Φ of tiles (other than the first and last ones) that are fully covered by these "extended" locked fragments, that is, $\Phi := \{i \in (\,0\,..\,\beta_S\,) : [\,s_{i-1}\,..\,s_i\,) \subseteq \bigcup_{j\in[\,1\,..\,\ell^S\,]}[\,\ell_j - \Delta - 13\hat{\kappa}\,..\,h_j + 13\hat{\kappa}\,)\}$. We have

$$\sum_{i\in\Phi}(\tau - |S[\,s_{i-1}\,..\,s_i\,)|) \le \sum_{i\in\Phi}\mathsf{ed}(S[\,s_{i-1}\,..\,s_i\,), Q[\,(i-1)\tau\,..\,i\tau\,)) \le \mathsf{ed}(S, {}^*Q^*)$$

and hence

$$\phi \cdot \tau - \mathsf{ed}(S, {}^*Q^*) \le \sum_{i\in\Phi}|S[\,s_{i-1}\,..\,s_i\,)| \le \|\mathcal{L}^S\| + 32\ell^S\hat{\kappa},$$

which is equivalent to

$$\phi \le \frac{\|\mathcal{L}^S\| + 32\ell^S\hat{\kappa} + \mathsf{ed}(S, {}^*Q^*)}{\tau}.$$

Finally, we have to account for the at most $2\ell^S$ tiles that overlap "extended" locked fragments but are not fully contained in them; we have at most two such tiles for each $L \in \mathcal{L}^S$.

Since $\ell^S = O(\mathsf{ed}^w(S \to {}^*Q^*))$ and $\|\mathcal{L}^S\| = O(\mathsf{ed}^w(S \to {}^*Q^*) \cdot q)$ by Lemma 8.12, $\mathsf{ed}^w(S \to {}^*Q^*) = O(kW)$, and $\tau = \Theta(\max\{k, q\})$, we have

$$2\ell^S + \frac{\|\mathcal{L}^S\| + 32\ell^S\hat{\kappa} + \mathsf{ed}(S, {}^*Q^*)}{\tau} = O\left(kW + \frac{kWq + (kW)^2 + k}{\tau}\right) = O(kW^2).$$

Let us now show how to efficiently construct the sets in scope. First, recall that $\mathsf{Special}(P)$ and $\mathsf{Special}(T)$ can be constructed in $O(kW^2)$ time in the PILLAR model due to Lemma 7.25. The remaining elements of $\mathsf{Red}(P)$ (resp. $\mathsf{Red}(T)$) can be computed in a simultaneous left-to-right scan of:

- the representation of starting positions of internal pieces p_i (resp. t_i) as $O(k)$ arithmetic progressions, which can be computed in $O(k)$ time due to Lemma 7.14;
- the $O(kW^2)$ locked fragments of P (resp. T) sorted with respect to their starting positions.

The $\widetilde{O}(kW^2)$ time required for sorting the starting positions of the locked fragments is the bottleneck of the algorithm in the PILLAR model. ◢

Definition 8.29 [CKW22, see Definition 6.10]. *For $j \in J$, set $\mathcal{I}'_j := (F_{j,1}, G_{j,1})\cdots(F_{j,z_j}, G_{j,z_j}) := \mathsf{Trim}(\mathcal{I}_j, \mathsf{Red}(P), \mathsf{Red}(T), 2\eta + 30)$, and $F_j := \mathsf{val}_\Delta(F_{j,1}, \ldots, F_{j,z_j})$ and $G_j := \mathsf{val}_\Delta(G_{j,1}, \ldots, G_{j,z_j})$.*

Further, let $F_{j,i}$ and $G_{j,i}$ correspond to the fragments $F_j[\,f_{j,i}\,..\,f_{j,i+1}+\Delta\,)$ and $G_j[\,g_{j,i}\,..\,g_{j,i+1}+\Delta\,)$, respectively, that is, we set $f_{j,1} = g_{j,1} = 0$ and, for $i \in [\,2\,..\,z_j+1\,]$, $f_{j,i} = \left(\sum_{x<i}|F_{j,x}|\right) - \Delta$ and $g_{j,i} = \left(\sum_{x<i}|G_{j,x}|\right) - \Delta$. ◢

The focus of the remainder of Section 8.5 is to prove the following lemma.

Lemma 8.43 [CKW22, see Lemma 6.11]**.**

$$\mathrm{Occ}^w_k(P,T) \cap \mathbb{L} = \left(\bigcup_{j\in J}(r_j + \mathrm{Occ}^w_k(\mathcal{I}'_j))\right) \cap \mathbb{L} = \left(\bigcup_{j\in J}(r_j + \mathrm{Occ}^w_k(F_j, G_j))\right) \cap \mathbb{L}.$$ ◢

The combination of Lemma 7.14 and Lemma 7.30 directly yields the following.

Corollary 8.30. *We have* $\left(\bigcup_{j\in J}(r_j + \mathrm{Occ}_k^w(F_j, G_j))\right) \supseteq \left(\bigcup_{j\in J}(r_j + \mathrm{Occ}_k^w(P, R_j))\right) = \mathrm{Occ}_k^w(P, T)$.

See [CKW22, Example 6.13] an example that illustrates that, for some $j \in J$ and $p \in \mathrm{Occ}_k(F_j, G_j)$, we might have $r_j + p \notin \mathrm{Occ}_k(P, T)$ if $r_j + p \in \mathbb{H}$.

It remains to show that, for any light position $p \in \left(\bigcup_{j\in J}(r_j + \mathrm{Occ}_k^w(F_j, G_j))\right)$, we have $p \in \mathrm{Occ}_k^w(P, T)$. The following lemma demonstrates that, for each $j \in J$, we can restrict our attention to a subset of the positions of R_j.

Lemma 8.31 [CKW22, see Lemma 6.14]**.** *Consider some $j \in J$ and a position p of R_j such that $r_j + p$ is a light position of T. If $\mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(r_j+p)}(Q)^*) \geq \partial_P$, then $p \notin \mathrm{Occ}_k^w(F_j, G_j)$.*

Proof. By the definition of $\mathsf{Red}(T)$, it follows that $|F_j| \geq |L_1^P|$ and any prefix of F_j of length at most $|L_1^P| + 13\hat{\kappa}$ is also a prefix of P. Combined with the upper bound on the sum of length-differences from Lemma 7.23, this also implies that any prefix of G_j of length at most $|L_1^P| + 13\hat{\kappa} - 3\kappa$ is also a prefix of R_j. Further, recall that we have $|R_j| \leq |P| + 3\kappa - k$ due to Lemma 7.23. Consequently, since $|R_j| - |F_j| = |P| - |G_j|$, we have

$$\mathrm{Occ}_k^w(F_j, G_j) \subseteq [\,0\,..\,|G_j| - |F_j| + k\,] \subseteq [\,0\,..\,3\kappa\,].$$

It thus suffices to consider $p \in [\,0\,..\,3\kappa\,]$. Let W denote a prefix of $G_j[\,p\,..\,|G_j|\,)$ that satisfies

$$\mathsf{ed}^w(F_j[\,0\,..\,|L_1^P|\,) \to W) = \min_t \mathsf{ed}^w(F_j[\,0\,..\,|L_1^P|\,) \to G_j[\,p\,..\,t\,)).$$

We distinguish between two cases.

- If $|W| \in [\,|L_1^P| - k\,..\,|L_1^P| + k\,]$, then W is a prefix of $T[\,r_j + p\,..\,n\,)$ since
$$p + |W| \leq 3\kappa + |L_1^P| + k \leq |L_1^P| + 13\hat{\kappa} - 3\kappa.$$
Thus, by Lemma 8.27 and since $\partial_P = \Theta(kW)$, we have
$$\begin{aligned}\mathsf{ed}^w(L_1^P \to W) &\geq \mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(r_j+p)}(Q)^*) - \mathrm{mk}(r_j + p, L_1^P)\\ &\geq \partial_P - \mathrm{mk}(r_j + p, L_1^P)\\ &= 2kW - \eta > kW \geq k.\end{aligned}$$
- Otherwise, we have $\mathsf{ed}^w(F_j[\,0\,..\,|L_1^P|\,) \to W^*) \geq \big||W| - |L_1^P|\big| > k$.

To conclude the proof, it suffices to observe that

$$\min_t \mathsf{ed}^w(F_j \to G_j[\,p\,..\,t\,)) \geq \min_t \mathsf{ed}^w(L_1^P \to G_j[\,p\,..\,t\,)) > k,$$

and hence $p \notin \mathrm{Occ}_k^w(F_j, G_j)$.

In what follows, for convenience, we assume that, for each run of plain pairs of $\mathcal{I}_j$ that has been trimmed, the deleted pairs correspond to a suffix of this run. This yields a natural mapping from pairs of $\mathcal{I}_j'$ to pairs of $\mathcal{I}_j$.

Definition 8.32 [CKW22, see Definition 6.15]**.** *For each $j \in J$ and each $i \in [\,1\,..\,z_j\,]$, let $\mathsf{orig}(j, i)$ denote the number of pairs to the left of pair $(F_{j,i}, G_{j,i})$ that were deleted in the process of obtaining $\mathcal{I}_j'$ from $\mathcal{I}_j$. We say that pair $(F_{j,i}, G_{j,i})$* originates *from pair $(P_{i+\mathsf{orig}(j,i)}, T_{j,i+\mathsf{orig}(j,i)})$.*

Let each internal pair of pieces $(F_{j,i}, G_{j,i})$ inherit the color of $(P_{i+\mathsf{orig}(j,i)}, T_{j,i+\mathsf{orig}(j,i)})$. In addition, mark the first and the last pairs as not plain.

Definition 8.33 [CKW22, see Definition 6.16]**.** *For $j \in J$ and $i \in [\,1\,..\,z_j\,]$, we say that $F_j[\,x_1\,..\,x_2\,)$ (or $G_j[\,x_1\,..\,x_2\,)$) has an* overlap *with a pair $(F_{j,i}, G_{j,i})$ if and only if $[\,x_1\,..\,x_2\,) \cap [\,f_{j,i}\,..\,f_{j,i+1} + \Delta\,) \neq \emptyset$ (or $[\,x_1\,..\,x_2\,) \cap [\,g_{j,i}\,..\,g_{j,i+1} + \Delta\,) \neq \emptyset$).*

Definition 8.34 [CKW22, see Definition 6.17]. *Consider some $j \in J$ and a contiguous sequence $\mathcal{M} = (F_{j,i_1}, G_{j,i_1}) \cdots (F_{j,i_2}, G_{j,i_2})$ of pairs in $\mathcal{I}'_j$ that are either all plain or all not plain (that is, $\mathcal{M}$ is* monochromatic*). Fix an $X \in \{F_j, G_j\}$. For $i \in [1 \,..\, z_j]$, if $X = F$, we set $x_{j,i} = f_{j,i}$; otherwise, we set $x_{j,i} = g_{j,i}$.*

For a non-negative integer γ, we say that a fragment $X[x_1 \,..\, x_2)$ of $X \in \{F_j, G_j\}$ is γ-contained *by $\mathcal{M}$ when the following two conditions are satisfied: (a) $x_1 \geq x_{j,i_1} + \gamma$ or $i_1 = 1$ and (b) $x_2 \leq x_{j,i_2+1} + \Delta - \gamma$ or $i_2 = z_j$.*

Fact 8.35 [CKW22, see Fact 6.18]. *If a fragment U of F or G is Δ-contained by a monochromatic sequence $\mathcal{M}$ of contiguous pairs in $\mathcal{I}'_j$, then U only overlaps pairs in $\mathcal{M}$.*

The proof of the following lemma is identical to that of [CKW22, Lemma 6.19].

Lemma 8.36 [CKW22, see Lemma 6.19]. *For $j \in J$, consider a monochromatic sequence $\mathcal{M} = (F_{j,i_1}, G_{j,i_1}) \cdots (F_{j,i_2}, G_{j,i_2})$ of contiguous pairs in $\mathcal{I}'_j$. Let $\{X, Y\} = \{F_j, G_j\}$ and $X[x_1 \,..\, x_2)$ be a fragment of X that is γ-contained by $\mathcal{M}$ for $\gamma \geq 13\kappa$. All the pairs of $\mathcal{I}'_j$ that overlap $Y[y_1 \,..\, y_2) \coloneqq Y[\max\{0, x_1 - 4\kappa\} \,..\, \min\{|Y|, x_2 + 4\kappa\})$ belong to $\mathcal{M}$.*

Lemma 8.37 [CKW22, see Lemma 6.19]. *Consider some $j \in J$ and a position $p \in \mathrm{Occ}^w_k(F_j, G_j)$ such that $r_j + p$ is a light position of T. Then, $r_j + p \in \mathrm{Occ}^w_k(P, T)$.*

Proof. First, observe that we may assume that $\mathcal{I}'_j \neq \mathcal{I}_j$; otherwise, the statement follows trivially. To avoid clutter, we drop the subscript j when referring to F_j and G_j and simply call them F and G, respectively.

Let b denote a position of G and let $\mathcal{B} : F \rightsquigarrow G[p \,..\, b)$ denote an alignment of cost $\min_t \mathrm{ed}^w(F \to G[p \,..\, t)) \leq k$. We intend to show that, in this case, there exist a position c of R_j and an alignment $\mathcal{C} : P \rightsquigarrow R_j[p \,..\, c)$ of the same cost.

Set

$$\mathcal{L}^T_{[r_j+p \,..\, r_j+p+m)} = \{L^T_i : i \in [i_1 \,..\, i_2]\}.$$

Observe that there is a natural mapping of each locked fragment $L^P_i \in \mathcal{L}^P$ to a fragment of F, which we denote by L^F_i; we denote the set of fragments in the image of this mapping by $\mathcal{L}^F$. Similarly, there is a natural mapping of each locked fragment $L^T_i \in \{L^T_i : i \in [i_1 \,..\, i_2]\}$ to a fragment of G, which we denote by L^G_i; we denote the set of fragments in the image of this mapping by $\mathcal{L}^G$. Let $\mathcal{D}'(p)$ consist of the images of the locked fragments in $\mathcal{D}(r_j + p)$ under these mappings. Observe that each fragment $L \in \mathcal{L}^F \cup \mathcal{L}^G$ only overlaps non-plain pairs. For a fragment $L^X_y \in \mathcal{L}^X$, where $X \in \{P, T, F, G\}$, let $L^X_y = X[\ell^X_y \,..\, r^X_y)$. The following claim follows instantly.

Claim 8.38 [CKW22, see Claim 6.21]. *Each fragment $L \in \mathcal{L}^F \cup \mathcal{L}^G$ is $13\hat{\kappa}$-contained by a sequence of contiguous non-plain pairs in $\mathcal{I}'_j$.*

We next essentially show that, if we were to mark positions of G_j based on overlaps of pairs of fragments in $\mathcal{L}^F \times \mathcal{L}^G$, consistently with Definition 8.14, position p of G_j would get the same number of marks as position $r_j + p$ of T.

Claim 8.39 [CKW22, see Claim 6.22]. *For all $x \in [1 \,..\, |\mathcal{L}^P|]$ and $y \in [i_1 \,..\, i_2]$, we have*

$$\left|[\ell^G_y - \hat{\kappa} \,..\, r^G_y + \hat{\kappa}) \cap [p + \ell^F_x \,..\, p + r^F_x)\right| = \left|[\ell^T_y - \hat{\kappa} \,..\, r^T_y + \hat{\kappa}) \cap [r_j + p + \ell^P_x \,..\, r_j + p + r^P_x)\right|.$$

Proof. Consider sequences

$$\mathcal{M}_x = (F_{j,w_1}, G_{j,w_1}) \cdots (F_{j,w_2}, G_{j,w_2}) \quad \text{and} \quad \mathcal{M}_y = (F_{j,w_3}, G_{j,w_3}) \cdots (F_{j,w_4}, G_{j,w_4})$$

of contiguous non-plain pairs of $\mathcal{I}'_j$ that $13\hat{\kappa}$-contain L^F_x and L^G_y, respectively, and are maximal in the sense that they cannot be extended and remain monochromatic. (Such sequences exist by Claim 8.38.)

First, consider the case where $\mathcal{M}_x$ and $\mathcal{M}_y$ do not coincide. We treat the case where $\mathcal{M}_x$ lies to the left of $\mathcal{M}_y$; the other case can be handled analogously. In this case, $w_2 < z_j$ and $w_3 > 1$. Recall that $p \in [\,0\,..\,3\kappa\,]$. By Lemma 8.36, we have that $G[\,p + \ell_x^F\,..\,p + r_x^F\,)$ only overlaps pairs in $\mathcal{M}_x$ and hence it is disjoint from $[\,\ell_y^G - \hat{\kappa}\,..\,r_y^G + \hat{\kappa}\,)$ since $p + r_x^F \le g_{j,w_3} \le \ell_y^G - \hat{\kappa}$. Now, observe that $\mathsf{orig}(j, w_1) \le \mathsf{orig}(j, w_3)$ and hence

$$r_j + p + r_x^P = r_j + \mathsf{orig}(j, w_1)\tau + p + r_x^F < r_j + \mathsf{orig}(j, w_3)\tau + \ell_y^G - \hat{\kappa} = \ell_y^T - \hat{\kappa}.$$

Thus, in this case, both considered intersections are empty.

Otherwise, $\mathcal{M}_x$ and $\mathcal{M}_y$ coincide. We then have

$$r_j + \mathsf{orig}(j, w_1)\tau + [\,p + \ell_x^F\,..\,p + r_x^F\,) = r_j + [\,p + \ell_x^P\,..\,p + r_x^P\,) = [\,r_j + p + \ell_x^P\,..\,r_j + p + r_x^P\,)$$

and

$$r_j + \mathsf{orig}(j, w_1)\tau + [\,\ell_y^G - \hat{\kappa}\,..\,r_y^G + \hat{\kappa}\,) = r_j + [\,\ell_y^T - r_j - \hat{\kappa}\,..\,r_y^T - r_j + \hat{\kappa}\,) = [\,\ell_y^T - \hat{\kappa}\,..\,r_y^T + \hat{\kappa}\,).$$

The statement readily follows in the considered case. ⌟

Let $\mathcal{B} = (f_v, g_v)_{v=0}^{u}$. For each $L_i^F \in \mathcal{D}'(p)$, let $[\,a_i^F\,..\,b_i^F\,) \subseteq [\,0\,..\,u\,]$ so that $L_i^F = F[\,f_{a_i^F}\,..\,f_{b_i^F}\,)$ and $\mathcal{B}(L_i^F) = G[\,g_{a_i^F}\,..\,g_{b_i^F}\,)$. Symmetrically, for each $L_i^G \in \mathcal{D}'(p)$, let $[\,a_i^G\,..\,b_i^G\,) \subseteq [\,0\,..\,u\,]$ so that $L_i^G = G[\,g_{a_i^G}\,..\,g_{b_i^G}\,)$ and $\mathcal{B}^{-1}(L_i^G) = F[\,f_{a_i^G}\,..\,f_{b_i^G}\,)$. Further, let $\mathcal{E}$ denote the multiset union of the multisets

$$\mathcal{E}^F = \{[\,a_i^F\,..\,b_i^F\,) : L_i^F \in \mathcal{L}^F \cap \mathcal{D}'(p)\} \quad \text{and} \quad \mathcal{E}^G = \{[\,a_i^G\,..\,b_i^G\,) : L_i^G \in \mathcal{L}^G \cap \mathcal{D}'(p)\}.$$

In fact, the following claim implies that the multiplicity of each element of $\mathcal{E}$ is one.

Claim 8.40 [CKW22, see Claim 6.23]. *The intervals in $\mathcal{E}$ are pairwise disjoint.*

Proof. First, observe that the elements of each of $\mathcal{L}^F$ and $\mathcal{L}^G$ are pairwise disjoint and hence the elements of each of $\mathcal{E}^F$ and $\mathcal{E}^G$ are pairwise disjoint.

Now, consider any two fragments $L_x^F \in \mathcal{D}'(p)$ and $L_y^G \in \mathcal{D}'(p)$. Observe that, since $L_x^P, L_y^T \in \mathcal{D}(r_j + p)$, we have $\mathsf{mk}(r_j + p, \hat{\kappa}, L_x^P, L_y^T) = 0$. Hence, $[\,\ell_y^T - \hat{\kappa}\,..\,r_y^T + \hat{\kappa}\,) \cap [\,r_j + p + \ell_x^P\,..\,r_j + p + r_x^P\,) = \varnothing$. By Claim 8.39, we also have $[\,g_{a_y^G} - \hat{\kappa}\,..\,g_{b_y^G} + \hat{\kappa}\,) \cap [\,p + f_{a_x^F}\,..\,p + f_{b_x^F}\,) = \varnothing$.

Since the cost of $\mathcal{B}$ is no more than k, $g_i \in [\,p + f_i - k\,..\,p + f_i + k\,]$ holds for all $i \in [\,0\,..\,u\,]$. In particular, we have $[\,g_{a_y^G} - k - 1\,..\,g_{b_y^G} + k\,] \supseteq [\,p + f_{a_y^G} - 1\,..\,p + f_{b_y^G} + 1\,)$. Since $[\,g_{a_y^G} - \hat{\kappa}\,..\,g_{b_y^G} + \hat{\kappa}\,) \supseteq [\,g_{a_y^G} - k - 1\,..\,g_{b_y^G} + k\,]$, we then have $[\,f_{a_y^G} - 1\,..\,f_{b_y^G} + 1\,) \cap [\,f_{a_x^F}\,..\,f_{b_x^F}\,) = \varnothing$, that is, $[\,f_{a_y^G}\,..\,f_{b_y^G}\,) \cap [\,f_{a_x^F}\,..\,f_{b_x^F}\,] = \varnothing$. This implies $[\,a_y^G\,..\,b_y^G\,) \cap [\,a_x^F\,..\,b_x^F\,) = \varnothing$; consequently, the intervals in $\mathcal{E}$ are pairwise disjoint. ⌟

$$\text{Set } \Lambda := \mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(r_j+p)}(Q)^*) + \sum_{i=2}^{\ell^P} \mathsf{ed}^w(L_i^P \to {}^*Q^*) + \sum_{i=i_1}^{i_2} \mathsf{ed}^w(L_i^T \to {}^*Q^*).$$

Claim 8.41 [CKW22, see Claim 6.24].

$$\sum_{L_i^F \in \mathcal{D}_j'(p)} \mathsf{ed}^w(L_i^F \to \mathcal{B}(L_i^F)) + \sum_{L_i^G \in \mathcal{D}_j'(p)} \mathsf{ed}^w(L_i^G \to \mathcal{B}^{-1}(L_i^G)) \ge \Lambda - 2\mathsf{mk}(r_j + p).$$

Proof. Using Lemma 8.27, we lower-bound each individual term of the left-hand side of the proved inequality. We consider three cases.

▫1 Consider some $L_i^G = G[\, p+\ell \,..\, p+r \,) \in \mathcal{D}'_j(p)$ and let $L_i^T = T[\, r_j+p+\ell' \,..\, r_j+p+r' \,)$. Since $\mathsf{ed}^w(F \to G[\, p \,..\, b \,)) \le k$, we have that $\mathcal{B}^{-1}(L_i^G)$ is a substring of $F[\, \max\{0,\ell-k\} \,..\, \min\{|F|, r+k\} \,)$. By Claim 8.38, L_i^G is $13\hat{\kappa}$-contained by a sequence $\mathcal{M}$ of contiguous non-plain pairs in $\mathcal{I}'_j$. Then, by Lemma 8.36, $F[\, \max\{0,\ell-k\} \,..\, \min\{|F|, r+k\} \,)$ only overlaps pairs of $\mathcal{I}'_j$ that are in $\mathcal{M}$ and hence it is a substring of $P[\, (\ell'-\ell)+\max\{0,\ell-k\} \,..\, (r'-r)+\min\{|F|, r+k\} \,)$, which in turn is a substring of $P[\, \max\{0,\ell'-k\} \,..\, \min\{|P|, r'+k\} \,)$. Thus, $\mathsf{ed}^w(L_i^G \to \mathcal{B}^{-1}(L_i^G)) \ge \mathsf{ed}^w(L_j^T \to {}^*\mathcal{Q}^*) - \mathrm{mk}(r_j+p, L_j^T)$ holds by Lemma 8.27.

▫2 Consider some $L_i^F = F[\, \ell \,..\, r \,) \in \mathcal{D}'_j(p) \setminus \{L_1^P\}$ and let $L_i^P = P[\, \ell' \,..\, r' \,)$. Since $\mathsf{ed}^w(F \to G[\, p \,..\, b \,)) \le k$, we that have $\mathcal{B}(L_i^F)$ is a substring of $G[\, \max\{p, p+\ell-k\} \,..\, \min\{|G|, p+r+k\} \,)$. As before, by combining Claim 8.38 and Lemma 8.36 we get that $G[\, \max\{p, p+\ell-k\} \,..\, \min\{|G|, p+r+k\} \,)$ is a substring of $T[\, r_j+(\ell'-\ell)+\max\{p, p+\ell-k\} \,..\, r_j+(r'-r)+\min\{|G|, p+r+k\} \,)$, which in turn is a substring of $T[\, r_j+p+\max\{0,\ell'-k\} \,..\, r_j+p+\min\{|T|, r'+k\} \,)$. Thus, $\mathsf{ed}^w(L_i^F \to \mathcal{B}(L_i^F)) \ge \mathsf{ed}^w(L_i^P \to {}^*\mathcal{Q}^*) - \mathrm{mk}(r_j+p, L_i^P)$ holds by Lemma 8.27.

▫3 Lastly, since $\mathsf{ed}^w(F \to G[\, p \,..\, b \,)) \le k$, we have that $\mathcal{B}(L_1^F)$ is a prefix of $T[\, r_j+p \,..\, r_j+p+|L_1^P|+k \,)$, and hence $\mathsf{ed}^w(L_1^F \to \mathcal{B}(L_1^F)) \ge \mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(r_j+p)}(Q)^*) - \mathrm{mk}(r_j+p, L_1^P)$ holds by Lemma 8.27.

We now put everything together. In the following inequalities, we use the fact that, for each $L \in (\mathcal{L}^P \cup \mathcal{L}^T_{[\, r_j+p \,..\, r_j+p+m \,)}) \setminus \mathcal{D}(r_j+p, B)$, we have $\mathsf{ed}^w(L \to {}^*\mathcal{Q}^*) \le \mathrm{mk}(r_j+p, L)$, and hence the sum of $\mathsf{ed}^w(L \to {}^*\mathcal{Q}^*) - \mathrm{mk}(r_j+p, L)$ over all such L is at most zero.

$$
\begin{aligned}
&\sum_{L_i^F \in \mathcal{D}'_j(p)} \mathsf{ed}^w(L_i^F \to \mathcal{B}(L_i^F)) + \sum_{L_i^G \in \mathcal{D}'_j(p)} \mathsf{ed}^w(\mathcal{B}^{-1}(L_i^G) \to L_i^G) \\
&\quad \ge \mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(r_j+p)}(Q)^*) - \mathrm{mk}(r_j+p, L_1^P) + \sum_{L_i^P \in \mathcal{D}(r_j+p) \setminus \{L_1^P\}} (\mathsf{ed}^w(L_i^P \to {}^*\mathcal{Q}^*) - \mathrm{mk}(r_j+p, L_i^P)) \\
&\qquad\qquad + \sum_{L_i^T \in \mathcal{D}(r_j+p)} (\mathsf{ed}^w(L_i^T \to {}^*\mathcal{Q}^*) - \mathrm{mk}(r_j+p, L_i^T)) \\
&\quad \ge \mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(r_j+p)}(Q)^*) + \sum_{i=2}^{\ell^P} \mathsf{ed}^w(L_i^P \to {}^*\mathcal{Q}^*) + \sum_{i=i_1}^{i_2} \mathsf{ed}^w(L_i^T \to {}^*\mathcal{Q}^*) - \sum_{i=1}^{\ell^P} \mathrm{mk}(r_j+p, L_i^P) \\
&\qquad\qquad - \sum_{i=i_1}^{i_2} \mathrm{mk}(r_j+p, L_i^T) \\
&\quad = \Lambda - \sum_{i=1}^{\ell^P} \sum_{v=1}^{\ell^T} \mathrm{mk}(r_j+p, \hat{\kappa}, L_v^P, L_i^T) - \sum_{i=i_1}^{i_2} \sum_{v=1}^{\ell^P} \mathrm{mk}(r_j+p, \hat{\kappa}, L_v^P, L_i^T) \\
&\quad \ge \Lambda - 2\mathrm{mk}(r_j+p).
\end{aligned}
$$

This concludes the proof of the claim. ⌟

Next, due to Lemma 8.31, we have $\mathsf{ed}^w(L_1^P \to \mathrm{rot}^{-\rho(r_j+p)}(Q)^*) < \partial_P$. By a direct application of Lemma 8.23, we then have that $\min_t \mathsf{ed}^w(P \to T[\, r_j+p \,..\, t \,)) \le \Lambda$. If $\Lambda \le k$, then we are done. For the remainder of the proof we thus consider the case where $\Lambda > k$, which, combined with the fact that r_j+p is a light position of T, means that $\Lambda - 2\mathrm{mk}(r_j+p) > k - 2\mathrm{mk}(r_j+p) \ge k - 2\eta$.

⌜ Claim 8.42 [CKW22, see Claim 6.25]. *For any run $M = (F_{j,i_1}, G_{j,i_1}) \cdots (F_{j,i_2}, G_{j,i_2})$ of $2\eta+30$ consecutive plain pairs in $\mathcal{I}'_j$, there exists some $i \in (\, i_1 \,..\, i_2 \,)$ for which $\mathsf{ed}^w(F[\, f_{j,i} \,..\, f_{j,i+1} \,) \to \mathcal{B}(F[\, f_{j,i} \,..\, f_{j,i+1} \,))) = 0$.*

Proof. We have $|f_{j,i_1+14} - f_{j,i_1}| \ge 14\tau - d_T \ge 14\kappa/2 - d_T \ge 6\kappa = \Delta$. Similarly, $|f_{j,i_2-13} - f_{j,i_2+1}| \ge \Delta$. Thus, $U = F[\, f_{j,i_1+14} \,..\, f_{j,i_2-13} \,)$ is Δ-contained by M and hence only overlaps pairs in M by Fact 8.35.

Now, each fragment $L_i^F \in \mathcal{L}^F$ is $13\hat{\kappa}$-contained by a sequence of contiguous non-plain pairs and hence only overlaps non-plain pairs. Further, for each fragment $L_i^G \in \mathcal{L}^G$, as shown in the proof of Claim 8.41, $\mathcal{B}^{-1}(L_i^G)$ only overlaps non-plain pairs. Observe that two fragments of F are necessarily disjoint if the sets of pairs that they overlap are disjoint, and hence $[\, f_{j,i_1+14} \,..\, f_{j,i_2-13} \,)$ is disjoint from all elements of $\mathcal{E}$. As the intervals in $\mathcal{E}$ are pairwise disjoint by Claim 8.40, using Claim 8.41, we obtain

$$\begin{aligned} k &\geq \mathsf{ed}^w(F \to G[\, p \,..\, b \,)) \\ &\geq \mathsf{ed}^w(U \to \mathcal{B}(U)) + \sum_{L_i^F \in \mathcal{D}'_j(p)} \mathsf{ed}^w(L_i^F \to \mathcal{B}(L_i^F)) + \sum_{L_i^G \in \mathcal{D}'_j(p)} \mathsf{ed}^w(\mathcal{B}^{-1}(L_i^G) \to L_i^G) \\ &\geq \mathsf{ed}^w(U \to \mathcal{B}(U)) + \Lambda - 2\mathsf{mk}(r_j + p) \\ &> \mathsf{ed}^w(U \to \mathcal{B}(U)) + k - 2\eta. \end{aligned}$$

Hence, we have $\mathsf{ed}^w(U \to \mathcal{B}(U)) < 2\eta$. This implies that

$$\sum_{i=i_1+14}^{i_2-14} \mathsf{ed}^w(F[\, f_{j,i} \,..\, f_{j,i+1} \,) \to \mathcal{B}(F[\, f_{j,i} \,..\, f_{j,i+1} \,))) < 2\eta,$$

and hence, since the number of summands in in the left-hand side of the inequality is $i_2 - 14 - (i_1 + 14) + 1 = 2\eta + 30 - 28 > \eta$ and each of these summands is a non-negative integer, the claim follows. ⌟

We are now ready to conclude the proof of the lemma. Let $\mathcal{M}_1, \ldots, \mathcal{M}_w$ denote the trimmed runs of $2\eta + 30$ consecutive plain pairs in $\mathcal{I}'_j$ such that, for all i, $\mathcal{M}_i$ originated from a run with $\chi(i)$ more plain pairs. For $i \in [\,1 \,..\, w\,]$, let $\mathcal{M}_i = (F_{j,s_i}, G_{j,s_i}) \cdots (F_{j,e_i}, G_{j,e_i})$ let $\psi(i) \in (\, s_i \,..\, e_i \,)$ be such that and $\mathsf{ed}^w(F[\, f_{j,\psi(i)} \,..\, f_{j,\psi(i)+1} \,) \to \mathcal{B}(F[\, f_{j,\psi(i)} \,..\, f_{j,\psi(i)+1} \,))) = 0$; observe that $\psi(i)$ exists due to Claim 8.42. Let us now show how to construct C given $\mathcal{B} = (f_v, g_v)_{v=0}^u$. We intuitively achieve this by inserting $\chi(i)$ copies of $Q^\infty[\,0 \,..\, \tau\,)$ after each of $F[\, f_{j,\psi(i)} \,..\, f_{j,\psi(i)+1} \,)$ and $\mathcal{B}(F[\, f_{j,\psi(i)} \,..\, f_{j,\psi(i)+1} \,))$ and aligning them without errors, thus restoring the original length of each trimmed plain run, without changing the cost of the alignment. Initially, set $C := \mathcal{B}$. Then, for each $\psi(i)$, in decreasing order

- replace each pair (f_v, g_v) of C that satisfies $f_v \geq f_{j,\psi(i)+1}$ with $(\chi(i) \cdot \tau + f_v, \chi(i) \cdot \tau + g_v)$, and
- insert $(f_{j,\psi(i)+1} + \phi, g_{j,\psi(i)+1} + \phi)_{\phi=0}^{\chi(i)\cdot\tau-1}$ after $(f_{j,\psi(i)+1} - 1, g_{j,\psi(i)+1} - 1)$. ⌟

We conclude with the proof of Lemma 8.43 as promised.

Lemma 8.43 [CKW22, see Lemma 6.11]**.**

$$\mathrm{Occ}_k^w(P,T) \cap \mathbb{L} = \Big(\bigcup_{j \in J} (r_j + \mathrm{Occ}_k^w(\mathcal{I}'_j))\Big) \cap \mathbb{L} = \Big(\bigcup_{j \in J} (r_j + \mathrm{Occ}_k^w(F_j, G_j))\Big) \cap \mathbb{L}.$$

Proof. ($\subseteq$): This direction is an immediate consequence of Corollary 8.30.

($\supseteq$): This direction is an immediate consequence of Lemma 8.37. ⌟

8.6 Combining the Partial Results: Faster AlignedPeriodicMatches

We are ready to prove the main result of this section—Lemma 8.44—which we restate here for convenience.

Lemma 8.44. *AlignedPeriodicMatches can be solved in $\widetilde{O}(k^{3.5}W^4)$ time in the* PILLAR *model if w is an integer metric weight function, with the output set $\mathrm{Occ}_k^w(P,T)$ represented as a collection of disjoint arithmetic progressions.*

Proof. If $W > \sqrt{k}$, Lemma 7.40 yields a better running time, so we henceforth assume $W \leq \sqrt{k}$. Consistently with all previous sections, set

$$\kappa := k + d_P + d_T = \Theta(k) \quad \text{and} \quad \hat{\kappa} := 112(kW + d_P^w + d_T^w) = O(kW) \quad \text{and} \quad \eta := \max\{1, \lfloor \sqrt{k}W \rfloor\} \leq \hat{\kappa}.$$

We proceed in roughly four steps.

- First, in a preprocessing step, we identify the heavy positions $\mathbb{H}$ and the light positions $\mathbb{L}$ in T, as well as a filter $\mathcal{F}$ (according to Lemma 7.14) for where potential k-error occurrences may start.
- Next, we compute all k-error occurrences starting at a position in $\mathbb{H}$, using the algorithm from Section 8.4. In particular, we obtain $\mathbb{H} \cap \mathrm{Occ}_k^w(P,T)$ as a set of positions.
- Next, we use `TrimmedPM` from Theorem 7.37 to obtain a candidate set $\mathcal{R}$ of potential starting positions of k-error occurrences (represented as disjoint arithmetic progressions with difference τ). From Section 8.5, we have $(\mathcal{R} \cap \mathcal{F}) \setminus (\mathbb{H} \cap \mathcal{F}) = \mathrm{Occ}_k^w(P,T) \cap \mathbb{L}$; hence we proceed to compute $\mathbb{H} \cap \mathcal{F}$ (represented as a set), and $\mathcal{R} \cap \mathcal{F}$ (represented as disjoint arithmetic progressions with difference τ), and thereafter compute their set difference to obtain $\mathrm{Occ}_k^w(P,T) \cap \mathbb{L}$ (represented as disjoint arithmetic progressions with difference τ).
- In a post-processing step, we union the two sets $\mathbb{H} \cap \mathrm{Occ}_k^w(P,T)$ and $\mathrm{Occ}_k^w(P,T) \cap \mathbb{L}$ and compute a representation as disjoint arithmetic progressions with difference τ.

Preprocessing. We compute the sets of locked fragments $\mathcal{L}^P = \mathtt{Locked}(P, Q, d_P^w, \partial_P, w)$ and $\mathcal{L}^T = \mathtt{Locked}(T, Q, d_T^w, 0, w)$ in $O(d^2)$ time using Corollary 8.13 and call $\mathtt{Heavy}(P, T, d, k, Q, \mathcal{L}^P, \mathcal{L}^T, \hat{\kappa}, \eta)$ to obtain the set $\mathbb{H}$ of heavy positions, represented as the union of $O(k^2W^2)$ disjoint integer ranges, in $\widetilde{O}(k^2W^3)$ time (cf. Lemma 8.18).

Define J and each of r_j and $\mathcal{I}'_j$, for $j \in J$, as in Lemma 7.14 and Definition 8.29, and set

$$\mathcal{R} \coloneqq \bigcup_{j\in J}(r_j + \mathrm{Occ}_k^w(\mathcal{I}'_j)).$$

Observe that, by Lemma 8.43,

$$\mathrm{Occ}_k^w(P,T) = (\mathrm{Occ}_k^w(P,T) \cap \mathbb{H}) \cup (\mathrm{Occ}_k^w(P,T) \cap \mathbb{L}) = (\mathrm{Occ}_k^w(P,T) \cap \mathbb{H}) \cup (\mathcal{R} \cap \mathbb{L}).$$

Finally, for our filter, Lemma 7.14 and Fact 3.7 yield

$$\mathcal{F} \coloneqq \bigcup_{j\in\mathbb{Z}}[\, jq - x_T - \kappa - d_T \mathinner{..} jq - x_T + \kappa + d_T \,] \supseteq \mathrm{Occ}_k(P,T) \supseteq \mathrm{Occ}_k^w(P,T).$$

Observe that we thus have

$$\mathrm{Occ}_k^w(P,T) \cap \mathbb{L} = (\mathcal{R} \cap \mathcal{F}) \setminus (\mathbb{H} \cap \mathcal{F}). \tag{4}$$

Heavy occurrences. By Lemma 8.20, we can compute $\mathrm{Occ}_k^w(P,T) \cap \mathbb{H}$ in time $\widetilde{O}((kW/\eta + 1)k^3W^3) = \widetilde{O}(k^{3.5}W^4)$.

Light occurrences. Let us now proceed to computing the remaining (k,w)-error occurrences, namely those that start at light positions. To this end, we first compute the $O(kW^2)$-size sets $\mathsf{Red}(P)$ and $\mathsf{Red}(T)$, defined in the beginning of Section 8.5, in $\widetilde{O}(kW^2)$ time (cf. Lemma 8.28). Then, we call $\mathtt{TrimmedPM}(T, P, k, Q, \mathcal{A}_P, \mathcal{A}_T, \mathsf{Red}(P), \mathsf{Red}(T), 2\eta + 30)$ from Theorem 7.37, which returns a representation of $\mathcal{R}$ as $O(k^3W^4 \cdot \eta) = O(k^{3.5}W^4)$ arithmetic progressions with difference $\tau \coloneqq q\lceil \kappa/2q \rceil$. Plugging in Theorem 9.10 for the DynamicPuzzleMatching data structure, `TrimmedPM` runs in time

$$\widetilde{O}(k^3W + k \cdot k^2W^4\eta + k^2W^4 \cdot kW) = \widetilde{O}(k^{3.5}W^4 + k^3W^5) = \widetilde{O}(k^{3.5}W^4).$$

We move on to remove surplus positions from the candidate set $\mathcal{R}$. We start by computing $\mathbb{H} \cap \mathcal{F}$.

Claim 8.45. *The set $\mathbb{H} \cap \mathcal{F}$ is of size $O(k^{2.5}W^3)$ and can be computed in time $\widetilde{O}(k^{2.5}W^3)$.*

Proof. The proof is similar to a part of the proof of Lemma 8.20.

Due to Lemma 8.18, our representation of $\mathbb{H}$ consists of $O(k^2W^2)$ disjoint integer ranges. In addition, due to Lemma 8.19, we have

$$|\mathbb{H}| = O((kW/\eta + 1)kW(\hat{\kappa} + q)) = O(k^{1.5}W^2 \cdot (kW + q)).$$

Let us first upper-bound the size of $|\mathbb{H} \cap \mathcal{F}|$. We distinguish between two cases.

- First, if $q \leq kW$, we have $|\mathbb{H} \cap \mathcal{F}| \leq |\mathbb{H}| = \mathcal{O}(k^{2.5}W^3)$.
- As for the complementary case where $q > kW$, observe that only $\mathcal{O}(k)$ out of any $\mathcal{O}(q)$ consecutive integers are in $\mathcal{F}$. Hence, $\mathbb{H} \cap \mathcal{F}$ is of size $\mathcal{O}(k^2W^2 + |\mathbb{H}| \cdot k/q) = \mathcal{O}(k^2W^2 + k^{2.5}W^2) = \mathcal{O}(k^{2.5}W^3)$.

As for computing $\mathbb{H} \cap \mathcal{F}$, we first sort the $\mathcal{O}(k^2W^2)$ integer ranges that comprise $\mathbb{H}$ with respect to their starting positions in $\widetilde{\mathcal{O}}(k^2W^2)$ time and then scan them from left to right, skipping positions in $\mathbb{Z} \setminus \mathcal{F}$. The running time of the scan is proportional to the total number of input ranges and output positions, and hence we are done. ⌟

We move on to compute $\mathcal{R} \cap \mathcal{F}$. In what follows, for any integer x, let us say that the *residue modulo* x of a non-empty arithmetic progression whose difference is a multiple of x is the residue of any element of this arithmetic progression modulo x.

⌜ Claim 8.46. *We can compute a representation of $\mathcal{R} \cap \mathcal{F}$ as $\mathcal{O}(k^{3.5}W^4)$ disjoint arithmetic progressions with difference τ, sorted according to their starting positions, in $\widetilde{\mathcal{O}}(k^{3.5}W^4)$ time.*

Proof. In a linear scan of the $\mathcal{O}(k^{3.5}W^4)$ arithmetic progressions that comprise $\mathcal{R}$ as returned by the call to the algorithm `TrimmedPM`, we delete any arithmetic progression whose elements are not in $\mathcal{F}$. Then, we sort all arithmetic progressions according to their starting positions in time $\widetilde{\mathcal{O}}(k^{3.5}W^4)$ and distribute them among $\mathcal{O}(k/q \cdot \tau) = \mathcal{O}(k^2)$ buckets according to their residues modulo τ. Finally, we scan linearly the arithmetic progressions in each bucket, greedily merging progressions that overlap (that is, we merge progressions if their union is also a valid arithmetic progression with difference τ). The number of the resulting arithmetic progressions is clearly upper-bounded by the size of the input, that is, $\mathcal{O}(k^{3.5}W^4)$; we sort them according to their starting positions in $\mathcal{O}(k^{3.5}W^4)$ time. ⌟

Finally, we compute the set difference $(\mathcal{R} \cap \mathcal{F}) \setminus (\mathbb{H} \cap \mathcal{F}) = \mathrm{Occ}_k^w(P,T) \cap \mathbb{L}$.

⌜ Claim 8.47. *A representation of $\mathrm{Occ}_k^w(P,T) \cap \mathbb{L}$ as $\mathcal{O}(k^{3.5}W^4)$ disjoint arithmetic progressions with difference τ, sorted according to their starting positions, can be computed in time $\widetilde{\mathcal{O}}(k^{3.5}W^4)$ time the* `PILLAR` *model.*

Proof. We intend to use equation (4), relying on Claims 8.45 and 8.46 to compute $\mathbb{H} \cap \mathcal{F}$ and a representation of $\mathcal{R} \cap \mathcal{F}$ as $\mathcal{O}(k^{3.5}W^4)$ disjoint arithmetic progressions with difference τ in $\widetilde{\mathcal{O}}(k^{3.5}W^4)$ time in total. Then, we process the elements of these two sets in $\mathcal{O}(d/q \cdot \tau) = \mathcal{O}(k^2)$ batches, where each batch contains all elements with a specific residue modulo τ. For such a fixed residue, we scan in parallel the arithmetic progressions in $\mathcal{R} \cap \mathcal{F}$ and the positions in $\mathbb{H} \cap \mathcal{F}$, both of which are sorted in increasing order. When some element of $\mathbb{H} \cap \mathcal{F}$ is contained in some arithmetic progression in $\mathcal{R} \cap \mathcal{F}$, this arithmetic progression is split into two parts, either of which may be empty. The number of the resulting arithmetic progressions is upper-bounded by the total size of the input, that is, $\mathcal{O}(k^{3.5}W^4)$; we sort them according to their starting positions in $\widetilde{\mathcal{O}}(k^{3.5}W^4)$ time. ⌟

This concludes the proof of this lemma. ⌟

9 Fast Data Structures for (Bleach-Commit) Dynamic Puzzle Matching

9.1 On Matrix Multiplication

Before we discuss how to efficiently compute ferns, it is instructive to take a step back to abstractly describe two different schemes that we use to efficiently *multiply* several matrices.

To this end, we introduce the problem DynamicMatrixMultiplication, which is of a similar flavor compared to DynamicPuzzleMatching. The main difference of DynamicMatrixMultiplication and DynamicPuzzleMatching is that the former directly operates on matrices, while the latter operates on puzzle pieces (that is, pairs of strings).

Problem DynamicMatrixMultiplication.

Maintained Object. A sequence of matrices $\mathcal{F} := F_1, \ldots, F_z \in \mathbb{R}_{\geq 0}^{q \times q}$ that share a common matrix property P; see Definition 5.31.

Initialization. `DMM-Init(`q`,` $\mathcal{F}$`)`: Given a dimension q and an initial sequence $\mathcal{F}$ of matrices that satisfy P, initialize the sequence of matrices to $\mathcal{F}$.

Update Operations. `DMM-Delete(`i`)`: Delete the i-th element of $\mathcal{F}$.

`DMM-Insert(`F`,` i`)`: Insert a matrix $F \in \mathbb{R}_{\geq 0}^{q \times q}$ with property P after the i-th element of $\mathcal{F}$.

`DMM-Substitute(`F`,` i`)`: Substitute the i-th element of $\mathcal{F}$ with a matrix $F \in \mathbb{R}_{\geq 0}^{q \times q}$ with property P.

`DMM-Split(`i`)`: Split the sequence after the i-th element and return an instance of the data structure for the prefix and an instance of the data structure for the suffix.

`DMM-Merge(`$\mathcal{F}'$`)`: Given an instance of DynamicMatrixMultiplication representing $\mathcal{F}'$ (a sequence of matrices with property P), append the sequence $\mathcal{F}'$ to the sequence $\mathcal{F}$.

Query Operations. `DMM-Matrix()`: Return $F_1 \oplus \cdots \oplus F_z$.

`DMM-Vector(`v`)`: Given a vector $v \in \mathbb{R}_{\geq 0}^{q \times 1}$, return $F_1 \oplus \cdots \oplus F_z \oplus v$.

We discuss two different implementations for DynamicMatrixMultiplication; we start with an implementation that allows for fast `DMM-Matrix` operations at the cost of only moderately fast update operations.

Lemma 9.1. *Let us fix a matrix property P with $T_M(q)$-time matrix multiplication and $T_V(q)$-time vector multiplication.*

Then, there is an implementation for DynamicMatrixMultiplication such that

- `DMM-Init(`q`,` $\mathcal{F}'$`)` *takes time $O(|\mathcal{F}'| \cdot T_M(q))$.*
- *Any update operation takes time $O(\log |\mathcal{F}| \cdot T_M(q))$.*
- `DMM-Matrix()` *takes time proportional to the time required to write down the output (that is, typically $O(q^2)$).*
- `DMM-Vector(`v`)` *takes time $O(T_V(q))$ plus time proportional to the time required to write down the output (that is, typically $O(q)$).*

Additionally, we support an operation `DMM-Root()` *with constant-time read-only access to a pointer to a matrix corresponding to the result of* `DMM-Matrix()`.

Proof. We maintain a balanced binary tree; we store F_i at leaf i and at an internal vertex we store the (min, +)-product of its (up to) two children.

At initialization, we build the tree in a bottom-up fashion using a linear number of (min, +)-products in total.

For updates, we insert, delete, or substitute the corresponding element of the sequence and then update its (former) parents and rebalance as necessary. Further, binary trees readily support both split and merge operations using a logarithmic number of matrix multiplications.

For the query operations, we either return the root of our tree (as we also do for the extra `DMM-Root()`) or the root multiplied with the given vector.

Associativity of the (min, +)-product ensures correctness; the running time guarantees follow from basic properties of balanced binary trees.

Secondly, we discuss an even easier implementation with faster update and `DMM-Vector` operations at the cost of slower `DMM-Matrix` operations.

Lemma 9.2. *Let us fix a matrix property P with $T_M(q)$-time matrix multiplication and $T_V(q)$-time vector multiplication.*

Then, there is an implementation for DynamicMatrixMultiplication *such that*

- `DMM-Init(`q`,` $\mathcal{F}'$`)` *takes time* $O(|\mathcal{F}'|)$ *(where we assume that input matrices do not need to be copied).*
- *Any update operation takes time* $O(1)$ *(where we assume that input matrices do not need to be copied).*
- `DMM-Matrix()` *takes time* $O(|\mathcal{F}| \cdot T_M(q))$.
- `DMM-Vector(`v`)` *takes time* $O(|\mathcal{F}| \cdot T_V(q))$.

Proof. We maintain a (linked) list of pointers to the input matrices. For updates, we insert, delete, or substitute the corresponding element of the sequence by updating pointers or concatenating the (linked) lists of pointers.

Now, for the query operations, we naively multiply the elements of the sequence. For the matrix operation, this involves $|\mathcal{F}| - 1$ multiplications; for the vector operation, we exploit associativity and compute $F_1 \oplus \cdots \oplus F_z \oplus v$ as

$$F_1 \oplus (\cdots \oplus (F_z \oplus v) \cdots);$$

that is, we start at the right and always (min, +)-multiply a matrix and a vector.

Now, both correctness and running time guarantees are immediate. ◢

9.2 Growing and Caring for Large Ferns

In the setting of BleachCommitDPM, we unfortunately have no direct guarantees on the lengths of the input strings. Fortunately, we are able to show that the guaranteed bound on the summed edit distance implies that there are large parts of the input strings that repeat in many strings. This in turn allows us to split the input strings into shorter strings, compute ferns for the shorter strings using Lemma 6.14, and then obtain ferns for the original strings by applying Lemma 9.1.

Theorem 9.3. *For any family of strings $\mathcal{S}$ (of maximum length n) such that $\mathsf{ed}(\mathcal{S}) \leq d$, as well as a positive integer k, there is an algorithm that given $\mathcal{S}$, d, k, and oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$, computes for each pair $(\dot{P}, \dot{T}) \in \mathcal{S} \times \mathcal{S}$*[21]

- *a Monge* $(\min(|\dot{T}|, d + k), k)$*-fern in total time* $O((d + k)^4 \log^2(n(d + k)))$ *in the* PILLAR *model; and*
- *a* $(W + 1)$*-bounded difference Monge* $(\min(|\dot{T}|, d + k), k)$*-fern in total time* $O(W(d + k)^3 \log^2(n(d + k)))$ *in the* PILLAR *model if all weights of w are integer.*

Proof. Set $\nabla := 8d + 2k$.

Now, write $\hat{S}$ for a 2-approximate center of $\mathcal{S}$. That is, we have $\sum_{\dot{S} \in \mathcal{S}} \mathsf{ed}(\hat{S}, \dot{S}) \leq 2\mathsf{ed}(\mathcal{S})$.[22] Further, for each $\dot{S} \in \mathcal{S}$ write $\mathcal{A}_{\dot{S}} : \hat{S} \rightsquigarrow \dot{S}$ for an optimal alignment.

Next, for a fragment $V = U[i..j]$ of a string U, by $V^{+\ell}$, we denote the fragment $U[i..\min\{j+\ell\}, |U|)$.

We start with structural characterizations and then discuss how to turn said characterizations into efficient algorithms.

Claim 9.4 (Decomposing $\hat{S}$). *There is a decomposition $\hat{S} = \bigcirc_{i=0}^{\tau-1} \hat{S}_i$ such that all of the following hold.*

1. *We have $\tau \in O(d)$.*
2. *For every $i \in [0..\tau - 1)$, we have $|\hat{S}_i| \geq \nabla$.*
3. *For every $i \in [0..\tau)$, either $|\hat{S}_i| = O(\nabla)$ or, for all $\dot{S} \in \mathcal{S}$, we have $\mathcal{A}_{\dot{S}}(\hat{S}_i^{+\nabla}) = \hat{S}_i^{+\nabla}$.*

[21] Recall from Remark 3.9 that $\mathsf{ed}(\mathcal{S}) \leq d$ implies $|\mathcal{S}| = O(d)$.

[22] It is worth highlighting that the approximate center is with respect to the unweighted edit distance.

Proof. If $|\hat{S}| < 100\nabla$, we return it as-is.

Otherwise, we decompose $\hat{S}$ into length-∇ fragments (where the last fragment has a length in $[2\nabla..4\nabla]$); $\hat{S} := \bar{S}_0 \cdots \bar{S}_{\bar{\tau}-1}$. In particular, we have $|\bar{S}_i| = \nabla$ for all $i \in [0..\bar{\tau}-1)$.

Observe that each $\bar{S}_i$ satisfies Item 3 due to $|\bar{S}_i| = O(\nabla)$. However, $\bar{\tau}$ might not satisfy Item 1.

Hence, let us say that an integer $i \in [0..\bar{\tau})$ is *dirty* if and only if $\bar{S}_i \neq \mathcal{A}_{\dot{S}}(\bar{S}_i)$ for any $\dot{S} \in \mathcal{S}$. An integer is *pure* if it is not dirty. Now, for each maximal interval $[i..j] \subseteq [0..\bar{\tau})$ of pure integers, merge tiles with indices in $[i..j-1]$ in the natural order and re-index the resulting tiles.

Now, for a dirty i, the string $\bar{S}_i$ remains unchanged (albeit with a potentially new index) and has all claimed properties. Further, we only ever merge a run of pure integers; which ensures that the second condition of Item 3 is satisfied. In total, we thus obtain the claim. ⌟

If $|\hat{S}| < 100\nabla$, in the following, we construct $(\min(|\dot{T}|, d+k), k)$-ferns; only in this case we might have $|\dot{T}| < d+k$. For notational simplicity, we hence ignore this case in our notation from here onward.

Next, we prove useful properties of the decomposition of Claim 9.4. To this end, for each $i \in [0..\tau)$, for each $\dot{S} \in \mathcal{S}$, set $\dot{S}_i := \mathcal{A}_{\dot{S}}(\hat{S}_i)$; and set $\mathcal{S}_i := \{\dot{S}_i^{+\nabla} : \dot{S} \in \mathcal{S}\}$.

We refer to $\mathcal{S}_i$ as the i-th tile collection. Further, we say that an index i is *pure* if $|\mathcal{S}_i| = 1$. Otherwise, i is *dirty*.

It is instructive to highlight an easy property of the decomposition of Claim 9.4.

Claim 9.5. *For all dirty i, we have $|\dot{S}| = O(\nabla)$ for all $\dot{S} \in \mathcal{S}_i$.* ⌟

Next, we bound the total size of tile collections of dirty indices.

Claim 9.6. *If i is pure, then $\mathsf{ed}(\mathcal{S}_i) = 0$. Further, we have $\sum_i \mathsf{ed}(\mathcal{S}_i) = O(d)$.*

Proof. First, for pure i, the corresponding tile collections contain only a single element each; hence the claim follows.

For the second claim, we prove that for all $\dot{S} \in \mathcal{S}$, we have

$$\sum_{i=0}^{\tau-1} \mathsf{ed}(\hat{S}_i^{+\nabla}, \dot{S}_i^{+\nabla}) \le 3\mathsf{ed}(\hat{S}, \dot{S}).$$

To this end, observe that aligning all $\hat{S}_i$'s with $\dot{S}_i$'s costs $\mathsf{ed}(\hat{S}, \dot{S})$ by construction as partitions of $\hat{S}$ and $\dot{S}$.

Next, we bound the contribution of the overlaps. To this end, observe that aligning $\hat{S}_i^{+\nabla} \setminus \hat{S}_i$ with $\dot{S}_i^{+\nabla} \setminus \dot{S}_i$ costs $2\mathsf{ed}(\hat{S}, \dot{S})$, as in addition to the original edits (which we have to account for again), the alignment $\mathcal{A}_S$ might change the lengths of the overlaps by at most $\mathsf{ed}(\hat{S}, \dot{S})$ characters in total. ⌟

In particular, Claim 9.6 implies that $\sum_{i=0}^{\tau-1} |\mathcal{S}_i| = O(d)$ (recall Remark 3.9).

Next, we argue that we obtain good puzzles out of our tile collections.

Claim 9.7 (Bounded torsion). *For every $\dot{P}, \dot{T} \in \mathcal{S}$, the torsion of the ∇-puzzle $(P_0^{+\nabla}, T_0^{+\nabla}), \ldots, (P_{\tau-1}^{+\nabla}, T_{\tau-1}^{+\nabla})$ is at most $4d = \nabla/2 - k$.*

Proof. For each $i \in [0..\tau)$, we have

$$\|P_i^{+\nabla}| - |T_i^{+\nabla}\| = \|P_i| - |T_i\| \le \mathsf{ed}(P_i, T_i) \le \mathsf{ed}(\hat{S}_i, P_i) + \mathsf{ed}(\hat{S}_i, T_i).$$

Hence, as $\hat{S}$ is a 2-approximate center of $\mathcal{S}$, we obtain

$$\mathsf{tor}((P_0^{+\nabla}, T_0^{+\nabla}), \ldots, (P_{\tau-1}^{+\nabla}, T_{\tau-1}^{+\nabla})) = \sum_{i=0}^{\tau-1} \|P_i^{+\nabla}| - |T_i^{+\nabla}\| \le 2d + 2d = 4d.$$ ⌟

Claim 9.8. *Fix $\dot{P}, \dot{T} \in \mathcal{S}$ and write ${}_{\circ}F_k^{P_0,T_0}, {}_{\circ}^{\circ}F^{P_1,T_1} \dots, {}_{\circ}^{\circ}F_k^{P_{\tau-2},T_{\tau-2}}, {}^{\circ}F_k^{P_{\tau-1},T_{\tau-1}}$ for (∇, k)-ferns for the ∇-puzzle $\mathcal{PP} := (P_0^{+\nabla}, T_0^{+\nabla}), \dots, (P_{\tau-1}^{+\nabla}, T_{\tau-1}^{+\nabla})$. Then,*

$$F_k^{\dot{P},\dot{T}} := {}_{\circ}F_k^{P_0,T_0} \oplus {}_{\circ}^{\circ}F_k^{P_1,T_1} \oplus \cdots \oplus {}_{\circ}^{\circ}F_k^{P_{\tau-2},T_{\tau-2}} \oplus {}^{\circ}F_k^{P_{\tau-1},T_{\tau-1}}$$

is a (∇, k)-fern (and thus a $(d+k, k)$-fern after suitable trimming) for $(\dot{P}, \dot{T})$.

Proof. We wish to employ Theorem 5.25. To this end, it suffices to observe that indeed $\nabla/2 - \mathsf{tor}(\mathcal{PP}) \geq 4d + k - 4d = k$. Finally, for the trimming we use Lemma 5.20. ⌟

With our structural insights gathered, it is time to implement them. We start by recalling an algorithm to compute a 2-approximate center from [CKW22, Lemma 9.9].

Claim 9.9 [CKW22, Lemma 9.9]. *Given a string family $\mathcal{S}$, we can construct a string $\hat{S} \in \mathcal{S}$ with $\sum_{S \in \mathcal{S}} \mathsf{ed}(S, \hat{S}) \leq 2\mathsf{ed}(\mathcal{S})$ in $O(1 + \mathsf{ed}(\mathcal{S})^3)$ time in the* PILLAR *model. Further, in the same time, we can compute alignments $\mathcal{A}_S : \hat{S} \rightsquigarrow\!\!\!\!\rightarrow S$ for every $S \in \mathcal{S}$.*

Proof sketch. For each pair of strings $S, S' \in \mathcal{S}$, we use Lemma 3.20 to compute $\mathsf{ed}(S, S')$, as well as a corresponding alignment. We return the string $\hat{S}$ that minimizes $\sum_{S \in \mathcal{S}} \mathsf{ed}(S, \hat{S})$, as well as the corresponding alignments.

We obtain the correctness from the triangle inequality.

For the running time, we compute

$$O\Big(\sum_{S,S' \in \mathcal{S}} \mathsf{ed}(S, S')^2\Big) = O\Big(\sum_{S,S' \in \mathcal{S}} \mathsf{ed}(S, \hat{S})^2 + \mathsf{ed}(S', \hat{S})^2\Big) = O(|\mathcal{S}| + \mathsf{ed}(\mathcal{S})^2) = O(1 + \mathsf{ed}(\mathcal{S})^3).$$ ⌟

Now, we run the algorithm from Claim 9.9: we compute a 2-approximate center $\hat{S}$ and corresponding alignments $\hat{S} \rightsquigarrow\!\!\!\!\rightarrow S$ for every $S \in \mathcal{S}$. Using the alignments obtained from Claim 9.9 (which runs in time $O(1 + d^3)$ in our case), we then compute the decomposition of Claim 9.4, again in time $O(1 + d^3)$. This in turn then allows to compute the tile collections in the same time.

Next, for each tile collection $\mathcal{S}_i$, we compute ferns for each pair of strings $\dot{P}, \dot{T} \in \mathcal{S}_i$; the details differ depending on the guarantees that we have for the weight function w.

General weights. For each pure index i, we use `GrowFern` of Lemma 6.14, on (suitably trimmed versions of) $\hat{S}_i, \hat{S}_i, \nabla$, and w to obtain a Monge (∇, ∇) fern for this index. As there are at most $O(d)$ pure indices, this costs in total $O(d\nabla^3 \log^2(n\nabla)) = O((d+k)^4 \log^2(n(d+k)))$ time.

For each dirty index i, we use Lemma 6.1, on each pair of $\dot{P}, \dot{T} \in \mathcal{S}_i$. In total, we call Lemma 6.1 at most $O(d^2)$ times; each time on a pair of strings of total length $O(\nabla)$. Hence, this step takes in total $O(d^2\nabla^2 \log \nabla) = O((d+k)^4 \log(d+k))$ time.

Integer weights. For each pure index i, we use `GrowFern` of Lemma 6.14, on (suitably trimmed versions of) $\hat{S}_i, \hat{S}_i, \nabla$, and w to obtain a $(W+1)$-bounded difference, Monge (∇, ∇) fern for this index. As there are at most $O(d)$ pure indices, this costs in total $O(dW\nabla^2 \log^2(n\nabla) = O(W(d+k)^3 \log^2(n(d+k)))$ time. After computing the ferns, we use Lemma 5.4 to construct the corresponding core-based matrix oracles.

For the dirty indices, we proceed akin to [CKW22], except that we use the dynamic weighted edit distance data structure from [GK25, Theorem 5.10] (see Lemma 5.36). We initialize the data structure with the piece $(\hat{S}_i, \hat{S}_i)$. Next, for each pair $(\dot{P}_i, \dot{T}_i) \in \mathcal{S}_i \times \mathcal{S}_i$, we use the update operations to make the data structure represent $(\dot{P}_i, \dot{T}_i)$, followed by a query operation to obtain a corresponding fern. Observe that the data structure already outputs the ferns as a core-based matrix oracle.

Across all at most $O(d)$ dirty indices, the initializations take time $O(dW\nabla^2 \log \nabla) = O(W(d+k)^3 \log(d+k))$. Further, in total, we query all data structures at most $O(d^2)$ times; each such query is preceded by a constant number of calls to update operations. In total, this takes time $O(W\nabla d^2 \log^2 \nabla) = O(W(d+k)^3 \log^2(d+k))$. For constructing the core-based matrix oracles (for pure indices), we spend $O((d+k)^2 + W(d+k)\log(W(d+k)))$ time per pure index (of which there are at most $O(d)$ in total), which is dominated by the running time for constructing the ferns.

Finally, to compute a fern for each pair $\dot{P}, \dot{T} \in \mathcal{S} \times \mathcal{S}$, we use Lemma 9.1 and proceed as follows. We initialize the data structure with ferns for[23]

$$(\hat{S}_0^{+\nabla}, \hat{S}_0^{+\nabla}), \ldots, (\hat{S}_{\tau-1}^{+\nabla}, \hat{S}_{\tau-1}^{+\nabla}).$$

Next, we iterate over each pair $\dot{P}, \dot{T} \in \mathcal{S} \times \mathcal{S}$; substitute the corresponding ferns, call `DMM-Matrix` and undo the substitution by substituting the original ferns. We then use Lemma 5.20 to trim the obtained ferns to $(d+k, k)$-ferns.

We obtain different running times depending on the type of ferns that we multiply.

General weights. As all of our ferns are Monge, they have $O((d+k)^2)$-time multiplication; hence our in total $O(d^2)$ substitution and updates cost $O((d+k)^4 \log d)$ time in total.

In total, we obtain all desired ferns in time $O((d+k)^4 \log^2(n(d+k)))$; thereby completing the proof.

Integer weights. As all of our ferns are Monge and $(W+1)$-bounded difference, they have $O(W(d+k)\log(d+k))$-time multiplication; hence our in total $O(d^2)$ substitution and updates cost $O(W(d+k)^3 \log^2(d+k))$ time in total.

In total, we obtain all desired ferns in time $O(W(d+k)^3 \log^2(n(d+k)))$; thereby completing the proof. ◢

9.3 Using Ferns: (Bleach-Commit) Dynamic Puzzle Matching

Theorem 9.3 allows us to efficiently compute ferns for a given collection of tiles (out of which then the puzzle pieces are built). What remains is to show how to efficiently maintain a sequence of said ferns under the operations to be supported in BleachCommitDPM. This, we tackle next.

Theorem 9.10 (Efficient implementation for BleachCommitDPM)**.** *There is an implementation for BleachCommitDPM that satisfies all of the following.*

- `BCDPM-Init`$(\mathcal{Y}'_{\min J}, \alpha, k, \Delta, \mathcal{S}_\beta, \mathcal{S}_\mu, \mathcal{S}_\varphi, Q, w)$ *can be performed in time*
  - $O(k^4 \log^2(k\lambda) + |\mathcal{Y}'|k^2)$ *for general weights and in time*
  - $O(k^3 W \log^2(k\lambda) + |\mathcal{Y}'|kW \log k)$ *for integer weights,*

  where λ is the length of the longest string in $\mathcal{S}_\beta \cup \mathcal{S}_\mu \cup \mathcal{S}_\varphi$;
- *any call to* `BCDPM-Set` *can be performed in constant time;*
- *any call to* `BCDPM-Query` *can be performed in $O(k\rho)$ time for general weights and in $O(k\rho \log \Delta)$ time for integer weights, where ρ is the number of blank pieces at the time of the query;*
- *any call to any other operation can be performed in time*
  - $O(k^2 \log(k + |\mathcal{Y}|))$ *for general weights and in time*
  - $O(kW \log^2(k + |\mathcal{Y}|))$ *for integer weights,*

  where $\mathcal{Y}$ is the BCDPM sequence to which the operation is applied.

Proof. We maintain an instance of the data structure of Lemma 9.2 on ferns that we maintain using multiple copies of the data structure from Lemma 9.1.

[23] Technically, we consider a sequence of ferns $F_0, \ldots, F_{\tau-1}$ with

$$F_0 \in {}_\circ\mathcal{F}^w_{\hat{S}_0^{+\nabla}, \hat{S}_0^{+\nabla}}|^k_\nabla, \quad F_i \in {}^\circ_\circ\mathcal{F}^w_{\hat{S}_i^{+\nabla}, \hat{S}_i^{+\nabla}}|^k_\nabla, \quad F_{\tau-1} \in {}^\circ\mathcal{F}^w_{\hat{S}_{\tau-1}^{+\nabla}, \hat{S}_{\tau-1}^{+\nabla}}|^k_\nabla.$$

Initialization. Given k, Δ, and the families $\mathcal{S}_\beta$, $\mathcal{S}_\mu$, $\mathcal{S}_\varphi$, we use Theorem 9.3 with $d := \Delta - k$ (and then Lemma 5.20 to cut out the required subferns) to obtain[24]

- a leading (Δ, k)-fern for each pair of strings from $\mathcal{S}_\beta$,
- an internal (Δ, k)-fern for each pair of strings from $\mathcal{S}_\mu$, and
- a trailing (Δ, k)-fern for each pair of strings from $\mathcal{S}_\varphi$.

Further, we compute the *neutral fern* $\mathcal{N}$ that corresponds to the canonical pair $\mathcal{Q}$ using Lemma 6.14. Observe that $\mathcal{N}$ is Monge, and $(W+1)$-bounded difference if w is integer. We then create an instance $I_\mathcal{N}$ of the data structure of Lemma 9.1 for a sequence of α pointers to $\mathcal{N}$.[25]

In the following, whenever we need a fern (or an instance of the data structure from Lemma 9.1) for a blank piece of order $\ell \le \alpha$, we use a corresponding `DMM-Split` operation on $I_\mathcal{N}$ to split of the first ℓ ferns. As we never change the underlying ferns of $I_\mathcal{N}$, we are able to precompute such a split for each $\ell \le \alpha$, and may thus assume that accessing a blank piece of order ℓ is a constant-time operation.

Next, we split the input sequence $\mathcal{Y}'$ into consecutive subsequences of pairs of strings and blank pieces. For each subsequence of pairs of strings, we initialize a data structure of Lemma 9.1 (using the ferns computed earlier).

Finally, we initialize an instance $I_\mathcal{Y}$ of the data structure of Lemma 9.2 on the sequence of pointers `DMM-Root()` of the data structures for the subsequences of pairs of strings and the blank pieces; so that the sequence stored in $I_\mathcal{Y}$ corresponds to the sequence $\mathcal{Y}'$.

As for the running time, we distinguish between the different types of weights that we consider.

General weights. As $d = O(k)$, the calls to Theorem 9.3 take time $O(k^4 \log^2(\lambda k))$ where λ is an upper bound on the lengths of the strings in $\mathcal{S}_\beta \cup \mathcal{S}_\mu \cup \mathcal{S}_\varphi$; we obtain Monge $(d+k, k)$-ferns in this case, which have $O((d+k)^2) = O(k^2)$ multiplication and $O(d+k) = O(d)$-time vector multiplication (see Lemma 5.33).

Next, precomputing (the powers of) the neutral fern for all blank pieces of all orders up to $\alpha = O(k)$ does not exceed $O(k^4 \log^2(\lambda k))$.

Next, the initialization of the data structures of Lemma 9.1 in total takes time $O(|\mathcal{Y}'|k^2)$; the initialization of the data structure for Lemma 9.2 runs in time $O(|\mathcal{Y}'|)$.

In total, `BCDPM-Init` thus takes time $O(k^4 \log^2(\lambda k) + |\mathcal{Y}'|k^2)$, as claimed.

Integer weights. As $d = O(k)$, the calls to Theorem 9.3 take time $O(k^3 W \log^2(\lambda k))$ where λ is an upper bound on the lengths of the strings in $\mathcal{S}_\beta \cup \mathcal{S}_\mu \cup \mathcal{S}_\varphi$; we obtain $(W+1)$-bounded difference and Monge $(d+k, k)$-ferns in this case, which have $O((W+1)(d+k)\log(d+k)) = O(Wk \log k)$ multiplication and $O(d+k) = O(d)$-time vector multiplication (see Lemma 5.35).

Next, precomputing (the powers of) the neutral fern for all blank pieces of all orders up to $\alpha = O(k)$ does not exceed $O(Wk^3 \log^2(\lambda k))$.

Next, the initialization of the data structures of Lemma 9.1 in total takes time $O(|\mathcal{Y}'|kW \log k)$; the initialization of the data structure for Lemma 9.2 runs in time $O(|\mathcal{Y}'|)$.

In total, `BCDPM-Init` thus takes time $O(k^3 W \log^2(\lambda k) + |\mathcal{Y}'|kW \log k)$, as claimed.

The `BCDPM-Set` operation. For the `BCDPM-Set` operation, we swap out the pointer to the old neutral fern for the pointer to the new neutral fern. This takes constant time.

The `BCDPM-Commit` operation. We use the `DMM-Merge` operation of the data structure corresponding to the blank piece and its two neighbors. This takes time $O(k^2 \log(|\mathcal{Y}| + \alpha)) = O(k^2 \log(|\mathcal{Y}| + k))$ for general weights and time $O(kW \log k \log(|\mathcal{Y}| + \alpha)) = O(kW \log^2(|\mathcal{Y}| + k))$ for integer weights.

[24] The torsion bound indeed implies $\Delta \ge 2k$ and thus $\mathrm{ed}(\mathcal{S}) \le k \le \Delta - k = d$.
[25] One might say that we grow a neutral fern tree of size α.

The `BCDPM-Bleach` operation. We use twice the `DMM-Split` operation of the data structure corresponding to the subsequence of pairs of strings to cut out the blank piece. In $I_{\mathcal{Y}}$, we use one `DMM-Delete` to delete the pointer to the old instance of the data structure from Lemma 9.1 and then three calls to `DMM-Insert` to insert pointers to the prefix, the corresponding neutral fern, and the suffix.

The only part that has a significant time cost is the `DMM-Split` operation, which again takes time $O(k^2 \log(|\mathcal{Y}| + \alpha)) = O(k^2 \log(|\mathcal{Y}| + k))$ for general weights and time $O(kW \log k \log(|\mathcal{Y}| + \alpha)) = O(kW \log^2(|\mathcal{Y}| + k))$ for integer weights.

The remaining update operations. We forward the update to the corresponding instance of the data structure from Lemma 9.1; the claimed running times follow.[26]

The `BCDPM-Query` operation. We call `DMM-Vector` of $I_{\mathcal{Y}}$ with the all-zeroes vector and return any index of the resulting vector whose entry is at most k.

The correctness of this operation follows from Theorem 5.25: we use (Δ, k)-ferns of a Δ puzzle with $k \le \Delta/2 - \mathsf{tor}(\mathsf{Unpack}(\mathcal{Y}))$.

For the running time, as at any point, $I_{\mathcal{Y}}$ stores an alternating sequence of ordinary ferns and a (power of the) neutral fern, we have $|I_{\mathcal{Y}}| = \rho$, where ρ is the number of blank pieces in $\mathcal{Y}$. Hence, this operation takes time $O(k\rho)$ for general weights and $O(k\rho \log \Delta)$ for integer weights (observe that for integer weights we have to pay an extra logarithmic factor for explicitly producing the resulting vector).

In total, this completes the proof. ◢

10 Fast Algorithms in Important Settings

In this section, we obtain efficient algorithms for PMwWE in important settings by combining the `PILLAR` model algorithm from Main Theorem 2 with efficient implementations of the `PILLAR` model in those settings.

Main Theorem 2. *PMwWE on strings with $n < 3/2\, m + k$ can be solved in the `PILLAR` model*
- *in time $O(k^4 \log^2(mk))$ for general weights; and*
- *in time $\widetilde{O}(k^{3.5} \cdot W^4)$ if w is a metric weight function with values in $[0\,..\,W]$.*

The output set $\mathrm{Occ}_k^w(P, T)$ is represented as a collection of disjoint arithmetic progressions. ◢

The Standard Setting

The `PILLAR` model admits an optimal implementation in the standard setting [Far97, BF00, KRRW24]; see [CKW20, Theorem 7.2].

Theorem 10.1 [Far97, BF00, KRRW24]**.** *After an $O(n)$-time preprocessing of a collection of strings of total length n, each `PILLAR` operation can be performed in $O(1)$ time.* ◢

Combined, the standard trick, Main Theorem 2 and Theorem 10.1 yield Main Theorems 3 and 4.

Main Theorem 3. *PMwWE can be solved in $O(n + k^4 \cdot n/m \cdot \log^2(mk))$ time.* ◢

Main Theorem 4. *PMwWE can be solved in $\widetilde{O}(n + k^{3.5} \cdot W^4 \cdot n/m)$ time for metric weight functions w with values in $[0\,..\,W]$.* ◢

Recall that for the standard setting, we also have a non-`PILLAR`-based algorithm that outperforms Main Theorems 3 and 4 for large values of k.

Main Theorem 1. *PMwWE can be solved in $O((n \log m + m \log^2 m) \cdot k)$ time.* ◢

[26] While not explicitly forbidden, we typically assume that such operations do not target a blank piece. If they do, we potentially first split the corresponding neutral fern.

We compliment our algorithms for the standard setting with an easy generalization of the conditional lower bound of [CKW23].

Theorem 10.2 [CKW23, Main Theorem 2]. *For real parameters $\delta > 0$ and $0.5 \leq \kappa \leq 1$, assuming the APSP Hypothesis, no algorithm, given strings X, Y of length at most m, a real threshold $1 \leq k \leq m^\kappa$, and oracle access to a normalized weight function w, decides if $\mathsf{ed}^w(X \to Y) \leq k$ in time $\mathcal{O}(m^{0.5+1.5\kappa-\delta})$.*

Corollary 10.3. *For real parameters $\delta > 0$ and $0 < \kappa \leq 1$, assuming the APSP Hypothesis, there is no algorithm that solves all instances of PMwWE satisfying $k \leq m^\kappa$ in time $\mathcal{O}(n \cdot m^{\min(1.5\kappa-0.5,\, 2.5\kappa-1)-\delta})$.*

Proof. Let us first consider the case of $0.5 \leq \kappa \leq 1$, assuming $\delta \leq 0.25$ without loss of generality. Suppose that we are given strings X, Y of length at most M over an alphabet Σ, a threshold $d \in \mathbb{R}_{\geq 1}$ satisfying $d \leq M^\kappa$, and oracle access to a normalized weight function w. Define $k = \lceil d \rceil + 1$ and $N = \lceil k^{1/\kappa} \rceil = \mathcal{O}(d^{1/\kappa}) = \mathcal{O}(M)$. We construct an instance of PMwWE with pattern $P = \$X\$^{N+1}$, text $T = \$Y\$^N\#$, where $\$, \# \notin \Sigma$, integer threshold k, and the weight function w extended so that $w(\$ \mapsto \$) = w(\# \mapsto \#) = 0$, $w(\$ \mapsto \#) = k - d \geq 1$, and other edits involving \$ or # cost $10k$.

If $\mathsf{ed}^w(X \to Y) \leq d$, then $\mathsf{ed}^w(P \to T) \leq \mathsf{ed}^w(\$ \to \$) + \mathsf{ed}^w(X \to Y) + \mathsf{ed}^w(\$^{N+1} \to \$^N\#) \leq 0 + d + k - d = k$, so $\mathrm{Occ}_k^w(P, T) \neq \varnothing$. Conversely, if $\mathsf{ed}^w(P \to T[\,i\,..\,j\,)) \leq k$ holds for some fragment of T, then every \$ in P has to be matched with a \$ or a # in T (otherwise, it would contribute at least $10k$), so $\mathsf{ed}^w(X \to Y) \leq k - \mathsf{ed}^w(\$ \to \#) = d$. Hence, to decide $\mathsf{ed}^w(X \to Y) \leq d$, it suffices to check $\mathrm{Occ}_k^w(P, T) \neq \varnothing$ using the hypothetical PMwWE algorithm, which is applicable because $m > N \geq k^{1/\kappa}$. This approach takes $\mathcal{O}(m + m^{\min(0.5+1.5\kappa,2.5\kappa)-\delta}) = \mathcal{O}(M + M^{0.5+1.5\kappa-\delta}) = \mathcal{O}(M^{0.5+1.5\kappa-\delta})$ time, contradicting Theorem 10.2.

For $0.4 < \kappa \leq 0.5$, we assume $\delta \leq 1.25 - 0.5/\kappa$ without loss of generality and, instead of $d \leq M^\kappa$, pick $d \leq M^{0.5}$ so that $N = \mathcal{O}(d^{1/\kappa}) = \mathcal{O}(M^{0.5/\kappa})$. The hypothetical approach takes $\mathcal{O}(m + m^{\min(0.5+1.5\kappa,2.5\kappa)-\delta}) = \mathcal{O}(M^{0.5/\kappa} + M^{0.5/\kappa\cdot(2.5\kappa-\delta)}) = \mathcal{O}(M^{1.25-\delta})$ time, contradicting Theorem 10.2.

In the final case of $0 \leq \kappa \leq 0.4$, the hypothetical running time for PMwWE is $\mathcal{O}(m^{2.5\kappa-\delta}) = \mathcal{O}(m^{1-\delta})$, so it suffices to observe that solving PMwWE requires reading the entire pattern.

The Compressed Setting

A straight-line program (SLP) is a context-free grammar $\mathcal{G}$ that consists of a set Σ of terminals and a set $N_\mathcal{G} := \{A_1, \ldots, A_n\}$ of non-terminals such that each $A_i \in N_\mathcal{G}$ is associated with a unique production rule $A_i \to f_\mathcal{G}(A_i) \in (\Sigma \cup \{A_j : j < i\})^*$. We may assume, without loss of generality, that each production rule is of the form $A \to BC$ for some symbols B and C.

Every symbol $A \in S_G := N_G \cup \Sigma$ generates a unique string, which we denote by $gen(A) \in \Sigma^*$; we obtain this string from A by repeatedly replacing each non-terminal with its production. We say that $\mathcal{G}$ generates $gen(A_n)$.

Combining a recent result of Duyster and Kociumaka [DK24, Theorem 1] on answering `IPM` queries over run-length straight-line programs into [CKW20, Theorem 7.13], which summarizes results from [BLR+15, I17, CKW20], we obtain the following implementation of the `PILLAR` model in the compressed setting.[27]

Theorem 10.4 [BLR+15, I17, CKW20, DK24]. *Given a collection of SLPs of total size n, generating strings of total length N, each PILLAR operation can be performed in $\mathcal{O}(\log^2 N)$ time after an $O(n \log N)$-time preprocessing.*

Combining Main Theorem 2 and Theorem 10.4 in a non-trivial, but by now standard procedure (see [BKW19, CKW20, CKW22]), we obtain the following result.

[27] See also the discussion just after [DK24, Corollary 2].

Corollary 10.5 (Compressed Setting). *Let $\mathcal{G}_T$ denote an SLP of size n generating a string T, let $\mathcal{G}_P$ denote an SLP of size m generating a string P, and set $N := |T|$ and $M := |P|$.*

Given $\mathcal{G}_T$, $\mathcal{G}_P$, an integer threshold k, and oracle access to a normalized weight function $w : \bar{\Sigma}^2 \to [0, W]$, we can compute $|\mathrm{Occ}_k^w(P,T)|$

- *in time $\widetilde{O}(m + n\,k^4)$ (for general w), and*
- *in time $\widetilde{O}(m + n\,k^{3.5}W^4)$ if w is an integer metric weight function.*

Further, we can report the elements of $\mathrm{Occ}_k^w(P,T)$ within $O(|\mathrm{Occ}_k^w(P,T)|)$ extra time.

The Dynamic Setting

Write $\mathcal{X}$ for an initially empty collection of strings that is subject to the following updates.

- `Makestring`(U): Insert a non-empty string U to $\mathcal{X}$.
- `Concat`(U,V): Insert string UV to $\mathcal{X}$, for $U,V \in \mathcal{X}$.
- `Split`(U,i): Insert $U[0..i)$ and $U[i..|U|)$ to $\mathcal{X}$, for $U \in \mathcal{X}$ and $i \in [0..|U|)$.

Let N denote an upper bound on the cumulative length of all strings in $\mathcal{X}$ throughout the execution of the algorithm.

The collection $\mathcal{X}$ can be maintained with operations `Makestring`(U), `Concat`(U,V), `Split`(U,i) requiring time $O(\log N + |U|)$, $O(\log N)$ and $O(\log N)$, respectively, so that `PILLAR` operations can be performed in time $O(\log^2 N)$; see [GKK+18, DK24].[28]

Observe that Kempa and Kociumaka [KK22, Section 8] presented an alternative deterministic implementation at the cost of extra $\mathrm{poly}(\log\log N)$-factor overheads in the running time.

By combining the data structures of [GKK+18, DK24, KK22], Main Theorem 2, and the standard trick (recall the first paragraph of Section 7) we obtain the following result.

Corollary 10.6 (Dynamic Setting). *We can maintain a collection $\mathcal{X}$ of non-empty persistent strings of total length N subject to* `makestring`(U) *in time $O(\log N + |U|)$,* `concat`(U,V) *in time $O(\log N)$, and* `split`(U,i) *in time $O(\log N)$, such that given two strings $P,T \in \mathcal{X}$, an integer threshold $k > 0$, and oracle access to a weight function $w : \bar{\Sigma}^2 \to [0, W]$, we can compute $\mathrm{Occ}_k^w(P,T)$*[29]

- *in time $\widetilde{O}(|T|/|P| \cdot k^4)$ (for general w); and*
- *in time $\widetilde{O}(|T|/|P| \cdot k^{3.5}W^4)$ if w is an integer metric weight function.*

The Quantum Setting

We say an algorithm on an input of size n succeeds *with high probability* if the success probability can be made at least $1 - 1/n^c$ for any desired constant $c > 1$.

Let us turn our attention to the quantum setting, where we assume that the input strings can be accessed in a quantum query model [Amb04, BdW02]. We are interested in the time complexity of quantum algorithms [BBC+95]. The near-optimal quantum algorithm of Hariharan and Vinay [HV03] for the decision version of exact pattern matching, encapsulated in the following statement, implies that `PILLAR` operations on strings of length $O(m)$ can be performed in $\widetilde{O}(\sqrt{m})$ time without any preprocessing. See [CPR+24] for details.

Theorem 10.7 [HV03]. *Consider a string T of length n and a string P of length $m \le n$. We can decide whether P occurs in T in $\widetilde{O}(\sqrt{n})$ time in the quantum model with high probability. If the answer is YES, then the algorithm returns a witness occurrence.*

Combined, the standard trick, Main Theorem 2 and Theorem 10.7 yield the following result.

[28] All stated time complexities hold with probability $1 - 1/N^{\Omega(1)}$.

[29] All running time bounds hold with probability $1 - 1/N^{\Omega(1)}$. This result can be derandomized at the cost of a $\mathrm{poly}(\log\log N)$-factor overhead.

Corollary 10.8 (Quantum Setting)**.** *In the quantum model, PMwWE can be solved w.h.p.*
- *in time* $\widetilde{O}(k^4 \cdot n/\sqrt{m})$ *for general weights; and*
- *in time* $\widetilde{O}(k^{3.5}W^4 \cdot n/\sqrt{m})$ *if* w *is an integer metric weight function.*

An open question is whether faster quantum algorithms can be designed for PMwWE as in [KNW24, KNW25] for PMwE.

The Packed Setting

In the Real RAM model of computation with word size $\Theta(\log n)$, we may store up to $\Omega(\log_{|\Sigma|} n)$ characters in a single machine word. The *packed representation* of a string S over an integer alphabet $[0\,..\,\sigma)$ is a list obtained by storing $\Theta(\log_\sigma n)$ characters per machine word—thus allowing us to store S in just $O(|S|/\log_\sigma n)$ machine words.

Theorem 10.9 [KK19, KRRW24]**.** *Given a packed representation of a string* S *of length* n*, after an* $O(n/\log_\sigma n)$*-time preprocessing, we can perform* `PILLAR` *operations in* $O(1)$ *time.*

Combined, the standard trick, Main Theorem 2 and Theorem 10.9 yield the following result.

Corollary 10.10 (Packed Setting)**.** *Given packed representations of a text* T *of length* n *and a pattern* P *of length* m *over alphabet* $[0\,..\,\sigma)$*, an integer threshold* k*, and oracle access to a normalized weight function* $w : \bar{\Sigma}^2 \to [0\,,\,W]$*, we can compute* $\mathrm{Occ}_k^w(P,T)$
- *in time* $O(n/\log_\sigma n + k^4 \cdot n/m \cdot \log^2(mk))$ *(for general* w*); and*
- *in time* $O(n/\log_\sigma n) + \widetilde{O}(k^{3.5}W^4 \cdot n/m)$ *if* w *is an integer metric weight function.*

The Read-only Setting

Finally, we consider the read-only setting, where we have random access to P and T, but our extra working space is limited.

Theorem 10.11 [BCS24, KS24]**.** *Suppose that we have read-only random access to a* n*-length string* S *of length* n *over an integer alphabet. For any* $\tau \in [1\,..\,n]$*, after a preprocessing step that uses* $\widetilde{O}(n)$ *time and* $\widetilde{O}(n/\tau)$ *extra space,* `PILLAR` *operations can be performed in time* $\widetilde{O}(\tau)$*.*

Combined, the standard trick, Main Theorem 2 and Theorem 10.11 yield the following result.

Corollary 10.12 (Read-only Setting)**.** *Suppose that we have read-only random access to a text* T *of length* n*, a pattern* P *of length* m*, and . Given an integer threshold* k*, and oracle access to a normalized weight function* $w : \bar{\Sigma}^2 \to [0\,,\,W]$*, for any* $\tau \in [1\,..\,n]$*, we can compute* $\mathrm{Occ}_k^w(P,T)$
- *in time* $\widetilde{O}(n + k^4\tau \cdot n/m)$ *using* $\widetilde{O}(m/\tau + k^4)$ *extra space (for general* w*); and*
- *in time* $\widetilde{O}(n + k^{3.5}W^4\tau \cdot n/m)$ *using* $\widetilde{O}(m/\tau + k^{3.5}W^4)$ *extra space if* w *is an integer metric weight function.*

Bibliography

Abr87 Karl R. Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16(6):1039–1051, 1987. doi:10.1137/0216067. 42

AKM+87 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359. 2, 3, 5, 6, 10, 16, 32

Amb04 Andris Ambainis. Quantum query algorithms and lower bounds. In *Classical and New Paradigms of Computation and their Complexity Hierarchies*, pages 15–32, 2004. doi:10.1007/978-1-4020-2776-5_2. 93

BBC+95 Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52:3457–3467, 1995. doi:10.1103/PhysRevA.52.3457. 93

BCS24 Gabriel Bathie, Panagiotis Charalampopoulos, and Tatiana Starikovskaya. Internal pattern matching in small space and applications. In *35th Annual Symposium on Combinatorial Pattern Matching, CPM 2024*, pages 4:1–4:20, 2024. doi:10.4230/LIPICS.CPM.2024.4. 94

BdW02 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X. 93

BF00 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium*, pages 88–94, 2000. doi:10.1007/10719839_9. 91

BI18 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM Journal on Computing*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128. i

BKW19 Karl Bringmann, Marvin Künnemann, and Philip Wellnitz. Few Matches or Almost Periodicity: Faster Pattern Matching with Mismatches in Compressed Texts. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 1126–1145. SIAM, 2019. doi:10.1137/1.9781611975482.69. 92

BLR+15 Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random Access to Grammar-Compressed Strings and Trees. *SIAM Journal on Computing*, 44(3):513–539, 2015. doi:10.1137/130936889. 92

CH02 Richard Cole and Ramesh Hariharan. Approximate String Matching: A Simpler Faster Algorithm. *SIAM Journal on Computing*, 31(6):1761–1782, 2002. doi:10.1137/S0097539700370527. 1, 3, 33

CKW20 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In *61st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2020*, pages 978–989, 2020. Full version: arXiv:2004.08350v2. arXiv:2004.08350v2, doi:10.1109/FOCS46700.2020.00095. 3, 4, 5, 8, 9, 11, 18, 42, 44, 45, 46, 61, 62, 63, 66, 67, 68, 91, 92

CKW22 Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster pattern matching under edit distance : A reduction to dynamic puzzle matching and the seaweed monoid of permutation matrices. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 698–707. IEEE, 2022. Full version: arXiv:2204.03087v1. arXiv:2204.03087v1, doi:10.1109/FOCS54457.2022.00072. i, 2, 3, 4, 5, 9, 10, 11, 15, 18, 19, 42, 43, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 61, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 88, 92

CKW23 Alejandro Cassis, Tomasz Kociumaka, and Philip Wellnitz. Optimal algorithms for bounded weighted edit distance. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*, pages 2177–2187. IEEE, 2023. Full version: arXiv:2305.06659v2. arXiv:2305.06659v2, doi:10.1109/FOCS57990.2023.00135. i, 1, 2, 3, 8, 12, 15, 18, 33, 34, 37, 92

CPR+24 Panagiotis Charalampopoulos, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Approximate circular pattern matching under edit distance. In *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024*, pages 24:1–24:22, 2024. doi:10.4230/LIPICS.STACS.2024.24. 11, 93

DGH+23 Debarati Das, Jacob Gilbert, MohammadTaghi Hajiaghayi, Tomasz Kociumaka, and Barna Saha. Weighted edit distance computation: Strings, trees, and dyck. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023*, pages 377–390. ACM, 2023. doi:10.1145/3564246.3585178. 1, 2, 14

DK24 Anouk Duyster and Tomasz Kociumaka. Logarithmic-time internal pattern matching queries in compressed and dynamic texts. In *31st International Symposium on String Processing and Information Retrieval, SPIRE 2024*, pages 102–117. Springer, 2024. doi:10.1007/978-3-031-72200-4_8. 92, 93

Far97 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual IEEE Symposium on Foundations of Computer Science, FOCS '97*, pages 137–143, 1997. doi:10.1109/SFCS.1997.646102. 91

FR06 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006. doi:10.1016/j.jcss.2005.05.007. 16

GK25 Egor Gorbachev and Tomasz Kociumaka. Bounded edit distance: Optimal static and dynamic algorithms for small integer weights. In *57th Annual ACM Symposium on Theory of Computing, STOC 2025*, pages 2157–2166. ACM, 2025. Full version: arXiv:2404.06401v3. doi:10.1145/3717823.3718168. 4, 5, 11, 16, 17, 27, 28, 33, 34, 35, 36, 37, 88

GKK+18 Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Łącki, and Piotr Sankowski. Optimal dynamic strings. In *29th ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1509–1528. SIAM, 2018. doi:10.1137/1.9781611975031.99. 93

HV03 Ramesh Hariharan and V. Vinay. String matching in Õ(sqrt(n)+sqrt(m)) quantum time. *Journal of Discrete Algorithms*, 1(1):103–110, 2003. doi:10.1016/S1570-8667(03)00010-8. 93

I17 Tomohiro I. Longest common extensions with recompression. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, pages 18:1–18:15, 2017. doi:10.4230/LIPIcs.CPM.2017.18. 92

KK19 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368. 94

KK22 Dominik Kempa and Tomasz Kociumaka. Dynamic suffix array with polylogarithmic queries and updates. In *STOC 2022: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1657–1670. ACM, 2022. doi:10.1145/3519935.3520061. 93

Kle05 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *16th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005*, pages 146–155. SIAM, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070454. 2, 6, 7, 16, 33

KMP77 Donald E. Knuth, James H. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024. 1

KNW24 Tomasz Kociumaka, Jakob Nogler, and Philip Wellnitz. On the communication complexity of approximate pattern matching. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, 2024. doi:10.1145/3618260.3649604. 5, 94

KNW25 Tomasz Kociumaka, Jakob Nogler, and Philip Wellnitz. Near-optimal-time quantum algorithms for approximate pattern matching. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025*, pages 517–534. SIAM, 2025. doi:10.1137/1.9781611978322.15. 5, 94

KRRW24 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. *SIAM J. Comput.*, 53(5):1524–1577, 2024. doi:10.1137/23M1567618. 91, 94

KS24 Dmitry Kosolobov and Nikita Sivukhin. Construction of sparse suffix trees and LCE indexes in optimal time and space. In *35th Annual Symposium on Combinatorial Pattern Matching, CPM 2024*, pages 20:1–20:18, 2024. doi:10.4230/LIPICS.CPM.2024.20. 94

Lev65 Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Dokl. Akad. Nauk SSSR*, 163:845–848, 1965. URL: http://mi.mathnet.ru/eng/dan31411. 1, 15

LMS98 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998. doi:10.1137/S0097539794264810. 5

LV86 Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986. doi:10.1016/0304-3975(86)90178-7. 3

LV88 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *J. Comput. Syst. Sci.*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1. 2, 4, 18

LV89 Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989. doi:10.1016/0196-6774(89)90010-2. 1, 2, 3, 18, 33

MBCT23 Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale Algorithm Design: Bioinformatics in the Era of High-Throughput Sequencing*. Cambridge University Press, 2 edition, 2023. 1

MP70 James H. Morris. and Vaughan R. Pratt. A linear pattern-matching algorithm. Technical report, University of California, Berkeley, Computation Center, 1970. Technical report. 1

Mye86 Eugene W. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986. doi:10.1007/BF01840446. 2

Nav01 Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001. doi:10.1145/375360.375365. 2

Sel80 Peter H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1(4):359–373, 1980. URL: https://doi.org/10.1016/0196-6774(80)90016-4. i, 1, 2

SV96 Süleyman Cenk Sahinalp and Uzi Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm (extended abstract). In *37th Annual IEEE Symposium on Foundations of Computer Science, FOCS 1996*, pages 320–328. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548491. 2

Tis15 Alexander Tiskin. Fast distance multiplication of unit-Monge matrices. *Algorithmica*, 71(4):859–888, 2015. URL: https://doi.org/10.1007/s00453-013-9830-z. 3, 10, 11, 16, 32, 33, 43

Ukk85 Esko Ukkonen. Finding approximate patterns in strings. *J. Algorithms*, 6(1):132–137, 1985. doi:10.1016/0196-6774(85)90023-9. 2

VW18 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893. 2

Index of Results

- **Section 3**: Definitions and generally useful facts and results.
  - **Definition 3.3**: An alignment $\mathcal{A} : P[\,p\,..\,p'\,) \rightsquigarrow T[\,t\,..\,t'\,)$ for strings P and T.
  - **Fact 3.5**: Alignments transfer decomposistions [CKW22, CKW23].
  - **Definition 3.1**: The alignment graph $\mathsf{AG}^w(P,T)$ of a weight function w and strings P, T [CKW23].
  - **Lemma 3.16**: Basic properties of the alignment graph [GK25, Lemma 5.2].
  - **Definition 3.14**: The augmented alignment graph $\overline{\mathsf{AG}}^w(P,T)$ for a weight function w and strings P and T [GK25].
  - **Theorem 3.10**: Efficient MSSP on planar graphs [Kle05].
  - **Theorem 3.12**: SMAWK algorithm for efficient (min, +)-product computation [AKM+87].
  - **Fact 3.4**: The weighted edit distance is a metric for metric weight functions [DGH+23, Fact 2.5].
  - **Fact 3.7**: For normalized weight functions, weighted occurrences are unweighted occurrences.
- **Main Theorem 1**: PMwWE is in $\widetilde{O}(nk)$ time.
  - **Remark 4.1**: Standard Trick: Can reduce to $O(n/k)$ instances (P,S) where $S = T[\,i\,..\,i+m+2k\,)$ with $i \equiv_k 0$.
  - **Lemma 4.2**: After preprocessing a string T of length n and a pattern P of length $m \le n+k$ for $O(n)$ time, can compute for any fragment S of T of length at most $(m+2k)$ an unweighted alignment $\mathcal{A} : P \rightsquigarrow S$ of cost at most $d = 4k$ or determine that $\mathrm{Occ}_k^w(P,S) = \emptyset$ in time $O(d^2)$ for a single S.
    - **Lemma 3.20**: Computing an optimal alignment of two strings of cost at most d (or deciding that no such alignment exists) takes $O(1+d^2)$ time in the PILLAR model, `Alignment (`S`,` T`)` [LV88].
    - **Theorem 10.1** [Far97, BF00, KRRW24].
  - **Definition 4.3**: The i-th d-slice $P_{d,i}$ of a string P, the graphs $G^{P,S,d}_{\ell,r}$ (and $G^{P,d}_{\ell,r}$) as subgraphs of $\overline{\mathsf{AG}}^w(P,S)$ (and $\overline{\mathsf{AG}}^w(P,P)$) and the corresponding distance matrices $D^{P,S,d}_{\ell,r}$ (and $D^{P,d}_{\ell,r}$).
  - **Lemma 4.5**: $O(d \log m)$-time computation of any $D^{P,d}_{\ell,r} \oplus v$ (`DMVO-Query`) after global $O(md \log^2 m)$-time preprocessing (`DMVO-Init`).
  - **Lemma 4.6**: Given alignment $\mathcal{A} : P \rightsquigarrow S$ of cost at most d and a $k \in [\,0\,..\,d\,]$, can compute $\mathrm{Occ}_k^w(P,S)$ using $O(d)$ calls to `DMVO-Query` plus $\widetilde{O}(d^2)$ extra time.
    - **Lemma 3.18**: Paths of cost k that connect the top and bottom of the alignment graph may use at most k non-diagonal edges.
- **Section 5**: Definitions and generally useful facts and results for Monge matrices and ferns.
  - **Definition 5.1**: The density matrix $A^\square$ and the core $\mathrm{core}(A)$ of size $\delta(A) := |\mathrm{core}(A)|$.
  - **Definition 5.7**: k-equivalence of integers, $a \overset{k}{=} b$.
  - **Definition 5.10**: The β-bounded difference property of an integer matrix.
  - **Definition 5.14** [GK25, Definitions 5.5 and 5.6].
  - **Remark 5.17**.
    - **Definition 5.16**.
    - **Fact 3.11**: Distance matrices of planar graphs are Monge [FR06, Section 2.3].
  - **Definition 5.18**.
  - **Definition 5.21**.
  - **Definition 5.23**.
  - **Definition 5.31**.
  - **Remark 5.34**.
    - **Lemma 6.14**: `GrowFern(`P`,` T`,` k`,` w`)`.

(refer to entry of Lemma 6.14)
- **Corollary 3.19**: Using MSSP to compute boundary-to-boundary paths in the alignment graph.
  - **Theorem 3.10**: Efficient MSSP on planar graphs [Kle05].

- **Main Theorem 3**: PMwWE in time $\widetilde{O}(n + k^4 \cdot n/m)$.
  - **Remark 7.1**: The standard trick to reduce the text-length to be roughly the pattern-length.
  - **Main Theorem 2**: Efficient PILLAR model algorithms for PMwWE.
    (refer to entry of Main Theorem 2)
  - **Theorem 10.1** [Far97, BF00, KRRW24].

- **Main Theorem 4**: PMwWE in time $\widetilde{O}(n + k^{3.5} \cdot W^4 \cdot n/m)$ for integer metrics $w : \bar{\Sigma}^2 \to [0\,..\,W]$.
  - **Remark 7.1**: The standard trick to reduce the text-length to be roughly the pattern-length.
  - **Main Theorem 2**: Efficient PILLAR model algorithms for PMwWE.
    (refer to entry of Main Theorem 2)
  - **Theorem 10.1** [Far97, BF00, KRRW24].

- **Corollary 10.3**: PMwWE not in $\min(k^{1.5-o(1)} \cdot n/m^{0.5}, k^{2.5-o(1)} \cdot n/m)$ time.
  - **Theorem 10.2**: Bounded weighted edit distance not in $m^{0.5}k^{1.5-o(1)}$ time [CKW23, Main Theorem 2].

- **Corollary 10.5**: Compressed Setting.
  - **Main Theorem 2**: Efficient PILLAR model algorithms for PMwWE.
    (refer to entry of Main Theorem 2)
  - **Theorem 10.4** [BLR+15, I17, CKW20, DK24].

- **Corollary 10.6**: Dynamic Setting.
  - **Remark 7.1**: The standard trick to reduce the text-length to be roughly the pattern-length.
  - **Main Theorem 2**: Efficient PILLAR model algorithms for PMwWE.
    (refer to entry of Main Theorem 2)

- **Corollary 10.8**: Quantum Setting.
  - **Remark 7.1**: The standard trick to reduce the text-length to be roughly the pattern-length.
  - **Main Theorem 2**: Efficient PILLAR model algorithms for PMwWE.
    (refer to entry of Main Theorem 2)
  - **Theorem 10.7** [HV03].

- **Corollary 10.10**: Packed Setting.
  - **Remark 7.1**: The standard trick to reduce the text-length to be roughly the pattern-length.
  - **Main Theorem 2**: Efficient PILLAR model algorithms for PMwWE.
    (refer to entry of Main Theorem 2)
  - **Theorem 10.9** [KK19, KRRW24].

- **Corollary 10.12**: Read-only Setting.
  - **Remark 7.1**: The standard trick to reduce the text-length to be roughly the pattern-length.
  - **Main Theorem 2**: Efficient PILLAR model algorithms for PMwWE.
    (refer to entry of Main Theorem 2)
  - **Theorem 10.11** [BCS24, KS24].

- **Lemma 6.5**: `GrowFern-SmallSED(`P`, `T`, `k`, `w`)` [GK25, adapted from Lemma 7.3].
  - **Lemma 6.1**: Any instance of GrowFern with $|P| + |T| = O(k)$ can be solved in $O(k^2 \log k)$.
    - **Corollary 3.19**: Using MSSP to compute boundary-to-boundary paths in the alignment graph.

- - **Theorem 3.10**: Efficient MSSP on planar graphs [Kle05].
- **Lemma 6.3** [GK25, Lemma 7.1]; see also [CKW23, Lemma 4.6 and Claim 4.11].
- **Definition 5.3**: The Core-based Matrix Oracle CMO(A) of a matrix A [GK25, Lemma 4.4].
- **Fact 3.11**: Distance matrices of planar graphs are Monge [FR06, Section 2.3].
- **Lemma 6.4** [GK25, see Lemma 7.2].
- **Lemma 5.6**: Given CMO(A) for some Monge matrix A, can compute CMO(C) for any contiguous submatrix C of A in time $O((N + \delta(A)) \log N)$ where N is the maximum dimension of A.
- **Fact 5.15** [GK25, Observation 5.7].
- **Lemma 5.5**: Given CMO(A) and CMO(B) for Monge A and B, can compute CMO($A \oplus B$) in time $O((N + \delta(A) + \delta(B)) \log N)$, where N is the maximum dimension of A and B [GK25, Lemma 4.6].
- **Lemma 5.13**: The $(\min, +)$-product of β-bounded difference matrices is β-bounded difference [GK25, Lemma 4.10].
- **Lemma 5.9**: Given k and CMO(A) and CMO(B) for Monge matrices A and B, can compute CMO(C') of a matrix C' that is k equivalent to $A \oplus B$ and $\delta(C') \le O(N\sqrt{k})$, in time $O((N + \delta(A) + \delta(B)) \log N)$, where N is the maximum dimension of A and B [GK25, Lemma 4.14].

▬ **Lemma 6.14**: `GrowFern(`P`, `T`, `k`, `w`)`.
- **Lemma 3.20**: Computing an optimal alignment of two strings of cost at most d (or deciding that no such alignment exists) takes $O(1 + d^2)$ time in the `PILLAR` model, `Alignment (`S`, `T`)` [LV88].
- **Lemma 5.20**.
  - **Fact 5.2**: The core of any contiguous submatrix of a matrix A is at most $\delta(A)$.
  - **Fact 5.11**: Contiguous submatrices of integer-valued β-bounded-difference matrices are integer-valued and β-bounded-difference.
- **Lemma 6.10** [CKW23, Lemma 4.5].
- **Lemma 6.5**: `GrowFern-SmallSED(`P`, `T`, `k`, `w`)` [GK25, adapted from Lemma 7.3].
  (refer to entry of Lemma 6.5)
- **Lemma 6.2**: Properties of self-ed [CKW23, Lemma 4.2] and [GK25, Lemma 6.2].
- **Lemma 3.18**: Paths of cost k that connect the top and bottom of the alignment graph may use at most k non-diagonal edges.
- **Lemma 6.11** [CKW23, part of Lemma 4.9].
- **Lemma 6.13** [GK25, Lemma 7.6].
- **Lemma 6.12** [CKW23, Lemma 4.4].
- **Lemma 3.21**: Computing the weighted edit distance of two strings takes $O(1 + k^3 \log^2 n)$ time in the `PILLAR` model, `WeightedED (`S`, `T`, `w`)` [CKW23, Theorem 4.1].
- **Lemma 5.5**: Given CMO(A) and CMO(B) for Monge A and B, can compute CMO($A \oplus B$) in time $O((N + \delta(A) + \delta(B)) \log N)$, where N is the maximum dimension of A and B [GK25, Lemma 4.6].
- **Lemma 5.9**: Given k and CMO(A) and CMO(B) for Monge matrices A and B, can compute CMO(C') of a matrix C' that is k equivalent to $A \oplus B$ and $\delta(C') \le O(N\sqrt{k})$, in time $O((N + \delta(A) + \delta(B)) \log N)$, where N is the maximum dimension of A and B [GK25, Lemma 4.14].

▬ **Corollary 6.16**: `ListAllOccs(`P`, `T`, `k`, `w`)`.
- **Corollary 3.19**: Using MSSP to compute boundary-to-boundary paths in the alignment graph.
  - **Theorem 3.10**: Efficient MSSP on planar graphs [Kle05].
- **Lemma 3.20**: Computing an optimal alignment of two strings of cost at most d (or deciding that no such alignment exists) takes $O(1 + d^2)$ time in the `PILLAR` model, `Alignment (`S`, `T`)` [LV88].
- **Lemma 6.5**: `GrowFern-SmallSED(`P`, `T`, `k`, `w`)` [GK25, adapted from Lemma 7.3].
  (refer to entry of Lemma 6.5)

- **Main Theorem 2**: Efficient PILLAR model algorithms for PMwWE.
  - **Definition 7.2**: A Δ-puzzle of strings and the value of a Δ-puzzle [CKW22, Definition 1.4].
  - **Lemma 7.13**: Reduction of PMwWE to AlignedPeriodicMatches and Verify.
    - **Lemma 7.4**: `Analyze(`P`,` k`)`, $O(k^2)$ time PILLAR model algorithm for structurally analyzing P to obtain either breaks, repetitive regions, or to conclude that P is almost periodic [CKW20, Lemma 6.4].
    - **Corollary 7.6**: $O(k^3)$-time PILLAR model reduction from PMwWE to Verify in the case of breaks.
      - **Fact 7.5**: $O(k^3)$-time PILLAR model algorithm for computing candidate positions in the case of breaks [CKW20, (proofs of) Lemmas 5.21 and 6.12].
    - **Corollary 7.9**: $\widetilde{O}(k^{3.5})$-time PILLAR model reduction from PMwWE to Verify in the case of repetitive regions.
      - **Lemma 7.8**: $\widetilde{O}(k^{3.5})$-time PILLAR model algorithm for computing candidate positions in the case of repetitive regions [CKW22, Sections 2.3 and 3.1].
        - **Fact 7.7**: NewPeriodicMatches is in time $\widetilde{O}(d^{3.5})$ in the PILLAR model.
    - **Lemma 7.12**: $O(k^2)$-time PILLAR model reduction from PMwWE to AlignedPeriodicMatches in the case of approximate periodicity.
      - **Fact 7.10**: $O(d^2)$ PILLAR algorithm for computing approximately periodic fragment of T containing all approximate occurrences [CKW20, compare Lemma 6.8].
      - **Fact 7.11**: `FindAWitness(`k`,` Q`,` S`)`, $O(k^2)$-time PILLAR algorithm that can be used to rotate Q such that $\mathsf{ed}(P, {}^{*}Q^{*}) = \mathsf{ed}(P, Q^{*})$ [CKW20, compare Lemma 6.5].
      - **Lemma 3.22**: Computing an optimal alignment of a string to an infinitely repeating string takes $O(1 + d^2)$ time in the PILLAR model, `Alignment (`S`,` Q`,` x`)` [CKW22, Lemma 2.10].
  - **Lemma 7.40**: AlignedPeriodicMatches is in time $O(k^4 \log^2(mk))$ in the PILLAR model.
    (refer to entry of Lemma 7.40)
  - **Lemma 8.44**: AlignedPeriodicMatches is in time $\widetilde{O}(k^{3.5}W^4)$ in the PILLAR model for metric integer weight functions.
    (refer to entry of Lemma 8.44)
  - **Corollary 6.17**: `Verify(`P`,` T`,` k`,` I`,` w`)`.
    - **Corollary 6.16**: `ListAllOccs(`P`,` T`,` k`,` w`)`.
      (refer to entry of Corollary 6.16)

- **Lemma 7.40**: AlignedPeriodicMatches is in time $O(k^4 \log^2(mk))$ in the PILLAR model.
  - **Lemma 7.18**: Efficient solution for the AlignedPeriodicMatches problem parameterized by the number of tiles in P.
    - **Lemma 7.15**: Efficient algorithm for the AlignedPeriodicMatches problem for the case $q \gg k$.
      - **Lemma 7.14**: $O(d_T + 1)$-time algorithm for computing a collection of short substrings R_j of T that capture all k-edit occurrences and there are only a few distinct ones [CKW22, Section 3.2].
      - **Corollary 6.17**: `Verify(`P`,` T`,` k`,` I`,` w`)`.
        - **Corollary 6.16**: `ListAllOccs(`P`,` T`,` k`,` w`)`.
          (refer to entry of Corollary 6.16)
    - **Fact 7.17**.
  - **Lemma 7.14**: $O(d_T + 1)$-time algorithm for computing a collection of short substrings R_j of T that capture all k-edit occurrences and there are only a few distinct ones [CKW22, Section 3.2].
  - **Lemma 7.19**: Δ-puzzle for P from its tile partition [CKW22, Lemma 4.6].
  - **Lemma 7.20**: Δ-puzzle for T from its tile partition [CKW22, Lemma 4.7].

- **Corollary 7.33.**
  - **Lemma 7.23**: Upper bound on the cumulative difference of length between puzzle pieces comprising R_j and P [CKW22, Lemma 4.10]**.**
    - **Definition 7.21**: Families of leading, internal, and trailing puzzle pieces [CKW22, Definition 4.8]**.**
  - **Lemma 7.30**: If the torsion is small, we can trim/expand sequences of at least $k+2$ plain pairs without changing the set of k-edit occurrences**.**
- **Theorem 7.37**: `TrimmedPM(`T, P, k, w, Q, $\mathcal{A}_P$, $\mathcal{A}_T$, $\mathit{Red}(P)$, $\mathit{Red}(T)$ α`)`, Reduction from the AlignedPeriodicMatches problem to the BleachCommitDPM problem**.**
  - **Definition 7.21**: Families of leading, internal, and trailing puzzle pieces [CKW22, Definition 4.8]**.**
  - **Lemma 7.25**: The median edit ditance of families of leading, internal, and trailing puzzle pieces is small, there are $O(k)$ special puzzle pieces that can be computed in $O(k)$ time [CKW22, Lemma 1.6]**.**
  - **Lemma 7.14**: $O(d_T+1)$-time algorithm for computing a collection of short substrings R_j of T that capture all k-edit occurrences and there are only a few distinct ones [CKW22, Section 3.2]**.**
  - **Lemma 7.23**: Upper bound on the cumulative difference of length between puzzle pieces comprising R_j and P [CKW22, Lemma 4.10]**.**
    - **Definition 7.21**: Families of leading, internal, and trailing puzzle pieces [CKW22, Definition 4.8]**.**
- **Lemma 7.25**: The median edit ditance of families of leading, internal, and trailing puzzle pieces is small, there are $O(k)$ special puzzle pieces that can be computed in $O(k)$ time [CKW22, Lemma 1.6]**.**
- **Theorem 9.10**: Efficient implementation for BleachCommitDPM**.**
  - **Lemma 9.2.**
  - **Lemma 9.1.**
  - **Theorem 9.3.**
    - **Remark 3.9**: Bounded median edit distance implies small size and small pairwise edit distance**.**
      - **Definition 3.8**: The median edit distance $\mathsf{ed}(\mathcal{S})$ of a family of strings $\mathcal{S}$**.**
    - **Theorem 5.25.**
      - **Lemma 3.18**: Paths of cost k that connect the top and bottom of the alignment graph may use at most k non-diagonal edges**.**
    - **Lemma 5.20.**
      - **Fact 5.2**: The core of any contiguous submatrix of a matrix A is at most $\delta(A)$**.**
      - **Fact 5.11**: Contiguous submatrices of integer-valued β-bounded-difference matrices are integer-valued and β-bounded-difference**.**
    - **Lemma 3.20**: Computing an optimal alignment of two strings of cost at most d (or deciding that no such alignment exists) takes $O(1+d^2)$ time in the `PILLAR` model, `Alignment (`S`,` T`)` [LV88]**.**
    - **Lemma 6.14**: `GrowFern(`P`,` T`,` k`,` w`).`
      (refer to entry of Lemma 6.14)
    - **Lemma 6.1**: Any instance of GrowFern with $|P|+|T|=O(k)$ can be solved in $O(k^2\log k)$**.**
      - **Corollary 3.19**: Using MSSP to compute boundary-to-boundary paths in the alignment graph**.**
        - **Theorem 3.10**: Efficient MSSP on planar graphs [Kle05]**.**

- • **Lemma 5.4**: Given a matrix A of size $p \times q$, we can construct a core-based matrix oracle for A in time $O(pq + \delta(A)\log(p+q))$ [GK25, Lemma 4.4].
- • **Lemma 5.36** [GK25, Theorem 5.10, simplified].
  - ○ **Lemma 5.6**: Given $\mathsf{CMO}(A)$ for some Monge matrix A, can compute $\mathsf{CMO}(C)$ for any contiguous submatrix C of A in time $O((N + \delta(A))\log N)$ where N is the maximum dimension of A.
  - ○ **Definition 5.3**: The Core-based Matrix Oracle $\mathsf{CMO}(A)$ of a matrix A [GK25, Lemma 4.4].
  - ○ **Lemma 5.12**: β-bounded-difference implies $\delta(A) \le 2\beta(\min\{p,q\} - 1)$ [GK25, Lemma 4.9].
- • **Lemma 9.1**.

- ○ **Lemma 5.20**.
  - • **Fact 5.2**: The core of any contiguous submatrix of a matrix A is at most $\delta(A)$.
  - • **Fact 5.11**: Contiguous submatrices of integer-valued β-bounded-difference matrices are integer-valued and β-bounded-difference.
- ○ **Lemma 6.14**: `GrowFern(`P`,` T`,` k`,` w`)`.
  (refer to entry of Lemma 6.14)
- ○ **Lemma 5.33**.
  - • **Fact 3.13**: The (min, +)-product of Monge matrices is Monge [Tis15, Theorem 2].
- ○ **Lemma 5.35**.
  - • **Lemma 5.12**: β-bounded-difference implies $\delta(A) \le 2\beta(\min\{p,q\} - 1)$ [GK25, Lemma 4.9].
  - • **Lemma 5.9**: Given k and $\mathsf{CMO}(A)$ and $\mathsf{CMO}(B)$ for Monge matrices A and B, can compute $\mathsf{CMO}(C')$ of a matrix C' that is k equivalent to $A \oplus B$ and $\delta(C') \le O(N\sqrt{k})$, in time $O((N + \delta(A) + \delta(B))\log N)$, where N is the maximum dimension of A and B [GK25, Lemma 4.14].
  - • **Fact 3.13**: The (min, +)-product of Monge matrices is Monge [Tis15, Theorem 2].
- ○ **Theorem 5.25**.
  - • **Lemma 3.18**: Paths of cost k that connect the top and bottom of the alignment graph may use at most k non-diagonal edges.

▬ **Lemma 8.20**: `HeavyMatches(`P`,` T`,` k`,` w`,` Q`,` $\mathcal{A}_P$`,` $\mathcal{A}_T$`,` $\mathbb{H}$`)` [CKW22, see Lemma 6.1].

- • **Lemma 8.18**: `Heavy(`$P, T, k, w, Q, \mathcal{L}^P, \mathcal{L}^T, \hat{\kappa}, \eta$`)` [CKW22, see Lemma 5.9].
  - ○ **Definition 8.14** [CKW22, see Definition 5.5].
  - ○ **Definition 8.17** [CKW22, Definition 5.8].
  - ○ **Fact 7.11**: `FindAWitness(`k`,` Q`,` S`)`, $O(k^2)$-time `PILLAR` algorithm that can be used to rotate Q such that $\mathsf{ed}(P, {}^*Q^*) = \mathsf{ed}(P, Q^*)$ [CKW20, compare Lemma 6.5].
- • **Lemma 7.14**: $O(d_T + 1)$-time algorithm for computing a collection of short substrings R_j of T that capture all k-edit occurrences and there are only a few distinct ones [CKW22, Section 3.2].
- • **Corollary 6.17**: `Verify(`P`,` T`,` k`,` I`,` w`)`.
  - ○ **Corollary 6.16**: `ListAllOccs(`P`,` T`,` k`,` w`)`.
    (refer to entry of Corollary 6.16)
- • **Lemma 8.19** [CKW22, see Lemma 5.10].
  - ○ **Lemma 8.15** [CKW22, see Lemma 5.6].
  - ○ **Definition 8.17** [CKW22, Definition 5.8].
  - ○ **Lemma 8.18**: `Heavy(`$P, T, k, w, Q, \mathcal{L}^P, \mathcal{L}^T, \hat{\kappa}, \eta$`)` [CKW22, see Lemma 5.9].
    - • **Definition 8.14** [CKW22, see Definition 5.5].
    - • **Definition 8.17** [CKW22, Definition 5.8].

- - **Fact 7.11**: `FindAWitness(`k`, `Q`, `S`)`, $O(k^2)$-time `PILLAR` algorithm that can be used to rotate Q such that $\mathsf{ed}(P, {}^{*}Q^{*}) = \mathsf{ed}(P, Q^{*})$ [CKW20, compare Lemma 6.5].

- **Lemma 8.44**: AlignedPeriodicMatches is in time $\widetilde{O}(k^{3.5}W^4)$ in the `PILLAR` model for metric integer weight functions.
  - **Lemma 7.40**: AlignedPeriodicMatches is in time $O(k^4 \log^2(mk))$ in the `PILLAR` model.
    (refer to entry of Lemma 7.40)
  - **Lemma 7.14**: $O(d_T + 1)$-time algorithm for computing a collection of short substrings R_j of T that capture all k-edit occurrences and there are only a few distinct ones [CKW22, Section 3.2].
  - **Section 8.4.**
  - **Theorem 7.37**: `TrimmedPM(`T`, `P`, `k`, `w`, `Q`, `$\mathcal{A}_P$`, `$\mathcal{A}_T$`, `$Red(P)$`, `$Red(T)$` `α`)`, Reduction from the AlignedPeriodicMatches problem to the BleachCommitDPM problem.
    - **Definition 7.21**: Families of leading, internal, and trailing puzzle pieces [CKW22, Definition 4.8].
    - **Lemma 7.25**: The median edit ditance of families of leading, internal, and trailing puzzle pieces is small, there are $O(k)$ special puzzle pieces that can be computed in $O(k)$ time [CKW22, Lemma 1.6].
    - **Lemma 7.14**: $O(d_T + 1)$-time algorithm for computing a collection of short substrings R_j of T that capture all k-edit occurrences and there are only a few distinct ones [CKW22, Section 3.2].
    - **Lemma 7.23**: Upper bound on the cumulative difference of length between puzzle pieces comprising R_j and P [CKW22, Lemma 4.10].
      - **Definition 7.21**: Families of leading, internal, and trailing puzzle pieces [CKW22, Definition 4.8].
  - **Corollary 8.13.**
    - **Corollary 6.17**: `Verify(`P`, `T`, `k`, `I`, `w`)`.
      - **Corollary 6.16**: `ListAllOccs(`P`, `T`, `k`, `w`)`.
        (refer to entry of Corollary 6.16)
    - **Lemma 8.12**: `Locked(`S`, `Q`, `d`, `h`,`w`)`, see [CKW20, Lemma 6.9].
      - **Lemma 8.10** [CKW20, see Lemma 5.11].
        - **Lemma 8.5** [CKW20, see Lemma 5.6].
      - **Fact 7.11**: `FindAWitness(`k`, `Q`, `S`)`, $O(k^2)$-time `PILLAR` algorithm that can be used to rotate Q such that $\mathsf{ed}(P, {}^{*}Q^{*}) = \mathsf{ed}(P, Q^{*})$ [CKW20, compare Lemma 6.5].
      - **Lemma 8.5** [CKW20, see Lemma 5.6].
  - **Lemma 8.18**: `Heavy(`$P, T, k, w, Q, \mathcal{L}^P, \mathcal{L}^T, \hat{\kappa}, \eta$`)` [CKW22, see Lemma 5.9].
    - **Definition 8.14** [CKW22, see Definition 5.5].
    - **Definition 8.17** [CKW22, Definition 5.8].
    - **Fact 7.11**: `FindAWitness(`k`, `Q`, `S`)`, $O(k^2)$-time `PILLAR` algorithm that can be used to rotate Q such that $\mathsf{ed}(P, {}^{*}Q^{*}) = \mathsf{ed}(P, Q^{*})$ [CKW20, compare Lemma 6.5].
  - **Definition 8.29** [CKW22, see Definition 6.10].
  - **Lemma 8.43** [CKW22, see Lemma 6.11].
    - **Corollary 8.30.**
      - **Lemma 7.14**: $O(d_T + 1)$-time algorithm for computing a collection of short substrings R_j of T that capture all k-edit occurrences and there are only a few distinct ones [CKW22, Section 3.2].
      - **Lemma 7.30**: If the torsion is small, we can trim/expand sequences of at least $k + 2$ plain pairs without changing the set of k-edit occurrences.
    - **Lemma 8.37** [CKW22, see Lemma 6.19].

- **Definition 8.14** [CKW22, see Definition 5.5].
- **Lemma 8.27** [CKW22, see Lemma 6.8].
  - **Lemma 8.22** [CKW22, see Lemma 6.3].
  - **Lemma 8.12**: `Locked(`S`,` Q`,` d`,` h`,`w`)`, see [CKW20, Lemma 6.9].
    - **Lemma 8.10** [CKW20, see Lemma 5.11].
      - **Lemma 8.5** [CKW20, see Lemma 5.6].
    - **Fact 7.11**: `FindAWitness(`k`,` Q`,` S`)`, $O(k^2)$-time `PILLAR` algorithm that can be used to rotate Q such that $\mathsf{ed}(P, {}^{*}Q^{*}) = \mathsf{ed}(P, Q^{*})$ [CKW20, compare Lemma 6.5].
    - **Lemma 8.5** [CKW20, see Lemma 5.6].
- **Lemma 8.36** [CKW22, see Lemma 6.19].
- **Lemma 8.31** [CKW22, see Lemma 6.14].
  - **Lemma 7.23**: Upper bound on the cumulative difference of length between puzzle pieces comprising R_j and P [CKW22, Lemma 4.10].
    - **Definition 7.21**: Families of leading, internal, and trailing puzzle pieces [CKW22, Definition 4.8].
  - **Lemma 8.27** [CKW22, see Lemma 6.8].
    - **Lemma 8.22** [CKW22, see Lemma 6.3].
    - **Lemma 8.12**: `Locked(`S`,` Q`,` d`,` h`,`w`)`, see [CKW20, Lemma 6.9].
      - **Lemma 8.10** [CKW20, see Lemma 5.11].
        - **Lemma 8.5** [CKW20, see Lemma 5.6].
      - **Fact 7.11**: `FindAWitness(`k`,` Q`,` S`)`, $O(k^2)$-time `PILLAR` algorithm that can be used to rotate Q such that $\mathsf{ed}(P, {}^{*}Q^{*}) = \mathsf{ed}(P, Q^{*})$ [CKW20, compare Lemma 6.5].
      - **Lemma 8.5** [CKW20, see Lemma 5.6].
- **Lemma 8.23** [CKW22, see Lemma 6.4].
  - **Lemma 8.12**: `Locked(`S`,` Q`,` d`,` h`,`w`)`, see [CKW20, Lemma 6.9].
    - **Lemma 8.10** [CKW20, see Lemma 5.11].
      - **Lemma 8.5** [CKW20, see Lemma 5.6].
    - **Fact 7.11**: `FindAWitness(`k`,` Q`,` S`)`, $O(k^2)$-time `PILLAR` algorithm that can be used to rotate Q such that $\mathsf{ed}(P, {}^{*}Q^{*}) = \mathsf{ed}(P, Q^{*})$ [CKW20, compare Lemma 6.5].
    - **Lemma 8.5** [CKW20, see Lemma 5.6].
  - **Fact 8.21** [CKW22, see Fact 6.2].
  - **Lemma 8.22** [CKW22, see Lemma 6.3].
- **Fact 8.35** [CKW22, see Fact 6.18].

- **Lemma 8.20**: `HeavyMatches(`P`,` T`,` k`,` w`,` Q`,` $\mathcal{A}_P$`,` $\mathcal{A}_T$`,` $\mathbb{H}$`)` [CKW22, see Lemma 6.1].
  (refer to entry of Lemma 8.20)
- **Lemma 8.28** [CKW22, see Lemma 6.9].
  - **Lemma 7.25**: The median edit ditance of families of leading, internal, and trailing puzzle pieces is small, there are $O(k)$ special puzzle pieces that can be computed in $O(k)$ time [CKW22, Lemma 1.6].
  - **Lemma 8.12**: `Locked(`S`,` Q`,` d`,` h`,`w`)`, see [CKW20, Lemma 6.9].
    - **Lemma 8.10** [CKW20, see Lemma 5.11].
      - **Lemma 8.5** [CKW20, see Lemma 5.6].
    - **Fact 7.11**: `FindAWitness(`k`,` Q`,` S`)`, $O(k^2)$-time `PILLAR` algorithm that can be used to rotate Q such that $\mathsf{ed}(P, {}^{*}Q^{*}) = \mathsf{ed}(P, Q^{*})$ [CKW20, compare Lemma 6.5].
    - **Lemma 8.5** [CKW20, see Lemma 5.6].
  - **Lemma 7.14**: $O(d_T + 1)$-time algorithm for computing a collection of short substrings R_j of T that capture all k-edit occurrences and there are only a few distinct ones [CKW22, Section 3.2].

- **Theorem 7.37**: `TrimmedPM`$(T, P, k, w, Q, \mathcal{A}_P, \mathcal{A}_T, \mathit{Red}(P), \mathit{Red}(T)\ \alpha)$, Reduction from the AlignedPeriodicMatches problem to the BleachCommitDPM problem.
  - **Definition 7.21**: Families of leading, internal, and trailing puzzle pieces [CKW22, Definition 4.8].
  - **Lemma 7.25**: The median edit ditance of families of leading, internal, and trailing puzzle pieces is small, there are $O(k)$ special puzzle pieces that can be computed in $O(k)$ time [CKW22, Lemma 1.6].
  - **Lemma 7.14**: $O(d_T + 1)$-time algorithm for computing a collection of short substrings R_j of T that capture all k-edit occurrences and there are only a few distinct ones [CKW22, Section 3.2].
  - **Lemma 7.23**: Upper bound on the cumulative difference of length between puzzle pieces comprising R_j and P [CKW22, Lemma 4.10].
    - **Definition 7.21**: Families of leading, internal, and trailing puzzle pieces [CKW22, Definition 4.8].
- **Theorem 9.10**: Efficient implementation for BleachCommitDPM.
  - **Lemma 9.2**.
  - **Lemma 9.1**.
  - **Theorem 9.3**.
    - **Remark 3.9**: Bounded median edit distance implies small size and small pairwise edit distance.
      - **Definition 3.8**: The median edit distance $\mathsf{ed}(\mathcal{S})$ of a family of strings $\mathcal{S}$.
    - **Theorem 5.25**.
      - **Lemma 3.18**: Paths of cost k that connect the top and bottom of the alignment graph may use at most k non-diagonal edges.
    - **Lemma 5.20**.
      - **Fact 5.2**: The core of any contiguous submatrix of a matrix A is at most $\delta(A)$.
      - **Fact 5.11**: Contiguous submatrices of integer-valued β-bounded-difference matrices are integer-valued and β-bounded-difference.
    - **Lemma 3.20**: Computing an optimal alignment of two strings of cost at most d (or deciding that no such alignment exists) takes $O(1 + d^2)$ time in the `PILLAR` model, `Alignment (`S`,` T`)` [LV88].
    - **Lemma 6.14**: `GrowFern`(P, T, k, w).
      (refer to entry of Lemma 6.14)
    - **Lemma 6.1**: Any instance of GrowFern with $|P| + |T| = O(k)$ can be solved in $O(k^2 \log k)$.
      - **Corollary 3.19**: Using MSSP to compute boundary-to-boundary paths in the alignment graph.
        - **Theorem 3.10**: Efficient MSSP on planar graphs [Kle05].
    - **Lemma 5.4**: Given a matrix A of size $p \times q$, we can construct a core-based matrix oracle for A in time $O(pq + \delta(A) \log(p + q))$ [GK25, Lemma 4.4].
    - **Lemma 5.36** [GK25, Theorem 5.10, simplified].
      - **Lemma 5.6**: Given $\mathsf{CMO}(A)$ for some Monge matrix A, can compute $\mathsf{CMO}(C)$ for any contiguous submatrix C of A in time $O((N + \delta(A)) \log N)$ where N is the maximum dimension of A.
      - **Definition 5.3**: The Core-based Matrix Oracle $\mathsf{CMO}(A)$ of a matrix A [GK25, Lemma 4.4].
      - **Lemma 5.12**: β-bounded-difference implies $\delta(A) \le 2\beta(\min\{p, q\} - 1)$ [GK25, Lemma 4.9].
    - **Lemma 9.1**.

- **Lemma 5.20.**
  - **Fact 5.2**: The core of any contiguous submatrix of a matrix A is at most $\delta(A)$.
  - **Fact 5.11**: Contiguous submatrices of integer-valued β-bounded-difference matrices are integer-valued and β-bounded-difference.
- **Lemma 6.14**: `GrowFern(`P`, `T`, `k`, `w`)`.
  (refer to entry of Lemma 6.14)
- **Lemma 5.33.**
  - **Fact 3.13**: The (min, +)-product of Monge matrices is Monge [Tis15, Theorem 2].
- **Lemma 5.35.**
  - **Lemma 5.12**: β-bounded-difference implies $\delta(A) \leq 2\beta(\min\{p, q\} - 1)$ [GK25, Lemma 4.9].
  - **Lemma 5.9**: Given k and $\mathsf{CMO}(A)$ and $\mathsf{CMO}(B)$ for Monge matrices A and B, can compute $\mathsf{CMO}(C')$ of a matrix C' that is k equivalent to $A \oplus B$ and $\delta(C') \leq O(N\sqrt{k})$, in time $O((N + \delta(A) + \delta(B)) \log N)$, where N is the maximum dimension of A and B [GK25, Lemma 4.14].
  - **Fact 3.13**: The (min, +)-product of Monge matrices is Monge [Tis15, Theorem 2].
- **Theorem 5.25.**
  - **Lemma 3.18**: Paths of cost k that connect the top and bottom of the alignment graph may use at most k non-diagonal edges.

- **Lemma 8.18**: `Heavy`$(P, T, k, w, Q, \mathcal{L}^P, \mathcal{L}^T, \hat{\kappa}, \eta)$ [CKW22, see Lemma 5.9].
  - **Definition 8.14** [CKW22, see Definition 5.5].
  - **Definition 8.17** [CKW22, Definition 5.8].
  - **Fact 7.11**: `FindAWitness(`k`, `Q`, `S`)`, $O(k^2)$-time `PILLAR` algorithm that can be used to rotate Q such that $\mathsf{ed}(P, {}^*Q^*) = \mathsf{ed}(P, Q^*)$ [CKW20, compare Lemma 6.5].
- **Lemma 8.19** [CKW22, see Lemma 5.10].
  - **Lemma 8.15** [CKW22, see Lemma 5.6].
  - **Definition 8.17** [CKW22, Definition 5.8].
  - **Lemma 8.18**: `Heavy`$(P, T, k, w, Q, \mathcal{L}^P, \mathcal{L}^T, \hat{\kappa}, \eta)$ [CKW22, see Lemma 5.9].
    - **Definition 8.14** [CKW22, see Definition 5.5].
    - **Definition 8.17** [CKW22, Definition 5.8].
    - **Fact 7.11**: `FindAWitness(`k`, `Q`, `S`)`, $O(k^2)$-time `PILLAR` algorithm that can be used to rotate Q such that $\mathsf{ed}(P, {}^*Q^*) = \mathsf{ed}(P, Q^*)$ [CKW20, compare Lemma 6.5].