This is the accepted version of:
P. Charalampopoulos, T. Kociumaka and P. Wellnitz,"Faster Pattern Matching under Edit Distance : A Reduction to Dynamic Puzzle Matching and the Seaweed Monoid of Permutation Matrices," 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS), 2022, pp. 698-707, doi: 10.1109/FOCS54457.2022.00072".

Faster Pattern Matching under Edit Distance

A Reduction to Dynamic Puzzle Matching and the Seaweed Monoid of Permutation Matrices

Panagiotis Charalampopoulos
Department of Computer Science and Information Systems
Birkbeck, University of London
London, UK
p.charalampopoulos@bbk.ac.uk

Tomasz Kociumaka
Max Planck Institute for Informatics
Saarland Informatics Campus
Saarbrücken, Germany
tomasz.kociumaka@mpi-inf.mpg.de

Philip Wellnitz
Max Planck Institute for Informatics
Saarland Informatics Campus
Saarbrücken, Germany
wellnitz@mpi-inf.mpg.de

***Abstract*—We consider the approximate pattern matching problem under the *edit distance*. Given a text T of length n, a pattern P of length m, and a threshold k, the task is to find the starting positions of all substrings of T that can be transformed to P with at most k *edits*. More than 20 years ago, Cole and Hariharan [SODA'98, J. Comput.'02] gave an $\mathcal{O}(n+k^4\cdot n/m)$-time algorithm for this classic problem, and this runtime has not been improved since.**

Here, we present an algorithm that runs in time $\mathcal{O}(n + k^{3.5}\sqrt{\log m \log k}\cdot n/m)$, thus breaking through this long-standing barrier. In the case where $n^{1/4+\varepsilon} \le k \le n^{2/5-\varepsilon}$ for some arbitrarily small positive constant ε, our algorithm improves over the state-of-the-art by polynomial factors: it is polynomially faster than both the algorithm of Cole and Hariharan and the classic $\mathcal{O}(kn)$-time algorithm of Landau and Vishkin [STOC'86, J. Algorithms'89].

We observe that the bottleneck case of the alternative $\mathcal{O}(n + k^4\cdot n/m)$-time algorithm of Charalampopoulos, Kociumaka, and Wellnitz [FOCS'20] is when the text and the pattern are (almost) periodic. Our new algorithm reduces this case to a new *Dynamic Puzzle Matching* problem, which we solve by building on tools developed by Tiskin [SODA'10, Algorithmica'15] for the so-called seaweed monoid of permutation matrices. Our algorithm relies only on a small set of primitive operations on strings and thus also applies to the fully-compressed setting (where text and pattern are given as straight-line programs) and to the dynamic setting (where we maintain a collection of strings under creation, splitting, and concatenation), improving over the state of the art.

***Index Terms*—approximate pattern matching, edit distance**

I. Introduction

Almost every introductory algorithms textbook covers the *pattern matching* problem: in a given text T of length n, we wish to find all occurrences of a given pattern P of length m. As fundamental as both this problem and its solutions are by today, as apparent are their limitations: a single surplus or missing character in the pattern (or in a potential occurrence) results in (potentially all) occurrences being missed. Hence, a large body of work focuses on *approximate pattern matching*, where we want to identify substrings of the text that are *close* to the pattern. In particular, in this paper, we consider a classic variant of approximate pattern matching where we allow for up to k character insertions, deletions, and substitutions (collectively: edits); that is, we consider approximate pattern matching under the edit distance.

Formally, for two strings X and Y, their edit distance (also known as the Levenshtein distance) $\delta_E(X,Y)$ is the minimum number of insertions, deletions, and substitutions of single characters required to transform X into Y. Now, in the *pattern matching with edits* problem, for a given text T, pattern P, and an integer threshold $k > 0$, the task is to find the starting positions of all *k-error* (or *k-edit*) *occurrences* of P in T. Specifically, we wish to list all positions v in T such that the edit distance between $T[\,v\,..\,w\,) \coloneqq T[\,v\,]T[\,v+1\,]\cdots T[\,w-1\,]$ and P is at most k for some position w; we write $\mathrm{Occ}_k^E(P,T)$ to denote the set of all such positions v.

Let us highlight the main prior results for pattern matching with edits; for a thorough review of other (in particular) early results on pattern matching with edits, we refer to the extensive survey of Navarro [21]. Back in 1980, Sellers [23] demonstrated how the standard dynamic-programming algorithm for computing $\delta_E(P,T)$ can be adapted to an $\mathcal{O}(nm)$-time algorithm for the pattern matching with edits problem. Around the same time, Masek and Paterson [20] reduced the running time by a poly-logarithmic factor using the Four-Russians technique. Only several year later, Landau and Vishkin [18] presented an $\mathcal{O}(nk^2)$-time solution, which they could then improve to the—by now—classic "kangaroo jumping" algorithm that solves this problem in $\mathcal{O}(nk)$ time [19]. In search of even faster methods, Sahinalp and Vishkin [22] developed an algorithm that runs in time $\mathcal{O}(n+nk^{8+1/3}(\log^* n)^{1/3}/m^{1/3})$—this algorithm was then improved by Cole and Hariharan [10], who gave an $\mathcal{O}(n+k^4n/m)$-time solution, which is asymptotically faster than the aforementioned Landau–Vishkin algorithm

Funding: Panagiotis Charalampopoulos: Work done while at Reichman University, Israel, partly supported by Israel Science Foundation grant 810/21. Tomasz Kociumaka: Work done while at University of California, Berkeley, partly supported by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship.

A full version of this paper is available at arxiv.org/abs/2204.03087. Proofs of the claims marked with $\diamondsuit$ are presented only in the full version.

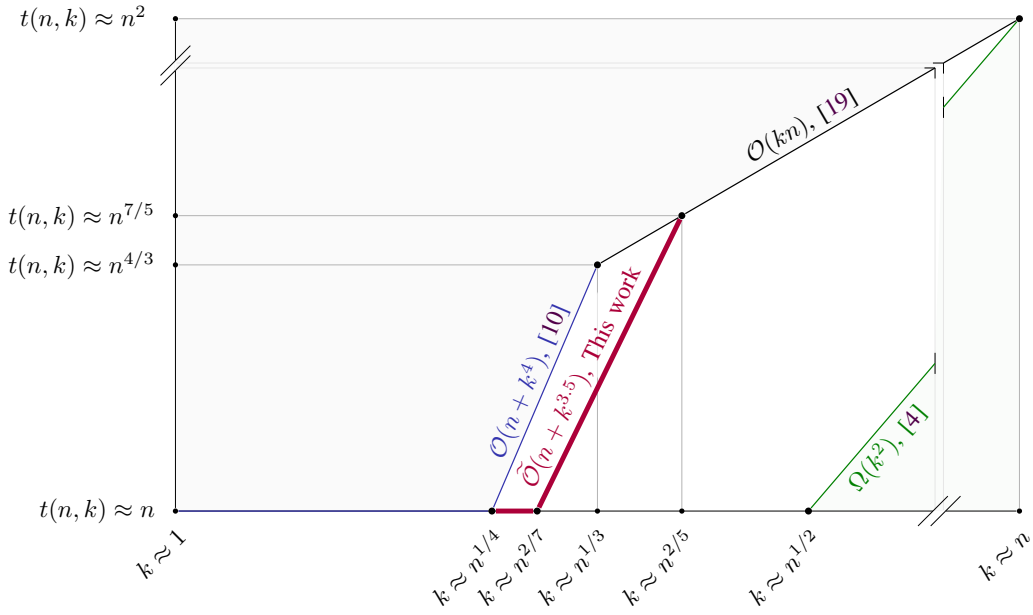

Fig. 1. The running time $t(n,k)$ of algorithms for the approximate pattern matching problem under the edit distance as a function of k for the important special case where $m = \Theta(n)$. The scale is doubly logarithmic and sub-polynomial factors are hidden; running times below n and above n^2 are not relevant, neither are values of k that lie above n. Any point that lies strictly to the bottom-right of the green line segment is unattainable unless SETH fails.

when $k = o(\sqrt[3]{m})$, and in that setting remained the fastest known algorithm prior to this work.

From a lower-bound perspective, we can benefit from the discovery that the classic quadratic-time algorithm for computing the edit distance of two strings is essentially optimal: Backurs and Indyk [4] recently proved that any polynomial-factor improvement would yield a major breakthrough for the satisfiability problem. For pattern matching with edits, this means that there is no hope for an algorithm running in time $\mathcal{O}(n + k^{2-\varepsilon}n/m)$ for any constant $\varepsilon > 0$: given an $\mathcal{O}(n + k^{2-\varepsilon}n/m)$-time algorithm for pattern matching with edits, we could compute the edit distance of any two given strings X and Y of total length N over an alphabet Σ in time $\mathcal{O}(N^{2-\varepsilon}\log N)$. Specifically, we pad X and Y to $P \coloneqq \$^{2N}X\$^{2N}$ and $T \coloneqq \$^{2N}Y\$^{2N}$, where $\$ \notin \Sigma$. Now, due to $\min_{v,w} \delta_E(P, T[v\,..\,w)) = \delta_E(P,T) = \delta_E(X,Y)$, we can binary search for the smallest value of k such that $\mathrm{Occ}_k^E(P,T)$ is not empty.

Despite the large gap between the quadratic and bi-quadratic dependency on k, no further advancements have been made to settle the running time of the pattern matching with edits problem. In particular, there has not even been any progress on resolving the 24-year-old conjecture of Cole and Hariharan [10] that an $\mathcal{O}(n + k^3 n/m)$-time algorithm *should be possible*—until now. We give the first algorithm that improves over the running time achieved by Cole and Hariharan [10]:

Theorem I.1. *Given a text T of length n, a pattern P of length m, and an integer threshold $k > 0$, we can compute the set $\mathrm{Occ}_k^E(P,T)$ in $\mathcal{O}(n + n/m \cdot k^{3.5}\sqrt{\log m \log k})$ time.*

Observe that if k is roughly between $n^{1/4}$ and $n^{2/7}$, we obtain the first linear-time algorithm for the important special case where text and pattern are close in length. Further, we still obtain polynomial improvements in the running time for values of k that are roughly less than $n^{2/5}$. Consult Figure 1 for a graphical comparison of the running times of our algorithm with the previous state-of-the-art and the conditional lower bound discussed above.

The PILLAR *Model and Faster Algorithms in Other Settings*

The generality of our approach allows for an easy adaptation to settings where the text and the pattern are not given explicitly, such as the dynamic setting (cf., Theorem I.3) and the fully-compressed setting (cf., Theorem I.4). In particular, we follow the approach by Charalampopoulos, Kociumaka, and Wellnitz [8] and implement the algorithm in the so-called PILLAR model. In that model, one bounds the running times of algorithms in terms of the number of calls to a small set of very common operations (the PILLAR operations) on strings, such as computing the length of their longest common prefix. Then, for any setting, an efficient implementation of the PILLAR operations yields a fast algorithm for approximate pattern matching. For pattern matching with edits, [8] presented an algorithm that runs in $\mathcal{O}(n/m \cdot k^4)$ time in the PILLAR model. We improve upon their algorithm.

Theorem I.2. *Given a pattern P of length m, a text T of length n, and an integer threshold $k > 0$, we can compute a representation of the set $\mathrm{Occ}_k^E(P,T)$ as $\mathcal{O}(n/m \cdot k^3)$ arithmetic progressions with the same difference in $\mathcal{O}(n/m \cdot k^{3.5}\sqrt{\log m \log k})$ time in the* PILLAR *model.*

Consistently with [8], we represent the set $\mathrm{Occ}_k^E(P,T)$ as $\mathcal{O}(k^3)$ disjoint arithmetic progressions with a common differ-

ence. Unless P is almost periodic, though, $\mathrm{Occ}_k^E(P,T)$ is of size $\mathcal{O}(k^2)$, and we can list the k-error occurrences explicitly; see [8] for a structural characterization of $\mathrm{Occ}_k^E(P,T)$.

In the standard setting, where the text and the pattern are both given explicitly, after an $\mathcal{O}(n)$-time preprocessing, we can perform each primitive PILLAR operation in constant time. We thus instantly obtain Theorem I.1. The same PILLAR implementation remains valid in the *internal setting* introduced in [14]. Specifically, after a linear-time preprocessing of an input string X, the algorithm of Theorem I.2 can efficiently compute $\mathrm{Occ}_k^E(P,T)$ for any two fragments P and T of the string X.

In the full version (see also [8]), we show that existing implementations of the primitive operations of the PILLAR model allow us to also obtain efficient algorithms for pattern matching under edit distance in the fully-compressed setting (where the text and the pattern are given as straight-line programs) and in the dynamic setting (where we maintain a collection of strings under creation, splitting, and concatenation). Our algorithms improve over the state-of-the-art algorithms of [8] for these settings: we trade a $\sqrt{k}$ factor for a factor that is asymptotically upper-bounded by the logarithm of the length of the considered pattern. Formally, we obtain the following results.

Theorem I.3 ($\diamondsuit$)**.** *We can maintain a collection $\mathcal{X}$ of non-empty persistent strings of total length N subject to* $\texttt{makestring}(U)$, $\texttt{concat}(U,V)$, *and* $\texttt{split}(U,i)$ *operations that require $\mathcal{O}(\log N + |U|)$, $\mathcal{O}(\log N)$, and $\mathcal{O}(\log N)$ time, respectively, so that given two strings $P,T \in \mathcal{X}$, and an integer threshold $k > 0$, we can compute a representation of $\mathrm{Occ}_k^E(P,T)$ as $\mathcal{O}(|T|/|P| \cdot k^3)$ arithmetic progressions with the same difference in time $\mathcal{O}(|T|/|P| \cdot k^{3.5}\sqrt{\log|P|\log k}\log^2 N)$.*[1]

Theorem I.4 ($\diamondsuit$)**.** *Let $\mathcal{G}_T$ denote a straight-line program of size n generating a string T, let $\mathcal{G}_P$ denote a straight-line program of size m generating a string P, let $k > 0$ denote an integer threshold, and set $N := |T|$ and $M := |P|$. We can compute $|\mathrm{Occ}_k^E(P,T)|$ in time $\mathcal{O}(m\log N + n\,k^{3.5}\log^2 N\sqrt{\log M\log k}\log\log N)$ and we can report the elements of $\mathrm{Occ}_k^E(P,T)$ within $\mathcal{O}(|\mathrm{Occ}_k^E(P,T)|)$ extra time.*

II. Related Work

Pattern Matching with Mismatches

The Hamming distance of two (equal-length) strings is the number of positions where the strings differ. This metric is more restrictive than edit distance since it allows substitutions but does not support insertions or deletions.

In the *pattern matching with mismatches* problem, we are given a text T of length n, a pattern P of length m, and an integer threshold $k > 0$, and we wish to compute the k-*mismatch occurrences* of P in T, that is, all length-m substrings of T that are at Hamming distance at most k from P. This problem has been extensively studied since the 1980s. A long line of works [1], [3], [6], [9], [11], [12], [15], [17] has culminated in an $\widetilde{\mathcal{O}}(n + kn/\sqrt{m})$-time algorithm,[2] presented by Gawrychowski and Uznański [12], who also showed that no significantly faster "combinatorial" algorithm exists. In other words, any polynomial-factor improvement over the result of [12] would require using fast matrix multiplication; nevertheless, the existence of such an "algebraic" solution remains a challenging open question.

As shown in [8], pattern matching with mismatches admits an $\widetilde{\mathcal{O}}(k^2 \cdot n/m)$-time algorithm in the PILLAR model. Analogously to pattern matching with edits, this solution constitutes the basis of the state-of-the-art algorithms in the internal, fully-compressed, and dynamic settings.

Online Algorithms for Pattern Matching with Edits

The pattern matching with edits problem has also been considered in the online setting, where the text arrives character by character and, by the time $T[\,w\,]$ becomes available, the algorithm needs to decide whether $\min_v \delta_E(P, T[\,v\,..\,w\,)) \le k$. Landau, Myers, and Schmidt [16] provided an online algorithm that runs in $\mathcal{O}(k)$ time per character. Subsequent work focused on the streaming model, whether the main emphasis is on reducing the space complexity of an online algorithm, usually at the cost of introducing Monte-Carlo randomization. Starikovskaya [24] presented an algorithm for this setting with both the space usage and the time required to process each character of the text being proportional to $\sqrt{m}(k\log m)^{\mathcal{O}(1)}$. Very recently, Kociumaka, Porat, and Starikovskaya [13], improved upon this result, presenting an algorithm that uses $\widetilde{\mathcal{O}}(k^5)$ space and processes each character of the text in $\widetilde{\mathcal{O}}(k^8)$ amortized time.

Approximating Pattern Matching with Edits

Chakraborty, Das, and Koucký [5] presented an $\widetilde{\mathcal{O}}(nm^{3/4})$-time algorithm that produces, for each position w of the text, a constant-factor approximation of $\min_v \delta_E(P, T[\,v\,..\,w\,))$. They also provided an online algorithm with a weaker approximation guarantee.

III. Technical Overview

For a string P (also called a *pattern*), a string T (also called a *text*), and an integer $k > 0$ (also called a *threshold*), we say that P has a k-*error occurrence* in T at position v if $\delta_E(P, T[\,v\,..\,w\,)) \le k$ holds for some $w \ge v$. We write $\mathrm{Occ}_k^E(P,T)$ to denote the set of the starting positions of k-error occurrences of P in T, that is,

$$\mathrm{Occ}_k^E(P,T) := \{v : \exists_{w \ge v}\ \delta_E(P, T[\,v\,..\,w\,)) \le k\}.$$

We now formally state the *pattern matching with edits* problem.

PMWithEdits(P,T,k)
Input: A pattern P of length m, a text T of length n, and a positive integer $k \le m$.
Output: The set $\mathrm{Occ}_k^E(P,T)$.

[1] All running time bounds hold with high probability (that is, $1 - 1/N^{\Omega(1)}$). A deterministic version can be obtained at the cost of a $\mathrm{poly}(\log\log N)$-factor overhead.

[2] Here and throughout, $\widetilde{\mathcal{O}}(\star)$ hides $\log^{\mathcal{O}(1)} n$ factors.

A. The AlignedPeriodicMatches Problem

Let us start with a short exposition of parts of our notation.[3] A string S is *primitive* if it cannot be expressed as U^y for a string U and an integer $y > 1$. For two strings U and V, we write

$$\delta_E(U, V^*) \coloneqq \min\{\delta_E(U, V^\infty[\,0\,..\,j\,)) : j \in \mathbb{Z}_{\geq 0}\}$$

to denote the minimum edit distance between U and any prefix of $V^\infty = V \cdot V \cdots$. Further, we write

$$\delta_E(U, {}^*V^*) \coloneqq \min\{\delta_E(U, V^\infty[\,i\,..\,j\,)) : i, j \in \mathbb{Z}_{\geq 0}, i \leq j\}$$

to denote the minimum edit distance between U and any substring of V^∞.

As we explain in the full version, a recent algorithm of Charalampopoulos, Kociumaka, and Wellnitz [8] reduces the PMWITHEDITS problem to several instances of the following restricted variant. The input of this variant additionally contains a string Q that is a common approximate period of P and T, as well as witness edit-distance alignments $\mathcal{A}_P$ and $\mathcal{A}_T$, which describe how to transform strings P and T, respectively, to appropriate substrings of Q^∞ using $\mathcal{O}(d)$ edits.

ALIGNEDPERIODICMATCHES$(P, T, k, d, Q, \mathcal{A}_P, \mathcal{A}_T)$
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 $q \coloneqq |Q| \leq m/8d$,
- an edit-distance alignment $\mathcal{A}_P : P \rightsquigarrow Q^\infty[\,0\,..\,y_P\,)$ of cost $d_P \coloneqq \delta_E(P, {}^*Q^*) = \delta_E(P, Q^*) \leq d$, and
- an edit-distance alignment $\mathcal{A}_T : T \rightsquigarrow Q^\infty[\,x_T\,..\,y_T\,)$ of cost $d_T \coloneqq \delta_E(T, {}^*Q^*) \leq 3d$, where $x_T \in [\,0\,..\,q\,)$.

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

Specifically, [8] implies the following reduction.

Fact III.1 ($\diamondsuit$)**.** *Given an instance (P, T, k) of the* PMWITHEDITS *problem, one can compute a representation of the set $\mathrm{Occ}_k^E(P, T)$ as $\mathcal{O}(n/m \cdot k^3)$ disjoint arithmetic progressions with the same difference in time $\mathcal{O}(n/m \cdot k^3)$ in the* PILLAR *model plus the time required for solving several instances* ALIGNEDPERIODICMATCHES$(P_i, T_i, k_i, d_i, Q_i, \mathcal{A}_{P_i}, \mathcal{A}_{T_i})$*, where $\sum_i |P_i| = \mathcal{O}(n)$ and, for each i, we have $|P_i| \leq m$ and $d_i = \lceil 8k/m \cdot |P_i|\rceil$.*

Remark III.2. When Fact III.1 is applied to an instance of the PMWITHEDITS problem such that the pattern P is approximately periodic, the input $(P_i, T_i, k_i, d_i, Q_i, \mathcal{A}_{P_i}, \mathcal{A}_{T_i})$ to each produced instance of ALIGNEDPERIODICMATCHES satisfies the following conditions: $P_i = P$, T_i is a fragment of T, $k_i = k$, and $d = \mathcal{O}(k)$. For the purposes of this technical overview, one can focus solely on that case.

Using the algorithm of [8, Lemma 6.11] to solve the ALIGNEDPERIODICMATCHES problem in $\mathcal{O}(d^4)$ time in the PILLAR model, the time (in the PILLAR model) required for solving all instances of the ALIGNEDPERIODICMATCHES problem that are generated by Fact III.1 is

$$\begin{aligned}\sum_i \mathcal{O}(d_i^4) &= \sum_i \mathcal{O}(k^4/m^4 \cdot |P_i|^4)\\ &= \sum_i \mathcal{O}(k^4/m \cdot |P_i|) = \mathcal{O}(n/m \cdot k^4).\end{aligned}$$

In particular, we can reinterpret the $\mathcal{O}(n/m \cdot k^4)$-time algorithm of [8] for the PMWITHEDITS problem as a combination of Fact III.1 and [8, Lemma 6.11]. Our main contribution is the following faster algorithm for the ALIGNEDPERIODICMATCHES problem.

Lemma III.3 (`AlignedPeriodicMatches(`P`, `T`, `k`, `d`, `Q`, `$\mathcal{A}_P$`, `$\mathcal{A}_T$`)`, $\diamondsuit$)**.** *We can solve the* ALIGNEDPERIODICMATCHES *problem in $\mathcal{O}(d^{3.5}\sqrt{\log n \log d})$ time in the* PILLAR *model.*

By combining Fact III.1 and Lemma III.3, we obtain Theorem I.2.

Theorem I.2. *Given a pattern P of length m, a text T of length n, and an integer threshold $k > 0$, we can compute a representation of the set $\mathrm{Occ}_k^E(P, T)$ as $\mathcal{O}(n/m \cdot k^3)$ arithmetic progressions with the same difference in $\mathcal{O}(n/m \cdot k^{3.5}\sqrt{\log m \log k})$ time in the* PILLAR *model.*

Proof. By Fact III.1, in $\mathcal{O}(n/m \cdot k^3)$ time, we can reduce the PMWITHEDITS problem to several instances ALIGNEDPERIODICMATCHES$(P_i, T_i, k_i, d_i, Q_i, \mathcal{A}_{P_i}, \mathcal{A}_{T_i})$, where $\sum_i |P_i| = \mathcal{O}(n)$ and, for each i, we have $|P_i| \leq m$ and $d_i = \lceil 8k/m \cdot |P_i|\rceil$. By Lemma III.3, the time required for solving all the obtained instances (in the PILLAR model) is

$$\begin{aligned}&\sum_i \mathcal{O}(d_i^{3.5}\sqrt{\log |T_i| \log d_i})\\ &\quad= \sum_i \mathcal{O}(k^{3.5}/m^{3.5} \cdot |P_i|^{3.5}\sqrt{\log m \log k})\\ &\quad= \sum_i \mathcal{O}(k^{3.5}/m \cdot |P_i|\sqrt{\log m \log k})\\ &\quad= \mathcal{O}(n/m \cdot k^{3.5}\sqrt{\log m \log k}).\end{aligned}$$

□

B. A Fast Algorithm for the AlignedPeriodicMatches Problem

We continue with a high-level description of the algorithm that underlies Lemma III.3. In what follows we assume for simplicity that $m/k \gg q \gg k$ and that both $Q^\infty[\,0\,..\,y_P\,)$ and $Q^\infty[\,x_T\,..\,y_T\,)$ are powers of Q.

A First Solution via the DynamicPuzzleMatching Problem: Let us first discuss how the (almost) periodicity of P and T yields a simple way to *filter out* many potential starting positions of k-error occurrences.

[3] Consult also the full version, where we provide a comprehensive exposition of the notation used throughout this paper, including those we consider standard.

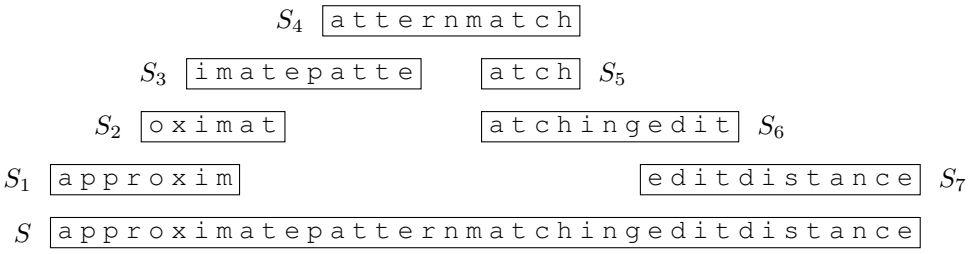

Fig. 2. $S_1, \ldots, S_7$ is a 4-puzzle whose value is S.

As an introductory example, suppose that P and T are perfectly periodic with period Q, that is, $P = Q^\infty[\,0\,..\,m\,)$ and $T = Q^\infty[\,0\,..\,n\,)$. Observe that in this special case, $\mathcal{A}_P$ and $\mathcal{A}_T$ are cost-0 alignments (that is, $d_T = d_P = 0$), and we have $m = y_P$, $0 = x_T$, and $n = y_T$. Next, we argue that all k-error occurrences of P in T start around the positions in T where exact occurrences of Q start, that is, in the intervals $[\,jq - k\,..\,jq + k\,]$ for $j \in \mathbb{Z}$.[4] To that end, observe that, for any alignment of cost at most k mapping P to a fragment $T[\,v\,..\,w\,)$ of T, at least one of the copies of Q that comprise P must match exactly; otherwise the edit distance would be much larger than k. Suppose that the i-th copy of Q, that is, $P[\,iq\,..\,(i+1)q\,)$, is matched exactly. As Q is primitive and hence does not match any of its non-trivial rotations, $P[\,iq\,..\,(i+1)q\,)$ must be matched with a fragment $T[\,i'q\,..\,(i'+1)q\,)$ of T. As the entire alignment makes at most k insertions and deletions, this implies that $v \in [\,(i'-i)q - k\,..\,(i'-i)q + k\,]$.

In general, the strings P and T are only *almost* periodic—in particular, the edits in $\mathcal{A}_T$ and $\mathcal{A}_P$ may widen the intervals of potential starting positions, albeit only by a $d_T + d_P \leq 4d$ additive term. Since $k < d$, we have

$$\mathrm{Occ}^E_k(P,T) \subseteq \bigcup_{j\in\mathbb{Z}} [\,jq - 5d\,..\,jq + 5d\,].$$

Hence, for each j, we define a fragment $R_j = T[\,r_j\,..\,r'_j\,)$ of T that is of length $m + \mathcal{O}(d)$ and, in the considered instance, is *responsible* for capturing k-error occurrences of P in T that start in $[\,jq - 5d\,..\,jq + 5d\,]$; specifically, we have

$$r_j + \mathrm{Occ}^E_k(P,R_j) \supseteq \mathrm{Occ}^E_k(P,T) \cap [\,jq - 5d\,..\,jq + 5d\,].$$

In addition, we identify a set $J \subseteq \mathbb{Z}$ of size $\mathcal{O}(m/q)$ such that

$$\mathrm{Occ}^E_k(P,T) = \bigcup_{j\in J} \big(r_j + \mathrm{Occ}^E_k(P,R_j)\big).$$

Our goal is to compute occurrences of P in each R_j separately. To that end, observe that both P and all R_js essentially decompose into (possibly slightly "edited") copies of Q. In particular, for $j, j+1 \in J$, we can obtain R_{j+1} from R_j by replacing $\mathcal{O}(d)$ such "edited" copies. As a first step toward capturing the notions of P and R_j being decomposed into *pieces* and our algorithm *replacing* pieces of R_j, we define Δ-*puzzles*; consult Figure 2 for a visualization of an example of a Δ-puzzle.

Definition III.4. *For an integer $\Delta \in \mathbb{Z}_{\geq 0}$, 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 all* $i \in [\,1\,..\,z\,)$.

The value $\mathsf{val}_\Delta(S_1, \ldots, S_z)$ *of the puzzle is*

$$S_1 \cdot S_2[\,\Delta\,..\,|S_2|\,) \cdot S_3[\,\Delta\,..\,|S_3|\,) \cdots S_z[\,\Delta\,..\,|S_z|\,).$$

In the full version, we define *pieces* $P_1, \ldots, P_z$ and $T_{j,1}, \ldots, T_{j,z}$ (for each $j \in J$) that form Δ-puzzles with values P and R_j, respectively, where $\Delta := 6(d_P + d_T + k)$. Let us intuitively describe these pieces.[5] First, let us partition both P and T into *tiles*, that is, maximal fragments that are aligned to different copies of Q by $\mathcal{A}_P$ and $\mathcal{A}_T$, respectively. Observe that all but $\mathcal{O}(d)$ tiles are exact copies of Q. Further, the endpoints R_j are $\mathcal{O}(d)$ positions apart from tile boundaries. We then obtain an induced partition for R_j by extending the first and last tiles that it fully contains these $\mathcal{O}(d)$ positions. Finally, we extend all tiles of the partition of P and the induced partition of R_j, other than the trailing ones, by Δ characters to the right. Consult Figure 3 for a visualization of this setting.

We call pieces $P_2, \ldots, P_{z-1}$ and $T_{j,2}, \ldots, T_{j,z-1}$ (for $j \in J$) *internal*. Observe that, for each i, all internal pieces of the form $T_{j,i'}$ with $j + i' = i$ coincide; that is, overlapping parts of different R_js share their internal pieces. Hence, for each $i \in (\,\min J + 1\,..\,\max J + z\,)$, we define $T_i := T_{j,i'}$ for any $j \in J$ and $i' \in (\,1\,..\,z\,)$ with $j + i' = i$. This is an essential property for our approach to work: when moving from R_j to R_{j+1}, we only need to shift the pieces T_i, and not to recompute them altogether.

Now, suppose that we can efficiently maintain a pair of Δ-puzzles so that we can at any time efficiently query for the k-error occurrences of the value of the first puzzle in the value of the second one. Then, as a warm-up solution, we can initialize the two puzzles as $P_1, \ldots, P_z$ and $T_{\min J,1}, \ldots, T_{\min J,z}$ and then replace pieces of the second puzzle as necessary in order

[4] Under our earlier assumption that $q \gg k$, this claim indeed allows for filtering out some positions where no occurrence may start as we have $jq + k \ll (j+1)q - k$ in that case.

[5] This description provides an oversimplified definition of pieces. In particular, as defined in the full version, P_1 covers at least 2 tiles whereas P_z covers 17 tiles. This is due to complications arising without the assumption $q \gg k$.

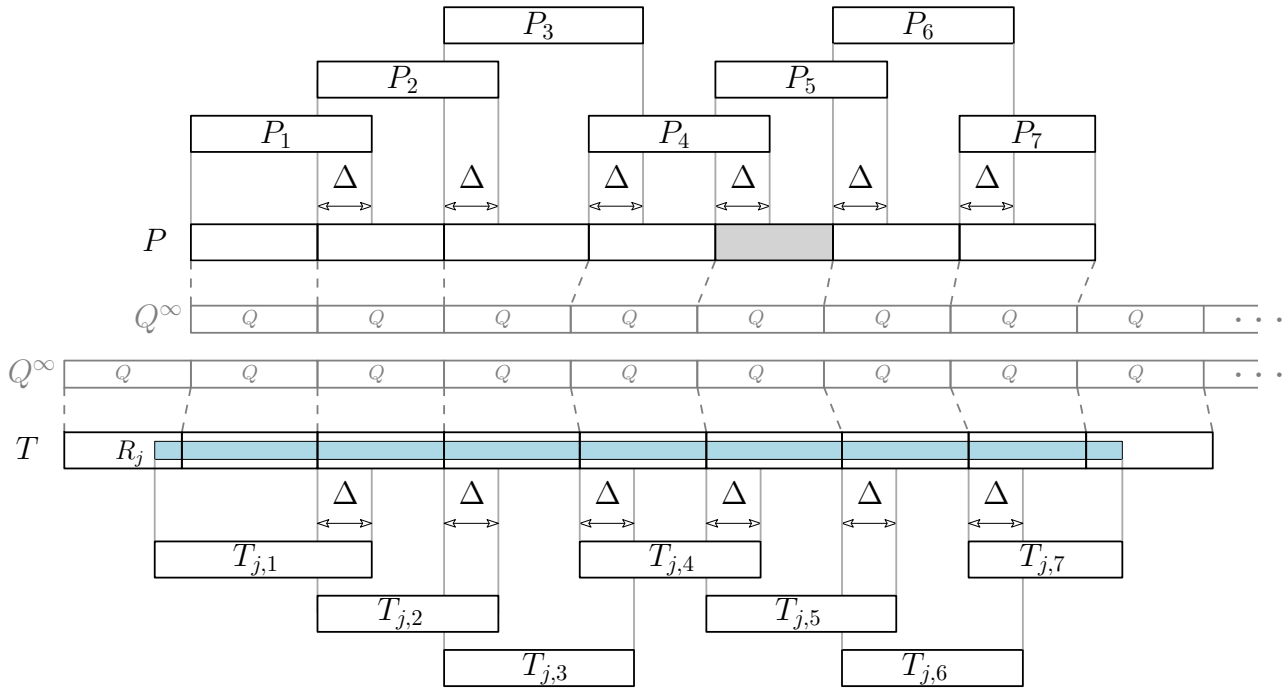

Fig. 3. Both P and T are partitioned into tiles. Specifically, dashed lines indicate the copy of Q to which a tile of P (or T) is aligned by $\mathcal{A}_P$ (or $\mathcal{A}_T$). For example, $\mathcal{A}_P$ aligns the shaded tile of P with the fifth copy of Q. The fragment R_j starts $\mathcal{O}(d)$ positions prior to the start of the second tile of T and ends $\mathcal{O}(d)$ positions after the end of the eighth tile. The pieces $P_1, \ldots, P_7$ and $T_{j,1}, \ldots, T_{j,7}$ form Δ-puzzles with values P and R_j, respectively.

to iterate over puzzles $T_{j,1}, \ldots, T_{j,z}$ for all $j \in J$. In fact, our final algorithm iterates over carefully trimmed versions of such puzzles, where we omit *plain* pieces that do not contribute to the solution set in an interesting manner. Formally, we capture the problem of maintaining such a pair of puzzles with the DYNAMICPUZZLEMATCHING problem.

DYNAMICPUZZLEMATCHING$(k, \Delta, \mathcal{S}_\beta, \mathcal{S}_\mu, \mathcal{S}_\varphi)$
Input: Positive integers k and Δ, as well as families $\mathcal{S}_\beta$, $\mathcal{S}_\mu$, and $\mathcal{S}_\varphi$ of *leading*, *internal*, and *trailing* pieces, respectively.
Maintained object: A *DPM-sequence* $\mathcal{I} = (U_1, V_1)(U_2, V_2)\cdots(U_z, V_z)$ of ordered pairs of strings, that additionally satisfies the following two conditions:
(a) $U_1, V_1 \in \mathcal{S}_\beta$, $U_z, V_z \in \mathcal{S}_\varphi$, and, for all $i \in (1\,..\,z)$, $U_i, V_i \in \mathcal{S}_\mu$,
(b) The torsion $\mathrm{tor}(\mathcal{I}) := \sum_{i=1}^{z} \big||U_i| - |V_i|\big|$ satisfies $\mathrm{tor}(\mathcal{I}) \le \Delta/2 - k$.
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').

It is assumed that $\mathcal{I}$ satisfies conditions (a) and (b) at initialization time and after each update.
Query (`DPM-Query`)**:** Under a promise that $U_1, \ldots, U_z$ and $V_1, \ldots, V_z$ are Δ-puzzles, return

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

We move on to our main result for the DYNAMICPUZZLEMATCHING problem. For a precise statement, we need to be able to quantify the complexity of the input families of strings; formally we define the *median edit distance* of a family $\mathcal{S}$ of strings over an alphabet Σ as $\delta_E(\mathcal{S}) := \min_{\hat{S} \in \Sigma^*} \sum_{S \in \mathcal{S}} \delta_E(S, \hat{S})$. Now, our result reads as follows.

Theorem III.5 ($\diamondsuit$)**.** *There is a data structure for* DYNAMICPUZZLEMATCHING$(k, \Delta, \mathcal{S}_\beta, \mathcal{S}_\mu, \mathcal{S}_\varphi)$ *supporting* $\mathcal{O}(\Delta \log z \log \Delta)$*-time updates and queries,* $\mathcal{O}(\Delta z \log \Delta)$*-time initialization, and* $\mathcal{O}((d^3 + \Delta^2 d)\log^2(d + \Delta))$*-time preprocessing, where* $d = \delta_E(\mathcal{S}_\beta) + \delta_E(\mathcal{S}_\mu) + \delta_E(\mathcal{S}_\varphi)$.[6]

Let us defer a detailed discussion of the proof of Theorem III.5 to the end of this overview. Here, we discuss its application to the ALIGNEDPERIODICMATCHES problem with the following string families.

$$\begin{aligned}
\mathcal{S}_\beta &:= \{P_1\} \cup \{T_{j,1} : j \in J\},\\
\mathcal{S}_\mu &:= \{P_i : i \in (1\,..\,z)\}\\
&\qquad \cup \{T_i : i \in (\min J + 1\,..\,\max J + z)\},\\
\mathcal{S}_\varphi &:= \{P_z\} \cup \{T_{j,z} : j \in J\}.
\end{aligned}$$

Next, we define multisets $\mathsf{Special}(P)$, $\mathsf{Special}(T)$, and $\mathsf{Special}_{\beta\varphi}(T)$ of *special pieces*—in this overview, we focus

[6]Recall that z is the length of the DPM-sequence that we maintain in the data structure.

on the two former multisets. In our fixed instance, we have

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

As mentioned earlier, there are only very few special pieces—crucially, we show the following lemma.

Lemma III.6 ($\diamondsuit$)**.** *The median edit distance of each of the families $\mathcal{S}_\beta$, $\mathcal{S}_\mu$, and $\mathcal{S}_\varphi$ is bounded by $\mathcal{O}(d)$.*

Further, each of the multisets $\mathsf{Special}(P)$, $\mathsf{Special}(T)$, and $\mathsf{Special}_{\beta\varphi}(T)$ is of size $\mathcal{O}(d)$ and can be computed in $\mathcal{O}(d)$ time in the PILLAR *model.*

For $j \in J$, let $\mathcal{I}_j$ denote the DPM-sequence $(P_1, T_{j,1})$ $(P_2, T_{j,2}) \cdots (P_z, T_{j,z})$. Now for $j \in J \setminus \max J$, each of $\mathcal{I}_j$ and $\mathcal{I}_{j+1}$ contains $\mathcal{O}(d)$ *special* pairs, that is, pairs that contain special pieces. Hence, we can naively iterate over all $\mathcal{I}_j$s in an instance of the DYNAMICPUZZLEMATCHING problem using $\mathcal{O}(d \cdot |J|)$ updates; in the considered instance we have $d \cdot |J| = \mathcal{O}(d \cdot m/q)$. Consult the full version for details of this reduction in the general case. As a preliminary improvement, also in the full version, we show how to reduce the number of updates to $\mathcal{O}(d^3)$. Let us give a brief sketch of this reduction.

We call a pair of pieces $(P_i, T_{j,i})$ *plain* if $i \in (1\,..\,z)$ and neither P_i nor $T_{j,i}$ is special; in the restricted case that we are considering here, the second condition is equivalent to $P_i = T_{j,i} = Q^{\infty}[0\,..\,q+\Delta)$. For $j \in J$, let $\mathcal{I}'_j$ denote the DPM-sequence obtained from $\mathcal{I}_j$ by trimming each run (that is, maximal contiguous subsequence) of plain pairs in $\mathcal{I}_j$ to length $k+1$ by deleting excess pairs.

The main idea is that we do not gain or lose any k-error occurrences by trimming the DPM-sequences, that is, we have $\mathrm{Occ}^E_k(\mathcal{I}_j) = \mathrm{Occ}^E_k(\mathcal{I}'_j)$. One direction is easy: removing the same substring from two strings P and T may only decrease the edit distance between P and T; this naturally translates to DPM-sequences. For the other direction, observe that if a DPM-sequence contains a run of at least $k+1$ plain pairs, then any cost-k alignment between the corresponding strings has to perfectly match at least one copy of Q in such a run—we can hence duplicate said copy by adding more plain pairs in the DPM-sequence without increasing the cost of the alignment. Induction then yields the claim.

With the aim of obtaining an $\mathcal{O}(d^3)$ upper bound on the number of required updates for iterating over the $\mathcal{I}'_j$s, let us think of the process of shifting P along T. For each j, each run of plain pairs in $\mathcal{I}_j$ can be attributed to a run of plain pieces in $P_2, \ldots, P_{z-1}$ that overlap a run of plain pieces in $T_{j,2} \ldots T_{j,z-1}$. As we shift P, in the most general case, the length of the overlap first increases, then it remains static, and, finally, it decreases. Overall, as j gets incremented, a run of plain pairs that is attributed to a specific pair of runs of plain pieces may change length $\omega(d)$ times. However, after trimming the lengths of all runs of plain pairs to $k+1 = \mathcal{O}(d)$, the length of such a run gets incremented/decremented $\mathcal{O}(d)$ times. As we have $\mathcal{O}(d)$ special pieces in each of P and T, we have $\mathcal{O}(d^2)$ pairs of runs of plain pieces, and hence we get the desired $\mathcal{O}(d^3)$ upper bound, as we can bound the number of updates other than insertions/deletions of plain pairs by $\mathcal{O}(d^2)$.

Note that we cannot always iterate explicitly over all $\mathcal{I}'_j$s as this would require $\Omega(m/q)$ calls to `DPM-Query`. We circumvent this problem by observing that if we have $\mathcal{I}'_{j-1} = \mathcal{I}'_j$ (for some $j \in J \setminus \min J$), then $r_j + \mathrm{Occ}^E_k(P, R_j) = q + r_{j-1} + \mathrm{Occ}^E_k(P, R_{j-1})$. Consequently, for any maximal interval $[j_1\,..\,j_2] \subseteq J$ where $\mathcal{I}'_{j_1} = \cdots = \mathcal{I}'_{j_2}$, we only process $\mathcal{I}'_{j_1}$; then, for each position $u \in \mathrm{Occ}^E_k(P, R_{j_1})$, we report an arithmetic progression $\{r_{j_1} + u + iq : i \in [0\,..\,j_2 - j_1]\}$ of k-error occurrences of P in T. On a high level, we are offloading the computation of $\bigcup_{j=j_1}^{j_2} \left(r_j + \mathrm{Occ}^E_k(P, R_j)\right)$ to the computation of $\mathrm{Occ}^E_k(\mathcal{I}'_{j_1})$.

A Faster Solution: To obtain a faster solution for the ALIGNEDPERIODICMATCHES problem, we intend to trim runs of plain pairs even further, to a length of roughly $\widetilde{\mathcal{O}}(\sqrt{d})$. Now, naively processing the obtained DPM-sequences, we may obtain "false-positive" occurrences, but—as we prove in the full version—not too many. In particular, we can extend existing tools to filter out such "false-positive" occurrences.

For a slightly more detailed overview, for any two positions $v < w$ of T, let us write $Q^{\infty}[\rho(v)\,..\,\rho(w)\,)$ for the fragment of Q^{∞} that $\mathcal{A}_T$ aligns with $T[v\,..\,w)$. Suppose that we have

$$\begin{aligned}\delta_E(P, {}^*Q^*) &= \delta_E(P, Q^{\infty}[\rho(v)\,..\,\rho(w)\,)) \\ &\leq \delta_E(T[v\,..\,w), {}^*Q^*) \\ &= \delta_E(T[v\,..\,w), Q^{\infty}[\rho(v)\,..\,\rho(w)\,)).\end{aligned}$$

Then, the triangle inequality yields

$$\begin{aligned}\Lambda &\coloneqq \delta_E(P, {}^*Q^*) + \delta_E(T[v\,..\,w), {}^*Q^*) \\ &\geq \delta_E(P, T[v\,..\,w)) \\ &\geq \delta_E(T[v\,..\,w), {}^*Q^*) - \delta_E(P, {}^*Q^*).\end{aligned}$$

We see that, intuitively, the best case is when all the errors of P with $Q^{\infty}[\rho(v)\,..\,\rho(w)\,)$ cancel out with errors of $T[v\,..\,w)$ with $Q^{\infty}[\rho(v)\,..\,\rho(w)\,)$. Now, roughly speaking, for each position v of T, we quantify the "potential savings" that an alignment $P \rightsquigarrow T[v\,..\,w)$ of cost at most k may yield compared to $\min_x \delta_E(P, Q^{\infty}[\rho(v)\,..\,x)) + \min_y \delta_E(T[v\,..\,w), Q^{\infty}[\rho(v)\,..\,y))$. To this end, we use the notion of *locked fragments* from [8] to mark each position of the text with a number of marks proportional to said "potential savings". (A similar notion was used in [10].) Based on a threshold $\eta \coloneqq \widetilde{\Theta}(\sqrt{d})$ on the number of marks (and a few technical conditions), we then classify each position as either *heavy* or *light*. Details on locked fragments and our marking scheme can be found in the full version.

Now, for our solution for ALIGNEDPERIODICMATCHES, we first show that the set of heavy positions intersects $\widetilde{\mathcal{O}}(\sqrt{d})$ ranges, each of size $\mathcal{O}(d)$, where a k-error occurrences of P may start (recall that $\mathrm{Occ}^E_k \subseteq \bigcup_{j \in \mathbb{Z}} [jq - 5d\,..\,jq + 5d]$). We can then compute the intersection of $\mathrm{Occ}^E_k(P, T)$ with heavy positions efficiently, that is, in $\widetilde{\mathcal{O}}(d^{3.5})$ time, using known tools.

Having taken care of the heavy positions, we can return to DYNAMICPUZZLEMATCHING for the light positions. To that end, consider again $T[v\,..\,w)$, supposing that v is a light position of T. We then have that $\Lambda \geq \delta_E(P, T[v\,..\,w)) \geq \Lambda - \eta$. Now, the optimal alignment $\mathcal{A}$ from P to $T[v\,..\,w)$ has to make $\Lambda - \eta$ edit operations just to align the locked fragments of the text and the pattern. This means that the number of edit operations that $\mathcal{A}$ makes in aligning portions of P disjoint from the locked fragments of P to portions of T disjoint from the locked fragments of T is at most η.

Now, we define a set $\mathsf{Red}(P) \supseteq \mathsf{Special}(P)$ that additionally contains all pieces of P that overlap some locked fragment of P; we similarly define a set $\mathsf{Red}(T)$ of pieces of T. Importantly, both $\mathsf{Red}(P)$ and $\mathsf{Red}(T)$ are of size $\mathcal{O}(d)$. Redefining plain pairs to be those that contain no red piece, we show that we can trim each run of plain pairs to have a length of $\mathcal{O}(\eta) = \widetilde{\Theta}(\sqrt{d})$. This allows us to reduce our problem to an instance of the DYNAMICPUZZLEMATCHING problem with $\widetilde{\Theta}(d^{2.5})$ updates in total; as before, we can essentially charge all but $\mathcal{O}(d^2)$ updates to $\mathcal{O}(d^2)$ pairs of runs of plain pairs, so that each such pair gets charged with $\widetilde{\Theta}(\sqrt{d})$ updates.

C. A Solution for the DynamicPuzzleMatching Problem

For our solution to the DYNAMICPUZZLEMATCHING problem, we rely on a framework of Tiskin [25], [26], [27] (which we recall and extend in the full version). A key observation behind this framework is that *semi-local* alignments between strings U and V can be represented as paths between boundary vertices of a certain *alignment graph*: a grid on vertices $[0\,..\,|V|] \times [0\,..\,|U|]$, augmented with diagonal edges. All horizontal and vertical edges have weight 1 (they represent insertions and deletions), whereas each diagonal edge $(u, v) \leftrightarrow (u+1, v+1)$ has weight 0 (for a match) or 1 (for a substitution). Then, $\delta_E(V[v\,..\,w), U)$ corresponds to the distance from $(v, 0)$ to $(w, |U|)$. As observed in [27], even though there are quadratically many such distances, they can be encoded in linear space using a certain *permutation matrix* that we denote by $P_{V,U}$. Moreover, we can *stitch* alignment graphs by computing a certain *seaweed product* of permutation matrices. For example, $P_{V,UU'}$ can be expressed as the seaweed product of $P_{V,U}$ and $P_{V,U'}$ (shifted appropriately so that the characters of U' are indexed from $|U|$ rather than from 0). Tiskin [26] provided an $\mathcal{O}(n \log n)$-time algorithm for computing the seaweed product of two $n \times n$ permutation matrices, but we cannot hope to compute $P_{V,U}$ in truly subquadratic time because it encodes $\delta_E(U, V)$.

In our setting, though, the strings U and V are of similar length (that is, $\big||V| - |U|\big| \leq \mathrm{tor}(\mathcal{I}) \leq \Delta/2 - k$) and we only care about alignments of cost at most k. The underlying paths corresponding to such alignments are fully contained within a narrow *diagonal band* of the alignment graph: all of their vertices (u, v) satisfy $u - v \in I := [-k\,..\,|V| - |U| + k]$ (in short, they belong to band I of the alignment graph); see Figure 4 for an illustration. In order to capture this scenario, we restrict the alignment graph to band I, which corresponds to zeroing out the costs of all diagonal edges outside band I.

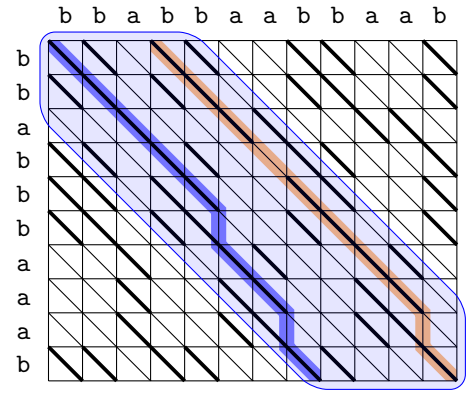

Fig. 4. The alignment graph for $U = \texttt{bbabbbaaab}$ and $V = \texttt{bbabbaabbaab}$. Thin edges have cost 1 whereas thick edges have cost 0. The blue and orange path represent cost-2 alignments $U \rightsquigarrow V[0\,..\,8)$ and $U \rightsquigarrow V[3\,..\,12)$, respectively. The diagonal band $I = [-2\,..\,4] = [-k\,..\,|V| - |U| + k]$ corresponding to $k = 2$ is shaded in blue.

We prove that the permutation matrix $P_{U,V}|_I$ of the restricted graph can be encoded in $\mathcal{O}(|I|)$ space and computed in $\widetilde{\mathcal{O}}(|I|^2)$ time (in the PILLAR model). Moreover, we show that $P_{U,V}|_I$ can be expressed solely in terms of $P_{U,V}$, which leads to a new operation of *restricting* a permutation matrix P to a given interval I. We write $P|_I$ for the result of said operation and we present a linear-time algorithm that computes $P|_I$ directly from P and I.

Let us now explain how these techniques are helpful in solving the DYNAMICPUZZLEMATCHING problem. Our high-level idea is to express $P_{V,U}|_I$ as the seaweed product of z smaller permutation matrices $P_1, \ldots, P_z$, with P_i depending only on the i-th pair (U_i, V_i). For a first attempt, we could use P_{V_i,U_i}, but the corresponding parts of the alignment overlap and thus cannot be stitched easily. Thus, we trim each piece U_i to U'_i so that $U = \mathsf{val}_\Delta(U_1, \ldots, U_z) = U'_1 \cdots U'_z$. Now, the seaweed product of matrices P_{V_i,U'_i} (shifted appropriately), restricted *a posteriori* to interval I, yields $P_{V,U}|_I$. However, the individual matrices P_{V_i,U'_i} are still too large, so we need to restrict them *a priori* as well. Thus, we actually use $P_{V_i,U'_i}|_{I_i}$, for appropriate intervals I_i of size at most Δ; see Figure 5 for an illustration. We build a balanced binary tree on top of the permutation matrices $P_{V_i,U'_i}|_{I_i}$ in order to maintain their seaweed product (so that every update requires recomputing $\mathcal{O}(\log z)$ partial products). For each query, we retrieve $P_{V,U}|_I$ and apply the SMAWK algorithm [2] in order to check, for every $v \in [0\,..\,|V| - |U| + k]$, whether $\delta_E(U, V[v\,..\,w)) \leq k$ holds for some $w \in [|U| - k\,..\,|W|]$ in $\widetilde{\mathcal{O}}(\Delta)$ time in total.

The remaining challenge is to build the matrices $P_{V_i,U'_i}|_{I_i}$. For this, we exploit the small median edit distance of the families $\mathcal{S}_\beta, \mathcal{S}_\mu, \mathcal{S}_\varphi$ to show that all such matrices can be precomputed in $\widetilde{\mathcal{O}}(d^3 + \Delta^2 d)$ time. If the puzzle pieces were of size $\mathcal{O}(d + \Delta)$, we could simply use an algorithm of Charalampopoulos, Kociumaka, and Mozes [7] that maintains $P_{X,Y}$ subject to edits of X, Y. In general, though, we decompose each piece into $\mathcal{O}(d)$ parts: perfect parts, which

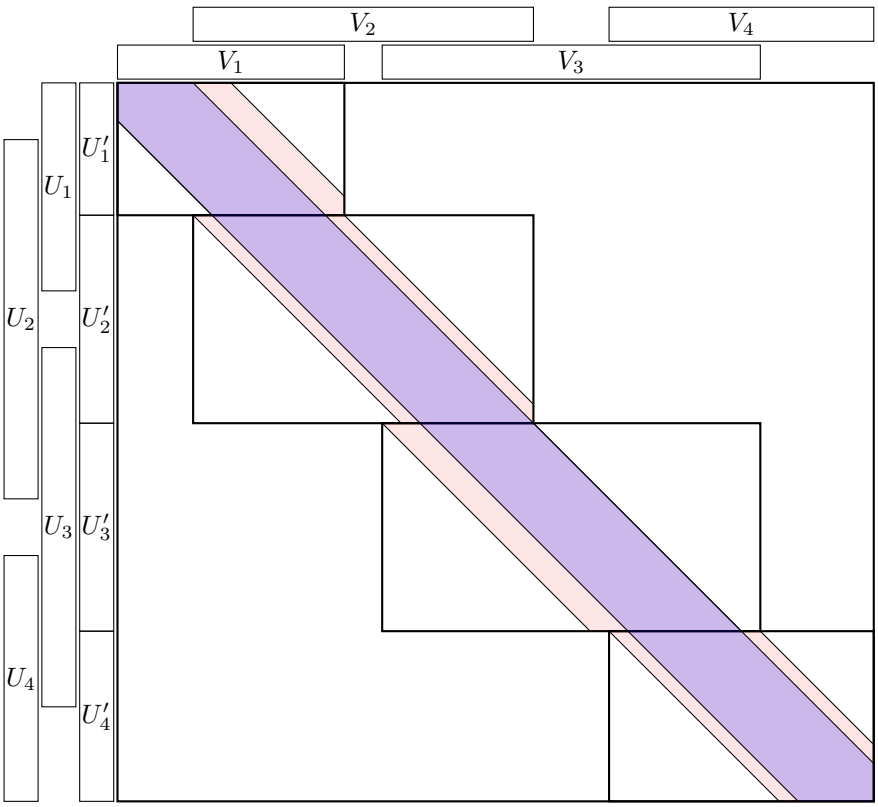

Fig. 5. A schematic illustration explaining why $P_{V,U}|_I$, which corresponds to the purple band, can be obtained by the seaweed products of matrices $P_{V_i,U'_i}|_{I_i}$, which correspond to the pink bands within the rectangles representing the alignment graphs of U'_i and V_i (subgraphs of the alignment graph of U and V).

can be arbitrarily long but are kept intact among all the puzzle pieces, and imperfect parts, which can contain edits but are of size $\mathcal{O}(\Delta)$. For each perfect part, we compute a single restricted permutation matrix in $\widetilde{\mathcal{O}}(\Delta^2)$ time. For imperfect parts, we use the dynamic algorithm of [7]. Finally, the restricted permutation matrix of a pair of pieces is obtained by stitching the matrices for pairs of parts similarly to how we obtain $P_{V,U}|_I$ from $P_{V_i,U'_i}|_{I_i}$s.

IV. Open Problems

The most important and obvious open problem is to close the gap between upper and lower bounds for the pattern matching with edits problem; as is depicted in Figure 1. In the quest for faster algorithms, one could try to relax the problem in scope, for instance, by considering its (easier) decision version where we only need to check whether $\mathrm{Occ}_k^E(P,T)$ is empty, or by allowing for some approximation by also reporting an arbitrary subset of the positions in $\mathrm{Occ}^E_{(1+\varepsilon)k}(P,T) \setminus \mathrm{Occ}_k^E(P,T)$ for a small $\varepsilon > 0$.

Another research direction could be to devise an algorithm with an analogous running time as the one presented here that reports all fragments of T that are at edit distance at most k from P (in appropriate batches); recall that $\mathrm{Occ}_k^E(P,T)$ is only the set of the starting positions of such fragments. While we think that the $\mathcal{O}(k^4 \cdot n/m)$-time `PILLAR` algorithm of [8] can be generalized to report all such fragments, our $\widetilde{\mathcal{O}}(k^{3.5} \cdot n/m)$-time solution *does not* seem to generalize. We remark that Landau, Myers, and Schmidt [16] showed that all the sought fragments can be listed in $\mathcal{O}(nk)$ time; for this, they adapted the algorithm of [19].

References

[1] K. R. Abrahamson, "Generalized string matching," *SIAM Journal on Computing*, vol. 16, no. 6, pp. 1039–1051, 1987. [Online]. Available: https://doi.org/10.1137/0216067

[2] A. Aggarwal, M. M. Klawe, S. Moran, P. W. Shor, and R. E. Wilber, "Geometric applications of a matrix-searching algorithm," *Algorithmica*, vol. 2, pp. 195–208, 1987. [Online]. Available: https://doi.org/10.1007/BF01840359

[3] A. Amir, M. Lewenstein, and E. Porat, "Faster algorithms for string matching with k mismatches," *Journal of Algorithms*, vol. 50, no. 2, pp. 257–275, 2004. [Online]. Available: https://doi.org/10.1016/S0196-6774(03)00097-X

[4] A. Backurs and P. Indyk, "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)," *SIAM Journal on Computing*, vol. 47, no. 3, pp. 1087–1097, 2018. [Online]. Available: https://doi.org/10.1137/15M1053128

[5] D. Chakraborty, D. Das, and M. Koucký, "Approximate online pattern matching in sublinear time," in *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019*, ser. LIPIcs, vol. 150. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019, pp. 10:1–10:15. [Online]. Available: https://doi.org/10.4230/LIPIcs.FSTTCS.2019.10

[6] T. M. Chan, S. Golan, T. Kociumaka, T. Kopelowitz, and E. Porat, "Approximating text-to-pattern Hamming distances," in *52nd Annual ACM Symposium on Theory of Computing, STOC 2020*. ACM, 2020, pp. 643–656. [Online]. Available: https://doi.org/10.1145/3357713.3384266

[7] P. Charalampopoulos, T. Kociumaka, and S. Mozes, "Dynamic string alignment," in *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, ser. LIPIcs, vol. 161. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, pp. 9:1–9:13. [Online]. Available: https://doi.org/10.4230/LIPIcs.CPM.2020.9

[8] P. Charalampopoulos, T. Kociumaka, and P. Wellnitz, "Faster approximate pattern matching: A unified approach," in *61st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, 2020, pp. 978–989, full version: https://arxiv.org/abs/2004.08350v2. [Online]. Available: https://doi.org/10.1109/FOCS46700.2020.00095

[9] R. Clifford, A. Fontaine, E. Porat, B. Sach, and T. Starikovskaya, "The k-mismatch problem revisited," in *27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*. SIAM, 2016, pp. 2039–2052. [Online]. Available: https://doi.org/10.1137/1.9781611974331.ch142

[10] R. Cole and R. Hariharan, "Approximate String Matching: A Simpler Faster Algorithm," *SIAM Journal on Computing*, vol. 31, no. 6, pp. 1761–1782, 2002. [Online]. Available: https://doi.org/10.1137/S0097539700370527

[11] Z. Galil and R. Giancarlo, "Improved string matching with k mismatches," *SIGACT News*, vol. 17, no. 4, pp. 52–54, 1986. [Online]. Available: https://doi.org/10.1145/8307.8309

[12] P. Gawrychowski and P. Uznański, "Towards unified approximate pattern matching for Hamming and L_1 distance," in *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, ser. LIPIcs, vol. 107. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, pp. 62:1–62:13. [Online]. Available: https://doi.org/10.4230/LIPIcs.ICALP.2018.62

[13] T. Kociumaka, E. Porat, and T. Starikovskaya, "Small space and streaming pattern matching with k edits," in *62nd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2021*. IEEE, 2021, pp. 885–896. [Online]. Available: https://doi.org/10.1109/FOCS52979.2021.00090

[14] T. Kociumaka, J. Radoszewski, W. Rytter, and T. Walen, "Internal pattern matching queries in a text and applications," in *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*. SIAM, 2015, pp. 532–551. [Online]. Available: https://doi.org/10.1137/1.9781611973730.36

[15] S. Kosaraju, "Efficient string matching," 1987, unpublished manuscript.

[16] G. M. Landau, E. W. Myers, and J. P. Schmidt, "Incremental string comparison," *SIAM Journal on Computing*, vol. 27, no. 2, pp. 557–582, 1998. [Online]. Available: https://doi.org/10.1137/S0097539794264810

[17] G. M. Landau and U. Vishkin, "Efficient string matching with k mismatches," *Theoretical Computer Science*, vol. 43, pp. 239–249, 1986. [Online]. Available: https://doi.org/10.1016/0304-3975(86)90178-7

[18] ——, "Fast string matching with k differences," *Journal of Computer and System Sciences*, vol. 37, no. 1, pp. 63–78, 1988. [Online]. Available: https://doi.org/10.1016/0022-0000(88)90045-1

[19] ——, "Fast parallel and serial approximate string matching," *Journal of Algorithms*, vol. 10, no. 2, pp. 157–169, 1989. [Online]. Available: https://doi.org/10.1016/0196-6774(89)90010-2

[20] W. J. Masek and M. Paterson, "A faster algorithm computing string edit distances," *Journal of Computer and System Sciences*, vol. 20, no. 1, pp. 18–31, 1980. [Online]. Available: https://doi.org/10.1016/0022-0000(80)90002-1

[21] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, vol. 33, no. 1, pp. 31–88, 2001. [Online]. Available: https://doi.org/10.1145/375360.375365

[22] S. C. Sahinalp and U. 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*. IEEE, 1996, pp. 320–328. [Online]. Available: https://doi.org/10.1109/SFCS.1996.548491

[23] P. H. Sellers, "The theory and computation of evolutionary distances: Pattern recognition," *Journal of Algorithms*, vol. 1, no. 4, pp. 359–373, 1980. [Online]. Available: https://doi.org/10.1016/0196-6774(80)90016-4

[24] T. Starikovskaya, "Communication and streaming complexity of approximate pattern matching," in *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, ser. LIPIcs, vol. 78. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017, pp. 13:1–13:11. [Online]. Available: https://doi.org/10.4230/LIPIcs.CPM.2017.13

[25] A. Tiskin, "Semi-local string comparison: algorithmic techniques and applications," 2007. [Online]. Available: https://arxiv.org/abs/0707.3619

[26] ——, "Fast distance multiplication of unit-Monge matrices," *Algorithmica*, vol. 71, no. 4, pp. 859–888, 2015. [Online]. Available: https://doi.org/10.1007/s00453-013-9830-z

[27] ——, "Semi-local string comparison: Algorithmic techniques and applications," *Mathematics in Computer Science*, vol. 1, no. 4, pp. 571–603, 2008. [Online]. Available: https://doi.org/10.1007/s11786-007-0033-3