The Tandem Duplication Distance Is NP-Hard

Manuel Lafond
Department of Computer Science, Universite de Sherbrooke, Sherbrooke, Quebec J1K 2R1, Canada
manuel.lafond@usherbrooke.ca

Binhai Zhu
Gianforte School of Computing, Montana State University, Bozeman, MT 59717, USA
bhz@montana.edu

Peng Zou
Gianforte School of Computing, Montana State University, Bozeman, MT 59717, USA
peng.zou@student.montana.edu

Abstract

In computational biology, tandem duplication is an important biological phenomenon which can occur either at the genome or at the DNA level. A tandem duplication takes a copy of a genome segment and inserts it right after the segment – this can be represented as the string operation $AXB \Rightarrow AXXB$. Tandem exon duplications have been found in many species such as human, fly or worm, and have been largely studied in computational biology.

The *Tandem Duplication* (TD) distance problem we investigate in this paper is defined as follows: given two strings S and T over the same alphabet, compute the smallest sequence of tandem duplications required to convert S to T. The natural question of whether the TD distance can be computed in polynomial time was posed in 2004 by Leupold et al. and had remained open, despite the fact that tandem duplications have received much attention ever since. In this paper, we prove that this problem is NP-hard, settling the 16-year old open problem. We further show that this hardness holds even if all characters of S are distinct. This is known as the *exemplar* TD distance, which is of special relevance in bioinformatics. One of the tools we develop for the reduction is a new problem called the *Cost-Effective Subgraph*, for which we obtain W[1]-hardness results that might be of independent interest. We finally show that computing the exemplar TD distance between S and T is fixed-parameter tractable. Our results open the door to many other questions, and we conclude with several open problems.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Tandem duplication, Text processing, Formal languages, Computational genomics, FPT algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.15

Related Version A full version of this paper is available at `https://arxiv.org/abs/1906.05266`.

Funding ML was supported by NSERC of Canada, and BZ was partially supported by NNSF of China under project 61628207.

1 Introduction

Tandem duplication is a biological process that creates consecutive copies of a segment of a genome during DNA replication. Representing genomes as strings, this event transforms a string AXB into another string $AXXB$. This process is known to occur either at small scale at the nucleotide level, or at large scale at the genome level [5, 6, 7, 23, 12]. For instance, it is known that the Huntington disease is associated with the duplication of 3 nucleotides `CAG` [13], whereas at genome level, tandem duplications are known to involve multiple genes during cancer progression [26]. Furthermore, gene duplication is believed to be the main driving force behind evolution, and the majority of duplications affecting organisms are believed to be of the tandem type (see e.g. [29]). As a result, around 3% of the human genome are formed of tandem repeats.

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 15; pp. 15:1–15:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

For these reasons, tandem duplications have received significant attention in the last decades, both in practice and theory. The combinatorial aspects of tandem duplications have been studied extensively by computational biologists [2, 14, 16, 22, 30], one question of interest being to reconstruct the evolution of a cluster of tandem repeats by duplications that could have given rise to the observed sequences. In parallel, various formal language communities [9, 31, 25] have investigated the expressive power of tandem duplications on strings.

From the latter perspective, a natural question arises: given a string S, what is the language that can be obtained starting from S and applying (any number of) tandem duplications, i.e. rules of the form $AXB \rightarrow AXXB$, where X can be any substring of S? This question was first asked in 1984 in the context of so-called *copying systems* [11]. Combined with results from [3], it was shown that this language is regular if S is on a binary alphabet, but not regular for larger alphabets. These results were rediscovered 15 years later in [9, 31]. In [25], it was shown that the membership, inclusion and regularity testing problems on the language defined by S can all be decided in linear time (still on binary alphabets). In [25, 24, 19], similar problems are also considered on non-binary alphabets, when the length $|X|$ of duplicated strings is bounded by a constant. More recently, Cho et al. [8] introduced a tandem duplication system where the *depth* of a character, i.e. the number of "generations" it took to generate it, is considered. In [18, 20], the authors study the *expressive power* of tandem duplications, a notion based on the subsequences that can be obtained from various types of copying mechanisms.

More directly related to our work, Alon et al. [1] recently investigated the minimum number of duplications required to transform a string S into another string T. We call this the *Tandem Duplication (TD) distance*. More specifically, the authors show that on binary strings, the maximum TD distance between a square-free string S and a string T of length n is $\Theta(n)$. They also mention the unsolved algorithmic problem of computing the TD distance between S and T. In fact, this question was posed earlier in [25] (pp. 306, Open Problem 3) by Leupold et al. and has remained open ever since. We settle this open problem in this paper for an unbounded alphabet. As will be seen, our technique is different from that used in [1], which only works for binary strings.

On the other hand, the TD distance is one of the many ways of comparing two genomes represented as strings in computational biology – other notable examples include breakpoint [17] and transpositions distances, the latter having recently been shown NP-hard in a celebrated paper of Bulteau et al. [4]. The TD distance has itself received special attention recently, owing to its role in cancer evolution [27].

Our results. In this paper, we solve the problem posed by Leupold et al. in 2004 and show that computing the TD distance from a string S to a string T is NP-hard. We show that this result holds even if S is *exemplar*, i.e. if each character of S is distinct. Exemplar strings are commonly studied in computational biology [28], since they represent genomes that existed prior to duplication events. We note that simply deciding if S can be transformed into T by a sequence of TDs still has unknown complexity. In our case, we show that the hardness of minimizing TDs holds on instances in which such a sequence is guaranteed to exist.

As demonstrated by the transpositions distance in [4], obtaining NP-hardness results for string distances can sometimes be an involving task. Our hardness reduction is also quite technical, and one of the tools we develop for it is a new problem we call the *Cost-Effective Subgraph*. In this problem, we are given a graph G with a cost c, and we must choose a subset X of $V(G)$. Each edge with both endpoints in X has a cost of $|X|$, every other edge costs c, and the goal is to find a subset X of minimum cost. We show that this problem is W[1]-hard for parameter $p + c$, where p is a parameter that asks if one can achieve a cost of at most $c|E(G)| - p$ (here $c|E(G)|$ is an upper bound on the cost[1]). The problem enforces optimizing the tradeoff between covering many edges versus having a large subset of high cost, which might be applicable to other problems. In our case it captures the main difficulty in computing TD distances. We then obtain some positive results by showing that if S is exemplar, then one can decide if S can be transformed into T using at most k duplications in time $2^{O(k^2)} + poly(n)$, where n is the length of T. The result is obtained through an exponential size kernel. All of our results concern strings with unbounded alphabet sizes. Finally, we conclude with several open problems that might be of interest to the theoretical computer science community.

This paper is organized as follows. In Section 2, we give basic definitions. In Section 3, we show that computing the TD distance is NP-hard through the Cost-Effective Subgraph problem. In Section 4, we show that computing the exemplar TD distance is FPT. In Section 5, we conclude the paper with several open problems.

2 Preliminary notions

We borrow the string terminology and notation from [15]. In particular, $[n]$ denotes the set of integers $\{1, 2, \ldots, n\}$. Unless stated otherwise, all the strings in the paper are on an alphabet denoted Σ. If S_1 and S_2 are two strings, we usually denote their concatenation by S_1S_2. For a string S, we write $\Sigma(S)$ for the subset of characters of Σ that have at least one occurrence in S. A string S is called *exemplar* if $|S| = |\Sigma(S)|$, i.e. each character present in S occurs only once. A *substring* of S is a contiguous sequence of characters within S. A *prefix* (resp. *suffix*) of S is a substring that occurs at the beginning (resp. end) of S, i.e. if $S = S_1S_2$ for some strings S_1 and S_2, then S_1 is a prefix of S and S_2 a suffix of S. A *subsequence* of S is a string that can be obtained by successively deleting characters from S.

A *tandem duplication* (TD) is an operation on a string S that copies a substring X of S and inserts the copy after the occurrence of X in S. In other words, a TD transforms $S = AXB$ into $AXXB$. Given another string T, we write $S \Rightarrow T$ if there exist strings A, B, X such that $S = AXB$ and $T = AXXB$. More generally, we write $S \Rightarrow_k T$ if there exist $S_1, \ldots, S_{k-1}$ such that $S \Rightarrow S_1 \Rightarrow \ldots \Rightarrow S_{k-1} \Rightarrow T$. We also write $S \Rightarrow_* T$ if there exists some k such that $S \Rightarrow_k T$.

▶ **Definition 1.** *The TD distance $dist_{TD}(S, T)$ between two strings S and T is the minimum value of k satisfying $S \Rightarrow_k T$. If $S \Rightarrow_* T$ does not hold, then $dist_{TD}(S, T) = \infty$.*

We use the term *distance* here to refer to the number of TD operations from a string S to another string T, but one may note that TD is not a metric in the formal sense. In particular, $dist_{TD}$ is not symmetric since duplications can only increase the length of a string.

A *square string* is a string of the form XX, i.e. a concatenation of two identical substrings. Given a string S, a *contraction* is the reverse of a tandem duplication. That is, it takes a square string XX contained in S and deletes one of the two copies of X. We write $T \rightarrowtail S$ if there exist strings A, B, X such that $T = AXXB$ and $S = AXB$. We also define $T \rightarrowtail_k S$ and $T \rightarrowtail_* S$ for contractions analogously as for TDs. Observe that by the symmetry of

[1] In other words, if we were to state the maximization version of the Cost-Effective Subgraph problem, p would be the value to maximize. The minimization version, however, is more convenient to use for our needs.

duplications and contractions, $T \rightarrowtail_k S$ if and only if $S \Rightarrow_k T$ and $T \rightarrowtail_* S$ if and only if $S \Rightarrow_* T$. When there is no possible confusion, we will sometimes write $T \rightarrowtail S$ instead $T \rightarrowtail_* S$.

We have the following problem.

The Tandem Duplication (TD) problem:
Input: Two strings S and T over the same alphabet Σ and an integer k.
Question: Is $dist_{TD}(S,T) \leq k$?

In the Exemplar-TD variant of this problem, S is required to be exemplar. In either variant, we may call S the *source string* and T the *target string*. We will often use the fact that S and T form a YES instance if and only if T can be transformed into S by a sequence of at most k contractions. See Fig.1 for a simple example.

$$\begin{array}{rl} \mathit{Sequence} & \mathit{Operations} \\ \mathit{Sequence} \quad T = \langle a, c, \underline{g, g}, a, c, g\rangle & \mathit{contraction\ on}\ \langle g, g\rangle \\ \langle\underline{a, c, g, a, c, g}\rangle & \mathit{contraction\ on}\ \langle a, c, g, a, c, g\rangle \\ \mathit{Sequence} \quad S = \langle a, c, g\rangle & \end{array}$$

Figure 1 An example for transforming sequence T to S by two contractions. The corresponding sequence of TDs from S to T would duplicate a, c, g, and then duplicate the first g.

We recall that although we study the minimization problem here, it is unknown whether the question $S \Rightarrow_* T$ can be decided in polynomial time. Nonetheless, our NP-hardness reduction applies to "promise" instances in which $S \Rightarrow_* T$ always holds.

3 NP-hardness of Exemplar-TD

To facilitate the presentation of our hardness proof, we first make an intermediate reduction using the Cost-Effective Subgraph problem, which we will then reduce to the promise version of the Exemplar-TD problem.

The Cost-Effective Subgraph problem

Suppose we are given a graph $G = (V, E)$ and an integer cost $c \in \mathbb{N}_{>0}$. For a subset $X \subseteq V$, let $E(X) = \{uv \in E : u, v \in X\}$ denote the edges inside of X. The *cost* of X is defined as

$$cost(X) = c \cdot (|E(G)| - |E(X)|) + |X| \cdot |E(X)|.$$

The Cost-Effective Subgraph problem asks for a subset X of minimum cost. In the decision version of the problem, we are given an integer r and we want to know if there is a subset X whose cost is at most r. Observe that $X = \emptyset$ or $X = V$ are possible solutions.

The idea is that each edge "outside" of X costs c and each edge "inside" costs $|X|$. Therefore, we pay for each edge not included in X, but if X gets too large, we pay more for edges in X. We must therefore find a balance between the size of X and its number of edges. The connection with the TD problem can be roughly described as follows: in our reduction, we will have many substrings which need to be deleted through contractions. We will have to choose an initial set of contractions X and then, each substring will have two ways to be contracted: one way requires c contractions, and the other requires $|X|$.

An obvious solution for a Cost-Effective Subgraph is to take $X = \emptyset$, which is of cost $c|E(G)|$. Another formulation of the problem could be whether there is a subset X of cost at most $c|E(G)| - p$, where p can be seen as a "profit" to maximize. Treating c and p as parameters, we show the NP-hardness and W[1]-hardness in parameters $c + p$ of the Cost-Effective Subgraph problem (we do not study the parameter r). Our reduction to the TD problem does not preserve W[1]-hardness and we only use the NP-hardness in this paper, but the W[1]-hardness might be of independent interest.

Before proceeding, we briefly argue the relevance of parameter c in the W[1]-hardness. If c is a fixed constant, then we may assume that any solution X satisfies $|X| \leq c$. This is because if $|X| > c$, every edge included in X will cost more than c and putting $X = \emptyset$ yields a lower cost. Thus for fixed c, it suffices to brute-force every subset X of size at most c and we get a $n^{O(c)}$ time algorithm. Our W[1]-hardness shows that it is difficult to remove this exponential dependence between n and c.

▶ **Theorem 2.** *The Cost-Effective Subgraph problem is NP-hard and W[1]-hard for parameter $c + p$.*

Proof. We reduce from CLIQUE. In this classic problem, we are given a graph G and an integer k, and must decide whether G contains a clique of size at least k, where a clique is a set of vertices in which every pair shares an edge. This problem is NP-hard [21] and also W[1]-hard in parameter k [10]. We will assume that k is even (which does not alter either hardness results).

Let (G, k) be a CLIQUE instance, letting $n := |V(G)|$ and $m := |E(G)|$. The graph in our Cost-Effective Subgraph instance is also G. We set the cost $c = 3k/2$, which is an integer since k is even, and set

$$r := c\left(m - \binom{k}{2}\right) + k\binom{k}{2} = cm + \binom{k}{2}(k - c) = cm - \frac{k}{2}\binom{k}{2}.$$

We ask whether G admits a subgraph X satisfying $cost(X) \leq r$. We show that (G, k) is a YES instance to CLIQUE if and only if G contains a set $X \subseteq V(G)$ of cost at most r. This will prove both NP-hardness and W[1]-hardness in $c + p$ (noting that here $p = k/2\binom{k}{2}$).

The forward direction is easy to see. If G is a YES instance, it has a clique X of size exactly k. Since $|E(X)| = \binom{k}{2}$, the cost of X is precisely r.

Let us consider the converse direction. Assume that (G, k) is a NO instance of CLIQUE. Let $X \subseteq V(G)$ be any subset of vertices. We will show that $cost(X) > r$. There are 3 cases to consider depending on $|X|$.

Case 1: $|X| = k$. Since G is a NO instance, X is not a clique and thus $|E(X)| = \binom{k}{2} - h$, where $h > 0$. We have that $cost(X) = c(m - \binom{k}{2} + h) + k(\binom{k}{2} - h) = cm + \binom{k}{2}(k - c) + h(c - k) = r + h(c - k)$. Since $c > k$ and $h > 0$, the cost of X is strictly greater than r.

Case 2: $|X| = k + l$ for some $l > 0$. Denote $|E(X)| = \binom{k+l}{2} - h$, where $0 \leq h \leq \binom{k+l}{2}$ The cost of X is

$$\begin{aligned} cost(X) &= c\left(m - \binom{k+l}{2} + h\right) + (k + l)\left(\binom{k+l}{2} - h\right) \\ &= cm + \binom{k+l}{2}(k + l - c) + h(c - k - l) \\ &= cm + \binom{k+l}{2}\left(l - \frac{k}{2}\right) + h\left(\frac{k}{2} - l\right). \end{aligned}$$

Considering the difference

$$\begin{aligned} cost(X) - r &= \binom{k+l}{2}\left(l - \frac{k}{2}\right) + h\left(\frac{k}{2} - l\right) - \left(-\frac{k}{2}\right)\binom{k}{2} \\ &= \frac{3kl^2}{4} - \frac{kl}{4} + \frac{l^3}{2} - \frac{l^2}{2} + h\left(\frac{k}{2} - l\right), \end{aligned}$$

if $k/2 - l \geq 0$, then the difference is clearly above 0 regardless of h, and then $cost(X) > r$ as desired. Thus we may assume that $k/2 - l < 0$. In this case, we may further assume that $h = \binom{k+l}{2}$, as this minimizes the difference. But in this case,

$$cost(X) = cm + \binom{k+l}{2}\left(l - \frac{k}{2}\right) + \binom{k+l}{2}\left(\frac{k}{2} - l\right) = cm > r,$$

which concludes this case.

Case 3: $|X| = k - l$, with $l > 0$. If $k = l$, then $X = \emptyset$ and $cost(X) = cm > r$. So we assume $k > l$. Put $|E(X)| = \binom{k-l}{2} - h$, where $h \geq 0$. We have

$$\begin{aligned} cost(X) &= c\left(m - \binom{k-l}{2} + h\right) + (k - l)\left(\binom{k-l}{2} - h\right) \\ &= cm + \binom{k-l}{2}(k - l - c) + h(c - k + l) \\ &= cm + \binom{k-l}{2}\left(-\frac{k}{2} - l\right) + h\left(\frac{k}{2} + l\right). \end{aligned}$$

The difference with this cost and r is

$$\begin{aligned} cost(X) - r &= \binom{k-l}{2}\left(-\frac{k}{2} - l\right) + h\left(\frac{k}{2} + l\right) - \left(-\frac{k}{2}\right)\binom{k}{2} \\ &= \frac{3kl^2}{4} + \frac{kl}{4} - \frac{l^3}{2} - \frac{l^2}{2} + h\left(\frac{k}{2} + l\right) \\ &> \frac{1}{4}(3l^3 + l^2) - \frac{1}{2}(l^3 + l^2) \geq 0, \end{aligned}$$

the latter since $k > l \geq 1$. Again, it follows that $cost(X) > r$. ◀

Reduction to Exemplar-TD (promise version)

Since the reduction is somewhat technical, we provide an overview of the techniques that we will use. Let (G, c, r) be a Cost-Effective Subgraph instance where c is the cost and r the optimization value, and with vertices $V(G) = \{v_1, \ldots, v_n\}$. We will construct strings S and T and argue on the number of contractions to go from T to S. We would like our source string to be $S = x_1 x_2 \ldots x_n$, where each x_i is a distinct character that corresponds to vertex v_i. Let S' be obtained by doubling every x_i, i.e. $S' = x_1 x_1 x_2 x_2 \ldots x_n x_n$. Our goal is to put $T = S' E_1 E_2 \ldots E_m$, where each E_i is a substring gadget corresponding to edge $e_i \in E(G)$ that we must remove to go from T to S. Assuming that there is a sequence of contractions that transforms T into S, we make it so that we first want to contract some, but not necessarily all, of the doubled x_i's of S', resulting in another string S''. Let t be the number of x_i's contracted from S' to S''. For instance, we could have $S'' = x_1 x_1 x_2 x_3 x_3 x_4 x_5 x_5$, where only x_2 and x_4 were contracted, and thus $t = 2$. The idea is that these contracted x_i's correspond to the vertices of a cost-effective subgraph. After T is transformed to $S'' E_1 \ldots E_m$, we then force each E_i to use S'' to contract it. For $m = 3$, a contraction sequence that we would like to enforce would take the form

$$\underline{S'}E_1E_2E_3 \rightarrowtail \underline{S''E_1}E_2E_3 \rightarrowtail \underline{S''E_2}E_3 \rightarrowtail \underline{S''E_3} \rightarrowtail \underline{S''} \rightarrowtail S,$$

where we underline the substring affected by contractions at each step. We make it so that when contracting $S''E_iE_{i+1}\dots E_m$ into $S''E_{i+1}\dots E_m$, we have two options. Suppose that v_j, v_k are the endpoints of edge e_i. If, in S'', we had chosen to contract x_j and x_k, we can contract E_i using a sequence of t moves. Otherwise, we must contract E_i using another more costly sequence of c moves. The total cost to eliminate the E_i gadgets will be $c(m - e) + te$, where e is the number of edges that can be contracted using the first choice, i.e. for which both endpoints were chosen in S''.

Unfortunately, constructing S' and the E_i's to implement the above idea is not straightforward. The main difficulty lies in forcing an optimal solution to behave as we describe, i.e. enforcing going from S' to S'' first, enforcing the E_i's to use S'', and enforcing the two options to contract E_i with the desired costs. In particular, we must replace the x_i's by carefully constructed substrings X_i. We must also repeat the sequence of E_i's a certain number p times. We now proceed with the technical details.

▶ **Theorem 3.** *The Exemplar-TD problem is NP-complete, even if for the given string S and T, $S \Rightarrow_* T$ is guaranteed to hold.*

Proof. To see that the problem is in NP, note that $dist_{TD}(S,T) \leq |T|$ since each contraction from T to S removes at least character. Thus a sequence of contractions can serve as a certificate, has polynomial size and is easy to verify.

For hardness, we reduce from the Cost-Effective Subgraph problem, which has been shown NP-hard in Theorem 2. Let (G,c,r) be an instance of Cost-Effective Subgraph, letting $n := |V(G)|$ and $m := |E(G)|$. Here c is the "outsider edge" cost and we ask whether there is a subset $X \subseteq V(G)$ such that $c(m - |E(X)|) + |X||E(X)| \leq r$. We denote $V(G) = \{v_1,\dots,v_n\}$ and $E(G) = \{e_1,\dots,e_m\}$. The ordering of vertices and edges is arbitrary but remains fixed for the remainder of the proof. For convenience, we allow the edge indices to loop through 1 to m, and so we put $e_i = e_{i+lm}$ for any integer $l \geq 0$. Thus we may sometimes refer to an edge e_h with an index $h > m$, meaning that e_h is actually the edge $e_{((h-1) \mod m)+1}$.

The construction. Let us first make an observation. If we take an exemplar string $X = x_1 \dots x_l$ (i.e. a string in which no character occurs twice), we can double its characters and obtain a string $X' = x_1x_1\dots x_lx_l$. The length of X' is only twice that of X and $dist_{TD}(X,X') = l$, i.e. going from X' to X requires l contractions. We will sometimes describe pairs of strings X and X' at distance l without explicitly describing X and X', but the reader can assume that X starts as an exemplar string of length l and we obtain X' by doubling each character, as above.

Now we show how to construct S and T. First let $d = m + 1$ and $p = m(n + m)^{10}$. The exact values of d and p are not crucial and will only refer to them when needed: for the most part, it is enough to think of d and p as simply "large enough". Note however that p is a multiple of m. For later reference, the value of k we will use in the reduction is $k = p/m \cdot d(r + nm) + 4cdn$.

Instead of doubling x_i's as in the intuition paragraph above, we will duplicate some characters d times. Moreover, we can't create a T string that behaves exactly as described above, but we will show that we can append p copies of carefully crafted substring to obtain the desired result. We need d and p to be high enough so that "enough" copies behave as we desire.

For each $i \in [n]$, define an exemplar string X_i of length d. Moreover, create enough characters so that no two X_i strings contain a character in common. Let X_i^d be a string satisfying $dist_{TD}(X_i, X_i^d) = d$.

Then for each $j \in \{0, 1, \ldots, 2p\}$, define an exemplar string B_j. Ensure that no B_j contains a character from an X_i string, and no two B_j's contain a common character. The B_j strings can consist of a single character, with the exception of B_0 and B_1 which are special. We assume that for B_0 and B_1, we have strings B_0^* and B_1^* such that

$$dist_{TD}(B_0, B_0^*) = dc + 2d - 2,$$
$$dist_{TD}(B_1, B_1^*) = dn + 2d - 1.$$

Again, this can be done using the doubling trick on exemplar strings. The B_j's are the building blocks of larger strings. For each $q \in [2p]$, define

$$\mathcal{B}_q = B_q B_{q-1} \ldots B_2 B_1 B_0, \qquad \mathcal{B}_q^0 = B_q B_{q-1} \ldots B_2 B_1 B_0^*,$$
$$\mathcal{B}_q^1 = B_q B_{q-1} \ldots B_2 B_1^* B_0, \qquad \mathcal{B}_q^{01} = B_q B_{q-1} \ldots B_2 B_1^* B_0^*.$$

These strings are used as "blockers" and prevent certain contractions from happening. Note that $\mathcal{B}_q^0$ and $\mathcal{B}_q^1$ can be turned into $\mathcal{B}_q$ using $dc+2d-2$ contractions and $dn+2d-1$ contractions, respectively. Moreover, $\mathcal{B}_q^{01}$ can be turned into $\mathcal{B}_q^0$ using $dn + 2d - 1$ contractions and into $\mathcal{B}_q^1$ using $dc + 2d - 2$ contractions.

Also define the strings

$$\mathcal{X} = X_1 X_2 \ldots X_n, \qquad \mathcal{X}^d = X_1^d X_2^d \ldots X_n^d,$$

and for edge $e_q = v_i v_j$ with $q \in [p]$ whose endpoints are v_i and v_j, define

$$\mathcal{X}_{e_q} = X_1^d \ldots X_{i-1}^d X_i X_{i+1}^d \ldots X_{j-1}^d X_j X_{j+1}^d \ldots X_n^d.$$

Thus in $\mathcal{X}_{e_q}$, all X_k substrings are turned into X_k^d, except X_i and X_j.

Finally, define a new additional character Δ, which will be used to separate some of the components of our string. We can now define S and T. We have

$$S = \mathcal{B}_{2p} \mathcal{X} \Delta = B_{2p} B_{2p-1} \ldots B_2 B_1 B_0 X_1 X_2 \ldots X_n \Delta.$$

It follows from the definitions of $\mathcal{B}_{2p}, \mathcal{X}$ and Δ that S is exemplar. Now for $i \in [p]$, define

$$E_i := \mathcal{B}_i^{01} \mathcal{X}_{e_i} \Delta \mathcal{B}_{2p}^1 \mathcal{X} \Delta,$$

which we will call the *edge gadget*. Define T as

$$\begin{aligned} T &= \mathcal{B}_{2p}^0 \mathcal{X}^d \Delta \mathcal{B}_{2p}^1 \mathcal{X} \Delta E_1 E_2 \ldots E_p \\ &= \mathcal{B}_{2p}^0 \mathcal{X}^d \Delta \mathcal{B}_{2p}^1 \mathcal{X} \Delta \left[\mathcal{B}_1^{01} \mathcal{X}_{e_1} \Delta \mathcal{B}_{2p}^1 \mathcal{X} \Delta\right] \left[\mathcal{B}_2^{01} \mathcal{X}_{e_2} \Delta \mathcal{B}_{2p}^1 \mathcal{X} \Delta\right] \ldots \left[\mathcal{B}_p^{01} \mathcal{X}_{e_p} \Delta \mathcal{B}_{2p}^1 \mathcal{X} \Delta\right]. \end{aligned}$$

We add brackets for clarity only – they indicate the distinct E_i substrings, but the brackets are not actual characters of T. The idea is that T starts with $S' = \mathcal{B}_{2p}^0 \mathcal{X}^d \Delta$, a modified S in which $\mathcal{B}_{2p}$ becomes $\mathcal{B}_{2p}^0$ and the X_i substrings are turned into X_i^d. This $\mathcal{X}^d$ substring serves as a choice of vertices in our cost-effective subgraph. Each edge e_i has a "gadget substring" $E_i = \mathcal{B}_i^{01} \mathcal{X}_{e_i} \Delta \mathcal{B}_{2p}^1 \mathcal{X} \Delta$. Since p is a multiple of m, the sequence of edge gadgets $E_1 E_2 \ldots E_m$ is repeated p/m times. Our goal to go from T to S is to get rid of all these edge gadgets by contractions. Note that because a E_i gadget starts with $\mathcal{B}_i^{01}$ and the gadget E_{i+1} starts with $\mathcal{B}_{i+1}^{01}$, the substring E_{i+1} has a character that the substring E_i does not have.

The hardness proof. We now show that G admits a subset of vertices W of cost at most r if and only if T can be contracted to S using at most $p/m \cdot d(r + nm) + 4cdn$ contraction operations. We include the forward direction, which is the most instructive, in the main text. The other direction can be found in the full version. Although we shall not dig into details here, it can be deduced from the ($\Rightarrow$) direction that $T \rightarrowtail_* S$ holds.

($\Rightarrow$) Suppose that G admits a subset of vertices W of cost at most r. Thus $c(m - |E(W)|) + |W| \cdot |E(W)| \leq r$. To go from T to S, first consider an edge e_i that does not have both endpoints in W. We show how to get rid of the gadget substring E_i for e_i using $dn + dc$ contractions. Note that T contains the substring $\mathcal{B}^1_{2p}\mathcal{X}\Delta E_i = \mathcal{B}^1_{2p}\mathcal{X}\Delta[\mathcal{B}^{01}_i\mathcal{X}_{e_i}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta]$, where brackets surround the E_i occurrence that we want to remove (note that here for $i > 1$, the prefix $\mathcal{B}^1_{2p}\mathcal{X}\Delta$ is the suffix of the previous E_{i-1} gadget, and for $i = 1$, it is the suffix of the starting block of T). We can first contract $\mathcal{B}^{01}_i$ to $\mathcal{B}^1_i$ using $dc + 2d - 2$ contractions, then contract $\mathcal{X}_{e_i}$ to $\mathcal{X}$ using $d(n - 2)$ contractions. The result is the $\mathcal{B}^1_{2p}\mathcal{X}\Delta[\mathcal{B}^1_i\mathcal{X}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta]$ substring, which becomes $\mathcal{B}^1_{2p}\mathcal{X}\Delta$ using two contractions (see below). This sums to $dc + 2d - 2 + d(n - 2) + 2 = dc + dn$ operations. More visually, the sequence of contractions works as follows (as before, brackets indicate the E_i substring and what remains of it, and the underlines are there to emphasize the substrings that participate in the contractions(s)):

$$\begin{aligned}
&\mathcal{B}^1_{2p}\mathcal{X}\Delta\left[\underline{\mathcal{B}^{01}_i}\mathcal{X}_{e_i}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\right] && (dc + 2d - 2 \text{ contractions})\\
\rightarrowtail&\mathcal{B}^1_{2p}\mathcal{X}\Delta\left[\mathcal{B}^1_i\underline{\mathcal{X}_{e_i}}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\right] && (d(n-2) = dn - 2d \text{ contractions})\\
\rightarrowtail&\mathcal{B}^1_{2p}\mathcal{X}\Delta\left[\mathcal{B}^1_i\mathcal{X}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\right] &&\\
=&B_{2p}B_{2p-1}\dots B_{i+1}\underline{\mathcal{B}^1_i\mathcal{X}\Delta}\left[\underline{\mathcal{B}^1_i\mathcal{X}\Delta}\mathcal{B}^1_{2p}\mathcal{X}\Delta\right] && (1 \text{ contraction})\\
\rightarrowtail&B_{2p}B_{2p-1}\dots B_{i+1}\mathcal{B}^1_i\mathcal{X}\Delta\left[\mathcal{B}^1_{2p}\mathcal{X}\Delta\right] &&\\
=&\underline{\mathcal{B}^1_{2p}\mathcal{X}\Delta}\left[\underline{\mathcal{B}^1_{2p}\mathcal{X}\Delta}\right] && (1 \text{ contraction})\\
\rightarrowtail&\mathcal{B}^1_{2p}\mathcal{X}\Delta.
\end{aligned}$$

This sequence of $dn + dc$ contractions effectively removes the E_i substring gadget. Observe that after applying this sequence, it is still true that every remaining E_j gadget substring is preceded by $\mathcal{B}^1_{2p}\mathcal{X}\Delta$. We may therefore repeatedly apply this contraction sequence to every e_i not contained in W (including those e_i gadgets for which $i > m$). This procedure is thus applied to $p/m \cdot (m - |E(W)|)$ gadgets. We assume that we have done so, and that every e_i for which the E_i gadget substring remains is in W. Call the resulting string T'.

Now, let $\mathcal{X}_W$ be the substring obtained from $\mathcal{X}^d$ by contracting, for each $v_i \in W$, the string X^d_i to X_i. We assume that we have contracted the $\mathcal{X}^d$ substring of T' to $\mathcal{X}_W$, which uses $d|W|$ contractions (note that there is only one occurrence of $\mathcal{X}^d$ in T', namely right before the first Δ). Call T'' the resulting string. At this point, for every E_i substring gadget that remains, where E_i corresponds to edge $e_i = v_jv_k$, $\mathcal{X}_W$ contains the substrings X_j and X_k (instead of X^d_j and X^d_k).

Let i be the smallest integer for which the e_i substring gadget E_i is still in T'. This is the leftmost edge gadget still in T'', meaning that T'' has the prefix

$$\mathcal{B}^0_{2p}\mathcal{X}_W\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\left[\mathcal{B}^{01}_i\mathcal{X}_{e_i}\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\right],$$

where brackets indicate the E_i substring. To remove E_i, first contract $\mathcal{B}^{01}_i$ to $\mathcal{B}^0_i$, and contract $\mathcal{X}_{e_i}$ to X_W (this is possible since $e_i \subseteq W$). The result is $\mathcal{B}^0_{2p}\mathcal{X}_W\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\left[\mathcal{B}^0_i\mathcal{X}_W\Delta\mathcal{B}^1_{2p}\mathcal{X}\Delta\right]$. One more contraction gets rid of the second half. This requires $dn + 2d - 1 + d(|W| - 2) + 1 =$

$dn + d|W|$ contractions. This procedure is applied to $p/m \cdot |E(W)|$ gadgets. To recap, the contraction sequence for E_i does as follows:

$$\begin{aligned}
&\mathcal{B}_{2p}^{0}\mathcal{X}_W\Delta\mathcal{B}_{2p}^{1}\mathcal{X}\Delta\left[\underline{\mathcal{B}_i^{01}}\mathcal{X}_{e_i}\Delta\mathcal{B}_{2p}^{1}\mathcal{X}\Delta\right] && (dn + 2d - 1 \text{ contractions})\\
\rightarrowtail &\mathcal{B}_{2p}^{0}\mathcal{X}_W\Delta\mathcal{B}_{2p}^{1}\mathcal{X}\Delta\left[\mathcal{B}_i^{0}\underline{\mathcal{X}_{e_i}}\Delta\mathcal{B}_{2p}^{1}\mathcal{X}\Delta\right] && (d(|W| - 2) \text{ contractions})\\
\rightarrowtail &\underline{\mathcal{B}_{2p}^{0}\mathcal{X}_W\Delta\mathcal{B}_{2p}^{1}\mathcal{X}\Delta}\left[\underline{\mathcal{B}_i^{0}\mathcal{X}_W\Delta\mathcal{B}_{2p}^{1}\mathcal{X}\Delta}\right] && (1 \text{ contraction})\\
\rightarrowtail &\mathcal{B}_{2p}^{0}\mathcal{X}_W\Delta\mathcal{B}_{2p}^{1}\mathcal{X}\Delta.
\end{aligned}$$

After we repeat this for every E_i, all that remains is the string $\mathcal{B}_{2p}^{0}\mathcal{X}_W\Delta\mathcal{B}_{2p}^{1}\mathcal{X}\Delta$. We contract $\mathcal{X}_W$ to $\mathcal{X}$ using $d(n-|W|)$ contractions (in total, going from $\mathcal{X}^d$ to $\mathcal{X}$ required dn contractions). Then contract $\mathcal{B}_{2p}^{0}$ and $\mathcal{B}_{2p}^{1}$ to $\mathcal{B}_{2p}$ using $dc+2d-2+dn+2d-1 = d(c+n+4)-3$ contractions. One more contraction of the second half of the string yields S. The summary of the number of contractions made is

$$\begin{aligned}
&\frac{p}{m}\cdot(m - |E(W)|)\cdot(dc + dn) + \frac{p}{m}\cdot|E(W)|\cdot(dn + d|W|) + dn + d(c + n + 4) - 3\\
\leq &\frac{p}{m}\cdot(m - |E(W)|)\cdot(dc + dn) + \frac{p}{m}\cdot|E(W)|\cdot(dn + d|W|) + 4cdn\\
= &\frac{p}{m}\cdot d\cdot(c + n)(m - |E(W)|) + \frac{p}{m}\cdot d\cdot(n + |W|)|E(W)| + 4cdn\\
= &\frac{p}{m}\cdot d\cdot[c(m - |E(W)|) + |W||E(W)| + nm] + 4cdn\\
\leq &\frac{p}{m}\cdot d(r + nm) + 4cdn,
\end{aligned}$$

as desired.

($\Leftarrow$): this direction of the proof is somewhat involved and can be found in the full version. The idea is to show that a minimum contraction sequence must have the form similar to that in the ($\Rightarrow$) direction. The challenging part is to show that each E_i substring must get removed separately in this sequence, and that "most" of them incur a cost of either $dn + dt - 2$ or $dn + dc - 2$ for some t (this "most" is the reason that we need a large p). ◀

4 An FPT algorithm for the exemplar problem

In this section, we will show that Exemplar-k-TD can be solved in time $2^{O(k^2)} + poly(n)$ by obtaining a kernel of size $O(k2^k)$, where n is the length of T.

We first note that there is a very simple, brute-force algorithm to solve the k-TD problem, which is the variant of the TD problem with parameter k, the number of TDs to turn S into T (including Exemplar-k-TD as a particular case). This only establishes membership in the XP class, but it will be useful to evaluate the complexity of our kernelization later on.

▶ **Proposition 4.** *The k-TD problem can be solved in time $O(n^{2k})$, where n is the size of the target string.*

Proof. Let (S, T) be a given instance of k-TD. Consider the branching algorithm that, starting from T, tries to contract every substring of the form XX in T and recurses on each resulting string, decrementing k by 1 each time (the branching stops when S is obtained or when k reaches 0 without attaining S). We obtain a search tree of depth at most k and degree at most n^2, and thus it has $O(n^{2k})$ nodes. Visiting the internal nodes of this search tree only requires enumerating $O(n^2)$ substrings, which form the set of children of the node. Hence, there is no added computation cost to consider when visiting a node. ◀

From now on, we assume that we have an Exemplar-k-TD instance (S, T), and so that S is exemplar.

Let x and y be two consecutive characters in S (i.e. xy is a substring of S). We say that xy is (S, T)-*stable* if in T, every occurrence of x in T is followed by y and every occurrence of y is preceded by x. That is, the direct successor of every x character is y, and the direct predecessor of every y character is y. An (S, T)-*stable substring* $X = x_1 \dots x_l$, where $l \geq 2$, is a substring of S such that $x_i x_{i+1}$ is (S, T)-stable for every $i \in [l - 1]$. We also define a string with a single character x_i to be a (S, T)-stable substring (provided x_i appears in S and T). If any substring of S that strictly contains X is not an (S, T)-stable substring, then X is called a *maximal* (S, T)-*stable substring.* Note that these definitions are independent of S and T, and so the same definitions apply for (X, Y)-stability, for any strings X and Y.

We will show that every maximal (S, T)-stable substring can be replaced by a single character, and that if T can be obtained from S using at most k tandem duplications, then this leaves strings of bounded size.

We first show that, roughly speaking, stability is maintained by all tandem duplications when going from S to T.

▶ **Lemma 5.** *Suppose that $dist_{TD}(S, T) = k$ and let X be an (S, T)-stable substring. Let $S = S_0, S_1, \dots, S_k = T$ be any minimum sequence of strings transforming S to T by tandem duplications. Then X is (S, S_i)-stable for every $i \in [k]$.*

Proof. Assume the lemma is false, and let S_i be the first of $S_1, \dots, S_k$ that does not satisfy the statement. Then there are two characters x, y belonging to X such that xy is (S, T)-stable, but xy is not (S, S_i)-stable.

We claim that, under our assumption, xy is not (S, S_j)-stable for any $j \in \{i, \dots, k\}$. As this includes $S_k = T$, this will contradict that xy is (S, T)-stable. We do this by induction – as a base case, xy is not (S, S_i)-stable so this is true for $j = i$. Assume that xy is not (S, S_{j-1})-stable, where $i < j \leq k$. Let D be the duplication transforming S_{j-1} to S_j (here $D = (a, b)$ contains the start and end positions of the substring of S_{j-1} to duplicate).

Suppose first that xy is not (S, S_{j-1})-stable because S_{j-1} has an occurrence of x that is not followed by y. Thus S_{j-1} has an occurrence of x, say at position p_x, followed by $z \neq y$. If we assume that xy is (S, S_j)-stable, then a y character must have appeared after this x from S_{j-1} to S_j. Changing the character next to this x is only possible if the last character duplicated by D is the x at position p_x and the first character of D is a y. In other words, denoting $S_{j-1} = A_1 y A_2 x z A_3$ for appropriate A_1, A_2, A_3 substrings, the D duplication must do the following

$$A_1 \underline{y A_2 x} z A_3 \Rightarrow A_1 \underline{y A_2 x y A_2 x} z A_3,$$

as otherwise, the character next to the above occurrence of x will remain z. But then, there is still an occurrence of x followed by z, and it follows that xy cannot be (S, S_j)-stable.

So suppose instead that xy is not (S, S_{j-1})-stable because S_{j-1} has an occurrence of y preceded by $z \neq x$. Assume the character preceding this y has changed in S_j and has become x. But one can verify that this is impossible. For completeness, we present each possible case: either D includes both z and y, includes one of them or none. These cases are represented below, and each one of them leads to an occurrence of y still preceded by z (the left-hand side represents S_{j-1} and the right-hand side represents S_j, and the A_i's are the relevant substrings in each case):

$$\begin{aligned}
\text{Include both: } & A_1\underline{A_2zyA_3}A_4 \Rightarrow A_1\underline{A_2zyA_3A_2zyA_3}A_4,\\
\text{Include } z \text{ only: } & A_1\underline{A_2z}yA_3 \Rightarrow A_1\underline{A_2zA_2z}yA_3,\\
\text{Include } y \text{ only: } & A_1z\underline{yA_2}A_3 \Rightarrow A_1z\underline{yA_2yA_2}A_3,\\
\text{Include none: } & A_1\underline{A_2}zyA_3 \Rightarrow A_1\underline{A_2A_2}zyA_3 \text{ or } A_1zy\underline{A_2}A_3 \Rightarrow A_1zy\underline{A_2A_2}A_3.
\end{aligned}$$

We have therefore shown that xy cannot be (S, S_j)-stable, and therefore not (S, T)-stable, which concludes the proof. ◀

Let S' be a substring obtained from S by tandem duplications, and let $X := S'[a..b]$ be the substring of S' at positions from a to b. Suppose that we apply a duplication $D = (c, d)$, which copies the substring $S'[c..d]$. Then we say that D *cuts* X if one of the following holds:

- $a < c \leq b$ and $b < d$, in which case we say that D cuts X *to the right*;
- $c < a$ and $a \leq d < b$, in which case we say that D cuts X *to the left*;
- $(a, b) \neq (c, d)$ and $a \leq c < d \leq b$, in which case D cuts X *inside.*

In other words, if we write $X = X_1X_2$ and $S' = UVX_1X_2WY$, cutting to the right takes the form $UVX_1\underline{X_2W}Y \Rightarrow UBX_1\underline{X_2WX_2W}Y$. Cutting to the left takes the form $U\underline{VX_1}X_2WY \Rightarrow U\underline{VX_1VX_1}X_2WY$. Rewriting $X = X_1X_2X_3$ and $S' = UX_1X_2X_3V$, cutting inside takes the form $UX_1\underline{X_2}X_3V \Rightarrow UX_1\underline{X_2X_2}X_3V$. Note that if D does not cut any occurrence of a maximal (S, S')-stable substring X and S'' is obtained by applying D on S', then X is (S, S'')-stable.

The next lemma shows that we can assume that maximal stable substrings never get cut, and thus always get duplicated together. The idea is that any duplication that cuts an X_j can be replaced by an equivalent duplication that doesn't.

▶ **Lemma 6.** *Suppose that $dist_{TD}(S, T) = k$, and let $X_1, \ldots, X_l$ be the set of maximal (S, T)-stable substrings. Then there exists a sequence of tandem duplications $D_1, \ldots, D_k$ transforming S into T such that no occurrence of an X_j gets cut by a D_i.*

In other words, for all $i \in [k]$ and all $j \in [l]$, the tandem duplication D_i does not cut any occurrence of X_j in the string obtained by applying $D_1, \ldots, D_{i-1}$ to S.

The above implies that we may replace each maximal (S, T)-stable substring X of S and T by a single character, since we may assume that characters of X are always duplicated together (assuming, of course, that S is exemplar). It only remains to show that the resulting strings are small enough. The proof of the following lemma has a very simple intuition. First, S has exactly 1 maximal (S, S)-stable substring. Each time we apply a duplication, we "break" at most 2 stable substrings, which creates 2 new ones. So if we apply k duplications, there are at most $2k + 1$ such substrings in the end.

▶ **Lemma 7.** *If $dist_{TD}(S, T) \leq k$, then there are at most $2k + 1$ maximal (S, T)-stable substrings.*

Proof. Let $S = S_0, S_1, \ldots, S_k = T$ be any minimum sequence of strings transforming S to T by tandem duplications. We show by induction that, for each $i \in \{0, 1, \ldots, k\}$, the number of maximal (S, S_i)-stable substrings is at most $2i + 1$. For $i = 0$, there is only one maximal (S, S)-stable substring, namely S itself. Now assume that there are at most $2(i-1)+1 = 2i-1$ maximal (S, S_{i-1})-stable substrings. Let $\mathbb{X} = \{X_1, \ldots, X_l\}$ be the set of these substrings, $l \leq 2i - 1$. We then know that S_{i-1} can be written as a concatenation of X_j's from $\mathbb{X}$

(with possible repetitions). The duplication D transforming S_{i-1} to S_i copies some of these X_j's entirely, except at most two X_j's at the ends which it may copy partially (i.e. D cuts at most two substrings from $\mathbb{X}$). In other words, the substring duplicated by D can be written as $X_j^2 X_{a_1} X_{a_2} \dots X_{a_r} X_h^1$, where $X_j = X_j^1 X_j^2$ and $X_h = X_h^1 X_h^2$ for some $j, h \in [l]$ (and $X_{a_1}, \dots, X_{a_r} \in \mathbb{X}$). Going further, S_{i-1} and S_i can be written, using appropriate substrings A, B, C that are concatenation of elements of $\mathbb{X}$, as

$$S_{i-1} = AX_j^1 \underline{X_j^2 B X_h^1} X_h^2 C \Rightarrow AX_j^1 \underline{X_j^2 B X_h^1 X_j^2 B X_h^1} X_h^2 C = S_i.$$

Now, any $X_r \in \mathbb{X} \setminus \{X_j, X_h\}$ is (S, S_i)-stable. Moreover, X_j^1, X_j^2, X_h^1 and X_h^2 are also (S, S_i)-stable. This shows that the number of maximal (S, S_i)-stable substrings is at most $2i - 1 - 2 + 4 = 2i + 1$, as desired. ◀

We can now transform an instance (S, T) of Exemplar-k-TD to a kernel, an equivalent instance (S', T') of size depending only on k.

▶ **Theorem 8.** *An instance (S, T) of Exemplar-k-TD admits a kernel (S', T') in which $|S'| \leq 2k + 1$ and $|T'| \leq (2k + 1)2^k$.*

Proof. Let S', T' be obtained from an instance (S, T) by replacing each maximal (S, T)-stable substring by a distinct character. We first prove that (S', T') is indeed a kernel by establishing its equivalence with (S, T). Clearly if (S', T') can be solved using at most k duplications, then the same applies to (S, T). By Lemma 6, the converse also holds: if (S, T) can be solved with at most k duplications, we may assume that these duplications never cut a maximal (S, T)-stable substring, and so these duplications can be applied on (S', T').

Then by Lemma 7, we know that S' has at most $2k + 1$ characters. If $dist_{TD}(S', T') \leq k$, then each duplication can at most double the size of the previous string. Therefore, T' must have size at most $(2k + 1)2^k$. ◀

The kernelization can be performed in polynomial time, as one only needs to identify maximal (S, T)-stable substrings and contract them (we do not bother with the exact complexity for now). Running the brute-force algorithm from Proposition 4 yields the following.

▶ **Corollary 9.** *The exemplar k-tandem duplication problem can be solved in time $O(((2k + 1)2^k)^{2k} + poly(n)) = 2^{O(k^2)} + poly(n)$, where n is the size of the input.*

5 Open problems

Although this work answers some open questions, many of them still deserve investigation. We conclude with some of these questions along with future research perspectives.

- Is the k-TD problem FPT in parameter k? As we observe in our Exemplar-k-TD kernelization, if T and S are large compared to k, they must share many long common substrings which could be exploited for an FPT algorithm. It is also an interesting question whether Exemplar-k-TD admits a polynomial size kernel.
- If $|\Sigma|$ is fixed, is k-TD in P? Even the $|\Sigma| = 2$ case is open. One possibility it to check whether we can reduce the alphabet of any instance to some constant by encoding each character appropriately.

- Can one decide in polynomial time whether $S \Rightarrow_* T$? The only known result on this topic is that it can be done if $|\Sigma| = 2$, as one can construct a finite automaton accepting all strings generated by S (though this automaton does not give the minimum number of duplications required).
- Does the k-TD problem admit a constant factor approximation algorithm? The answer might depend on the hardness of deciding whether $S \Rightarrow_* T$, but one might still consider the promise version of the problem.
- If the length of each duplicated string is bounded by d, is k-TD in P (with d treated as a constant)? We believe that it is FPT in $k + d$, but is it FPT in d?

References

1 Noga Alon, Jehoshua Bruck, Farzad F. Hassanzadeh, and Siddharth Jain. Duplication distance to the root for binary sequences. *IEEE Transactions on Information Theory*, 63(12):7793–7803, 2017. doi:10.1109/TIT.2017.2744608.

2 Gary Benson and Lan Dong. Reconstructing the duplication history of a tandem repeat. In *Proceedings of the 17th International Conference on Intelligent Systems for Molecular Biology*, ISMB'99, pages 44–53. AAAI, 1999.

3 Daniel P. Bovet and Stefano Varricchio. On the regularity of languages on a binary alphabet generated by copying systems. *Information Processing Letters*, 44(3):119–123, 1992. doi:10.1016/0020-0190(92)90050-6.

4 Laurent Bulteau, Guillaume Fertin, and Irena Rusu. Sorting by transpositions is difficult. *SIAM Journal on Discrete Mathematics*, 26(3):1148–1180, 2012. doi:10.1137/110851390.

5 Brian Charlesworth, Paul Sniegowski, and Wolfgang Stephan. The evolutionary dynamics of repetitive dna in eukaryotes. *Nature*, 371:215–220, 1994. doi:10.1038/371215a0.

6 Kamalika Chaudhuri, Kevin Chen, Radu Mihaescu, and Satish Rao. On the tandem duplication-random loss model of genome rearrangement. In *Proceedings of 17th ACM-SIAM Symposium on Discrete Algorithms*, SODA'06, pages 564–570. SIAM, 2006.

7 Zhi-Zhong Chen, Lusheng Wang, and Zhanyong Wang. Approximation algorithms for reconstructing the duplication history of tandem repeats. *Algorithmica*, 54(4):501–529, 2009. doi:10.1007/s00453-008-9209-8.

8 Da-Jung Cho, Yo-Sub Han, and Hwee Kim. Bound-decreasing duplication system. *Theoretical Computer Science*, 793:152–168, 2019. doi:10.1016/j.tcs.2019.06.018.

9 Juegen Dassow, Victor Mitrana, and Gheorghe Paun. On the regularity of the duplication closure. *Bulletin of the EATCS*, 69:133–136, 1999.

10 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness ii: On completeness for w[1]. *Theoretical Computer Science*, 141(1–2):109–131, 1995. doi:10.1016/0304-3975(94)00097-3.

11 Andrzej Ehrenfeucht and Grzegorz Rozenberg. On regularity of languages generated by copying systems. *Discrete Applied Mathematics*, 8(3):313–317, 1984. doi:10.1016/0166-218X(84)90129-X.

12 Andrew J. Sharp et al. and Even E. Eichler. Segmental duplications and copy-number variation in the human genome. *The American J. of Human Genetics*, 77(1):78–88, 2005. doi:10.1086/431652.

13 Marcy Macdonald et al. and Peter S. Harper. A novel gene containing a trinucleotide repeat that is expanded and unstable on huntington's disease. *Cell*, 72(6):971–983, 1993. doi:10.1016/0092-8674(93)90585-E.

14 Olivier Gascuel, Michael D. Hendy, Alain Jean-Marie, and Robert McLachlan. The combinatorics of tandem duplication trees. *Systematic Biology*, 52(1):110–118, 2003. doi:10.1080/10635150390132821.

15 Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology.* Cambridge University Press, 1997.

16 Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *Journal of Computer and System Sciences*, 69(4):525–546, 2004. doi:10.1016/j.jcss.2004.03.004.

17 Sridhar Hannenhalli and Pavel A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proceedings of IEEE 36th Symposium on Foundations of Computer Science*, FOCS'95, pages 581–592. IEEE, 1995. doi:10.1109/SFCS.1995.492588.

18 Farzad F. Hassanzadeh, Moshe Schwartz, and Jehoshua Bruck. The capacity of string-duplication systems. *IEEE Transactions on Information Theory*, 62(2):811–824, 2016. doi:10.1109/TIT.2015.2505735.

19 Masami Ito, Peter Leupold, and Kayoko Shikishima-Tsuji. Closure of languages under bounded duplications. In *Proceedings of the 10th International Conference on Developments in Language Theory*, volume 4036 of *Lecture Notes in Computer Science*, pages 238–247. Spring-Verlag, 2006. doi:10.1007/11779148_22.

20 Siddharth Jain, Farzad F. Hassanzadeh, and Jehoshua Bruck. Capacity and expressiveness of genomic tandem duplication. *IEEE Transactions on Information Theory*, 63(10):6129–6138, 2017. doi:10.1109/TIT.2017.2728079.

21 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.

22 Gad M. Landau, Jeanette P. Schmidt, and Dina Sokol. An algorithm for approximate tandem repeats. *Journal of Computational Biology*, 8(1):1–18, 2001. doi:10.1089/106652701300099038.

23 Ivica Letunic, Richard R. Copley, and Peer Bork. Common exon duplication in animals and its role in alternative splicing. *Human Molecular Genetics*, 11(13):1561–1567, 2002. doi:10.1093/hmg/11.13.1561.

24 Peter Leupold, Carlos Martin-Vide, and Victor Mitrana. Uniformly bounded duplication languages. *Discrete Applied Mathematics*, 146(3):301–310, 2005. doi:10.1016/j.dam.2004.10.003.

25 Peter Leupold, Victor Mitrana, and Jose M. Sempere. Formal languages arising from gene repeated duplication. In Gheorghe Paun Natasa Jonoska and Grzegorz Rozenberg, editors, *Aspects of Molecular Computing*, volume 2950 of *Lecture Notes in Computer Science*, pages 297–308. Springer-Verlag, Berlin, 2004. doi:10.1007/978-3-540-24635-0_22.

26 Layla Oesper, Anna M. Ritz, Sarah J. Aerni, Ryan Drebin, and Benjamin J. Raphael. Reconstructing cancer genomes from paired-end sequencing data. *BMC Bioinformatics*, 13(Suppl 6):S10, 2012. doi:10.1186/1471-2105-13-S6-S10.

27 Letu Qingge, Xiaozhou He, Zhihui Liu, and Binhai Zhu. On the minimum copy number generation problem in cancer genomics. In *Proceedings of the 10th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, BCB'18, pages 260–269, New York, NY, 2018. ACM Press. doi:10.1145/3233547.3233586.

28 David Sankoff. Gene and genome duplication. *Current opinion in genetics and development*, 11(6):681–684, 2001. doi:10.1016/S0959-437X(00)00253-7.

29 Jack W. Szostak and Ray Wu. Unequal crossing over in the ribosomal dna of saccharomyces cerevisiae. *Nature*, 284:426–430, 1980. doi:10.1038/284426a0.

30 Olivier Tremblay-Savard, Denis Bertrand, and Nadia El-Mabrouk. Evolution of orthologous tandemly arrayed gene clusters. *BMC Bioinformatics*, 12(Suppl 9):S2, 2011. doi:10.1186/1471-2105-12-S9-S2.

31 Ming-Wei Wang. On the irregularity of the duplication closure. *Bulletin of the EATCS*, 70:162–163, 2000.