Tandem Duplication Parameterized by the Length Difference

Peter Damaschke

Department of Computer Science and Engineering, Chalmers University
41296 Göteborg, Sweden
ptr@chalmers.se

Abstract. A tandem duplication in a string takes a substring and inserts another copy of it right beside it. Given two strings, we want to find a shortest sequence of tandem duplications that transform the shorter string into the longer one, or recognize that no such sequence exists. The problem, in particular with short tandem duplications, is of interest in genomics, and a number of complexity results are known. First we improve a recent simple XP algorithm. However, our main technical contributions are an FPT algorithm, where the parameter is the difference of lengths of the two given strings, and a polynomial kernel.

Keywords: tandem duplication, edit distance, periodic string, ynamic programming, parameterized algorithm, alignment graph

1 Introduction

1.1 Definitions and Problem

In this work we give combinatorial and algorithmic results for a particular problem on strings, i.e., sequences of symbols from an alphabet. For understanding the problem, some basic definitions are needed first.

We sometimes use $|X|$ to denote the length of a string X, that is, the number of symbols in X (where multiple occurrences of a symbol are counted that many times). A string where all symbols are distinct is called *exemplar*. A *square* is a string of the form XX, in other words, a concatenation of two equal strings. A *substring* of X consists of some consecutive symbols of X, that is, we use the term substring in the strict sense.

A *tandem duplication (TD)* transforms a string of the form AXB into $AXXB$, that is, it inserts another copy of a substring X besides the existing occurrence of X. The *TD distance* of a string T from a string S is the minimum number k of TDs needed to transform S into T. We define $k = \infty$ if no such sequence of TDs exists. For clarity we state our problem formally:

Given: two strings S and T, where $|S| \leq |T|$.
Find: a sequence with a minimum number of TDs that turns S into T.

We use $n := |T|$ for the length of the target string T.

Short tandem repeats appear to be a type of mutations of particular interest in genomics. To quote from [10], "Short tandem repeat (STR) ... are abundant throughout the human genome, and specific repeat expansions may be associated with human diseases. ... Thus, the knowledge of the normal repeat ranges of STRs is critically important to determine pathogenicity of observed repeats in known STRs or to discover novel disease-relevant repeat expansions", and from [11], "Very short tandem

Peter Damaschke: Tandem Duplication Parameterized by the Length Difference, pp. 18–29.
Proceedings of PSC 2023, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-07206-6

repeats bear substantial genetic, evolutional, and pathological significance in genome analyses." For a given sample of genomic strings we may want to recognize whether some strings result from others by sequences of short TDs, in order to figure out normal and irregular amounts of TDs. Also note that k TDs of length at most some ℓ can expand a string by at most $d \leq k\ell$ symbols.

The computational complexity of computing TD distances is only partly understood. To begin with, NP-hardness has been shown only rather recently: Computing the TD distance is NP-hard even when the alphabet size is 5 [2] or when S is an exemplar string [9]. The problem parameterized by the number k of duplications is fixed-parameter tractable (FPT) for exemplar strings S [9]. The latter paper also mentions a simple XP algorithm for arbitrary strings S and T that runs in $O(n^{2k})$ time. It is based on the operation opposite to TD:

A *contraction* transforms a string of the form $AXXB$ into AXB. In the present paper we call AXB a *contraction result* of $AXXB$.

The aforementioned XP algorithm simply branches at most k times on all possible contractions, whose number is trivially bounded by $n^2/2$. Hence the number of different sequences of contractions is at most $(n^2/2)^k$.

1.2 Contributions

First we improve the XP algorithm from [9]. By using some combinatorics of periods in strings, the time is reduced by essentially a factor n^k.

In the main part of the paper we first present an FPT algorithm for minimizing the number of TDs, parameterized by the length difference of T and S. This parameter d may be practically motivated as mentioned above, and from the parameterized complexity point of view, d is just a natural parameter to start with.

Intuitively one would expect this problem to be in FPT, since a small length difference allows only a few short TDs, and together with the linear structure of strings it should be possible to apply dynamic programming on subsets or a related technique. The basic idea is to decide which symbols in T shall be kept (matched) or deleted (unmatched), and (if possible) to delete exactly the unmatched symbols by a minimum number of contractions. However, it is not so obvious how to do the "local" technical details in the most efficient way, as one must care about dependencies among overlapping squares.

We also construct a polynomial kernel, which needs even more effort. Part of the preprocessing is an acyclic directed grid graph that encodes all possible alignments of the input strings, including invalid ones, and whose geometry allows to derive data reduction rules.

We remark that our problem can be seen as a variant of string editing, with TDs as edit operations. It is well known that string editing problems with insertions, deletions, and replacements as edit steps can be easily rephrased as shortest path problems in a certain alignment graph. For TD minimization we have to use an alignment graph in a more elaborated way (where periods in strings play an essential role), indicating that the concept might prove useful also for other string editing variants.

We conclude the paper with open questions.

1.3 Periods

We provide some terminology concerning periods in strings. For a positive integer p, a string $R = r_1 \dots r_m$ is said to have a *period* p if $r_i = r_{i+p}$ holds for every index i with $1 \leq i < i + p \leq m$. A *p-run* in a string is a substring that has a period p and length at least $2p$, and is maximal with these properties, in the sense that adding another adjacent symbol would destroy the period p. Note that every substring of length $2p$ in a p-run is a square. A substring which is a p-run for some number p is also simply called a *run*, without specifying p. The *exponent* of a run R is its length $|R|$ divided by its shortest period; note that this is in general a fractional number.

1.4 Fixed-Parameter Tractability and Problem Kernels

Technical introductions to parameterized algorithms and complexity can be found in a number of textbooks. In a nutshell, a problem with input size n and another input parameter d is in XP and in FPT, if some algorithm can solve it in time $O(n^{f(d)})$ and $O(f(d) \cdot n^{O(1)})$, respectively, where f is some computable function.

Given an instance of a parameterized problem, a *kernel* is another instance with the following properties: It is equivalent to the given instance, in that it yields the same output, its size is bounded by some function of d, and it is computable from the given instance in a time being polynomial in n. A problem is in FPT if and only if it possesses a kernel. But the kernel size is not always polynomial in d.

The O^* notation for XP and FPT time bounds depending on the input size n and a parameter d suppresses factors that are polynomial in n (with a constant exponent not depending on d) such that it focusses on the more critical dependency on the parameter.

2 Improved Branching for Arbitrary Strings

Not very surprisingly, the brute-force bound in [9] is a bit of an overestimate. Our idea is to avoid this brute-force branching on all substrings and potential contractions, and to branch only on contractions that produce *different* strings, taking advantage of some nice properties of periods. We begin with some combinatorial lemma.

Lemma 1. *All contractions of squares of length $2p$ in a p-run yield the same contraction result.*

Proof. We choose a square XX of length $2p$ as indicated, and write the p-run accordingly as $AXXB$. Let $CAXXBD$ be the entire string. Note that each of A, B, C, D might be empty. The result $CAXBD$ of the contraction can be obtained by deleting the right X and moving BD by p positions to the left. (If BD is empty, nothing happens in this step.) Since p is a period of $AXXB$, the prefix $CAXB$ of the contraction result has, at every position, the same symbol as $CAXXB$ had, and the suffix D is always the same, independently of the position of XX. □

This gives already an improvement of the trivial $O(n^2)$ bound on the number of different contraction results:

Proposition 2. *A string of length n has at most $n(\ln n + 1)$ different contraction results.*

Proof. First we observe, for every p, that any two squares of length $2p$ that overlap in at least p positions belong to the same p-run. Namely, let $T = t_1 \dots t_n$ be a string, and let $t_{i+1} \dots t_{i+2p}$ and $t_{i+k+1} \dots t_{i+k+2p}$ be squares, with some positive integer $k \leq p$. We have to show $t_j = t_{j+p}$ for all j with $i+1 \leq j < j+p \leq i+k+2p$, or simpler, $i+1 \leq j \leq i+k+p$. For $j \leq i+p$ this holds because of the first square. For $j \geq i+k+1$ this holds because of the second square. Every j in our range satisfies at least one of these conditions, since $j \geq i+p+1$ implies $j \geq i+k+1$ by $k \leq p$.

To prove the actual assertion, consider any fixed positive integer $p \leq n/2$, and all squares of length $2p$ in the string. Whenever the left ends of two squares are at most p positions away, they belong to the same p-run, and by Lemma 1 they yield the same contraction result. By contraposition this can be formulated also in this way: Two squares of length $2p$ can yield different contraction results only if their left ends have a distance at least p. Hence at most n/p different such contraction results can exist. Finally, summing over all p yields the claim, using the well-known sum of the harmonic series. □

However, we can do even better, using deeper combinatorics of strings: In [4] it was shown that the sum of exponents of all runs in a string of length n is at most $4.1n$, thus improving linear upper bounds with larger constants from earlier papers. This further reduces the XP time bound considerably:

Theorem 3. *For a string T of length n and another string S we can find a sequence of at most k TDs that transform S into T (or report that none exist) in $O^*((2.05n)^k)$ time. Moreover, there exist at most $(2.05n)^k$ different sequences of strings of the form $S = R_1, R_2 \dots, R_j = T$, where $j \leq k$, and every R_{i+1} is obtained from R_i by some TD.*

Proof. Simply branch on all possible contraction results, in a search tree with T at the root and with depth k, keep only the contraction sequences that result in S. The branching factor, i.e., the base of the time bound, is the maximum number of different contraction results in a string.

Proposition 2 would already yield $O^*((n \log n)^k)$. For the better linear base, observe that the number of different periods $p \leq k/2$ of a run of length k is at most half its exponent. Lemma 1 can be rephrased in the way that the contractions of any squares of the same fixed length $2p$ in the run, where the numbers $p \leq k/2$ are periods of the run, yield the same contraction results. Hence the number of distinct contraction results overall is at most half the sum of the exponents of all runs. With the bound $4.1n$ [4] we obtain $2.05n$. □

Remark 4. The bound of $O(n)$ distinct squares ($2n$ in [6], later improved by several authors like [7,5] to eventually less than n in [1]) does not simply imply a time bound as in Theorem 3, because squares that have different locations but are equal as strings are not counted there as distinct, but they can produce different contraction results. A similar remark holds for the known result that every string has only $O(n)$ different runs [8,3]. The catch is that a p-run (as defined here) is also a q-run for all integer multiples q of p until the half length of the run, and squares of different lengths yield, of course, different contraction results. Therefore, the linear bound on the sum of exponents of the runs is needed.

Remark 5. Since the $O(n^{2k})$ bound in [9] was used after kernelization in their FPT result when S is an exemplar string and k is the parameter, Theorem 3 improves the

running time for solving this problem kernel accordingly. Probably Theorem 3 can also improve other related results.

3 Length Difference as Parameter

Next we consider our TD minimization problem parameterized by the length difference $d = |T| - |S|$. (In the rest of the paper, symbol d is reserved for the exact difference.) Since the number k of TDs that transform S into T trivially satisfies $k \leq d$, Theorem 3 yields an $O^*((2.05n)^d)$ time bound for finding a shortest sequence of TDs. However, below we will obtain an FPT algorithm for the parameter d. The idea is quite natural: Since most of the symbols are not involved in TDs, certain sub-instances of the problem, consisting of certain pairs of substrings of T and S, can be solved independently, while a dynamic programming process cares of separators of matching substrings.

Theorem 6. *For a string T of length n and another string S, with length difference $d = |T| - |S|$, we can find a sequence with a minimum number of TDs that transforms S into T (or report that none exist) in $O^*((2d)^d)$ time, more precisely, in $O(d^3(2d)^d n)$ time.*

The remainder of this section is devoted to the proof of Theorem 6. First some more preparations and definitions are needed. Remember that a substring consists of consecutive symbols in a string. Sometimes we use $r_1 \dots r_n$ with $n = 0$ to denote an empty string.

Lemma 7. *Consider any sequence of contractions of T that deletes d symbols in total. Then there exists a partitioning of T into substrings of length at most $2d$, such that every contracted square is entirely contained in one of these substrings.*

Proof. We call a symbol in T active when it participates in some contracted square (no matter whether it belongs to the deleted or undeleted half of that square). All other symbols are called inactive. Since d symbols are deleted in total, at most $2d$ symbols are active. (It could be fewer than $2d$ symbols when the contracted squares overlap.) Any maximal substring of active symbols is called a block.

Consider any subset Q of symbols in T that becomes a contracted square sometimes during our contraction sequence, and consider any inactive symbol u in T. Assume that symbols of Q appear both to the left and to the right of u. Since, by definition, inactive symbols are never deleted in a sequence of contractions, u remains present all the time, such that Q will never become a substring (i.e., consist of consecutive symbols), which contradicts the assumption that Q becomes a contracted square.

It follows that every contracted square is contained in some block. Now we simply make every block a substring of our claimed partitioning, and divide the inactive symbols arbitrarily into substrings of length at most $2d$. □

Dynamic programming function. Given $S = s_1 \dots s_m$ and $T = t_1 \dots t_n$, we define $c(i, j)$ to be the minimum number of contractions needed to transform $t_1 \dots t_j$ into $s_1 \dots s_i$, and $c(i, j) := \infty$ if no such transformation exists. To account for empty prefixes we also define $c(0, 0) = 0$.

Principle of optimality. Let i and j be any indices such that $c(i,j) < \infty$, and consider some corresponding optimal solution, i.e., minimum sequence of contractions. By Lemma 7 there exists some index $b \geq j - 2d$ such that for every contracted square Q, the symbols of Q are either all in $t_1 \dots t_b$ or all in $t_{b+1} \dots t_j$. Let e be the index such that $t_1 \dots t_b$ is transformed into $s_1 \dots s_e$. It follows that $c(i,j)$ is the sum of $c(e,b)$ and the minimum number of contractions needed to transform $t_{b+1} \dots t_j$ into $s_{e+1} \dots s_i$.

Computing the optimal values. Based on this observation, we now describe the computation of $c(i,j)$ for any given pair of indices (i,j). If $i > j$ then $c(i,j) = \infty$. Values for $i = j$ are also clear: If $t_1 \dots t_j = s_1 \dots s_j$ then $c(j,j) = 0$, else $c(j,j) = \infty$.

Due to the principle of optimality, we may proceed as follows. Try all possible index pairs (e,b), where $j - 2d \leq b < j$ and $e \leq b \leq e + d$. For every such pair, transform $t_{b+1} \dots t_j$ into $s_{e+1} \dots s_i$ using a minimum number of contractions. Add this number to $c(e,b)$. Finally, $c(i,j)$ is the minimum of these sums, for all index pairs. The mentioned index pairs (e,b) are sufficient, since $b \geq j - 2d$ was shown in the paragraph above, the inequalities $e \leq b \leq j$ are trivial by the definitions, and e can differ from b by at most d, since a string of length b can be contracted only to strings of length at least $b - d$ (and $c(e,b)$ would be infinite otherwise).

Complexity analysis. We check $O(d^2)$ pairs of indices (e,b). Now we bound the time needed to optimally transform $t_{b+1} \dots t_j$ into $s_{e+1} \dots s_i$. The length of the former string is at most $2d$ and the number k of contractions is bounded by d. The XP algorithm from [9] enumerates all possible sequences of contractions, and takes the shortest one.

A sequence of, say, k contractions can be uniquely specified, e.g., by the left ends and the lengths $l_1, \dots, l_k$ of the deleted substrings. For the left ends we have (rather generously) at most $(2d-1)^k$ choices. Since $l_1 + \dots + l_k \leq d$, the number of possible sequences of k lengths is $\binom{d}{k}$. Hence, by the binomial formula, the total number of choices for all $k \leq d$ is at most $((2d-1)+1)^d = (2d)^d$.

Since $j \leq n$ and $i \leq j \leq i + d$, we must compute $O(dn)$ values $c(i,j)$. (To see that only these values are needed, remember the definition of $c(i,j)$ and of the parameter d.) The product of the three terms above yields the claimed overall time bound. This concludes the proof of Theorem 6.

From the optimal values, a specific solution can be reconstructed by backtracing in the standard way.

Remark 8. Direct application of the trivial $O(n^{2k})$ bound from [9] to $n \leq 2d$ and $k \leq d$ would only yield $O^*((2d)^{2d})$ time, but another counting argument above gave $O^*((2d)^d)$, since $2d$ is close to d. Also note that the direct application of Theorem 3, that was made for general n and k, would yield $O^*((2.05d)^d)$ in our case, which is slightly worse. The picture would change with an improved bound on the sum of exponents of runs.

4 Kernelization

In this section we construct a kernel that is polynomial in the parameter $d = |T| - |S|$. Recall the notation $S = s_1 \dots s_m$ and $T = t_1 \dots t_n$. With the help of an alignment graph that depicts possible "paths of insertions" of symbols, we find, in any large

enough instance, either two matching substrings not involved in TDs, or periodic substrings whose deletion does not change the result of the instance.

4.1 Alignment Graph Construction

As a first step we define a directed graph G that we call the *alignment graph*. Its vertices are certain (but not all) pairs of integers (i, j) in the rectangle specified by $0 \leq i \leq m = |S|$ and $0 \leq j \leq n = |T|$. They can be imagined as grid points in a Cartesian coordinate system. Also note that $d = n - m$.

We will create certain directed edges of the form either $(i, j-1) \to (i, j)$ or $(i-1, j-1) \to (i, j)$, called *vertical* and *diagonal edges*, respectively.

Before specifying exactly which vertices and edges exist in G, we outline the idea behind the graph: Traversing $(i, j-1) \to (i, j)$ shall model the deletion of t_j, and traversing $(i-1, j-1) \to (i, j)$ shall model the action of matching symbol s_i to symbol t_j. Thus every solution with exactly d unmatched symbols corresponds to some directed path from $(0, 0)$ to (m, n) with at most d vertical edges. Of course, the converse of the last statement is far from being true. We also "purify" our graph by not creating some obviously useless vertices and edges. Now we describe the actual construction. See also Figure 1 for the position of the alignment graph in the i-j coordinate system.

Definition 9. *The alignment graph of two strings S and T is obtained as follows. We (tentatively) create all vertices (i, j) with $0 \leq i \leq j \leq i + d$, and all vertical and diagonal edges between them. Next, any diagonal edge $(i-1, j-1) \to (i, j)$ is retained only if $s_i = t_j$, and otherwise deleted. Finally we also delete all vertices (and all their incident edges) that cannot be reached from $(0, 0)$ or cannot reach (m, n) via directed paths.*

The construction can be done in $O(dn)$ time by standard techniques: First, the graph has obviously $O(dn)$ vertices and edges, respectively, and it is a directed acyclic graph. For every diagonal edge it is decided locally, by testing the equality of two symbols, whether it is retained or deleted. Finally, the vertices reachable from $(0, 0)$ or (reversely to the edge orientations) from (m, n) can be determined by breadth-first search in linear time.

In the so obtained alignment graph G, the directed paths from $(0, 0)$ to (m, n) describe exactly all possible alignments of S and T after deleting d symbols from T, but still without caring whether these deletions can be realized by contractions of squares.

The alignment graph has two special directed paths from $(0, 0)$ to (m, n) that we call the *left* and *right greedy path*, defined as follows.

Definition 10. *The left greedy path always traverses a vertical edge, and whenever the vertical edge from the current vertex does not exist, it traverses the diagonal edge instead. Similarly, the right greedy path always traverses a diagonal edge, and whenever the diagonal edge from the current vertex does not exist, it traverses the vertical edge instead.*

Note that the alternative edge always exists by construction, because G contains only vertices from which (m, n) is reachable. That is, we never get stuck. We also observe that all vertices of the alignment graph are in the region of the plane bounded

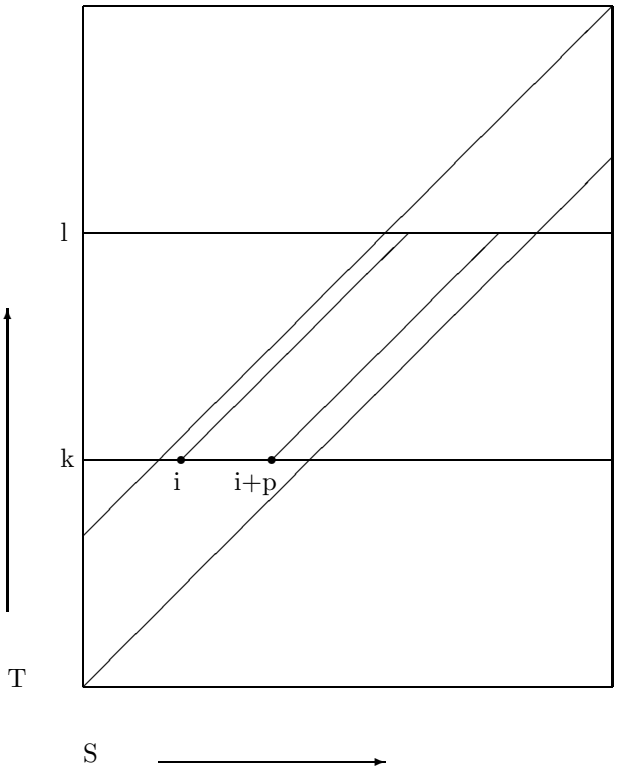

Figure 1. Diagram of the alignment graph. Its vertices are located in the region between the two long diagonal lines. The breadth of this stripe is d, both in horizontal and vertical direction. The diagram also depicts a pair of diagonal paths with coordinates as in Lemma 11.

by these two greedy paths. Otherwise some directed edge must leave this region, which would contradict the definition of the greedy paths.

We call any directed path merely consisting of diagonal edges a *diagonal path.*

4.2 Periods and Alignment Graph Properties

For convenience we say that a vertex (i, j) in the alignment graph is in column i and row j (see Figure 1).

Lemma 11. *Consider two rows k and l with $l-k > d$ in the alignment graph. Suppose that two (not necessarily distinct) directed paths from row k to row l are diagonal. Then either these paths are identical, or they have the form*

$$(i,k)\ldots(i+l-k,l) \text{ and } (i+p,k)\ldots(i+p+l-k,l)$$

for some integers i $(k-d \le i \le k)$ and p $(1 \le p \le d-(i-k))$, and in the latter case, both $t_{k+1}\ldots t_l$ and $s_{i+1}\ldots s_{i+p+l-k}$ have a period p.

Proof. We only have to show periodicity in the latter case. It follows directly from the condition for the existence of diagonal edges. Namely, for every q $(0 < q < p)$ we now obtain $s_{i+q} = t_{k+q} = s_{i+q+p} = t_{k+q+p} = s_{i+q+2p} = t_{k+q+2p} = \ldots$ (and so on in this way, as long as the indices are in the given interval), which yields the assertion. □

For formal clarity we need some further technical definitions.

A *valid sequence* for $T = t_1\ldots t_n$ and $S = s_1\ldots s_m$ means any sequence of contractions of squares that transforms T into S. Recall that every contracted square

has a length of at most $2d$. Without loss of generality we can assume that every contraction deletes the left half of the contracted square. With this convention, any valid sequence defines a partial function c from $\{t_1, \ldots, t_n\}$ onto $\{s_1, \ldots, s_m\}$, where $c(t_i) = s_j$ means that t_i is turned into s_j by the valid sequence, and $c(t_i)$ is undefined if t_i gets deleted by some contraction. This function can be naturally extended to substrings T^* and S^* of T and S, respectively: $c(T^*) = S^*$ means that T^* is turned into S^*. Note that this is possible only if $|T^*| = |S^*|$.

A *fresh symbol* is a symbol that does not yet appear in the strings at hand, that is, it may even extend the alphabet.

Lemma 12. *Let T^* be a substring of T, and let S^* be a substring of S, with the following properties:*

- $|T^*| = |S^*| \geq d + 2$.
- *In every valid sequence for T and S it holds that $c(T^*) = S^*$. (In particular, all symbols in T^* are matched symbols, in every valid sequence.)*

Next, we write T^* as a concatenation $T_1^*T_2^*$ where $|T_1^*| = d$, replace T_2^* (in T) with one fresh symbol f, and denote the resulting string T'. Similarly, we write S^* as a concatenation $S_1^*S_2^*$ where $|S_1^*| = d$, replace S_2^* (in S) with the same symbol f, and denote the resulting string S'.
Then, the valid sequences for T and S are exactly the same as the valid sequences for T' and S', in the sense that they contract the same squares, and in the same order.

Proof. The two directions of the equivalence are similar in structure, but they have to use slightly different arguments:

Consider any valid sequence for T and S. The assumption implies that every contracted square is completely to the right of T^* or contains at most the d leftmost symbols of T^*. Hence the same sequence is also valid for T' and S'.

Consider any valid sequence for T' and S'. Since f occurs only once in T' and S', it follows that f is a matched symbol, and f in T' is matched onto f in S'. Furthermore, every contracted square is completely to the right or to the left of f. Hence the same sequence is also valid for T and S. □

In the following we use a nice and simple property of periodic strings. Let X be any string that has a period p: Let us delete any substring P of length p and concatenate the two remaining substrings of X. Then the resulting string does not depend on the choice of P. Basically this was already stated in different phrasing in Lemma 1, but here we add a similar observation for the reverse operation: Let us choose any position between two neighbored symbols of X and insert there the suitable (and uniquely determined) string of length p that preserves periodicity. Then the resulting string does not depend on the choice of that position. We refer to these two operations as *shortening* and *enlarging* a p-periodic string X by p consecutive symbols.

We remark that, in the assumptions of the following lemma, C "starts d positions earlier" than D, while their end positions are the same. This is intended, as we need that to account for the deletion of up to d symbols from T until position $j + l$. Also remember that d denotes the exact difference $|T| - |S|$, not only an upper bound.

Lemma 13. *Suppose that the string T contains a substring $D = t_j \ldots t_{j+l}$ having a start position $j > d$, the length $l + 1 > d(d + 1) + 2d = d^2 + 3d$ and a period*

$p \le d$, and that the string S contains a substring $C = s_{j-d} \dots s_{j+l}$, also with period p. Then, shortening both D and C by p consecutive symbols yields an equivalent problem instance, that is, an instance with the same optimal number of contractions.

Proof. Consider any valid sequence for T and S. Since every contraction of a square of length $2q$ deletes q unmatched symbols, and d unmatched symbols exist in total, the (at most d) contracted squares together cover at most $2d$ symbols. There remain at least $d(d+1)$ symbols in D which are not covered by any contracted squares. We further observe that D can be uniquely partitioned into maximal substrings of covered symbols and uncovered symbols, respectively. Since at most d contractions are done, this partitioning has at most d substrings of covered symbols, hence at most $d+1$ substrings of uncovered symbols. It follows that some substring of uncovered symbols has a length at least d. In other words, D has some substring M of d symbols that are not involved in any contracted square. In particular, M is a substring of matched symbols.

The aforementioned substring M of T is matched onto a substring of S that we denote N. That is, $c(M) = N$. More specifically, N is a substring of C (since C starts at position $j-d$, and at most d symbols are deleted from D). Let us remove some substring P of length exactly $p \le d$ from M, and also remove the corresponding substring (its matching partner $c(P)$) from N. Let T' and S' denote the resulting strings, after these deletions from T and S, respectively. Then our valid sequence is also valid for T' and S', since M is not involved in any contraction.

Now remember the above definition of shortening and enlarging, and note that removing P from M and $c(P)$ from N means to shorten the p-periodic strings D and C, respectively, by p symbols. Similarly, we may insert, at corresponding positions in M and N, two equal substrings of length exactly p that preserve the period. The effect is that the p-periodic strings D and C are enlarged by p symbols. and our valid sequence is also valid for the resulting strings T' and S', for the same reason (M is not involved in any contraction).

Finally consider a valid sequence doing the optimal number k of contractions for transforming T into S. To summarize the observations above, shortening D and C yields a valid sequence with k contractions, furthermore, after this shortening there cannot exist another valid sequence with less than k contractions, since this would imply such a sequence also after enlarging D and C again (thus recovering the given T and S), which contradicts the minimality of k. This shows the assertion. □

4.3 The Kernel

Using the previous lemmas we can now finish up. Tandem duplication admits a kernel of size $O(d^3)$, or in more detail:

Theorem 14. *For a string T of length n and another string S, with length difference $d = |T| - |S|$, the problem of finding a sequence with a minimum number of TDs that transforms S into T (or report that none exist) has a kernel of size $O(d^3)$ that can be computed in $O(dn)$ time.*

Proof. We can assume $d > 3$ and $n > 6d^3$ (for simplicity with a generous constant factor), otherwise there is nothing to prove. For better orientation in the proof see Figure 1 again.

First we construct the alignment graph G. Since each of the two greedy paths uses d vertical edges, both greedy paths together can use at most $2d$ vertical edges.

Trivially, these $2d$ vertical edges $(i, j-1) \to (i, j)$ can use at most $2d$ different coordinates j on the T-axis, and these positions j cut the T-axis into at most $2d+1$ intervals. Furthermore, within these intervals, the greedy paths can use only diagonal edges. From these observations it follows the existence of an interval $D \subset [0, n]$ of length $n/(2d+1) > n/(3d)$ such that the sub-paths of both greedy paths restricted to the indices $j \in D$ are diagonal paths. By Lemma 11, either the two greedy paths restricted to D are identical, or T and S have aligned substrings with a period at most d and length $n/3d$. Since $n > 6d^3$, this length is larger than $2d^2 \geq d^2 + 4d > d + 2$.

In the former case, since G is bounded left and right by the two greedy paths, all alignments must use the greedy path restricted to D. Thus we are in the situation of Lemma 12, that is, we know two substrings of length at least $d+2$ that are necessarily matched to each other. As described there, using a fresh symbol we can shorten the instance to an equivalent one.

In the latter case we use Lemma 13 to shorten the instance to an equivalent one. (The substrings D and C specified there have lengths of at least $d^2 + 4d$.) This can be done only $O(n)$ times until $n \leq 6d^3$.

Thus we eventually obtain a kernel of size $O(d^3)$. As for the time for this kernelization, we first remark that polynomial time is evident. For the claimed specific time bound we observe: The initial alignment graph can be constructed in $O(dn)$ time, as noticed earlier, and the greedy paths are found in linear time. The shortening operations together take only linear (not quadratic) time as well, since they all cut away pairwise disjoint parts of the strings and the alignment graph, and updates after every shortening operation. are local □

A fully worked out substantial example would be lengthy, but a simple example just illustrating the principle could be a pair of strings like $S = aaaacacbbababababab$ and $T = aaaaacacbbababababababab$. The kernelization algorithm would recognize that *cacbb* must be mapped to *cacbb* and replace this pattern with a fresh symbol, and shorten the periodic substrings to the left and to the right of this pattern, by cutting out the same periods in both S and T, and the kernel would still reveal that some a and some ab must be doubled at the left and right end, in order to convert S into T.

In Lemma 12, for simplicity we did not care about the number of different fresh symbols. Substrings with more than d positions in between may reuse the same fresh symbols. Since at most d symbols of T are deleted in total, these remote occurrences of the same symbol would not interfere, and still the correct symbols would be matched to each other. Thus, extending the alphabet by only $O(d)$ different fresh symbols suffices. A technically more challenging question is whether one can avoid any extension of the alphabet, without sacrificing the kernel size, or at least lower the number of different fresh symbols further, e.g., by using carefully designed substrings instead of the fresh symbols.

5 Concluding Discussions

To our best knowledge, it is open whether the tandem duplication problem for arbitrary strings, parameterized by the number k of contractions, is fixed-parameter tractable (or perhaps W[1]-hard). A positive answer would imply our FPT result for parameter d, but even in that case, the question of complexity bounds would remain interesting. E.g., recall that the dependence of the time bound on d might be further improved.

We have studied the length difference d as parameter, but note that $k \leq d$, and a certain weakness of parameter d is that k can be arbitrarily smaller. A refinement worth investigating is the combined parameter (k, ℓ) where ℓ denotes the maximum length of substrings to be duplicated. Note that $d \leq k\ell$, hence our problem is in FPT also with parameter $k\ell$. Since trivially $\ell \leq d$, parameter ℓ alone would be much stronger than d, but the question whether tandem duplication parameterized by ℓ is in FPT seems to be as challenging as for k. Also a more fine-grained analysis in the parameter (k, ℓ) rather than d does not appear to be straightforward. But hardness results (provided that they hold) might be easier to prove for stronger parameters.

Finally, in our results we have not fixed the alphabet size. Would fixed alphabet sizes allow stronger time bounds? Can one construct a kernel without extending the alphabet?

Acknowledgements

Part of the algorithmic ideas have been developed during the supervision of the master's thesis project of Belmin Dervisevic and Mateo Raspudic. I would like to thank the former students for inspiring discussions, an anonymous reviewer of an earlier version for drawing my attention to exponents of runs, and the anonymous reviewers at PSC for several small corrections.

References

1. S. Brlek and S. Li: *On the number of distinct squares in finite sequences: Some old and new results*, in Combinatorics on Words - 14th International Conference, WORDS 2023, Umeå, Sweden, June 12-16, 2023, Proceedings, A. E. Frid and R. Mercas, eds., vol. 13899 of Lecture Notes in Computer Science, Springer, 2023, pp. 35–44.
2. F. Cicalese and N. Pilati: *The tandem duplication distance problem is hard over bounded alphabets*, in Combinatorial Algorithms - 32nd International Workshop, IWOCA 2021, Ottawa, ON, Canada, July 5-7, 2021, Proceedings, P. Flocchini and L. Moura, eds., vol. 12757 of Lecture Notes in Computer Science, Springer, 2021, pp. 179–193.
3. M. Crochemore, L. Ilie, and W. Rytter: *Repetitions in strings: Algorithms and combinatorics*. Theor. Comput. Sci., 410(50) 2009, pp. 5227–5235.
4. M. Crochemore, M. Kubica, J. Radoszewski, W. Rytter, and T. Walen: *On the maximal sum of exponents of runs in a string*. J. Discrete Algorithms, 14 2012, pp. 29–36.
5. A. Deza, F. Franek, and A. Thierry: *How many double squares can a string contain?* Discret. Appl. Math., 180 2015, pp. 52–69.
6. A. S. Fraenkel and J. Simpson: *How many squares can a string contain?* J. Comb. Theory, Ser. A, 82(1) 1998, pp. 112–120.
7. L. Ilie: *A note on the number of squares in a word*. Theor. Comput. Sci., 380(3) 2007, pp. 373–376.
8. R. M. Kolpakov and G. Kucherov: *Finding maximal repetitions in a word in linear time*, in 40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA, IEEE Computer Society, 1999, pp. 596–604.
9. M. Lafond, B. Zhu, and P. Zou: *Computing the tandem duplication distance is np-hard*. SIAM J. Discret. Math., 36(1) 2022, pp. 64–91.
10. Q. Liu, Y. Tong, and K. Wang: *Genome-wide detection of short tandem repeat expansions by long-read sequencing*. BMC Bioinform., 21-S(21) 2020, p. 542.
11. H. Yu, S. Zhao, S. Ness, H. Kang, Q. Sheng, D. C. Samuels, O. Oyebamiji, Y. Zhao, and Y. Guo: *Non-canonical RNA-DNA differences and other human genomic features are enriched within very short tandem repeats*. PLoS Comput. Biol., 16(6) 2020.