Comparing Degenerate Strings

Mai Alzamel, Lorraine a K Ayad, Giulia Bernardini, Roberto Grossi, Costas S Iliopoulos, Nadia Pisanti, Solon P Pissis, Giovanna Rosone

Fundamenta Informaticae XX (2020) 41–59 41 *DOI 10.3233/FI-2020-1947 IOS Press*

Comparing Degenerate Strings*

Mai Alzamel
Department of Informatics, King's College London, UK
Department of Computer Science, King Saud University, KSA
mai.alzamel@kcl.ac.uk

Lorraine A.K. Ayad
Department of Informatics, King's College London, UK
lorraine.ayad@kcl.ac.uk

Giulia Bernardini
Department of Computer Sciences, Systems and Communication, University of Milano - Bicocca, Italy
giulia.bernardini@unimib.it

Roberto Grossi
Department of Computer Science, University of Pisa, Italy
grossi@di.unipi.it

Costas S. Iliopoulos
Department of Informatics, King's College London, UK
costas.iliopoulos@kcl.ac.uk

Nadia Pisanti
Department of Computer Science, University of Pisa, Italy
pisanti@di.unipi.it

Solon P. Pissis
CWI Amsterdam, the Netherlands
solon.pissis@cwi.nl

Giovanna Rosone
Department of Computer Science, University of Pisa, Italy
giovanna.rosone@unipi.it

*The final publication is available at IOS Press through `https://doi.org/10.3233/FI-2020-1947`. Please cite: Mai Alzamel, Lorraine A. K. Ayad, Giulia Bernardini, Roberto Grossi, Costas S. Iliopoulos, Nadia Pisanti, Solon P. Pissis, Giovanna Rosone, Comparing Degenerate Strings. (2020) Fundamenta Informaticae, 175(1-4): 41-58.

Abstract. Uncertain sequences are compact representations of sets of similar strings. They highlight common segments by collapsing them, and explicitly represent varying segments by listing all possible options. A generalized degenerate string (GD string) is a type of uncertain sequence. Formally, a GD string $\hat{S}$ is a sequence of n sets of strings of total size N, where the ith set contains strings of the same length k_i but this length can vary between different sets. We denote by W the sum of these lengths $k_0, k_1, \ldots, k_{n-1}$. Our main result is an $\mathcal{O}(N + M)$-time algorithm for deciding whether two GD strings of total sizes N and M, respectively, over an integer alphabet, have a non-empty intersection. This result is based on a combinatorial result of independent interest: although the intersection of two GD strings can be exponential in the total size of the two strings, it can be represented in *linear* space. We then apply our string comparison tool to devise a simple algorithm for computing all palindromes in $\hat{S}$ in $\mathcal{O}(\min\{W, n^2\}N)$-time. We complement this upper bound by showing a similar conditional lower bound for computing maximal palindromes in $\hat{S}$. We also show that a result, which is essentially the same as our string comparison linear-time algorithm, can be obtained by employing an automata-based approach.

Keywords: degenerate strings, generalized degenerate strings, elastic-degenerate strings, string comparison, palindromes

1. Introduction

Uncertain sequences are useful for representing sets of similar strings in a compact form. They highlight common segments by collapsing them, and explicitly represent varying segments by listing all possible options. A *degenerate string* (also known as indeterminate string) over an alphabet Σ is a sequence of subsets of Σ. A great deal of research has been conducted on this type of uncertain sequence (see [1, 2, 3, 4, 5, 6] and references therein).

A more general definition of degenerate strings, with the aim of representing multiple closely-related sequences [7], was introduced in [8]: an *elastic-degenerate string* (ED string) $\tilde{S}$ over Σ is a sequence of subsets of Σ^* (see also network expressions [9]). That is, any set of $\tilde{S}$ does not contain, in general, only single letters, nor substrings of the same length, as it may contain strings of different lengths, including the empty string (see Figure 1). In a few recent papers, several algorithms for pattern matching on ED strings have been presented; specifically, for finding all exact [8, 10, 11, 12, 13] or approximate [14, 15] occurrences of a standard string pattern in an ED text.

$$\tilde{S} = \left\{ \begin{matrix} \mathtt{A} \\ \varepsilon \end{matrix} \right\} \cdot \left\{ \begin{matrix} \mathtt{GC} \\ \mathtt{A} \end{matrix} \right\} \cdot \left\{ \begin{matrix} \mathtt{CTC} \\ \mathtt{GCTT} \\ \mathtt{C} \\ \varepsilon \end{matrix} \right\} \cdot \left\{ \begin{matrix} \mathtt{CTA} \\ \mathtt{GAGA} \\ \mathtt{GT} \end{matrix} \right\} \cdot \{ \mathtt{T} \}$$

Figure 1. Example of an ED string.

In [16], the preliminary version of this paper, we introduced another special type of uncertain sequence called a generalized degenerate string; this can be viewed as an extension of degenerate

```
CA--AGCTCTATCTCGTA--TT        AGCTCTATCTCG
C---AGCCGAAGCTCGTATATT        AGCCGAAGCTCG
CATCAAGTCAACGCAG----TT        AAGTCAACGCAG
```

Figure 2. Multiple sequence alignment (left) and Local Gapless Alignment (right).

$$\hat{S} = \{\mathtt{A}\} \cdot \left\{ \begin{matrix} \mathtt{GC} \\ \mathtt{AG} \end{matrix} \right\} \cdot \left\{ \begin{matrix} \mathtt{TCT} \\ \mathtt{CGA} \\ \mathtt{TCA} \end{matrix} \right\} \cdot \{\mathtt{A}\} \cdot \left\{ \begin{matrix} \mathtt{TCTC} \\ \mathtt{GCTC} \\ \mathtt{CGCA} \end{matrix} \right\} \cdot \{\mathtt{G}\}$$

Figure 3. GD string obtained from the local gapless alignment of Figure 2.

strings or as a restricted variant of ED strings. Formally, a *generalized degenerate string* (GD string) $\hat{S}$ over Σ is a sequence of n sets of strings of total size N over Σ, where the ith set contains strings of the same length $k_i > 0$ but this length can vary between different sets. We denote the sum of these lengths $k_0, k_1, \ldots, k_{n-1}$ by W. A GD string can be used to represent in a compact form a *gapless* multiple sequence alignment (MSA) of fixed width, that is, for example, a high-scoring local alignment of multiple sequences (see Figure 2 and Figure 3).

In this paper we solve the problem of comparing two GD strings, that is, deciding whether two GD strings have a non-empty intersection. String comparison is the core computational task in several string-processing applications, whether they process standard or uncertain strings. For example, to extract frequent patterns from a single string, to find common substrings among several strings, or to check whether a string is palindromic, one must have a tool for comparing two strings. Therefore, we first develop an efficient algorithm to compare two GD strings, and then, as proof of concept, we apply this algorithm to compute all palindromes in a GD string.

In a standard string, a palindrome is a sequence that reads the same from left to right and from right to left. Detection of palindromic factors in texts is a classical and well-studied problem in algorithms on strings and combinatorics on words with a lot of variants arising out of different practical scenarios. A string $X = X[0]X[1]\ldots X[n-1]$ is said to have an initial palindrome of length k if its prefix of length k is a palindrome. Manacher first discovered an on-line algorithm that finds all initial palindromes in a string [17]. Later Apostolico et al. observed that the algorithm given by Manacher is actually able to find all maximal palindromic factors in the string in $\mathcal{O}(n)$ time [18]. Gusfield gave an off-line linear-time algorithm to find all maximal palindromes in a string and also discussed the relation between biological sequences and gapped palindromes [19].

With uncertain sequences, we first need to have an algorithm for efficient *string comparison*, where automata provide the following baseline. Let $\hat{X}$ and $\hat{Y}$ be two GD (or two ED) strings of total sizes N and M, respectively. We first construct the non-deterministic finite automaton (NFA) A of $\hat{X}$ and the NFA B of $\hat{Y}$ in time $\mathcal{O}(N + M)$. We then construct the product NFA C such that

$L(C) = L(A) \cap L(B)$ in time $\mathcal{O}(NM)$. The non-emptiness decision problem, namely, checking if $L(C) \neq \emptyset$, is decidable in time linear in the size of C, using breadth-first search (BFS). Hence the comparison of $\hat{X}$ and $\hat{Y}$ can be done in time $\mathcal{O}(NM)$. It is known that if there existed faster methods for obtaining the automata intersection, then significant improvements would be implied to many long standing open problems [20]. Hence an immediate reduction to the problem of NFA intersection does not particularly help. For GD strings, specifically, we show that we can construct an ad-hoc deterministic finite automaton (DFA) for $\hat{X}$ and $\hat{Y}$, so that the intersection can be performed efficiently, but this simple solution cannot achieve $\mathcal{O}(N + M)$ time for integer alphabets as its cost is alphabet-dependent.

Our Contribution. Our first result is an $\mathcal{O}(N + M)$-time algorithm for deciding whether the intersection of two GD strings of sizes N and M, respectively, over a constant-sized alphabet is non-empty. We show this by means of DFAs (see Section 3). This result is based on a combinatorial result of independent interest: although the intersection of two GD strings can be exponential in the total size of the two strings, it can be represented in linear space. We next present an efficient implementation of this result in the standard word RAM model with word size $w = \Omega(\log(N + M))$ that works also for integer alphabets. Specifically, we show an $\mathcal{O}(N + M)$-time algorithm for deciding whether the intersection of two GD strings of sizes N and M, respectively, over an integer alphabet is non-empty (see Section 4). We then apply our string comparison tool to compute palindromes in GD strings. We present a simple $\mathcal{O}(\min\{W, n^2\}N)$-time algorithm for computing all palindromes in $\hat{S}$; a proof-of-concept experiment is also presented (see Section 5). Notably, we complement this upper bound with a non-trivial $\Omega(n^2|\Sigma|)$ lower bound under the Strong Exponential Time Hypothesis [21, 22] for computing all maximal palindromes in $\hat{S}$ (see Section 6). Let us remark that there exists an infinite family of GD strings over an integer alphabet of size $|\Sigma| = \Theta(N)$ on which our algorithm requires time $\mathcal{O}(n^2N)$, thus matching the conditional lower bound. We conclude this paper with some open problems (see Section 7).

A preliminary version of this paper appeared in [16].

2. Preliminaries

A *string* X is a sequence of elements on an alphabet Σ, where the *alphabet* Σ is a non-empty finite set of letters of size $\sigma = |\Sigma|$. The set of all finite strings over an alphabet Σ, including the *empty string* ε of length 0, is denoted by Σ^*. The set of all strings of length $k > 0$ over Σ is denoted by Σ^k. For any string X, we denote by $X[i\,..\,j]$ the *substring* or *factor* of X that *starts* at position i and *ends* at position j. In particular, $X[0\,..\,j]$ is the *prefix* of X that ends at position j, and $X[i\,..\,|X| - 1]$ is the *suffix* of X that starts at position i, where $|X|$ denotes the *length* of X. The *suffix tree* of X (*generalized suffix tree* for a set of strings) is a compact trie representing all suffixes of X. We denote the *reversal* of X by string X^R, i.e., $X^R = X[|X| - 1]X[|X| - 2] \ldots X[0]$.

We say that a string P is a *palindrome* if and only if $P = P^R$. If factor $X[i\,..\,j]$, $0 \leq i \leq j \leq n - 1$, of string X of length n is a palindrome, then $\frac{i+j}{2}$ is the *center* of $X[i\,..\,j]$ in X and $\frac{j-i+1}{2}$ is the *radius* of $X[i\,..\,j]$. In other words, a palindrome is a string that reads the same forward and backward, i.e., a string P is a palindrome if $P = YaY^R$ where Y is a string, Y^R is the reversal of Y and a is either a single letter (when the center is an integer) or the empty string (when it is not). Moreover, $X[i\,..\,j]$ is

called a *palindromic factor* of X. It is said to be a *maximal palindrome* if there is no other palindrome in X with center $\frac{i+j}{2}$ and larger radius. Hence X has exactly $2n-1$ maximal palindromes. A maximal palindrome P of X can be encoded as a pair (c, r), where c is the center of P in X and r is the radius of P.

Definition 2.1. A *generalized degenerate string (GD string)* $\hat{S} = \hat{S}[0]\hat{S}[1]\dots\hat{S}[n-1]$ of *length* n over an alphabet Σ is a finite sequence of n degenerate letters. Every *degenerate letter* $\hat{S}[i]$ of *width* $k_i > 0$, denoted also by $w(\hat{S}[i])$, is a finite non-empty set of strings such that $\hat{S}[i][0], \dots, \hat{S}[i][|\hat{S}[i]|-1] \in \Sigma^{k_i}$. For any GD string $\hat{S}$, we denote by $\hat{S}[i]\dots\hat{S}[j]$ the *GD substring* of $\hat{S}$ that starts at position i and ends at position j.

In this work, we generally consider GD strings over an *integer alphabet* of size $\sigma = N^{\mathcal{O}(1)}$. We now define some parameters that describe the structure of a GD string.

Definition 2.2. The *total size* N and *total width* W, denoted also by $w(\hat{S})$, of a GD string $\hat{S}$ are respectively defined as $N = \sum_{i=0}^{n-1} |\hat{S}[i]| \cdot k_i$ and $W = \sum_{i=0}^{n-1} k_i$.

Example 2.3. The GD string $\hat{S}$ of Figure 3 has length $n=6$, size $N=28$, and $W=12$.

Definition 2.4. Given two degenerate letters $\hat{X}$ and $\hat{Y}$, their *Cartesian concatenation* is

$$\hat{X} \otimes \hat{Y} = \{xy \mid x \in \hat{X}, y \in \hat{Y}\}.$$

When $\hat{Y} = \emptyset$ (resp. $\hat{X} = \emptyset$) we set $\hat{X} \otimes \hat{Y} = \hat{X}$ (resp. $= \hat{Y}$). Notice that $\otimes$ is associative.

Definition 2.5. Consider a GD string $\hat{S}$ of length n. The *language of* $\hat{S}$ is

$$L(\hat{S}) = \hat{S}[0] \otimes \hat{S}[1] \otimes \cdots \otimes \hat{S}[n-1].$$

Our goal for GD string comparison is to establish, given two GD strings $\hat{R}$ and $\hat{S}$, whether the intersection of their language is non-empty. To this purpose, we define the notions of *chop* and *active suffixes*.

Definition 2.6. Let $\hat{X} = \{ x_i \in \Sigma^k \}$ and $\hat{Y} = \{ y_j \in \Sigma^h \}$ be two degenerate letters on alphabet Σ. Further let us assume without loss of generality that $w(\hat{Y}) < w(\hat{X})$ (i.e., $h < k$). We define the set *chop of* $\hat{X}$ *and* $\hat{Y}$ and the set *active suffixes of* $\hat{X}$ *and* $\hat{Y}$ as follows:

- $\text{chop}_{\hat{X},\hat{Y}} = \{ y_j \in \hat{Y} \mid y_j \text{ matches a prefix of some } x_i \in \hat{X} \}$
- $\text{active}_{\hat{X},\hat{Y}} = \{ x_i[h\,..\,k-1] \mid x_i[0\,..\,h-1] \in \text{chop}_{\hat{X},\hat{Y}} \}$

Let $w(\text{chop}_{\hat{X},\hat{Y}}) = \min\{w(\hat{X}), w(\hat{Y})\}$. When $\text{active}_{\hat{X},\hat{Y}} = \{\varepsilon\}$, we set $\text{active}_{\hat{X},\hat{Y}} = \emptyset$. We then have that $\text{active}_{\hat{X},\hat{Y}} = \emptyset$ either if $h=k$ or if there is no match between any of the strings in $\hat{Y}$ and the prefix of a string in $\hat{X}$; i.e., $\text{chop}_{\hat{X},\hat{Y}} = \emptyset$.

Example 2.7. (for Definition 2.6)
Consider the following degenerate letters $\hat{X}$ and $\hat{Y}$ where $w(\hat{Y}) < w(\hat{X})$. The underlined strings in letter $\hat{Y}$ are prefixes of strings in letter $\hat{X}$, hence they are in $\text{chop}_{\hat{X},\hat{Y}}$. The suffixes of such strings in $\hat{X}$ are the active suffixes in $\text{active}_{\hat{X},\hat{Y}}$.

$$\hat{X} = \begin{Bmatrix} \underline{\texttt{TCC}}\texttt{TA} \\ \texttt{ATCGA} \\ \underline{\texttt{TCC}}\texttt{AC} \\ \underline{\texttt{CAT}}\texttt{TA} \end{Bmatrix} \quad \hat{Y} = \begin{Bmatrix} \texttt{GCA} \\ \underline{\texttt{CAT}} \\ \underline{\texttt{TCC}} \end{Bmatrix} \quad \text{chop}_{\hat{X},\hat{Y}} = \{\texttt{CAT}, \texttt{TCC}\} \quad \text{active}_{\hat{X},\hat{Y}} = \{\texttt{TA}, \texttt{AC}\}$$

Definition 2.8. Let $\hat{R}$ and $\hat{S}$ be two GD strings of length r and s, respectively. $\hat{R}[0] \dots \hat{R}[i]$ is the *prefix* of $\hat{R}$ that ends at position i. It is called *proper* if $i \neq r-1$. We say that $\hat{R}[0] \dots \hat{R}[i]$ is *synchronized* with $\hat{S}[0] \dots \hat{S}[j]$ if $w(\hat{R}[0] \dots \hat{R}[i]) = w(\hat{S}[0] \dots \hat{S}[j])$. We call these the *shortest synchronized prefixes* of $\hat{R}$ and $\hat{S}$, respectively, when for all i', j' such that $i' < i$ and $j' < j$, we have that $w(\hat{R}[0] \dots \hat{R}[i']) \neq w(\hat{S}[0] \dots \hat{S}[j'])$.

Example 2.9. (for Definition 2.8)
The GD string $\hat{S}$ of Figure 3 and the GD string $\hat{R}$ below have no synchronized proper prefixes, while $\hat{S}$ and the GD string $\hat{T}$ below have synchronized prefix of length 1 and 6 (because $w(\hat{S}[0]) = w(\hat{T}[0]) = 1$ and $w(\hat{S}[0]\hat{S}[1]\hat{S}[2]) = w(\hat{T}[0]\hat{T}[1]) = 6$) , and thus their shortest synchronized prefix has length 1.

$$\hat{R} = \begin{Bmatrix} \texttt{CGCAC} \\ \texttt{AGCCG} \\ \texttt{AGCCG} \end{Bmatrix} \cdot \begin{Bmatrix} \texttt{AAT} \\ \texttt{TAG} \end{Bmatrix} \cdot \begin{Bmatrix} \texttt{CTCG} \\ \texttt{GCAG} \\ \texttt{CTCA} \end{Bmatrix} \qquad \hat{T} = \begin{Bmatrix} \texttt{C} \\ \texttt{A} \end{Bmatrix} \cdot \begin{Bmatrix} \texttt{CCACT} \\ \texttt{GCCCA} \\ \texttt{TCCTT} \end{Bmatrix} \cdot \begin{Bmatrix} \texttt{AT} \\ \texttt{AG} \end{Bmatrix} \cdot \begin{Bmatrix} \texttt{ATCG} \\ \texttt{AGCT} \\ \texttt{GGCA} \end{Bmatrix}$$

3. GD String Comparison for Small Alphabets Using Automata

In this section we describe a simple algorithm for GD string comparison that is based on DFAs. It works in linear time for constant-sized alphabets, and it is generalized in the subsequent section to work for integer alphabets.

Given $\hat{R}$ and $\hat{S}$ of total size N and M, respectively, each degenerate letter of $\hat{R}$ and $\hat{S}$ can be represented by a trie, where its leaves are collapsed to a single one. For every two consecutive degenerate letters, the collapsed leaves of the former trie coincide with the root of the latter trie. An acyclic DFA is obtained in this way, as illustrated in Example 3.1 below. We can perform the comparison of $\hat{R}$ and $\hat{S}$ by intersecting their corresponding DFAs using BFS on their product DFA. The trivial upper bound on the number of reachable states is $\mathcal{O}(NM)$, and the traditional method for constructing the DFA for $L(\hat{R}) \cap L(\hat{S})$ is non-linear, but this can be improved to $\mathcal{O}(N+M)$ by exploiting the structure of the two input DFAs. Each state in such a DFA has a unique level: the common length of paths from the initial state. This structure is *inherited* by the product DFA. In other words, a level-i state in the product DFA corresponds to a pair of level-i states in the input DFAs. Observe that a level-i state in one DFA is uniquely represented by the label of the path from the root of its trie, and for a fixed DFA

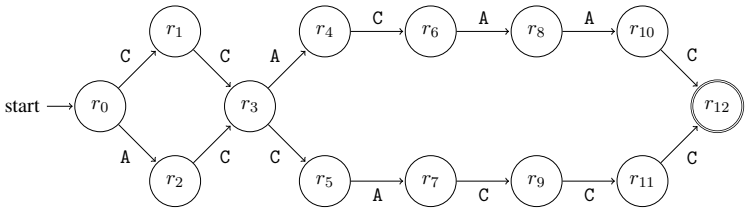


Figure 4. The DFA for $L(\hat{R})$

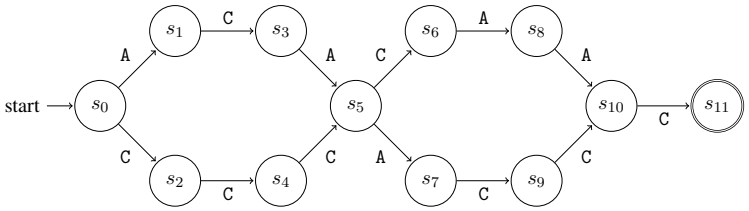


Figure 5. The DFA for $L(\hat{S})$

and level, these labels have uniform lengths. Considering the two states composing a reachable state in the product DFA, it is easy to see that the shorter label must be a suffix of the longer label. Hence, the state in the DFA with longer labels at level i uniquely determines the state in the DFA with shorter labels at level i. Consequently, the number of reachable level-i states in the product DFA is bounded by the number of level-i states in the input DFAs, and the size is $\mathcal{O}(N + M)$.

Note that computing the product DFA is alphabet-dependent, due to branching (transition function) on the same letter in the states of the two input DFAs.

Example 3.1. Say we want to compare the following two GD strings:

$$\hat{R} = \begin{Bmatrix} \mathtt{A\,C} \\ \mathtt{C\,C} \end{Bmatrix} \cdot \begin{Bmatrix} \mathtt{A\,C\,A\,A\,C} \\ \mathtt{C\,A\,C\,C\,C} \end{Bmatrix} \qquad \hat{S} = \begin{Bmatrix} \mathtt{A\,C\,A} \\ \mathtt{C\,C\,C} \end{Bmatrix} \cdot \begin{Bmatrix} \mathtt{A\,C\,C} \\ \mathtt{C\,A\,A} \end{Bmatrix} \cdot \{\mathtt{C}\}$$

We start by constructing the DFA for $L(\hat{R})$ and the DFA for $L(\hat{S})$, shown in Figure 4 and Figure 5, respectively. We then construct their product DFA, shown in Figure 6, which gives the intersection of $L(\hat{R})$ and $L(\hat{S})$: `ACACAAC` and `CCCACCC`.

4. GD String Comparison for Integer Alphabets

In this section we describe a more general algorithm for the problem of GD string comparison addressed also in Section 3. Let $\hat{R}$ and $\hat{S}$ be of total size N and M, respectively. We provide an $\mathcal{O}(N + M)$-time algorithm in the standard word RAM model with word size $w = \Omega(\log(N + M))$ to decide whether the intersection of $\hat{R}$ and $\hat{S}$ is non-empty, which also works for integer alphabets.

We show that, even if the size of $L(\hat{R}) \cap L(\hat{S})$ can be exponential in the total sizes of $\hat{R}$ and $\hat{S}$ (Fact 4.1), the problem of GD string comparison, i.e., deciding whether $L(\hat{R}) \cap L(\hat{S})$ is non-empty, can

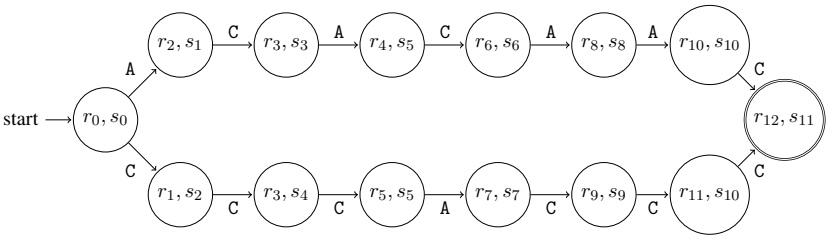


Figure 6. The product DFA for $L(\hat{R}) \cap L(\hat{S})$

be solved in time linear with respect to the sum of the total sizes of the two GD strings (Theorem 4.10) and is thus of independent interest.

Fact 4.1. Given two GD strings $\hat{R}$ and $\hat{S}$, $L(\hat{S}) \cap L(\hat{R})$ can have size exponential in the total sizes of $\hat{R}$ and $\hat{S}$. A trivial example of this is given when $\hat{R} = \hat{S}$ and they are a sequence of n degenerate letters of, for example, 2 strings of 2 characters each: the total size of both $\hat{R}$ and $\hat{S}$ is $4n$, while $|L(\hat{S}) \cap L(\hat{R})| = |L(\hat{R})| = 2^n$.

We next show when it is possible to factorize $L(\hat{R}) \cap L(\hat{S})$ into a Cartesian concatenation.

Lemma 4.2. Consider two GD strings $\hat{S} = \hat{S}'\hat{S}''$ and $\hat{R} = \hat{R}'\hat{R}''$ such that $w(\hat{S}) = w(\hat{R})$. If $\hat{S}'$ is synchronized with $\hat{R}'$, then $L(\hat{R}) \cap L(\hat{S}) = (L(\hat{R}') \cap L(\hat{S}')) \otimes (L(\hat{R}'') \cap L(\hat{S}''))$.

By applying Lemma 4.2 wherever $\hat{R}$ and $\hat{S}$ have synchronized prefixes, we are then left with the problem of intersecting GD strings with no synchronized proper prefixes. We now define an alternative decomposition within such strings (see also Example 4.4).

Definition 4.3. Let $\hat{R}$ and $\hat{S}$ be two GD strings of length r and s, respectively, with no synchronized proper prefixes. We define

$$\text{c-chain}(\hat{R}, \hat{S}) = \max_q\{0 \le q \le r + s - 2 \mid \text{chop}_q \neq \emptyset\},$$

where chop_i denotes the set $\text{chop}_{\hat{A}_i, \hat{B}_i}$, and $(\hat{A}_0, \hat{B}_0), (\hat{A}_1, \hat{B}_1), \ldots, (\hat{A}_q, \hat{B}_q)$, $\text{pos}(\hat{A}_i), \text{pos}(\hat{B}_i)$ are recursively defined as follows:

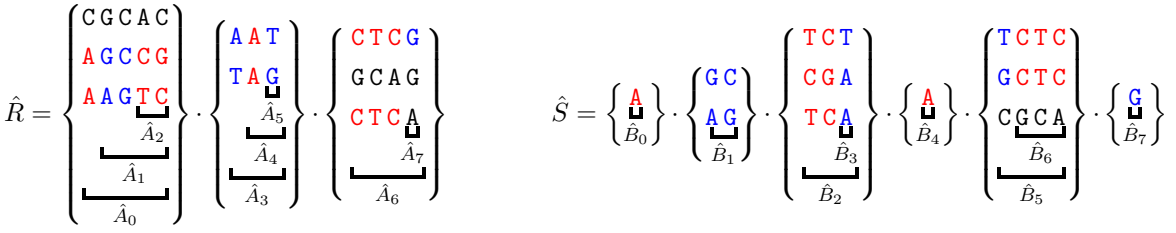
$\hat{A}_0 = \hat{R}[0], \hat{B}_0 = \hat{S}[0]$, and $\text{pos}(\hat{A}_0) = \text{pos}(\hat{B}_0) = 0$. For $0 < i \le r + s - 2$, if $\text{chop}_{i-1} \neq \emptyset$,

$$\hat{A}_i = \begin{cases} \hat{R}[\text{pos}(\hat{A}_{i-1}) + 1] \text{ and } \text{pos}(\hat{A}_i) = \text{pos}(\hat{A}_{i-1}) + 1 & \text{if } w(\text{chop}_{i-1}) = w(\hat{A}_{i-1}) \\ \text{active}_{\hat{A}_{i-1}, \hat{B}_{i-1}} \text{ and } \text{pos}(\hat{A}_i) = \text{pos}(\hat{A}_{i-1}) & \text{otherwise} \end{cases}$$

$$\hat{B}_i = \begin{cases} \hat{S}[\text{pos}(\hat{B}_{i-1}) + 1] \text{ and } \text{pos}(\hat{B}_i) = \text{pos}(\hat{B}_{i-1}) + 1 & \text{if } w(\text{chop}_{i-1}) = w(\hat{B}_{i-1}) \\ \text{active}_{\hat{A}_{i-1}, \hat{B}_{i-1}} \text{ and } \text{pos}(\hat{B}_i) = \text{pos}(\hat{B}_{i-1}) & \text{otherwise} \end{cases}$$

The generation of pairs $(\hat{A}_i, \hat{B}_i)$ stops at $i = q$ either if $q = r+s-2$, or when $\text{chop}_{q+1} = \emptyset$, in which case $\hat{R}$ and $\hat{S}$ only match until $(\hat{A}_q, \hat{B}_q)$. Intuitively, $\hat{A}_i$ (respectively, $\hat{B}_i$) represents suffixes of the current position of $\hat{R}$ (respectively, of $\hat{S}$), while $\text{pos}(\hat{B}_i)$ (respectively, $\text{pos}(\hat{A}_i)$) tells *which* position of $\hat{R}$ (respectively, $\hat{S}$) we are chopping.

Example 4.4. (for Definition 4.3)
Consider the following GD strings $\hat{R}$ and $\hat{S}$ with no synchronized proper prefixes: chop_0 is the first red set from the left, chop_1 is the first blue one, chop_2 is the second red one, etc. The c-chain$(\hat{R}, \hat{S})$ terminates when $q = 7$.



Definition 4.5. Let $\hat{R}$ and $\hat{S}$ be two GD strings of length r and s, respectively, with $w(\hat{R}) = w(\hat{S})$ and no synchronized proper prefixes. We define $G_{\hat{R},\hat{S}}$ as a directed acyclic graph with a structure of up to $r + s - 1$ levels, each node being a set of strings, as follows, where we assume without loss of generality that $w(\hat{R}[0]) > w(\hat{S}[0])$:

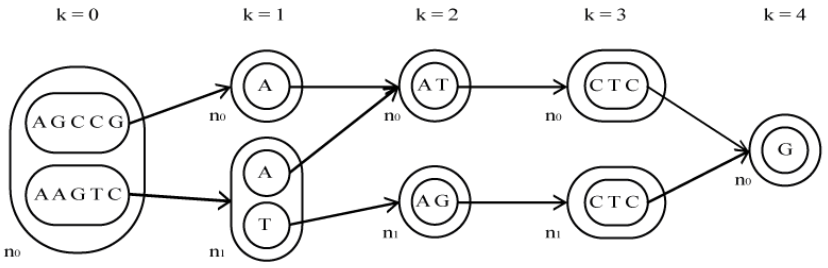
Level $k = 0$ consists of a single node:
$n_0 = \{x \in \hat{R}[0] \mid x = y_0 \dots y_{q_0} \text{with } y_j \in \text{chop}_j \; \forall j : 0 \le j \le q_0\}$, where q_0 is the index of the rightmost chop containing suffixes of $\hat{R}[0]$.

Level $k > 0$ consists of $\ell = |\text{chop}_{q_{k-1}}|$ nodes. Assuming without loss of generality that level $k-1$ has been built with suffixes of $\hat{R}[\text{pos}(\hat{A}_{q_{k-1}})]$, level k contains suffixes of a position of $\hat{S}$. Let $c_0, \dots, c_{\ell-1}$ denote the elements of $\text{chop}_{q_{k-1}}$. Then, for $0 \le i \le \ell-1$, the i-th node of level k is:
$n_i = \{y_{q_{k-1}+1} \dots y_{q_k} \mid c_i y_{q_{k-1}+1} \dots y_{q_k} \in \hat{B}_{q_{k-1}} \text{with } y_j \in \text{chop}_j \; \forall j : q_{k-1}+1 \le j \le q_k\}$, where q_k is the index of the rightmost chop containing suffixes of $\hat{S}[\text{pos}(\hat{B}_{q_{k-1}})]$.
Every string in level $k-1$ whose suffix is c_i is the source of an edge having the whole node n_i as a sink.

We define paths$(G_{\hat{R},\hat{S}})$ as the set of strings spelled by a path in $G_{\hat{R},\hat{S}}$ that starts at n_0 and ends at the last level.

Note that the size of $G_{\hat{R},\hat{S}}$ is at most linear in the sum of the sizes of $\hat{R}$ and $\hat{S}$, as the nodes contain strings either in $\hat{R}$ or in $\hat{S}$ with no duplications, and each node has out-degree equal to the number of strings it contains.

Example 4.6. (for Definition 4.5)
The graph $G_{\hat{R},\hat{S}}$ for the GD strings $\hat{R}$, $\hat{S}$ of Example 4.4 is the following:



$q_0 = 2$ and the strings in level 0 belong to $(\text{chop}_0 \otimes \text{chop}_1 \otimes \text{chop}_2) \cap \hat{R}[0]$. Level 1 contains suffixes of strings in $\hat{B}_2$ (and of strings in $\hat{B}_3$ as $\text{chop}_3 = \{\mathtt{A}, \mathtt{T}\}$ and indeed $q_1 = 3$), level 2 suffixes of strings in $\hat{A}_3$ (as $q_2 = 5$), level 3 suffixes of strings in $\hat{B}_5$ ($q_3 = 6$), level 4 suffixes of strings in $\hat{A}_6$ ($q_4 = 7$). The three paths from level 0 to level 4 correspond to the three strings in $L(\hat{R}) \cap L(\hat{S})$: `AGCCGAATCTCG`, `AAGTCAATCTCG`, `AAGTCTAGCTCG`.

Let $G^k_{\hat{R},\hat{S}}$ be $G_{\hat{R},\hat{S}}$ truncated at level k, and let $|G^k_{\hat{R},\hat{S}}|$ be the length of the strings it spells. Let $L_k(\hat{S})$ denote the set of prefixes of length $|G^k_{\hat{R},\hat{S}}|$ of $L(\hat{S})$.

Lemma 4.7. Let $\hat{R}, \hat{S}$ be two GD strings with $w(\hat{R}) = w(\hat{S}) = W$ and no synchronized proper prefixes. Then $L_k(\hat{S}) \cap L_k(\hat{R}) = \text{paths}(G^k_{\hat{R},\hat{S}})$ for all levels k of $G_{\hat{R},\hat{S}}$ such that $L_k(\hat{S}) \cap L_k(\hat{R}) \neq \emptyset$.

Proof:
Again, let us assume without loss of generality that $w(\hat{R}[0]) > w(\hat{S}[0])$. We prove the result by induction on k.
Level $k = 0$. By construction, n_0 contains strings in $\hat{R}[0] \cap (\text{chop}_0 \otimes \cdots \otimes \text{chop}_{q_0})$, which have length $|G^0_{\hat{R},\hat{S}}|$, and are also in $\hat{S}[0]$, and hence belong to both $L_0(\hat{S})$ and $L_0(\hat{R})$.
Level $k > 0$. By inductive hypothesis, we have that $L_{k-1}(\hat{S}) \cap L_{k-1}(\hat{R}) = \text{paths}(G^{k-1}_{\hat{R},\hat{S}})$; suppose that $L_k(\hat{S}) \cap L_k(\hat{R}) \neq \emptyset$, otherwise the graph ends at level $k-1$. We first show that $\text{paths}(G^k_{\hat{R},\hat{S}}) \subseteq L_k(\hat{S}) \cap L_k(\hat{R})$: by Definition 4.5, any $z \in \text{paths}(G^k_{\hat{R},\hat{S}})$ can be written as $z = z'z''$ with z' in $\text{paths}(G^{k-1}_{\hat{R},\hat{S}})$ and with z'' that belongs to some node at level k of $G^k_{\hat{R},\hat{S}}$ reached by an edge leaving a suffix of z'. By inductive hypothesis $z' \in L_{k-1}(\hat{S}) \cap L_{k-1}(\hat{R})$ and, again by Definition 4.5, $z'' \in \text{chop}_{q_{k-1}+1} \otimes \cdots \otimes \text{chop}_{q_k}$; since $L_k(\hat{S}) \cap L_k(\hat{R}) \neq \emptyset$ these chops are not empty, their concatenation contains the suffix of length $|G^k_{\hat{R},\hat{S}}| - |G^{k-1}_{\hat{R},\hat{S}}|$ of strings in both $L_k(\hat{R})$ and $L_k(\hat{S})$, and hence $z \in L_k(\hat{S}) \cap L_k(\hat{R})$.
We now show that $L_k(\hat{S}) \cap L_k(\hat{R}) \subseteq \text{paths}(G^k_{\hat{R},\hat{S}})$. Consider string $u \in L_k(\hat{S}) \cap L_k(\hat{R})$ that can be written as $u = u'u''$ with u' the prefix of u having length $|G^{k-1}_{\hat{R},\hat{S}}|$ which then belongs to $L_{k-1}(\hat{S}) \cap L_{k-1}(\hat{R})$; then, by inductive hypothesis, $u' \in \text{paths}(G^{k-1}_{\hat{R},\hat{S}})$ and, since $u \in L_k(\hat{S}) \cap L_k(\hat{R})$, then there is an edge linking a suffix of u' at level $k-1$ with a node at level k of $G^k_{\hat{R},\hat{S}}$ containing a $|G^k_{\hat{R},\hat{S}}| - |G^{k-1}_{\hat{R},\hat{S}}|$ long suffix u'' of u, and hence $u \in \text{paths}(G^k_{\hat{R},\hat{S}})$. □

As a special case of Lemma 4.7, if $L(\hat{S}) \cap L(\hat{R}) \neq \emptyset$, then $G_{\hat{R},\hat{S}}$ is built up to the last level and the following holds.

Theorem 4.8. Let $\hat{R}, \hat{S}$ be two GD strings having lengths, respectively, r and s, with $w(\hat{R}) = w(\hat{S})$ and no synchronized proper prefixes. Then $G_{\hat{R},\hat{S}}$ has exactly $r + s - 1$ levels, and we have that $L(\hat{S}) \cap L(\hat{R}) = \text{paths}(G_{\hat{R},\hat{S}})$.

$G_{\hat{R},\hat{S}}$ is thus a linear-sized representation of the possibly exponential-sized (see Fact 4.1) set $L(\hat{S}) \cap L(\hat{R})$.

We now show an $\mathcal{O}(N + M)$-time algorithm for the standard word RAM model, denoted by GDSC, that decides whether $L(\hat{R})$ and $L(\hat{S})$ share at least one string (returns 1) or not (returns 0). GDSC starts with constructing the generalized suffix tree $T_{\hat{R},\hat{S}}$ of all the strings in $\hat{R}$ and $\hat{S}$. Then it scans $\hat{R}$ and $\hat{S}$ starting with $\hat{R}[0]$ and $\hat{S}[0]$, storing in $\text{chop}_{\hat{R},\hat{S}}$ the latest chop_i and in $\text{active}_{\hat{R},\hat{S}}$ the latest $\text{active}_{\hat{A}_i,\hat{B}_i}$, using $T_{\hat{R},\hat{S}}$. To efficiently implement GDSC, rather than explicitly storing suffixes in $\text{active}_{\hat{R},\hat{S}}$, they are stored as index positions of $\hat{R}[i]$ or $\hat{S}[j]$. A variable *suff* is used to keep track of the starting position of the suffixes. Note that this starting position is the same for every index that has been stored in $\text{active}_{\hat{R},\hat{S}}$. Given that $w(\hat{S[0]}) < w(\hat{R[0]})$, the next comparison is made between the corresponding suffixes of $\hat{R}[0]$ of length $w(\hat{R[0]}) - \textit{suff}$ and $\hat{S}[1]$, proceeding with the same process. The comparison of letters can be: (i) between $\hat{R}[i]$ and $\hat{S}[j]$; or (ii) between the corresponding suffixes of $\text{active}_{\hat{R},\hat{S}}$ and $\hat{R}[i]$; or (iii) between the corresponding suffixes of $\text{active}_{\hat{R},\hat{S}}$ and $\hat{S}[j]$. If the two GD strings have a synchronized proper prefix, this will result in $\text{active}_{\hat{R},\hat{S}} = \emptyset$ at positions i in $\hat{R}$ and j in $\hat{S}$. At this point, the comparison is restarted with the immediately following pair of degenerate letters.

Example 4.9. Consider the GD strings $\hat{R}$ and $\hat{S}$ below. The set $L(\hat{R}) \cap L(\hat{S})$ contains the strings ATACGACTAACGTT, ATACGACTAGCACT, TATCCGACTACGTT, TATCCGACTGCACT.

$$\hat{R} = \begin{Bmatrix} \underline{\texttt{ATA}} \\ \underline{\texttt{TAT}} \end{Bmatrix} \cdot \begin{Bmatrix} \texttt{GACC} \\ \underline{\texttt{CCGA}} \\ \underline{\texttt{CGAC}} \end{Bmatrix} \cdot \begin{Bmatrix} \texttt{CG} \\ \underline{\texttt{CT}} \\ \underline{\texttt{TA}} \end{Bmatrix} \cdot \begin{Bmatrix} \underline{\texttt{ACGT}} \\ \underline{\texttt{GCAC}} \end{Bmatrix} \cdot \{\underline{\texttt{T}}\}$$

$$\hat{S} = \begin{Bmatrix} \underline{\texttt{TATCC}} \\ \texttt{TATGA} \\ \underline{\texttt{ATACG}} \end{Bmatrix} \cdot \begin{Bmatrix} \underline{\texttt{ACTA}} \\ \underline{\texttt{GACT}} \end{Bmatrix} \cdot \begin{Bmatrix} \underline{\texttt{GCA}} \\ \texttt{AAT} \\ \underline{\texttt{ACG}} \end{Bmatrix} \cdot \begin{Bmatrix} \underline{\texttt{C}} \\ \texttt{A} \\ \underline{\texttt{T}} \end{Bmatrix} \cdot \{\underline{\texttt{T}}\}$$

Theorem 4.10. Algorithm GDSC is correct. Given two GD strings $\hat{R}$ and $\hat{S}$ of total sizes N and M, respectively, over an integer alphabet, algorithm GDSC requires $\mathcal{O}(N + M)$ time.

Proof:
The correctness follows directly from Lemma 4.2, Lemma 4.7, and Theorem 4.8.

As for the complexity, we have the following: constructing the generalized suffix tree $T_{\hat{R},\hat{S}}$ can be done in time $\mathcal{O}(N + M)$ [23]; for the sets pair $(\hat{A}_i, \hat{B}_i)$ as in Definition 4.3, such that $w(\hat{A}_i) = k$

and $w(\hat{A}_i) \leq w(\hat{B}_i)$, we query $T_{\hat{R},\hat{S}}$ with the k-length prefixes of strings in $\hat{B}_i$; for integer alphabets, instead of spelling the strings from the root of $T_{\hat{R},\hat{S}}$, we locate the corresponding terminal nodes for $(\hat{A}_i, \hat{B}_i)$. It then suffices to find longest common prefixes between these suffixes to simulate the querying process. Since all suffixes are lexicographically sorted during the construction of $T_{\hat{R},\hat{S}}$, we can also have the suffixes considered by pair $(\hat{A}_i, \hat{B}_i)$ lexicographically ranked with respect to $(\hat{A}_i, \hat{B}_i)$. Hence we do not perform the longest common prefix operation for all possible suffix pairs, but only for the lexicographically adjacent ones within this group. This can be done in $\mathcal{O}(1)$ time per pair after $\mathcal{O}(N + M)$-time pre-processing over $T_{\hat{R},\hat{S}}$ [24]. chop_i is thus populated with the k-length prefixes of strings in $\hat{B}_i$ found in $\hat{A}_i$. The set $\text{active}_{\hat{A}_i,\hat{B}_i}$ of active suffixes can be found by chopping the suffixes of the string in $\hat{B}_i$ from their prefixes successfully queried in $T_{\hat{R},\hat{S}}$. This requires time $\mathcal{O}(|\hat{A}_i| + |\hat{B}_i|)$ for processing $(\hat{A}_i, \hat{B}_i)$.

Let $\hat{R}$ and $\hat{S}$ be of length r and s, respectively. Assume that $\hat{R}$ and $\hat{S}$ have no synchronized proper prefixes. Then Theorem 4.8 ensures that the total number of comparisons cannot exceed $r + s - 2$: this results in a time complexity of $\mathcal{O}(N + M + \sum_{i=0}^{r+s-2}(|\hat{A}_i| + |\hat{B}_i|)) = \mathcal{O}(N + M)$.

If $\hat{R}$ and $\hat{S}$ have synchronized proper prefixes, we perform the comparison up to the shortest synchronized prefixes (i.e., the set of active suffixes becomes empty) and then restart the procedure from the immediately following pair of degenerate letters. Clearly the total number of comparisons also in this case cannot be more than $r + s - 2$. □

5. Computing Palindromes in GD Strings

Armed with the efficient GD string comparison tool, we shift our focus on computing palindromes in GD strings.

Definition 5.1. A GD string $\hat{S}$ is a *GD palindrome* if there exists a string in $L(\hat{S})$ that is a palindrome.

A GD palindrome $\hat{S}[i] \dots \hat{S}[j]$ in $\hat{S}$, whose total width is $w(\hat{S}[i] \dots \hat{S}[j])$, can be encoded as a pair (c, r), where its *center* is $c = \frac{w(\hat{S}[0]\dots\hat{S}[i-1])+w(\hat{S}[0]\dots\hat{S}[j])-1}{2}$, when $i > 0$, otherwise, $c = \frac{w(\hat{S}[0]\dots\hat{S}[j])-1}{2}$, when $i = 0$; its *radius* is $r = \frac{w(\hat{S}[i]\dots\hat{S}[j])}{2}$. $\hat{S}[i] \dots \hat{S}[j]$ is called *maximal* if no other GD palindrome (c, r') exists in $\hat{S}$ with $r' > r$. Note that we only consider the GD palindromes $\hat{S}[i] \dots \hat{S}[j]$ that start with the first letter of some string $X \in \hat{S}[i]$ and end with the last letter of some string $Y \in \hat{S}[j]$, while the center can be anywhere: in between or inside degenerate letters. That is, in $\hat{S}$ there are $2 \cdot w(\hat{S}) - 1 = 2W - 1$ possible centers.

Example 5.2. (for Definition 5.1)
In the GD string

$$\hat{S} = \{\mathtt{G}\} \cdot \left\{\begin{matrix}\underline{\mathtt{A\,C\,A}}\\ \underline{\overline{\mathtt{T\,T\,T}}}\\ \mathtt{G\,T\,C}\end{matrix}\right\} \cdot \left\{\begin{matrix}\overline{\mathtt{T\,G}}\\ \mathtt{A\,G}\\ \underline{\mathtt{T\,T}}\end{matrix}\right\} \cdot \left\{\begin{matrix}\underline{\mathtt{G\,A}}\\ \overline{\mathtt{G\,T}}\end{matrix}\right\} \cdot \left\{\begin{matrix}\mathtt{C\,A\,G\,G\,C\,T\,T\,T}\\ \underline{\mathtt{C\,C\,A\,G\,T\,T\,A\,C}}\\ \mathtt{A\,T\,T\,T\,C\,A\,G\,G}\end{matrix}\right\} \cdot \{\underline{\mathtt{A}}\}$$

we have palindrome ACATTGACCAGTTACA (underlined) at $\hat{S}[1] \dots \hat{S}[5]$ which corresponds to GD palindrome (8.5,8), palindrome TTT at $\hat{S}[1]$ (underlined twice) which corresponds to GD palindrome (2,1.5) and palindrome TGGT at $\hat{S}[2] \dots \hat{S}[3]$ (overlined) which corresponds to GD palindrome (5.5,2). Observe that GD palindrome (5.5,2) is maximal for center 5.5. Note that CTGGTC does not correspond to a valid GD palindrome as it neither starts at the beginning of a degenerate letter, nor ends at the end of a degenerate letter.

In this section, we consider the following problem. Given a GD string $\hat{S}$ of length n, total size N, and total width W, find all GD strings $\hat{S}[i] \dots \hat{S}[j]$, with $0 \leq i \leq j \leq n-1$, that are GD palindromes. We give two alternative algorithms: one finds all GD palindromes seeking them for all (i, j) pairs of starting and ending positions, and the other one finds them starting from all possible centers. The two algorithms have different time complexities: which one is faster depends on W, N, and n. In fact, they compute all GD palindromes, but report only the maximal ones.

5.1. Algorithms for Computing GD Palindromes

We first describe algorithm MAXPALPAIRS. For all i, j positions within $\hat{S}$, in order to check whether $\hat{S}[i] \dots \hat{S}[j]$ is a GD palindrome, we apply the GDSC algorithm to $\hat{S}[i] \dots \hat{S}[j]$ and its reverse, denoted by *rev*$(\hat{S}[i] \dots \hat{S}[j])$; the reverse is defined by reversing the sequence of degenerate letters and also reversing the strings in every degenerate letter. GD palindromes are, finally, sorted per center, and the maximal GD palindromes are reported. Sorting the (i, j) pairs by their centers can be done in $\mathcal{O}(W)$ time using bucket sort, which is bounded by $\mathcal{O}(N)$ since $N \geq W$.

Since there are $\mathcal{O}(n^2)$ pairs (i, j), and since by Theorem 4.10 algorithm GDSC takes time proportional to the total size of $\hat{S}[i] \dots \hat{S}[j]$ to check whether $\hat{S}[i] \dots \hat{S}[j]$ is a GD palindrome, algorithm MAXPALPAIRS takes $\mathcal{O}(n^2 N)$ time in total. In algorithm MAXPALCENTERS, we consider all possible centers c of $\hat{S}$. In the case when c is in between two degenerate letters we simply try to extend to the left and to the right via applying GDSC. In the case when c is inside a degenerate letter we intuitively split the letter vertically into two letters and try to extend to the left and to the right via applying GDSC. At each extension step of this procedure we maintain two GD strings $\hat{L}$ (left of the center) and $\hat{R}$ (right of the center) such that they are of the same total width. We consider the reverse of $\hat{L}$ (similar to algorithm MAXPALPAIRS) for the comparison. In the case where c occurs inside a degenerate letter to make sure we do not identify palindromes which do not exist, for all j split strings of the degenerate letter, we check that $\hat{L}^R[0][j][0 \dots k-1] = \hat{R}[0][j][0 \dots k-1]$ where $\hat{L}^R = \mathit{rev}(\hat{L})$ and $k = \min(w(L^R[0]), w(\hat{R}[0]))$. If no matches are found, we move onto the next center. Otherwise, when a match is found, we update *rev*$(\hat{L})$ and $\hat{R}$ with the remainder of the split degenerate letter (if its length is greater than k), as well as the next degenerate letters. Algorithm GDSC is applied to compare *rev*$(\hat{L})$ and $\hat{R}$. After a positive comparison, we overwrite $\hat{L}$ and $\hat{R}$ by adding the degenerate letters of the current extension until $w(\hat{L}) = w(\hat{R})$ (or until the end of the string is reached). This process is repeated as long as GDSC returns a positive comparison, that is, until the maximal GD palindrome with center c is found. The radius reported is then the total sum of all values of $w(\hat{L})$. If GDSC returns a negative comparison at center c, we proceed with the next center, because we clearly cannot have a GD palindrome centered at c extended further if *rev*$(\hat{L}) \cap \hat{R}$ is empty.

By Theorem 4.10 and the fact that there are $2W-1$ possible centers, we have that algorithm MAX-PALCENTERS takes $\mathcal{O}(WN)$ time in total. We obtain the following result.

Theorem 5.3. Given a GD string of length n, total size N, and total width W, over an integer alphabet, all (maximal) GD palindromes can be computed in time $\mathcal{O}(\min\{W, n^2\}N)$.

The problem that gained significant attention recently is the factorization of a string X of length n into a sequence of palindromes [25, 26, 27, 28, 29, 30]. We say that $X_1, X_2, \ldots, X_\ell$ is a (maximal) palindromic factorization of string X, if every X_i is a (maximal) palindrome, $X = X_1X_2 \ldots X_\ell$, and ℓ is minimal. In biological applications we need to factorize a sequence into palindromes in order to identify *hairpins*, patterns that occur in single-stranded DNA or, more commonly, in RNA. Next, we define and solve the same problem for GD strings.

Alatabbi et al. gave an off-line $\mathcal{O}(n)$-time algorithm for finding a maximal palindromic factorization of X [25]. Fici et al. presented an on-line $\mathcal{O}(n \log n)$-time algorithm for computing a palindromic factorization of X [26]; a similar algorithm was presented by I et al. [31]. In addition, Rubinchik and Shur [27] devised an $\mathcal{O}(n)$-sized data structure that helps to locate palindromes in a string; they also showed how it can be used to compute a palindromic factorization of X in $\mathcal{O}(n \log n)$ time. Borozdin et al. recently improved this by presenting an $\mathcal{O}(n)$-time algorithm [28]. Alzamel et al. considered the factorization problem for weighted sequences [29] and Adamczyk et al. considered the factorization problem with gaps and errors [30]. In what follows, we define and solve the same problem for GD strings.

Definition 5.4. A *(maximal) GD palindromic factorization* of a GD string $\hat{S}$ is a sequence $\hat{P}_1, \ldots, \hat{P}_\ell$ of GD strings, such that: (i) every $\hat{P}_i$ is either a (maximal) GD palindrome or a degenerate letter of $\hat{S}$; (ii) $\hat{S} = \hat{P}_1 \ldots \hat{P}_\ell$; (iii) ℓ is minimal.

After locating all (maximal) GD palindromes in $\hat{S}$ using Theorem 5.3, we are in a position to amend the algorithm of Alatabbi et al. [25] to find a (maximal) GD palindromic factorization of $\hat{S}$. We define a directed graph $\mathcal{G}_{\hat{S}} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{i \mid 0 \leq i \leq n\}$ and $\mathcal{E} = \{(i, j+1) \mid \hat{S}[i] \ldots \hat{S}[j]$ (maximal) GD palindrome of $\hat{S}\} \cup \{(i, i+1) | 0 \leq i < n\}$. Note that $\mathcal{V}$ contains a node n that is the sink of edges representing (maximal) GD palindromes ending at $\hat{S}[n-1]$. For maximal GD palindromes, $\mathcal{E}$ contains no more than $3W$ edges, as the maximum number of maximal GD palindromes is $2W-1$. For GD palindromes, $\mathcal{E}$ contains $\mathcal{O}(n^2)$ edges, as the maximum number of GD palindromes is $\mathcal{O}(n^2)$. A shortest path in $\mathcal{G}_{\hat{S}}$ from 0 to n gives a (maximal) GD palindromic factorization. For maximal GD palindromes, the size of $\mathcal{G}_{\hat{S}}$ is $\mathcal{O}(W)$, as $n \leq W$, and so finding this shortest path requires $\mathcal{O}(W)$ time using a standard algorithm. For GD palindromes, the size of $\mathcal{G}_{\hat{S}}$, and thus the time, is $\mathcal{O}(n^2)$.

Theorem 5.5. Given a GD string $\hat{S}$ of length n, total size N, and total width W, over an integer alphabet, a (maximal) GD palindromic factorization of $\hat{S}$ can be computed in time $\mathcal{O}(\min\{W, n^2\}N)$.

Remark 5.6. As a final remark, notice that a way to check whether a GD string is a GD palindrome is to compare it with its reverse. Since it is not a simple pattern matching between two GD strings, one can reduce the number of comparisons by comparing the r-th string in $\hat{S}[s]$ and the r-th string in

$\hat{S}[n-1-s]^R$, for all $0 \leq s \leq n-1$. It means that the comparison is not free, but constrained to the positions that contain the first half of the palindrome contained in $\hat{S}$.

5.2. Computing GD Palindromes in Protein Sequences

	Hypervariable Region			
	I		II	
V	[32]	**This paper**	[32]	**This paper**
V_kII	18-27	11-36	119-130	118-131
	104-113	104-113	169-180	169-180
V_kIII	18-27	11-30	132-142	131-145
V_λII	63-74	62-81	140-152	140-152
V_λIII	51-74	50-75	132-143	131-144
V_λV	96-104	95-104	134-141	134-141

Table 1. Coordinates of (maximal) palindromes identified within hypervariable regions I and II.

We have implemented algorithm MAXPALPAIRS in C++. We present here a proof-of-concept experiment but we anticipate that the algorithmic tools developed in this paper are applicable in a wide range of biological applications.

We first obtained the amino acid sequences of 5 immunoglobulins within the human V regions [33] and converted these into mRNA sequences [34]. The letters X, S, T, Y, Z, R and H were replaced by degenerate letters according to IUPAC [35]. Each other letter, $c \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$, was treated as a single degenerate letter $\{c\}$. An average of 47% of the total number of positions within the 5 sequences consisted of one of the following: X, S, T, Y, Z, R and H. We then used algorithm MAXPALPAIRS to find all maximal palindromes in the 5 sequences. Table 1 shows the palindromes identified within hypervariable regions I and II. Our results are in accordance with Wuilmart et al [32] who presented a *statistical* (fundamentally different) method to identify the location of palindromes within regions of immunoglobulin genes. The ranges we report are greater than or equal to the ones of [32] due to the *maximality* criterion used in the computation of GD palindromes by algorithm MAXPALPAIRS.

6. A Conditional Lower Bound under SETH

In this section, we show a conditional lower bound for computing palindromes in degenerate strings. Let us first define the 2-Orthogonal Vectors problem. Given two sets $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and $B = \{\beta_1, \beta_2, \ldots, \beta_n\}$ of d-bit vectors, where $d = \omega(\log n)$, the *2-Orthogonal Vectors* problem asks the following question: is there any pair α_i, β_j of vectors that is orthogonal? Namely, is $\sum_{k=0}^{d-1} \alpha_i[k] \cdot \beta_j[k]$ equal to 0 for some $i, j \in [1, n]$? For the moderate dimension of this problem, we follow [36], assuming $n^{2-\epsilon} d^{\mathcal{O}(1)} \leq n^2 d$. The following result is known.

Theorem 6.1. ([36, 21, 22, 37])
The 2-Orthogonal Vectors problem cannot be solved in $\mathcal{O}(n^{2-\epsilon} \cdot d^{\mathcal{O}(1)})$ time, for any $\epsilon > 0$, unless the *Strong Exponential Time Hypothesis* fails.

We next show that the 2-Orthogonal Vectors problem can be reduced to computing maximal palindromes in degenerate strings thus obtaining a conditional lower bound similar to the upper bound obtained in Theorem 5.3 for computing all GD palindromes.

Theorem 6.2. Given a degenerate string of length $4n$ over an alphabet of size $\sigma = \omega(\log n)$, all maximal GD palindromes cannot be computed in $\mathcal{O}(n^{2-\epsilon} \cdot \sigma^{\mathcal{O}(1)})$ time, for any $\epsilon > 0$, unless the *Strong Exponential Time Hypothesis* fails.

Proof:
Let $d = \sigma$ and consider the alphabet $\Sigma = \{0, 1, \ldots, \sigma - 1\}$. We say that two subsets of Σ *match* if they have a common element. Given a d-bit vector α, we define $\mu(\alpha)$ to be the following subset of Σ: $s \in \mu(\alpha)$ if and only if $\alpha[s] = 1$. Thus, two vectors α and β are orthogonal if and only if the sets $\mu(\alpha)$ and $\mu(\beta)$ are disjoint. In the string comparison setting, two degenerate letters $\mu(\alpha)$ and $\mu(\beta)$ *do not match* if and only if α and β are orthogonal. The reduction works as follows. Given $A = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and $B = \{\beta_1, \beta_2, \ldots, \beta_n\}$, we construct the following simple degenerate string of length $4n$ in time $\mathcal{O}(n\sigma)$:

$$S = \underbrace{\mu(\alpha_1)\mu(\beta_1)\mu(\alpha_2)\mu(\beta_2)\ldots\mu(\alpha_n)\mu(\beta_n)}\,\mu(\alpha_1)\mu(\beta_1)\mu(\alpha_2)\mu(\beta_2)\ldots\mu(\alpha_n)\mu(\beta_n).$$

$$\ldots \qquad \underline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$$

Then the 2-Orthogonal Vectors problem for the sets A and B has a positive answer if and only if at any position of S, from 0 to $2n$, there *does not occur* a palindrome of length at least $2n$. All such occurrences can be easily verified from the respective palindrome centers in time $\mathcal{O}(n)$. In other words, if at any position of S there does not occur a palindrome of length at least $2n$, this is because we have a mismatch between a pair $\mu(\alpha_i), \mu(\beta_j)$ of letters, which implies that there exists a pair α_i, β_j of orthogonal vectors. Also, by the construction, all such pairs are to be (implicitly) compared, and thus, if there exists any pair that is orthogonal, the corresponding mismatch will result in a palindrome of length less than $2n$. □

7. Concluding Remarks and Open Problems

In this paper we solved the problem of comparing two GD strings, that is, to decide whether two GD strings have a non-empty intersection. We then applied our string comparison tool to devise a simple algorithm for computing all palindromes in a GD string. We also complemented the latter algorithm by showing a similar conditional lower bound.

In Section 1, we sketched how automata can be used for comparing two ED strings. Recall that an ED string is a more general notion of degenerate string, where a degenerate letter generally contains strings of arbitrary lengths as well as the empty string. For GD strings, we showed that this comparison

can be done in linear time for integer alphabets (Theorem 4.10). An interesting open problem is whether we can devise a more efficient (than the $\mathcal{O}(NM)$-time automata-based sketched in Section 1) approach for deciding whether the two languages represented by two ED strings of sizes N and M have a non-empty intersection; or, more generally, whether they share a sufficiently long substring.

Acknowledgements

NP and GR are partially supported by the project MIUR-SIR CMACBioSeq ("Combinatorial methods for analysis and compression of biological sequences") grant n. RBSI146R5L.

References

[1] Soldano H, Viari A, Champesme M. Searching for flexible repeated patterns using a non-transitive similarity relation. *Pattern Recognition Letters*, 1995. **16**(3):233–246.

[2] Pisanti N, Soldano H, Carpentier M. Incremental Inference of Relational Motifs with a Degenerate Alphabet. In: 16th Annual Symposium on Combinatorial Pattern Matching (CPM), volume 3537 of *Springer LNCS*. 2005 pp. 229–240.

[3] Pisanti N, Soldano H, Carpentier M, Pothier J. A Relational Extension of the Notion of Motifs: Application to the Common 3D Protein Substructures Searching Problem. *Journal of Computational Biology*, 2009. **16**(12):1635–1660.

[4] Abrahamson K. Generalized String Matching. *SIAM Journal of Computing*, 1987. **16**(6):1039–1051.

[5] Crochemore M, Iliopoulos CS, Kociumaka T, Radoszewski J, Rytter W, Walen T. Covering problems for partial words and for indeterminate strings. *Theoretical Computer Science*, 2017. **698**:25–39.

[6] Iliopoulos CS, Radoszewski J. Truly Subquadratic-Time Extension Queries and Periodicity Detection in Strings with Uncertainties. In: 27th Annual Symposium on Combinatorial Pattern Matching (CPM), volume 54 of *LIPIcs*. 2016 pp. 8:1–8:12.

[7] Consortium TCPG. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 2016. pp. 1–18.

[8] Iliopoulos CS, Kundu R, Pissis SP. Efficient Pattern Matching in Elastic-Degenerate Texts. In: 11th International Conference on Language and Automata Theory and Applications (LATA), volume 10168 of *Springer LNCS*. 2017 pp. 131–142.

[9] Myers EW. Approximate Matching of Network Expressions with Spacers. *Journal of Computational Biology*, 1996. **3**(1):33–51.

[10] Grossi R, Iliopoulos CS, Liu C, Pisanti N, Pissis SP, Retha A, Rosone G, Vayani F, Versari L. On-line pattern matching on a set of similar texts. In: 28th Annual Symposium on Combinatorial Pattern Matching (CPM), volume 78 of *LIPIcs*. 2017 pp. 9:1–9:14.

[11] Aoyama K, Nakashima Y, Inenaga S, Bannai H, Takeda M, et al. Faster online elastic degenerate string matching. In: 29th Annual Symposium on Combinatorial Pattern Matching (CPM), volume 105 of *LIPIcs*. 2018 pp. 9:1–9:10.

[12] Pissis SP, Retha A. Dictionary Matching in Elastic-Degenerate Texts with Applications in Searching VCF Files On-line. In: 17th International Symposium on Experimental Algorithms (SEA), volume 103 of *LIPIcs*. 2018 pp. 16:1–16:14.

[13] Bernardini G, Gawrychowski P, Pisanti N, Pissis SP, Rosone G. Even Faster Elastic-Degenerate String Matching via Fast Matrix Multiplication. In: 46th International Colloquium on Automata, Languages, and Programming (ICALP), volume 132 of *LIPIcs*. 2019 pp. 21:1–21:15.

[14] Bernardini G, Pisanti N, Pissis SP, Rosone G. Pattern Matching on Elastic-Degenerate Text with Errors. In: 24th International Symposium on String Processing and Information Retrieval (SPIRE), volume 10508 of *Springer LNCS*. 2017 pp. 74–90.

[15] Bernardini G, Pisanti N, Pissis S, Rosone G. Approximate pattern matching on elastic-degenerate text. *Theoretical Computer Science*, 2020. **812**:109–122.

[16] Alzamel M, Ayad L, Bernardini G, Grossi R, Iliopoulos C, Pisanti N, Pissis S, Rosone G. Degenerate string comparison and applications. In: 18th International Workshop on Algorithms in Bioinformatics (WABI), volume 113 of *LIPIcs*. 2018 pp. 21:1–21:14.

[17] Manacher G. A New Linear-Time "On-Line" Algorithm for Finding the Smallest Initial Palindrome of a String. *Journal of the ACM*, 1975. **22**(3):346–351.

[18] Apostolico A, Breslauer D, Galil Z. Parallel detection of all palindromes in a string. *Theoretical Computer Science*, 1995. **141**(1):163–173.

[19] Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, New York, NY, USA, 1997.

[20] Lipton RJ. On The Intersection of Finite Automata, pp. 145–148. Springer US, Boston, MA, 2010.

[21] Impagliazzo R, Paturi R. On the Complexity of k-SAT. *Journal of Computer and System Science*, 2001. **62**(2):367–375.

[22] Impagliazzo R, Paturi R, Zane F. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Science*, 2001. **63**(4):512–530.

[23] Farach M. Optimal suffix tree construction with large alphabets. In: 38th Annual Symposium on Foundations of Computer Science (FOCS). IEEE, 1997 pp. 137–143.

[24] Bender MA, Farach-Colton M. The LCA Problem Revisited. In: 9th Latin American Symposium (LATIN), volume 1776 of *LNCS*. Springer Berlin Heidelberg, 2000 pp. 88–94.

[25] Alatabbi A, Iliopoulos CS, Rahman MS. Maximal Palindromic Factorization. In: Prague Stringology Conference (PSC). 2013 pp. 70–77.

[26] Fici G, Gagie T, Kärkkäinen J, Kempa D. A Subquadratic Algorithm for Minimum Palindromic Factorization. *Journal of Discrete Algorithms*, 2014. **28**:41–48.

[27] Rubinchik M, Shur AM. EERTREE: An Efficient Data Structure for Processing Palindromes in Strings. In: 27th International Workshop on Combinatorial Algorithms (IWOCA), volume 9538 of *Springer LNCS*, pp. 321–333. 2015.

[28] Borozdin K, Kosolobov D, Rubinchik M, Shur AM. Palindromic Length in Linear Time. In: 28th Annual Symposium on Combinatorial Pattern Matching (CPM), volume 78 of *LIPIcs*. 2017 pp. 23:1–23:12.

[29] Alzamel M, Gao J, Iliopoulos CS, Liu C, Pissis SP. Efficient Computation of Palindromes in Sequences with Uncertainties. In: 18th Engineering Applications of Neural Networks (EANN), volume 744 of *Springer CCIS*, pp. 620–629. 2017.

[30] Adamczyk M, Alzamel M, Charalampopoulos P, Iliopoulos CS, Radoszewski J. Palindromic Decompositions with Gaps and Errors. In: The 12th International Computer Science Symposium in Russia (CSR), volume 10304 of *Springer LNCS*. 2017 pp. 48–61.

[31] I T, Sugimoto S, Inenaga S, Bannai H, Takeda M. Computing Palindromic Factorizations and Palindromic Covers On-line. In: 25th Symposium on Combinatorial Pattern Matching (CPM), volume 8486 of *Springer LNCS*. 2014 pp. 150–161.

[32] Wuilmart C, Urbain J, Givol D. On the location of palindromes in immunoglobulin genes. *Proceedings of the National Academy of Sciences of the United States of America*, 1977. **74**(6):2526–2530.

[33] Gally JA, Edelman GM. The Genetic Control of Immunoglobulin Synthesis. *Annual Review of Genetics*, 1972. **6**(1):1–46.

[34] Schuh RT. Major Patterns in Vertebrate Evolution. *Systematic Biology*, 1978. **27**(2):172.

[35] IUPAC-IUB Commission on Biochemical Nomenclature. Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. *Biochemistry*, 1970. **9**(20):4022–4027.

[36] Gao J, Impagliazzo R. Orthogonal Vectors is hard for first-order properties on sparse graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 2016. **23**:53.

[37] Williams R. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci*, 2005. **348**(2-3):357–365.