Order-Preserving Pattern Matching Indeterminate Strings

Diogo Costa\textsuperscript{a}, Luís M. S. Russo\textsuperscript{a}, Rui Henriques\textsuperscript{a}, Hideo Bannai\textsuperscript{b}, Alexandre P. Francisco\textsuperscript{a}

\textsuperscript{a}INESC-ID and Instituto Superior Técnico, Universidade de Lisboa, Portugal
\textsuperscript{b}Department of Computer Science, Kyushu University, Japan

Abstract

Given an indeterminate string pattern $p$ and an indeterminate string text $t$, the problem of order-preserving pattern matching with character uncertainties ($\mu$OPPM) is to find all substrings of $t$ that satisfy one of the possible orderings defined by $p$. When the text and pattern are determinate strings, we are in the presence of the well-studied exact order-preserving pattern matching (OPPM) problem with diverse applications on time series analysis. Despite its relevance, the exact OPPM problem suffers from two major drawbacks: 1) the inability to deal with indetermination in the text, thus preventing the analysis of noisy time series; and 2) the inability to deal with indetermination in the pattern, thus imposing the strict satisfaction of the orders among all pattern positions.

This paper provides the first polynomial algorithm to answer the $\mu$OPPM problem when indetermination is observed on the pattern or text. Given two strings with length $m$ and $O(r)$ uncertain characters per string position, we show that the $\mu$OPPM problem can be solved in $O(mr \lg r)$ time when one string is indeterminate and $r \in \mathbb{N}^+$. Mappings into satisfiability problems are provided when indetermination is observed on both the pattern and the text, and results concerning the general problem complexity are presented as well, with $\mu$OPPM problem proved to be $\text{NP}$-hard in general.

Keywords: order-preserving pattern matching, indeterminate string analysis, generic pattern matching, satisfiability

1. Introduction

Given a pattern string $p$ and a text string $t$, the exact order preserving pattern matching (OPPM) problem is to find all substrings of $t$ with the same relative orders as $p$. The problem is applicable to strings with characters drawn from numeric or ordinal alphabets. Illustrating, given $p=(1,5,3,3)$ and $t =$

*Corresponding author

Email address: aplf@tecnico.ulisboa.pt (Alexandre P. Francisco)
(5, 1, 4, 2, 2, 5, 2, 4), substring $t[1..4] = (1, 4, 2, 2)$ is reported since it satisfies the character orders in $p$, $p[0] \leq p[2] = p[3] \leq p[1]$. Despite its relevance, the OPPM problem has limited potential since it prevents the specification of errors, uncertainties or don’t care characters within the text.

Indeterminate strings allow uncertainties between two or more characters per position. Given indeterminate strings $p$ and $t$, the problem of order preserving pattern matching uncertain text ($\mu$OPPM) is to find all substrings of $t$ with an assignment of values that satisfy the orders defined by $p$. For instance, let $p = (1, 2, 3, 4)$ and $t = (5, 0, 1, 2, 5, 2, 3, 4)$. The substrings $t[1..4]$ and $t[4..7]$ are reported since there is an assignment of values that preserve either $p[0] < p[1] < p[2] = p[3]$ or $p[0] < p[2] = p[3] < p[1]$ orderings: respectively $t[1..4] = (0, 1, 2, 2)$ and $t[4..7] = (2, 5, 3, 3)$.

Order-preserving pattern matching captures the structural isomorphism of strings, therefore having a wide-range of relevant applications in the analysis of financial times series, musical sheets, physiological signals and biological sequences [1, 2, 3]. Uncertainties often occur across these domains. In this context, although the OPPM problem is already a relaxation of the traditional pattern matching problem, the need to further handle localized errors is essential to deal with noisy strings [4]. For instance, given the stochasticity of gene regulation (or markets), the discovery of order-preserving patterns in gene expression (or financial) time series needs to account for uncertainties [5, 6]. Numerical indexes of amino-acids (representing physiochemical and biochemical properties) are subjected to errors difficulting the analysis of protein sequences [7]. Another example are ordinal strings obtained from the discretization of numerical strings, often having two uncertain characters in positions where the original values are near a discretization boundary [4].

Let $m$ and $n$ be the length of the pattern $p$ and text $t$, respectively. The exact OPPM problem has a linear solution on the text length $O(n + m \log m)$ based on the Knuth-Morris-Pratt algorithm [8, 2, 9]. Alternative algorithms for the OPPM problem have also been proposed [10, 11, 12]. Contrasting with the large attention given to the resolution of the OPPM problem, to our knowledge there are no polynomial-time algorithms to solve the $\mu$OPPM problem. Naive algorithms for $\mu$OPPM assess all possible pattern and text assignments, bounded by $O(nr^m)$ when considering up to $r$ uncertain characters per position.

This work proposes the first polynomial time algorithms able to answer the $\mu$OPPM problem. Accordingly, the contributions are organized as follows. First, we show that an indeterminate string of length $m$ order-preserving matches a determinate string with the same length in $O(nr \log r)$ time based on their monotonic properties. Second, and given two indeterminate strings with the same size, we provide a linear encoding of the $\mu$OPPM into a satisfiability formula with properties of interest. Furthermore, we extend this encoding and we present results concerning the computational complexity of $\mu$OPPM problem variations, namely a proof of that the $\mu$OPPM problem is \textbf{NP}-hard in general. Third, given a pattern and text strings with lengths $m$ and $n$, only one of them indeterminate, we show that the $\mu$OPPM problem can be solved in linear space and its average efficiency boosted under effective filtration procedures.
A preliminary version of this work was presented at the Annual Symposium on Combinatorial Pattern Matching (CPM) [13]. In this paper, we revise previous results and we present new results concerning the computational complexity of $\mu$OPPM problem; Sections 3.3, 3.4 and 5 are new.

2. Background

Let $\Sigma$ be a totally ordered alphabet and an element of $\Sigma^*$ be a string. The length of a string $w$ is denoted by $|w|$. The empty string $\varepsilon$ is a string of length 0. For a string $w = xyz$, $x$, $y$ and $z$ are called a prefix, substring, and suffix of $w$, respectively. The $i$-th character of a string $w$ is denoted by $w[i]$ for each $0 \leq i < |w|$. For a string $w$ and integers $0 \leq i < |w|$, $w[i..j]$ denotes the substring of $w$ from position $i$ to position $j$. For convenience, let $w[i..j] = \varepsilon$ when $i > j$.

Given strings $x$ and $y$ with equal length $m$, $y$ is said to order-preserving against $x$ [8], denoted by $x \approx y$, if the orders between the characters of $x$ and $y$ are the same, i.e. $x[i] \leq x[j] \iff y[i] \leq y[j]$ for any $0 \leq i, j < m$. A non-empty pattern string $p$ is said to order-preserving match (op-match in short) a non-empty text string $t$ if and only if there is a position $i$ in $t$ such that $p \approx t[i..|p|+1..i]$. The order-preserving pattern matching (OPPM) problem is to find all such text positions.

2.1. The Problem

Given a totally ordered alphabet $\Sigma$, an indeterminate string is a sequence of disjoint sets of characters $x[0]..x[n-1]$ where $x[i] \subseteq \Sigma$. Each position is given by $x[i] = \sigma_1..\sigma_r$, where $r \geq 1 \land \sigma_i \in \Sigma$.

Given an indeterminate string $x$, a valid assignment $\$x$ is a (determinate) string with a single character at position $i$, denoted $\$x[i]$, contained in the $x[i]$ set of characters, i.e. $\$x[0] \subseteq x[0]$, ..., $\$x[m-1] \subseteq x[m-1]$. For instance, the indeterminate string $(1,[3,3],[4,2],[3,1],[2])$ has $2^4$ valid assignments. Given an indeterminate position $x[i] \subseteq \Sigma$, $\$x[i]$ is the $j$th ordered value of $x[i]$ (e.g. $\$x[0]=1$ for $x[0]=1[2]$). Given an indeterminate string $x$, let a partially assigned string $\$x$ be an indeterminate string with an arbitrary number of uncertain characters removed, i.e. $\$x[0] \subseteq x[0]$, ..., $\$x[m-1] \subseteq x[m-1]$.

Given a determinate string $x$ of length $m$, an indeterminate string $y$ of equal length is said to be order-preserving against $x$, identically denoted by $x \approx y$, if there is a valid assignment $\$y$ such that the relative orders of the characters in $x$ and $\$y$ are the same, i.e. $x[i] \leq x[j] \iff \$y[i] \leq \$y[j]$ for any $0 \leq i, j < m$. Given two indeterminate strings $x$ and $y$ with length $m$, $y$ preserves the orders of $x$, $x \approx y$, if exists $\$y$ in $y$ that respects the orders of a valid assignment $\$x$ in $x$.

A non-empty indeterminate pattern string $p$ is said to order-preserving match (op-match in short) a non-empty indeterminate text string $t$ if and only if there is a position $i$ in $t$ such that $p \approx t[i..|p|+1..i]$. The problem of order-preserving pattern matching with character uncertainties ($\mu$OPPM) problem is to find all such text positions.
To understand the complexity of the \( \mu \)OPPM problem, let us look to its solution from a naive stance yet considering state-of-the-art OPPM principles. The algorithmic proposal by Kubica et al. \[8\] is still up to this date the one providing a lowest bound, \( O(n+q) \), where \( q = m \) for alphabets of size \( m^{O(1)} \) \((q = m \log m \) otherwise). Given a determinate string \( x \) of length \( m \), an integer \( i \) \((0 \leq i < m) \) is said in the context of this work to be an order-preserving border of \( x \) if \( x[0..i] \approx x[m - i + 1..m] \). In this context, given a pattern string \( p \), the orders between the characters of \( p \) are used to linearly infer the order borders. The order borders can then be used within the Knuth-Morris-Pratt algorithm to find op-matches against a text string \( t \) in linear time \[8\].

Given a determinate string \( p \) of length \( m \) and an indeterminate string \( t \) of length \( n \), the previous approach is a direct candidate to the \( \mu \)OPPM problem by decomposing \( t \) in all its possible assignments, \( O(r^n) \). Since determinate assignments to \( t \) are only relevant in the context of \( m \)-length windows, this approach can be improved to guarantee a maximum of \( O(r^m) \) assignments at each text position. Despite its simplicity, this solution is bounded by \( O(nr^m) \). This complexity is further increased when indetermination is also considered in the pattern, stressing the need for more efficient alternatives.

2.2. Related work

The exact OPPM problem is well-studied in literature. Kubica et al. \[8\], Kim et al. \[2\] and Cho et al. \[9\] presented linear time solutions on the text length by respectively combining order-borders, rank-based prefixes and grammars with the Knuth–Morris–Pratt (KMP) algorithm \[14\]. Cho et al. \[10\], Belazzougui et al. \[11\], and Chhabra et al. \[12\] presented \( O(nm) \) algorithms that show a sublinear average complexity by either combining bad character heuristics with the Boyer–Moore algorithm \[15\] or applying filtration strategies. Recently, Chhabra et al. \[16\] proposed further principles to solve OPPM using word-size packed string matching instructions to enhance efficiency.

In the context of numeric strings, multiple relaxations to the exact pattern matching problem have been pursued to guarantee that approximate matches are retrieved. In norm matching \[17, 18, 19, 20\], matches between numeric strings occur if a given distance threshold \( f(x, y) \leq \theta \) is satisfied. In \((\delta, \gamma)\)-matching \[21, 22, 23, 24, 25, 26, 27\], strings are matched if the maximum difference of the corresponding characters is at most \( \delta \) and the sum of differences is at most \( \gamma \).

In the context of nominal strings, variants of the pattern matching task have also been extensively studied to allow for don’t care symbols in the pattern \[28, 29, 30\], transposition-invariant \[22\], parameterized matching \[31, 32\], less than matching \[33\], swapped matching \[34, 35\], gaps \[36, 37, 38\], overlap matching \[39\], and function matching \[40, 41\].

Despite the relevance of the aforementioned contributions to answer the exact order-preserving pattern matching and generic pattern matching, they cannot be straightforwardly extended to efficiently answer the \( \mu \)OPPM problem.
3. On solving $\mu$OPPM

Section 3.1 introduces the first efficient algorithm to solve the $\mu$OPPM problem when one string is indeterminate ($r \in \mathbb{N}^+$). Section 3.2 discusses the existence of efficient solvers when both strings are indeterminate. Section 3.3 introduces then a polynomial time algorithm for the Alternate-$\mu$OPPM as a subproblem of $\mu$OPPM where both strings may have indeterminate characters, but never in the same position. Given the formulations proposed in Section 3.2, we hypothesize that op-matching indeterminate strings with an arbitrary number of uncertain characters per position ($r \in \mathbb{N}^+$) is in class NPC. Furthermore, we show in Section 3.4 that the problem $\{3,3\}$-$\mu$OPPM, defined as the subproblem of $\mu$OPPM where both the pattern and the text have indeterminate characters in any position (although at least one position must have at least three indeterminate characters in both pattern and text), is NP-hard. We still leave a gap in between these two groups, namely for the strings where there are at most two indeterminate characters in both strings at the same position. It remains open whether or not this problem is NP-hard.

3.1. $O(mn \log r)$ time $\mu$OPPM when one string is indeterminate

Given a determinate string $x$ of length $m$, there is a well-defined permutation of positions, $\pi$, that specifies a non-monotonic ascending order of characters in $x$. For instance, given $x=(1,4,3,1)$, then $x[0] = x[3] < x[2] < x[1]$ and $\pi = (0, 3, 2, 1)$. Given a determinate string $y$ with the same length, $y$ op-matches $x$ if it $y$ satisfies the same $m$-1 orders. For instance, given $x = (1, 4, 3, 1)$ and $y = (2, 5, 4, 3)$, $x$ orders are not preserved in $y$ since $y[0] \neq y[3] < y[2] < y[1]$

The monotonic properties can be used to answer $\mu$OPPM when one string is indeterminate. Given an indeterminate string $y$, let $x_{\pi}$ and $y_{\pi}$ be the permuted strings in accordance with $\pi$ orders in $x$. To handle equality constraints, positions in $y_{\pi}$ with identical characters in $x_{\pi}$ can be intersected, producing a new string $y'_{\pi}$ with $s$ length ($s \leq m$). Illustrating, given $x=(4,1,4,2)$ and $y = (2,7,2,7,8,1,4,8)$, then $\pi = (1,3,0,2)$, $x_{\pi}=(1,2,4,4)$. $y_{\pi} = (2,8,4,1,7,2,8,7)$ and $y'_{\pi} = (y_{\pi}[0], y_{\pi}[1], y_{\pi}[2] \cap y_{\pi}[3]) = (2,8,4,1,7)$. To handle monotonic inequalities, $y'_{\pi}[i]$ characters can be concatenated in descending order to compose $z = y'_{\pi}[0]y'_{\pi}[1]...y'_{\pi}[s]$ and the orders between $x$ and $y$ verified by testing if the longest increasing subsequence (LIS) of $z$ has $s$ length. In the given example, $z = (2,8,4,1,7)$, and the LIS of $z = (2,8,4,1,7)$ is $w=(2,4,7)$. Since $|w| = |y'_{\pi}|=3$, $y$ op-matches $x$.

**Theorem 3.1.** Given a determinate string $x$ and an indeterminate string $y$, let $x_{\pi}$ and $y_{\pi}$ be the sorted strings in accordance with $\pi$ order of characters in $x$. Let the positions with equal characters in $x_{\pi}$ be intersected in $y_{\pi}$ to produce a new indeterminate string $y'_{\pi}$. Consider $z_i$ to be a string with $y'_{\pi}[i]$ characters in descending order and $z = z_1z_2...z_m$, then $|w| = |y'_{\pi}|$ if and only if $y \approx x$, where $w$ is a longest increasing subsequence in $z$.

**Proof.** ($\Rightarrow$) If the length of the longest increasing subsequence (LIS), $|w|$, equals the number of monotonic relations in $x$, $|y'_{\pi}|$, then $y \approx x$. By sorting characters
in descending order per position, we guarantee that at most one character per position in $y'_x$ appears in the LIS (respecting monotonic orders in $x$ given $y'_x$ properties). By intersecting characters in positions of $y$ with identical characters in $x$, we guarantee the eligibility of characters satisfying equality orders in $x$, otherwise empty positions in $y'_x$ are observed and the LIS length is less than $|y'_x|$. ($\Leftarrow$) If $|w| < |y'_x|$, there is no assignment in $y$ that op-matches $x$ due to one of two reasons: 1) there are empty positions in $y'_x$ due to the inability to satisfy equalities in $x$, or 2) it is not possible to find a monotonically increasing assignment to $y'_x$ and, given the properties of $y'_x$, $y_x$ cannot preserve the orders of $x_x$.

Solving the LIS task on a string of size $n$ is $O(n \lg n)$ \[42\] where $n = |z| = O(rm)$. In addition, set intersection operations are performed $O(m)$ times on sets with $O(r)$ size, which can be accomplished in $O(rm \lg r)$ time. As a result, the $\mu$OPPM problem with one indeterminate string can be solved in $O(rm \lg(rm))$.

Given the fact that the candidate string for the LIS task has properties of interest, we can improve the complexity of this calculus (Theorem \[3.2\]) in accordance with Algorithm \[1\]

\begin{algorithm}
\caption{O(mr \lg r) $\mu$OPPM algorithm with one indeterminate string.}
\begin{algorithmic}
\State \textbf{Input:} determine $x$, indeterminate $y$ ($|x| = |y| = m$)
\State \hspace{1em} $\pi \leftarrow \text{sortedIndexes}(x)$; \hspace{2cm} // $O(m)$ if $|\Sigma| = m^{O(1)}$; $O(m \lg m)$ otherwise
\State \hspace{1em} $x_x \leftarrow \text{permute}(x, \pi)$, $y_x \leftarrow \text{permute}(y, \pi)$; \hspace{2cm} // $O(m + mr)$
\State \hspace{1em} \hspace{1em} $j \leftarrow 0$; $y'_x[0] \leftarrow \{y_x[0]\}$;
\State \hspace{1em} \hspace{1em} \begin{algorithmic}
\State \hspace{2em} \textbf{foreach} $i \in 1..m$ \textbf{do} \hspace{2cm} // $O(mr \lg r)$
\State \hspace{3em} \textbf{if} $x_x[i] = x_x[i-1]$ \textbf{then} $y'_x[j] \leftarrow y'_x[j] \cap \{y_x[i]\}$; \hspace{2cm} // $O(r \lg r)$
\State \hspace{3em} \hspace{1em} else \hspace{1em} $j \leftarrow j+1$; $y'_x[j] \leftarrow \{y_x[i]\}$;
\State \hspace{3em} \hspace{1em} $s \leftarrow |y'_x|$, nextMin $\leftarrow -\infty$;
\State \hspace{2em} \textbf{foreach} $i \in 0..s-1$ \textbf{do} \hspace{2cm} // $O(mr)$
\State \hspace{3em} nextMin $\leftarrow \min(a \mid a \in y'_x[i], a > \text{nextMin})$; \hspace{2cm} // $O(r)$
\State \hspace{3em} \textbf{if} $\not\exists$ nextMin \textbf{then} return false;
\State \hspace{1em} \textbf{return} true;
\end{algorithmic}
\end{algorithmic}
\end{algorithm}

\textbf{Theorem 3.2.} $\mu$OPPM two strings of length $m$, one being indeterminate, is in $O(mr \lg r)$ time, where $r \in \mathbb{N}^+$. 

\textbf{Proof.} In accordance with Algorithm \[1\] $\mu$OPPM is bounded by the verification of equalities, $O(mr \lg r)$ \[43\]. Testing inequalities after set intersections can be linearly performed on the size of $y$, $O(mr)$ time, improving the $O(mr \lg(mr))$ bound given by the LIS calculus.

The analysis of Algorithm \[1\] further reveals that the $\mu$OPPM problem with one indeterminate string requires linear space in the text length, $O(mr)$. 


3.2. µOPPM with indeterminate pattern and text

As indetermination in real-world strings is typically observed between pairs of characters [4], a key question is whether µOPPM on two indeterminate strings is in class $\mathbf{P}$ when $r = 2$. To explore this possibility, new concepts need to be introduced. In OPPM research, character orders in a determinate string of length $m$ can be decomposed in 3 sequences with $m$ unit sets:

**Definition 3.3.** For $i = 0, \ldots, m - 1$:

- $\text{Leq}_x[i] = \{\max\{k \mid k < i, x[i] = x[k]\}\} \ (\emptyset$ if there is no eligible $k$),
- $\text{Lmax}_x[i] = \{\max\{\text{argmax}_k\{x[k] \mid k < i, x[i] > x[k]\}\}\} \ (\emptyset$ if there is no eligible $k$),
- $\text{Lmin}_x[i] = \{\max\{\text{argmin}_k\{x[k] \mid k < i, x[i] < x[k]\}\}\} \ (\emptyset$ if there is no eligible $k$).

$\text{Leq}$, $\text{Lmax}$ and $\text{Lmin}$ capture $=$, $>$ and $<$ relationships between each character $x[i]$ in $x$ and the closest preceding character $x[k]$. These orders can be inferred in linear time for alphabets of size $m^{O(1)}$ and in $O(m \log m)$ time for other alphabets by answering the “all nearest smaller values” task on the sorted indexes [8]. Figure 1 depicts $\text{Leq}$, $\text{Lmax}$ and $\text{Lmin}$ for $x = (1, 4, 3, 1)$. Given determinate strings $x$ and $y$, $A = \text{Leq}_x[t+1]$, $B = \text{Lmax}_x[t+1]$ and $C = \text{Lmin}_x[t+1]$, if $x[0..t] \approx y[0..t]$, then $x[0..t + 1] \approx y[0..t + 1]$ if and only if

$$\forall a \in A \ (y[t + 1] = y[a]) \land \forall b \in B \ (y[t + 1] > y[b]) \land \forall c \in C \ (y[t + 1] < y[c]).$$

When allowing uncertainties between pairs of characters, previous research on the OPPM problem cannot be straightforwardly extended due to the need to trace $O(2^m)$ assignments on indeterminate strings.
Lemma 3.4. Given a determinate string $x$, an indeterminate string $y$, and the singleton sets $A = \text{Leq}_x[0..t+1]$, $B = \text{Lmax}_x[0..t+1]$ and $C = \text{Lmin}_x[0..t+1]$ containing a position in $\{0, \ldots , t\}$. If $x[0..t] \approx y[0..t]$ is verified on a specific assignment of $y$ characters, denoted $\$y$, then $x[0..t+1] \approx y[0..t+1]$ if and only if

$$
\exists y[t+1] \in \$y[t+1], \forall a \in A \exists y[a] \in \$y[a], \forall b \in B \exists y[b] \in \$y[b], \forall c \in C \exists y[c] \in \$y[c] \\
y[t+1] = \$y[a] \land \$y[t+1] > \$y[b] \land \$y[t+1] < \$y[c]
$$

Proof. ($\Rightarrow$) In accordance with $\text{Leq}$, $\text{Lmax}$ and $\text{Lmin}$ definition, for any $a \in A$, $b \in B$ and $c \in C$ we have $x[t+1] = x[a]$, $x[t+1] > x[b]$ and $x[t+1] < x[c]$. If there is an assignment to $y[0..t+1]$ in $\$y$ that preserves the orders of $x[0..t+1]$, then for each $a \in A$, $b \in B$ and $c \in C$ $y[t+1] = \$y[a]$, $y[t+1] > \$y[b]$ and $y[t+1] < \$y[c]$ (where $\$y[0..t] = \$y[a]$, $\$y[0..t] > \$y[b]$ and $\$y[0..t] < \$y[c]$). ($\Leftarrow$) We need to show that $x[0..t+1] \approx y[0..t+1]$. Since $x[0..t] \approx y[0..t]$, for $i < t$, $\exists y[i] \in \$y[i]$, $\exists y[t+1] \in \$y[t+1]$: $x[t+1] > x[i]$ implies $y[t+1] > y[i]$. Assuming $x[t+1] > x[i]$ for some $i \in \{0, \ldots , t\}$: by the definition of $\text{Lmax}$, $\forall b \in B x[b] > x[i]$; by the order-isomorphism of $x[0..t]$ and $y[0..t]$ in $\$y[0..t]$, there is $y[i] \in \$y[i]$ and $y[b] \in \$y[b]$ that $\forall b \in B y[b] > y[i]$; and by the assumption of the lemma, $\forall b \in B y[t+1] > y[b]$; hence $y[t+1] > y[i]$. Similarly, $x[t+1] < x[i]$ (and $x[t+1] = x[i]$) implies $y[t+1] < y[i]$ (and $y[t+1] = y[i]$), yielding the stated equivalence. \hfill \Box

Given two strings of equal length, the $\mu\text{OPPM}$ problem can be schematically represented according to the identified order restrictions. Figure 2 represents restrictions on the indeterminate string $y = (2, 4, 5, 3, 5, 1, 2)$ in accordance with the observed orders in $x = (1, 4, 3, 1)$. The left side edges are placed in accordance with Lemma 3.4 and capture assessments on the orders between pairs of characters. The right side edges capture incompatibilities detected after the assessments, i.e. pairs of characters that cannot be selected simultaneously (for instance, $y[0] = 2$ and $y[3] = 1$, or $y[1] = 4$ and $y[2] = 5$). For the given example, there are two valid assignments, $\$y_1 = (2, 4, 3, 2)$ and $\$y_2 = (2, 5, 3, 2)$, that satisfy $x[0] = x[3] < x[2] < x[1]$, thus $y$ op-matches $x$.

To verify whether there is an assignment that satisfies the identified ordering restrictions, we propose the reduction of $\mu\text{OPPM}$ problem to a Boolean satisfiability problem.

Given a set of Boolean variables, a formula in conjunctive normal form is a conjunction of clauses, where each clause is a disjunction of literals, and a literal corresponds to a variable or its negation. Let a 2CNF formula be a formula in the conjunctive normal form with at most two literals per clause. Given a CNF formula, the satisfiability (SAT) problem is to verify if there is an assigning of values to the Boolean variables such that the CNF formula is satisfied.

Theorem 3.5. The $\mu\text{OPPM}$ problem over two strings of equal length, one being indeterminate, can be reduced to a satisfiability problem with the following CNF
Let us show that if \( \phi \) is satisfiable, then \( x \) is assigned a value that \( \phi \) is not satisfiable. (\( \Rightarrow \)) When \( x \approx y \), there is an assignment of values to \( y \) such that \( \phi \) is satisfiable. \( \phi \) is satisfiable if there is at least one variable assigned to true per clause \( \lor_{y[i] \in y[i]} z_i \cdot s_y[i] \) given conflicts \( \neg z_i \cdot s_y[i] \lor \neg z_j \cdot s_y[j] \). \( \phi \) is satisfiable if there is at least one variable assigned to true per clause \( \lor_{y[i] \in y[i]} z_i \cdot s_y[i] \) given conflicts \( \neg z_i \cdot s_y[i] \lor \neg z_j \cdot s_y[j] \). \( \phi \) is satisfiable if there is at least one variable assigned to true per clause \( \lor_{y[i] \in y[i]} z_i \cdot s_y[i] \) given conflicts \( \neg z_i \cdot s_y[i] \lor \neg z_j \cdot s_y[j] \).

**Proof.** Let us show that if \( x \) is a match of \( y \) then \( \phi \) is satisfiable, and if \( x \) does not match \( y \) then \( \phi \) is not satisfiable. (\( \Rightarrow \)) When \( x \approx y \), there is an assignment of values to \( y \) such that \( \phi \) is satisfiable. \( \phi \) is satisfiable if there is at least one variable assigned to true per clause \( \lor_{y[i] \in y[i]} z_i \cdot s_y[i] \) given conflicts \( \neg z_i \cdot s_y[i] \lor \neg z_j \cdot s_y[j] \). As conflicts do not prevent the existence of a valid assignment (by assumption), then \( \exists y \land \forall i \in \{0, m-1\} \neg z_i \cdot s_y[i] \) and \( \phi \) is satisfiable. (\( \Leftarrow \)) When \( x \) does not match \( y \), there is no assignment of values \( y \in y \) that can satisfy the orders of \( x \). Per formulation, the conflicts \( \neg z_i \cdot s_y[i] \lor \neg z_j \cdot s_y[j] \) prevent the satisfiability of one or more clauses \( \lor_{y[i] \in y[i]} z_i \cdot s_y[i] \), leading to a non-satisfiable formula.

If the established \( \phi \) formula is satisfiable, there is a Boolean assignment to the variables that specify an assignment of characters in \( y \), \( s_y \), preserving the orders of \( x \) (as defined by \( \text{Leq}, \text{Lmax}, \text{Lmin} \)). Otherwise, it is not possible to select an assignment \( s_y \) matching \( x \). \( \phi \) has at most \( r \times m \) variables,
Theorem 3.6. Given two strings of length $|φ|$, the associated CNF formula is:

$$\{z_i, \sigma \mid i \in \{0..m - 1\}, \sigma \in \Sigma\}$$. The Boolean value assigned to a variable $z_{i, \sigma}$ simply defines that the associated character $\sigma$ from $y[i]$ can be either considered (when $\text{true}$) or not (when $\text{false}$) to compose a valid assignment $\$y$ that matches the given determinate string $x$. The reduced formula in (1) is composed of two major types of clauses: $\forall y[i] \in y[i] z_{i, y[i]}$, and $(\neg z_{i, y[i]} \lor \neg z_{i, y[j]} \lor \text{bool})$ where $\text{bool}$ is either given by $y[i] = y[j]$, $y[i] < y[j]$ or $y[i] > y[j]$. Clauses of the first type specify the need to select at least one character per position in $y$ to guarantee the presence of valid assignments. The remaining clauses specify ordering constraints between characters. If an inequality, such as $y[i] > y[j]$, is assessed as true, the associated clause is removed. Otherwise, $(\neg z_{i, \sigma_1} \lor \neg z_{j, \sigma_2})$ is derived, meaning that these $\sigma_1$ and $\sigma_2$ characters should not be selected simultaneously since they do not satisfy the orders defined by a given pattern. For instance, the pairs of characters in orange from Figure 2 should not be simultaneously selected due to order conflicts. To this end, $(\neg z_{0, 2} \lor \neg z_{3, 1})$ and $(\neg z_{1, 4} \lor \neg z_{2, 5})$ clauses need to be included to verify if $y \approx x$. Considering $y = (2, 4\{5, 4\{5, 1\}2)$ and $x = (1, 4, 3, 1)$, schematically represented in Figure 2, the associated CNF formula is:

$$\phi = z_{0, 2} \land (z_{1, 4} \lor z_{1, 5}) \land (z_{2, 4} \lor z_{2, 5}) \land (z_{3, 1} \lor z_{3, 2}) \land (\neg z_{0, 2} \lor \neg z_{3, 1}) \land (\neg z_{1, 4} \lor \neg z_{2, 5})$$

Theorem 3.7. Given two strings of length $m$, one being indeterminate with $r = 2$, the $\mu\text{OPPM}$ problem can be reduced to a 2SAT problem with a CNF formula with $O(m)$ size.

Proof. Given Theorem 3.6 and the fact that the reduced CNF formula has at most two literals per clause – $\phi$ is a composition of $\forall y[i] \in y[i] z_{i, y[i]}$ clauses with $|y[i]| = \{1, 2\}$ and $(\neg z_{i, y[i]} \lor \neg z_{i, y[j]} \lor \text{bool})$ clauses – $\mu\text{OPPM}$ with $r = 2$ and one indeterminate string is reducible to 2SAT. The reduced formula has at most 10$m$ clauses with 2 literals each, being linear in $m$:

- clauses that impose the selection of at least one character per position in $y$] Since $y$ has $m$ positions, and each position is either determinate (unitary clause) or defines an uncertainty between a pair of characters, there are $m$ clauses and at most 2$m$ literals;

- clauses that define the ordering restrictions between two variables] A position in the indeterminate string $y[i]$ needs to satisfy at most two order relations. Considering that $i$, $\text{Leq}[i]$, $\text{Lmax}[i]$ and $\text{Lmin}[i]$ specify uncertainties between pairs of characters, there are up to 12 restrictions per position: 4 ordering restrictions between characters in $y[i]$ and $y[\text{Leq}[i]]$, $y[\text{Lmax}[i]]$ and $y[\text{Lmin}[i]]$. Whenever the order between two characters is not satisfied, a clause is added per position, leading to at most 12$m$ clauses.

Theorem 3.7. The $\mu\text{OPPM}$ between determinate and indeterminate strings of equal length can be solved in linear time when $r = 2$. 

10
Given indeterminate strings, previous contributions are insufficient to answer the $\muOPPM$ problem. In this context, the $Leq$, $Lmax$ and $Lmin$ vectors need to be redefined to be inferred from an indeterminate string:

**Definition 3.8.** For $i = 0, \ldots, m - 1$:

- $Leq_x[i][j] = \{ k \mid k < i, \exists_p \, x_j[i] = x_p[k] \}$ ($\emptyset$ if there is no eligible $k$),
- $Lmax_x[i][j] = \{ k \mid k < i, \exists_p \, x_j[i] > x_p[k] \}$ ($\emptyset$ if there is no eligible $k$),
- $Lmin_x[i][j] = \{ k \mid k < i, \exists_p \, x_j[i] < x_p[k] \}$ ($\emptyset$ if there is no eligible $k$).

Figure 3 schematically represents the order relationships of $x = (2, 1|3, 3)$ and the associated $Leq$, $Lmax$ and $Lmin$ vectors. In this scenario, $x[2]$ needs to be verified not only against $x_0[1]$ but also against $x_1[1]$ in case $x_0[1]$ is disregarded.

**Remark 3.9.** Given $Leq$, $Lmax$ and $Lmin$ (Definition 3.8), there are $O((rm)^2)$ order relationships when $r \in \mathbb{N}^+$ since each character in a given position establishes at most $O(m)$ relationships with characters in preceding positions.

**Lemma 3.10.** Given indeterminate strings $x$ and $y$, let $A_j = Leq_x[t + 1][j]$, $B_j = Lmax_x[t + 1][j]$ and $C_j = Lmin_x[t + 1][j]$ (Definition 3.8) be the orders associated with $x_j[t + 1]$. If $x[1..t] \approx y[1..t]$ is verified on a partial assignment of $y$ characters, denoted by $\triangledown y$, then $x[1..t + 1] \approx y[1..t + 1]$ if and only if

$$\exists j \in \{0, 1\} \exists y[t + 1] \forall a \in A_j, b \in B_j, c \in C_j \exists y[a] \in y[a], y[b] \in y[b], y[c] \in y[c]
\begin{align*}
&y[t + 1] = y[a] \lor y[b] \lor y[c] \\
&y[t + 1] = y[a] \land y[t + 1] > y[b] \land y[t + 1] < y[c]
\end{align*}$$

**Proof.** ($\Rightarrow$) Similar to the proof of Lemma 3.4 yet $A$, $B$ and $C$ conditional to $x[t + 1]$ (Definition 3.3) are now given by $A_j$, $B_j$ and $C_j$ conditional to $x_j[t + 1]$ (Definition 3.8). If there is an assignment to $y_[1..t + 1]$ in $\triangledown y$ that preserves one

---

1SAT problems have linear time and space solutions on the size of the input formula. Consider for instance the original proposal [44], the formula $\phi$ is modeled by a directed graph $G = (V, E)$, with two nodes per variable $z_i$ in $\phi$ ($z_i$ and $\neg z_i$) and two directed edges for each clause $z_i \lor z_j$ (the equivalent implicative forms $\neg z_i \Rightarrow z_j$ and $\neg z_j \Rightarrow z_i$). Given $G$, the strongly connected components (SCCs) of $G$ can be discovered in $O(|V| + |E|)$. During the traversal if a variable and its complement belong to the same SCC, then the procedure stops as $\phi$ is determined to be unsatisfiable. Given the fact that both $|V| = O(m)$ and $|E| = O(m)$ by Lemma 3.6 this procedure is $O(m)$ time and space.
Figure 3: Order relationships of $x = (2, 1|3, 3)$ and the corresponding $Leq$, $Lmax$ and $Lmin$ vectors.

Figure 4: Conflicts when op-matching $y = (2, 0, 3|4)$ against $x = (2, 1|3, 3)$.

of the possible orders in $x_{1..t+1}$, then for any $a \in A_j$, $b \in B_j$ and $c \in C_j$: $y_{t+1} = y[a]$, $y[t+1] > y[b]$ and $y[t+1] < y[c]$ (where $y[t+1] \in y[t+1]$, $y[a] \in y[a]$, $y[b] \in y[b]$, and $y[c] \in y[c]$).

(⇐) We need to show that $x_{1..t+1} \approx y_{1..t+1}$. Since $x_{1..t} \approx y_{1..t}$, it is sufficient to prove that for $i \leq t$: exists $x[i] \in y[i]$, $x[t+1] \in y[t+1]$, $y[i] \in y[i]$, and $y[t+1] \in y[t+1]$ such that $x[t+1] = y[i] \iff y[t+1] = y[i]$, $x[t+1] > y[i] \iff y[t+1] > y[i]$ and $x[t+1] < y[i] \iff y[t+1] < y[i]$. This results from Definition 3.8 the order-isomorphism property and Lemma 3.4.

Figure 4 represents encountered restrictions when op-matching $x = (2, 1|3, 3)$ against $y = (2, 0, 3|4)$. The right side edges capture the detected incompatibilities, i.e. pairs of characters that cannot be selected simultaneously. For the given example, there are 2 valid assignments – $y_1 = (2, 0, 3)$ and $y_2 = (2, 0, 4)$ – satisfying $x_0[1] < y_0[0] < x_0[2]$, thus $x \approx y$.

To verify whether there is an assignment that satisfies the identified ordering restrictions, Theorem 3.11 extends the previously introduced SAT mapping.
assigned to these variables define whether characters \( \sigma \) belong to an op-match. The reduced formula is composed of two major types of assignments such that

**Theorem 3.11.** Given \( \text{Leq}, \text{Lmax} \) and \( \text{Lmin} \) (Definition 3.8), \( \mu \text{OPPM} \) problem over two indeterminate strings of equal length can be reduced to a satisfiability problem with the following CNF formula:

\[
\phi = \bigwedge_{i=0}^{m-1} z_i, x[i] \in x[i], y[i] \in y[i] \\
\wedge \bigwedge_{i=0}^{m-1} \bigg( \bigwedge_{j \in \text{Leq}[i]} \bigwedge_{y[j] \in y[j]} \left( \neg z_i, x[i], y[i] \lor \neg z_j, x[j], y[j] \lor y[i] = y[j] \right) \\
\wedge \bigwedge_{j \in \text{Lmax}[i]} \bigwedge_{y[j] \in y[j]} \left( \neg z_i, x[i], y[i] \lor \neg z_j, x[j], y[j] \lor y[i] > y[j] \right) \\
\wedge \bigwedge_{j \in \text{Lmin}[i]} \bigwedge_{y[j] \in y[j]} \left( \neg z_i, x[i], y[i] \lor \neg z_j, x[j], y[j] \lor y[i] < y[j] \right) \bigg)
\]

(2)

**Proof.** If \( x \approx y \) then \( \phi \) is satisfiable, and if \( x \) does not op-match \( y \) then \( \phi \) is not satisfiable.

(\( \Rightarrow \)) When \( x \) op-matches \( y \), there is an assignment of values in \( x \) and \( y \) such that \( x[i] \approx y[i] \). \( \phi \) is satisfiable if there is at least one valid assignment \( z_i, x[i], y[i] \) per \( i^{th} \) position. As conflicts \( \neg z_i, x[i], y[i] \lor \neg z_j, x[j], y[j] \) do not prevent the existence of a valid assignment (by assumption), one or more variables \( z_i, x[i], y[i] \) can be selected per position. \( \phi \) can then be satisfied by fixing a single variable \( z_i, x[i], y[i] \) per \( i^{th} \) position as true and the remaining variables as false. (\( \Leftarrow \)) When \( x \) does not op-match \( y \), there is no assignment of values \( x \in x \) and \( y \in y \) such that \( x \approx y \). Per formulation, in the absence of an order-preserving match, conflicts will prevent the assignment of at least one variable \( z_i, x[i], y[i] \) per \( i^{th} \) position, thus making \( \phi \) formula unsat.

If the formula in (2) is satisfiable, there is a Boolean assignment to the variables such that there is an assignment of characters in \( y \), \( \$y \), and in \( x \), \( \$x \), such that both strings op-match. Otherwise, it is not possible to select assignments such that \( x \approx y \). Given \( r = 2 \), the established \( \phi \) formula has at most \( 4m \) variables, \( \{ z_i, \sigma_1, \sigma_2 \mid i \in \{0 \ldots m-1\}, \sigma_1, \sigma_2 \in \Sigma \} \). The Boolean values assigned to these variables define whether characters \( x[i] \) and \( y[i] \) belong to an op-match. The reduced formula is composed of two major types of clauses:

- Those in the first line of (2) ensure that at least one combination of characters, \( \$x[i] \) and \( \$y[i] \), should be selected per \( i^{th} \) position.
• Remaining ones in (2) specify ordering constraints between pairs of characters \( \sigma_1 \in y[i] \) and \( y[\text{Leq}[i]] \), \( y[\text{Lmax}[i]] \) and \( y[\text{Lmin}[i]] \); if the inequalities \( y[i] = y[j] \), \( y[i] > y[j] \) and \( y[i] < y[j] \) are assessed as false, then it leads to clauses of the form \( \neg z_{i,\sigma_1} \lor \neg z_{j,\sigma_2} \), meaning that these characters should not be selected simultaneously in the given positions (see Figure 4).

To instantiate the proposed mapping, consider \( x = (2,1|3,3) \) and \( y = (2,0,3|4) \), schematically represented in Figure 3. The associated CNF formula is:

\[
\phi = z_{0,2,2} \land (z_{1,1,0} \lor z_{1,3,0}) \land (z_{2,3,3} \lor z_{2,3,4}) \\
\land (\neg z_{0,2,0} \lor \neg z_{1,3,0}) \land (\neg z_{1,3,0} \lor \neg z_{2,3,3}) \land (\neg z_{1,3,0} \lor \neg z_{2,3,4})
\]

**Theorem 3.12.** The \( \mu\text{OPPM} \) problem for two indeterminate strings of equal length is reducible into a satisfiability problem over a CNF formula with size \( O((m r)^2) \).

**Proof.** The reduced formula in (2) is in the two conjunctive normal form (CNF) with at most 4m clauses in the first line of (2) and a maximum of \( O(m r) \) orders per position (Remark 3.9), totalling at most \( O((m r)^2) \) order conflicts between characters, from the restriction clauses in the reammining of (2).

Although we are no longer in the conditions of Theorem 3.7 namely because the above satisfiability formulation is not a 2SAT instance, given its unique properties, effective backtracking in accordance with the clauses in the first line of (2), as well as dedicated conflict pruning principles derived from reammining clauses in (2), can be considered to develop efficient SAT solvers able to solve the \( \mu\text{OPPM} \) problem. And, as we will show later, we are not expected to do much better.

### 3.3. Polynomial time Alternate-\( \mu\text{OPPM} \)

In this section, we define Alternate-\( \mu\text{OPPM} \) as the subproblem of \( \mu\text{OPPM} \) where both strings \( (x \text{ and } y, \text{ interchangeable}) \) may have indeterminate characters, but never in the same position; we show that Alternate-\( \mu\text{OPPM} \) is polynomial in both the number of indeterminacies \( r \) (which may be different in each position and string) and length of the strings \( m \). To do this, we will present a set of 2SAT clauses, in the form of implications, that can represent every constraint of this problem. We will first assume that there are no repeated characters within each string and then extend the reduction to handle equalities.

Given a string \( x \) and position \( i \), we represent the set of indeterminate characters \( x[i] \) as the ascending sequence \( a_0|...|a_{r_i-1} \) where \( \forall j \ a_j \in x[i] \) and \( |x[i]| = r_i \). We will use only \( r \) when the context leads to no ambiguities, or to mean the largest possible \( r_i \). All of our 2SAT variables will be of the form \( g_{a_j} \), meaning that the chosen value \( $x[i] \) is greater than or equal to \( a_j \).
Consistency clauses. Here, we describe the clauses that maintain consistency between all the $q$ variables for individual positions. We only need to specify that, if we have chosen a value greater than $a_i$, we have also chosen a value greater than $a_{i-1}$, the value immediately below it, i.e.,

$$\forall i \in [1, r-1] (g_{a_i} \implies g_{a_{i-1}}).$$

This leads to a single clause per indeterminacy, per position, for both pattern and text, and so, at most, $2mr = O(mr)$ clauses.

Order clauses (Type 1). Here, we describe the clauses enforcing the order relation between each pair of positions. Given two strings $x$ and $y$, for positions $\alpha$ and $\beta$, if $x[\alpha] > x[\beta]$, then $y[\alpha] > y[\beta]$ (and the same for the $<$ relation).

This first set of clauses applies to Type 1 (see Table 1). We only need to find the index (in each string) that separates the cases where $x[\alpha] > x[\beta]$ from the cases where $x[\alpha] < x[\beta]$ and add a single constraint expressing it.

Let $i$ be the lowest index such that $b_i > a$ and $j$ the lowest index such that $a_j \geq b$, where $a$ and $b$ are as in Table 1. Then, we have

$$g_{b_i} \implies \neg g_{a_j},$$
$$\neg g_{b_i} \implies g_{a_j}.$$

This leads to two clauses for every pair of positions, and so, $O(m^2)$ clauses.

Order clauses (Type 2). Finally, we have a second set of clauses that applies to Type 2 (see Table 2). Here, we have the order between $\alpha$ and $\beta$ fixed already by whichever string $x$ or $y$ has no indeterminacies.

If $a > b$, for every index $i$ indexing $b_i$, and let $j$ be the lowest index such that $a_j > b_i$. Then we add

$$g_{b_i} \implies g_{a_j},$$

If there is no such $j$, we add instead

$$\neg g_{b_i}.$$
Similarly, if \( a < b \), for every index \( i \) indexing \( a_i \), let \( j \) be the lowest index such that \( a_i < b_j \). Then we add

\[
g_{a_i} \implies g_{b_j}.
\]

If there is no such \( j \), we add instead

\[
\neg g_{a_i}.
\]

This leads to at most \( r \) clauses for every pair of positions, and so \( O(rm^2) \) clauses. Because character order is a transitive property, this type of clauses may be reduced to \( O(rm) \) using a similar notion to the \( L_{max} \) and \( L_{min} \) sets introduced in Section 3.2 to consider only “adjacent” (taking adjacent to mean the closest position of the same type) pairs of positions, instead of every pair.

**Forcing choice.** With the clauses specified above, we can find coherent solutions to the problem. However, it is possible to satisfy the formula by assigning all possible values for a given variable to false (effectively skipping the position). This has a straightforward solution, given the chosen encoding of the variables. Each 2SAT variable represents a greater or equal value in the corresponding OPPM position, the variable corresponding to the lowest value for each position is trivially true, letting us force a value choice with a single added variable. For every position, with variables \( g_0, ..., g_{r_i} \), we add the clause \( g_0 \), forcing it to be true to satisfy the 2SAT formula.

**Extracting solutions.** Finally, we need to extract the solution to the OPPM problem from the 2SAT solution. This is easily done in linear time by sweeping every variable in ascending order, in each position. In each position, with variables \( g_0, ..., g_{r_i} \), we find the variable at index \( j \) such that \( g_j \) is true and \( g_{j+1} \) is false. The chosen value in the OPPM problem, for the given position, is the value at index \( j \).

**Dealing with equalities.** We now turn to cases where characters match and show how to adapt the encoding above to equalities. Let us consider Type II equalities, first, where \( a = b \). The easy solution to this is the same as the one presented before. We preprocess the two strings by grouping all the repeats into a single position and intersecting their indeterminacies. For Type I equalities, we need to add 4 clauses to each pair. Let \( i, j \) be indexes such that \( a = b_i \) and \( b = a_j \). We add

\[
\begin{align*}
g_{b_i} & \implies g_{a_j}, \\
g_{a_j} & \implies g_{b_i}, \\
\neg g_{b_{i+1}} & \implies \neg g_{a_{j+1}}, \\
\neg g_{a_{j+1}} & \implies \neg g_{b_{i+1}}.
\end{align*}
\]

If only \( i \) exists (or \( j \)), we simply remove \( b_i \) (or \( a_j \)) from the input, as such an assignment could never lead to a valid solution.
Pair incompatibility. All the clauses described above serve to maintain consistency between pairs. It may happen that a given pair is unsatisfiable by itself, and no clauses would be constructed. These cases can be dealt separately, as pre-processing. If we find a pair that can not be satisfied, we can terminate the program before ending the construction, since there is no solution to the OPPM instance.

Theorem 3.13. The Alternate-\(\mu\)OPPM can be solved in \(O(rm^2)\) time and space.

Proof. Property resulting from the encoding above and, as in the proof of Theorem 3.7, given the fact that a 2SAT problem can be solved in linear time \[14\].

3.4. \(\mu\)OPPM with 3 indeterminacies in both text and pattern is \(\text{NP}-\text{hard}\)

In this section, we define \(\{3, 3\}\)-\(\mu\)OPPM as the subproblem of \(\mu\)OPPM where both the pattern and the text have indeterminate characters in any position (although at least one position must have at least three indeterminate characters in both pattern and text) and prove it \(\text{NP}-\text{hard}\) (thus proving the same for general \(\mu\)OPPM). We do this with a direct reduction from 3CNF-SAT, first presenting the construction and then the proof of equivalence between the two instances. The construction is similar to the one by Bose et al. for the permutation matching problem \[13\].

Construction. To ease the description of the construction itself, we start by describing how we represent an instance of 3CNF-SAT. First, we assume that every literal and clause has some ordering. We have a set \(V\) of literals, and a set \(C\) of clauses. Each clause \(c\) is represented by two tuples, \((z_{c,0}, z_{c,1}, z_{c,2})\) and \((l_{c,0}, l_{c,1}, l_{c,2})\). \(z_{c,i} \in \{0, \ldots, |V| - 1\}\) represents the index of literal \(i\) of clause \(c\); \(l_{c,j} \in \{0, 1\}\) represents the value of the literal \(i\) in clause \(c\), having the value of 0 for positive literals and 1 for negative literals. For example, the clause \((v_1 \lor \neg v_2, \lor v_5)\) would be represented by the two tuples \(z = (1, 2, 5)\) and \(l = (0, 1, 0)\).

Although the designations of text or pattern are interchangeable in this section, we will use pattern for the simpler string (with less indeterminacies) and text for the more complicated string (with more indeterminacies). We use \(p\) and \(t\) for the pattern and text, respectively, or \(s\) when they are interchangeable.

Both text and pattern have two parts, one representing literals and the other representing clauses. Each literal, and clause, has a single position in each string to represent it, dividing \(s\) into \(sv = s[0..|V| - 1]\) and \(sc = s[|V|..|V| + |C| - 1]\). In \(pv\), we have a simple sequence of literals given by their indexes, so \(p[i] = i + 1\), for \(i \in \{0, \ldots, |V| - 1\}\); in \(tv\) we have a similar sequence, but each literal takes one of two variable values to represent an assignment of \(\text{true}\) or \(\text{false}\), so \(t[i] = 2 \times (i + 1)\) or \(2 \times (i + 1) - 1\). We choose the larger value to represent the assignment of \(\text{true}\). In \(sc\), each position has three indeterminacies, corresponding to the three variables of the clause. In \(pc\), we choose one of the three literals of the
Table 3: \( \mu \)OPPM instance corresponding to the 3CNF-SAT formula \((z_1 \lor \neg z_2 \lor z_3) \land (\neg z_1 \lor z_2 \lor z_4)\).

| \(i\) | 0 | 1 | 2 | 3 | 4 | 5 |
|------|---|---|---|---|---|---|
| Formula | \(z_1\) | \(z_2\) | \(z_3\) | \(z_4\) | \(c_1\) | \(c_2\) |
| Pattern | 1 | 2 | 3 | 4 | 1|2|3 | 1|2|4 |
| Text | 1|2|3|4 | 5|6 | 7|8 | 2|3|6 | 1|4|8 |

respectively clause. For clause \(c\), with literals \(v_1, v_2, v_5\) (regardless of their value being positive or negative), its position in \(p\), \(p[|V| + c] = 1|2|5\). In \(t_C\), as in \(p_C\), we choose one of the literals, but now the value of the literal must satisfy the clause. For clause \(c\), \((v_1 \lor \neg v_2 \lor v_5)\), \(t[|V| + c] = 2 \times 1 - 0 \mid 2 \times 2 - 1 \mid 2 \times 5 - 0 = 2|3|10\). An example of this construction is shown in Table 3.

**Lemma 3.14.** The construction above takes polynomial time.

**Proof.** It is easy to see that, assuming that variables and clauses are numbered, we can simply scan the formula once to construct our two strings in linear time.

**Lemma 3.15.** The initial 3CNF-SAT clause is satisfiable if and only if there is an order-isomorphic match between the two constructed strings.

**Proof.** We start by showing how solving the \(\mu\)OPPM instance solves the initial 3CNF-SAT instance. To solve \(\mu\)OPPM, we need to choose exactly one value for each position in \(p\) and \(t\) that leads to two order-isomorphic strings. To extract the solution, we can limit ourselves to look at the initial part of \(t\), \(t[0,|V| - 1]\), which sets the value of each literal.

First, note that \(p\) function is to maintain consistency between the values of literals chosen in \(t\). By choosing only literals in \(p\), and not their values, we force equality between all such literals. Because of order-isomorphism, this equality must be kept in \(t\), forcing a valid solution to use a single value for each literal (since different values match in \(p\) but mismatch in \(t\)). If we choose a literal to be positive/negative at some position in \(t\), we force the value of that literal to be positive/negative at every position in \(t\).

Now, we focus on \(t_C\). Every clause has exactly one position in \(t_C\), and each of these positions have three choices of value, matching only the three values that satisfy a clause. Because we must choose one value in each position to solve our \(\mu\)OPPM instance, we must choose one value that satisfies each clause, for every clause.

Putting these two properties together, to solve \(\mu\)OPPM we must choose a literal value that satisfies each clause and those literals must have consistent values. This establishes the equivalence between the solutions of the two instances.

We can easily extract the solution from \(\mu\)OPPM to 3CNF-SAT by checking whether the values in \(t_V\) are even or odd, true or false, respectively. There is a unique solution to 3CNF-SAT given an \(\mu\)OPPM solution.
To extract the solution from 3CNF-SAT to \( \mu \)OPPM, we take the values assigned to each variable and choose the respective values in \( t_V \). Then, we need to choose values for \( p_C \) and \( t_C \), which can easily be done by choosing any of the literals that satisfies its respective clause. There may be multiple \( \mu \)OPPM solutions for a given 3CNF-SAT solution.

**Theorem 3.16.** \( \{3,3\}\)-\( \mu \)OPPM is \( \text{NP} \)-hard.

**Proof.** Using Lemmas 3.14 and 3.15 we show that 3CNF-SAT \( \leq_p \{3,3\}\)-\( \mu \)OPPM by constructing an instance of \( \mu \)OPPM in polynomial time. The solutions can also be retrieved and translated in polynomial time.

**Theorem 3.17.** \( \mu \)OPPM is \( \text{NP} \)-hard.

**Proof.** Since \( \{3,3\}\)-\( \mu \)OPPM is a particular case of \( \mu \)OPPM, and it is \( \text{NP} \)-hard, then OPPM is \( \text{NP} \)-hard.

## 4. Polynomial time \( \mu \)OPPM

**Lemma 4.1.** Given a pattern string of length \( m \) and a text string of length \( n \), one being indeterminate, the \( \mu \)OPPM problem can be solved in \( O(nmr \lg r) \) time.

**Proof.** From Theorem 3.2 verifying if two strings of length \( m \) op-match can be done in \( O(mr \lg r) \) time (indetermination in one string) since at most \( n - m + 1 \) verifications need to be performed.

Lemma 4.1 confirms that the \( \mu \)OPPM problem with one indeterminate strings is in class \( \text{P} \). This lemma further triggers the research question “Is \( O(nmr \lg r) \) a tight bound to solve the \( \mu \)OPPM?”, here left as an open research question.

Irrespective of the answer, the analysis of the average complexity is of complementary relevance. State-of-the-art research on the exact OPPM problem shows that the average performance of algorithms in \( O(nm) \) time can outperform linear time algorithms [12, 46, 47].

Motivated by the evidence gathered by these works, we suggest the use of filtration procedures to improve the average complexity of the proposed \( \mu \)OPPM algorithm while still preserving its complexity bounds. A filtration procedure encodes the input pattern and text, and relies on this encoding to efficiently find positions in the text with a high likelihood to op-match a given pattern. Despite the diversity of string encodings, simplistic binary encodings are considered to be the state-of-the-art in OPPM research [12, 46]. In accordance with Chhabra et al. [12], a pattern \( p \) can be mapped into a binary string \( p' \) expressing increases (1), equalities (0) and decreases (0) between subsequent positions. By searching for exact pattern matches of \( p' \) in an analogously transformed text string \( t' \), we guarantee that the verification of whether \( p[i..i + m - 1] \) and \( t[i..i + m - 1] \) orders are preserved is only performed when exact binary matches occur. Illustrating, given \( p = (3,1,2,4) \) and \( t = (2,4,3,5,7,1,4,8) \), then \( p' = (1,0,1,1) \) and
$t' = (1, 1, 0, 1, 0, 1, 1)$, revealing two matches $t'[1..4]$ and $t'[4..7]$: one spurious match $t[1..4]$ and one true match $t[4..7]$.

When handling indeterminate strings the concept of increase, equality and decrease needs to be redefined. Given an indeterminate string $x$, consider $x'[i] = 1$ if $\max(x[i]) < \min(x[i] + 1)$, $x'[i] = 0$ if $\min(x[i]) \geq \max(x[i] + 1)$, and $x'[i] = \ast$ otherwise. Under this encoding, the pattern matching problem is identical under the additional guard that a character in $p'$ always matches a do not care position, $t'[i] = \ast$, and vice-versa. Illustrating, given $p = (6, 2|3, 5)$ and $t = (3|4, 5, 6|8, 6|7, 3, 5, 4|6, 7|8, 4)$, then $p' = (0, 1)$ and $t' = (11\ast01\ast01)$, leading to one true match $t[3..5]$ – e.g. $t[3..5] = (6, 3, 5)$ – and one spurious match $t[5..7]$. Exact pattern matching algorithms, such as Knuth-Morris-Pratt and Boyer-Moore, can be adapted to consider do not care positions while preserving complexity bounds [14, 15].

The properties of the proposed encoding guarantee that the exact matches of $p'$ in $t'$ cannot skip any op-match of $p$ in $t$. Thus, when combining the premises of Lemma 4.1 with the previous observation, we guarantee that the computed µOPPM solution is sound.

The application of this simple filtration procedure prevents the recurring $O(mr \lg r)$ verifications $n - m + 1$ times. Instead, the complexity of the proposed method to solve the µOPPM problem becomes $O(dnr \lg r + n)$ (when one string is indeterminate) where $d$ is the number of exact matches ($d \ll n$). According to previous work on exact OPPM with filtration procedures [12], SBNDM2 and SBNDM4 algorithms [48] (Boyer-Moore variants) were suggested to match binary encodings. In the presence of small patterns, Fast Shift-Or (FSO) [49] can be alternatively applied [12].

A given string text can be read and encoded incrementally from the standard input as needed to perform µOPPM, thus requiring $O(mr)$ space. When filtration procedures are considered, the aforementioned algorithms for exact pattern matching require $O(m)$ space [12], thus µOPPM space requirements are bound by substring verifications (Section 3): $O(mr)$ space when one string is indeterminate and $O((mr)^2)$ when indetermination is considered on both strings.

5. Open problem

We can look at the µOPPM by the number and position of the indeterminate characters. We have shown that, for any number of indeterminacies, µOPPM has a polynomial-time algorithm for indeterminate characters in a single string (Section 3.1), or in both strings, but never in both strings at the same position (Section 3.3). For indeterminate characters in both strings at the same position, we have also shown that for at least three indeterminacies (at select positions), the problem in NP-hard (Section 3.4).

There is a gap in between these two groups, however, for the strings where there are at most two indeterminate characters in both strings at the same position. It remains open whether or not this problem is NP-hard. Given that our reduction from Section 5.4 uses three indeterminate character in both strings,
it also remains open whether the problem with two indeterminate characters in one string and three in the other (at the same position) is NP-hard.

Following the pattern-avoidance precedent by Guillemot and Vialette for the related problem of permutation matching, we note that, for the case of $\mu$OPPM with at most two indeterminate characters (both strings, same position), there is a straightforward encoding in 2SAT for $(1|3, 2|4)$-avoiding strings, here taken to mean that, in a single string, for the pair of positions $(i, j)$, the rank of the characters (only for the pair in question) is not $1|3$ in $i$ and $2|4$ in $j$ (with $i$ and let $j$ being interchangeable). The full problem, however, remains open.

6. Concluding remark

This work addressed the relevant yet scarcely studied problem of finding order-preserving pattern matches on indeterminate strings ($\mu$OPPM). We showed that the problem has a linear time and space solution when one string is indeterminate. In addition, the $\mu$OPPM problem (when both strings are indeterminate) was mapped into a satisfiability formula of polynomial size and two simple types of clauses in order to study efficient solvers for the $\mu$OPPM problem. Moreover the $\mu$OPPM problem was shown to be NP-hard in general. Finally, we showed that solvers of the $\mu$OPPM problem can be boosted in the presence of filtration procedures and we identified a still open problem in what concerns the computational complexity of the $\mu$OPPM problem when restricted to at most two indeterminate characters in both strings at the same position.

Acknowledgments. This work was developed in the context of a secondment granted by the BIRDS MASC RISE project funded in part by EU H2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no.690941. This work was further supported by national funds through Fundação para a Ciência e Tecnologia (FCT), namely under projects PTDC/CCI-BIO/29676/2017, TUBITAK/0004/2014, SAICTPAC/0021/2015, and UID/CEC/50021/2019.

References

[1] X. Ge, Pattern matching in financial time series data, final project report for ICS 278 (1998).

[2] J. Kim, P. Eades, R. Fleischer, S.-H. Hong, C. S. Iliopoulos, K. Park, S. J. Puglisi, T. Tokuyama, Order-preserving matching, Theoretical Computer Science 525 (2014) 68–79 (2014).

[3] R. Henriques, A. Paiva, Seven principles to mine flexible behavior from physiological signals for effective emotion recognition and description in affective interactions., in: PhyCS, 2014, pp. 75–82 (2014).
[4] R. Henriques, Learning from high-dimensional data using local descriptive models, Ph.D. thesis, Instituto Superior Tecnico, Universidade de Lisboa, Lisboa (2016).

[5] R. Henriques, S. C. Madeira, Bicspam: flexible biclustering using sequential patterns, BMC bioinformatics 15 (1) (2014) 130 (2014).

[6] R. Henriques, C. Antunes, S. Madeira, Methods for the efficient discovery of large item-indexable sequential patterns, in: New Frontiers in Mining Complex Patterns, Vol. 8399 of LNCS, Springer International Publishing, 2014, pp. 100–116 (2014).

[7] S. Kawashima, M. Kanehisa, Aaindex: amino acid index database, Nucleic acids research 28 (1) (2000) 374–374 (2000).

[8] M. Kubica, T. Kulczyński, J. Radoszewski, W. Rytter, T. Waleń, A linear time algorithm for consecutive permutation pattern matching, Information Processing Letters 113 (12) (2013) 430–433 (2013).

[9] S. Cho, J. C. Na, K. Park, J. S. Sim, A fast algorithm for order-preserving pattern matching, Information Processing Letters 115 (2) (2015) 397–402 (2015).

[10] S. Cho, J. C. Na, K. Park, J. S. Sim, Fast order-preserving pattern matching, in: Combinatorial Optimization and Applications, Springer, 2013, pp. 295–305 (2013).

[11] D. Belazzougui, A. Pierrot, M. Raffinot, S. Vialette, Single and multiple consecutive permutation motif search, in: Int. Symposium on Algorithms and Computation, Springer, 2013, pp. 66–77 (2013).

[12] T. Chhabra, J. Tarhio, A filtration method for order-preserving matching, Information Processing Letters 116 (2) (2016) 71–74 (2016). doi:http://dx.doi.org/10.1016/j.ipl.2015.10.005

[13] R. Henriques, A. P. Francisco, L. M. S. Russo, H. Bannai, Order-preserving pattern matching indeterminate strings, in: Proceedings of the Symposium on Combinatorial Pattern Matching (CPM), 2018 (2018).

[14] D. E. Knuth, J. H. Morris, Jr, V. R. Pratt, Fast pattern matching in strings, SIAM Journal on Computing 6 (2) (1977) 323–350 (1977).

[15] R. S. Boyer, J. S. Moore, A fast string searching algorithm, Communications of the ACM 20 (10) (1977) 762–772 (1977).

[16] T. Chhabra, S. Faro, M. O. Külekci, J. Tarhio, Engineering order-preserving pattern matching with simd parallelism, Softw. Pract. Exper. 47 (5) (2017) 731–739 (May 2017). doi:10.1002/spe.2433

[17] A. Amir, O. Lipsky, E. Porat, J. Umanski, Approximate matching in the l1 metric, in: CPM, Vol. 5, Springer, 2005, pp. 91–103 (2005).
[18] O. Lipsky, E. Porat, Approximate matching in the $l\infty$ metric, Information Processing Letters 105 (4) (2008) 138 – 140 (2008). doi:http://dx.doi.org/10.1016/j.ipl.2007.08.012

[19] A. Amir, Y. Aumann, P. Indyk, A. Levy, E. Porat, Efficient computations of $l1$ and $l\infty$ rearrangement distances, Theoretical Computer Science 410 (43) (2009) 4382 – 4390 (2009). doi:http://dx.doi.org/10.1016/j.tcs.2009.07.019

[20] E. Porat, K. Efremenko, Approximating general metric distances between a pattern and a text, in: ACM-SIAM Symposium on Discrete algorithms, SIAM, 2008, pp. 419–427 (2008).

[21] E. Cambouropoulos, M. Crochemore, C. Iliopoulos, L. Mouchard, Y. Pinzon, Algorithms for computing approximate repetitions in musical sequences, Int. Journal of Computer Mathematics 79 (11) (2002) 1135–1148 (2002).

[22] M. Crochemore, C. S. Iliopoulos, T. Lecroq, W. Plandowski, W. Rytter, Three heuristics for delta-matching: delta-bm algorithms, in: CPM, Springer, 2002, pp. 178–189 (2002).

[23] R. Clifford, C. Iliopoulos, Approximate string matching for music analysis, Soft Computing-A Fusion of Foundations, Methodologies and Applications 8 (9) (2004) 597–603 (2004).

[24] P. Clifford, R. Clifford, C. Iliopoulos, Faster algorithms for $\delta$, $\gamma$-matching and related problems, in: Annual Symposium on Combinatorial Pattern Matching, Springer, 2005, pp. 68–78 (2005).

[25] I. Lee, R. Clifford, S.-R. Kim, Algorithms on extended ($\delta$, $\gamma$)-matching, Computational Science and Its Applications-ICCSA 2006 (2006) 1137–1142 (2006).

[26] I. Lee, J. Mendivelso, Y. J. Pinzón, $\delta\gamma$–parameterized matching, in: International Symposium on String Processing and Information Retrieval, Springer, 2008, pp. 236–248 (2008).

[27] J. Mendivelso, I. Lee, Y. J. Pinzón, Approximate function matching under $\delta$-and $\gamma$-distances., in: SPIRE, Springer, 2012, pp. 348–359 (2012).

[28] J. Holub, W. Smyth, S. Wang, Fast pattern-matching on indeterminate strings, Journal of Discrete Algorithms 6 (1) (2008) 37 – 50, selected papers from AWOCA 2005 (2008). doi:http://dx.doi.org/10.1016/j.jda.2006.10.003

[29] R. Cole, C. Iliopoulos, T. Lecroq, W. Plandowski, W. Rytter, On special families of morphisms related to $\delta$-matching and don’t care symbols, Information Processing Letters 85 (5) (2003) 227–233 (2003).
[30] A. Apostolico, Algorithms and theory of computation handbook, Chapman & Hall/CRC, 2010, Ch. General Pattern Matching, pp. 15–15 (2010).

[31] B. S. Baker, A theory of parameterized pattern matching: algorithms and applications, in: ACM symposium on Theory of computing, ACM, 1993, pp. 71–80 (1993).

[32] A. Amir, M. Farach, S. Muthukrishnan, Alphabet dependence in parameterized matching, Information Processing Letters 49 (3) (1994) 111 – 115 (1994). doi:http://dx.doi.org/10.1016/0020-0190(94)90086-8

[33] A. Amir, M. Farach, Efficient 2-dimensional approximate matching of half-rectangular figures, Information and Computation 118 (1) (1995) 1 – 11 (1995). doi:http://dx.doi.org/10.1006/inco.1995.1047

[34] A. Amir, Y. Aumann, G. M. Landau, M. Lewenstein, N. Lewenstein, Pattern matching with swaps, Journal of Algorithms 37 (2) (2000) 247 – 266 (2000). doi:http://dx.doi.org/10.1006/jagm.2000.1120

[35] S. Muthukrishnan, New results and open problems related to non-standard stringology, in: Combinatorial Pattern Matching, Springer, 1995, pp. 298–317 (1995).

[36] D. Cantone, S. Cristofaro, S. Faro, An efficient algorithm for δ-approximate matching with α-bounded gaps in musical sequences, in: IW on Experimental and Efficient Algorithms, Springer, 2005, pp. 428–439 (2005).

[37] D. Cantone, S. Cristofaro, S. Faro, On tuning the (δ, α)-sequential-sampling algorithm for δ-approximate matching with alpha-bounded gaps in musical sequences., in: ISMIR, 2005, pp. 454–459 (2005).

[38] K. Fredriksson, S. Grabowski, Efficient algorithms for pattern matching with general gaps, character classes, and transposition invariance, Information Retrieval 11 (4) (2008) 335–357 (2008).

[39] A. Amir, R. Cole, R. Hariharan, M. Lewenstein, E. Porat, Overlap matching, Information and Computation 181 (1) (2003) 57 – 74 (2003). doi:http://dx.doi.org/10.1016/S0890-5401(02)00035-4

[40] A. Amir, Y. Aumann, M. Lewenstein, E. Porat, Function matching, SIAM Journal on Computing 35 (5) (2006) 1007–1022 (2006).

[41] A. Amir, I. Nor, Generalized function matching, Journal of Discrete Algorithms 5 (3) (2007) 514 – 523, selected papers from Ad Hoc Now 2005 (2007). doi:http://dx.doi.org/10.1016/j.jda.2006.10.001

[42] M. L. Fredman, On computing the length of longest increasing subsequences, Discrete Mathematics 11 (1) (1975) 29 – 35 (1975). doi:https://doi.org/10.1016/0012-365X(75)90103-X
[43] E. D. Demaine, A. López-Ortiz, J. I. Munro, Adaptive set intersections, unions, and differences, in: In Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA, Citeseer, 2000 (2000).

[44] B. Aspvall, M. F. Plass, R. E. Tarjan, A linear-time algorithm for testing the truth of certain quantified boolean formulas, Information Processing Letters 8 (3) (1979) 121–123 (1979).

[45] P. Bose, J. F. Buss, A. Lubiw, Pattern matching for permutations, Inf. Process. Lett. 65 (5) (1998) 277–283 (Mar. 1998). doi:10.1016/S0020-0190(97)00209-3

[46] D. Cantone, S. Faro, M. O. Külkici, An efficient skip-search approach to the order-preserving pattern matching problem., in: Stringology, 2015, pp. 22–35 (2015).

[47] T. Chhabra, M. O. Külkici, J. Tarhio, Alternative algorithms for order-preserving matching., in: Stringology, 2015, pp. 36–46 (2015).

[48] B. ˇDurian, J. Holub, H. Peltola, J. Tarhio, Improving practical exact string matching, Information Processing Letters 110 (4) (2010) 148–152 (2010).

[49] K. Fredriksson, S. Grabowski, Practical and optimal string matching, in: SPIRE, Vol. 3772, Springer, 2005, pp. 376–387 (2005).

[50] S. Guillemot, S. Vialette, Pattern matching for 321-avoiding permutations, 2009 (12 2009). doi:10.1007/978-3-642-10631-6_107