On Distances Between Words with Parameters

Pierre Bourhis
Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Aaron Boussidan
LIGM, Université Gustave Eiffel, CNRS, Marne-la-Vallée, France

Philippe Gambette
LIGM, Université Gustave Eiffel, CNRS, Marne-la-Vallée, France

Abstract

The edit distance between parameterized words is a generalization of the classical edit distance where it is allowed to map particular letters of the first word, called parameters, to parameters of the second word before computing the distance. This problem has been introduced in particular for detection of code duplication, and the notion of words with parameters has also been used with different semantics in other fields. The complexity of several variants of edit distances between parameterized words has been studied, however, the complexity of the most natural one, the Levenshtein distance, remained open.

In this paper, we solve this open question and close the exhaustive analysis of all cases of parameterized word matching and function matching, showing that these problems are NP-complete. To this aim, we also provide a comparison of the different problems, exhibiting several equivalences between them. We also provide and implement a MaxSAT encoding of the problem, as well as a simple FPT algorithm in the alphabet size, and study their efficiency on real data in the context of theater play structure comparison.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases String matching, edit distance, Levenshtein, parameterized matching, parameterized words, parameter words, instantiable words, NP-completeness, MAX-SAT

Digital Object Identifier 10.4230/LIPIcs.CPM.2023.6

Supplementary Material *Software (Source Code)*: `https://github.com/AaronFive/paramatch`
archived at `swh:1:dir:3a72b0d85a4a2be9126900473b8f3e6d03c12a52`

Funding ANR-18-CE23-0003 CQFD

1 Introduction

Measuring the similarity between text strings is a fundamental problem in computer science, and has applications in bioinformatics [23], databases [1, 16] and natural language processing [27]. Among the measures of similarities between strings, the Levenshtein distance [28] is the most commonly used, both for its practicality and its ease of computation. This distance quantifies the minimum number of operations of insertion, deletion, and substitution needed to transform a string into another one. It has a wide range of applications, ranging from biological sequence alignment [33] to dialect pronunciation differences [25] or signature authentication [34]. Computing the edit distance between two strings of length n and m can be achieved in time $O(nm)$, by computing the distance between all their prefixes, and storing the results in a dynamic programming fashion [37]. The success of the Levenshtein distance generated many extensions and generalization on more complex models, such as trees [38] or automata [32].

However, a limitation of the Levenshtein distance is that it only captures proximity between strings (or objects) written on the same alphabet. Evaluating the proximity of strings written on different alphabets is a problem that arises in various applications, such as bioinformatics [35], image processing [17] and code duplication [6, 7]. In all those contexts, the technique used is the one of parameterized matching [6, 7]. Instead of using classical strings, parameterized matching uses "parameterized words" written using both constant parts, which are expensive to rename, and parameters, which are meant to be renamed freely. Formally, two equal-length strings u and v over an alphabet Π are said to be parameterized matching if there exists a 1-to-1 function $f : \Pi \to \Pi$ such that $f(u) = v$, where $f(u)$ is defined as $f(u_1) \dots f(u_{|u|})$.

Words with parameters also occur in other frameworks, and are often used in slightly different ways. The first of those frameworks was initially introduced in the context of Ramsey theory in the 80s [36], and is called "parameter words". In this context, parameters are labelled according to their order of first occurrence. Parameter words are also equipped with a composition operation, where parameters of the first word can be instantiated by characters or parameters of the second word. Parameter words can also be seen as equivalence classes of parameterized words, which are the main focus of this article.

A second framework using parameters is the one of parameterized regular expressions introduced in [10], where parameters can only be instantiated by constants, and not by other parameters. The focus in this context is therefore made on the set of all possible valuations of the parameters. Then, when defining algorithmic problems on such objects, two distinct semantics can be studied: either the "certainty semantics", where all valuations need to have some property, or the "possibility semantics", where at least one valuation needs to have this property. To make a difference with the parameterized word framework mentioned below, we choose to call these words "instantiable words". Finally, this notion of words with parameters can also be seen as a refined version of partial words (words containing a wildcard character) [15]. The notion of partial words is also important in the context of databases where paths of incomplete graphs can be interpreted as instantiable words [9].

This article aims at studying similarity by using edit distances in the framework of words with parameters. In this framework, the pattern matching problem, which consists in looking for the first string as a subword of the second string, has been extensively studied, either looking for exact occurrences, with efficient algorithms [4, 19, 30] or approximate ones, which is often NP-hard [21, 22]. In the case where we compare the two input strings in their entirety, various exact parameterized matching problems have been studied for parameterized pattern matching, namely string parameterized matching [7], single pattern parameterized matching [7, 3], multiple pattern parameterized matching, or 2-dimensional parameterized matching, many of those works being compiled in [29] and [31]. Different approximate variants of parameterized matching using edit distance have already been studied, but the problem has not been completely solved: the first work on the topic is [8], in which Baker introduces a form of approximate parameterized pattern matching in which the replacement of any substring by another one that is in parameterized matching with it is considered as a base edit operation. Parameterized matching under the Hamming distance, i.e., with a distance allowing only substitutions, has been covered in [24], where the authors prove that the string matching problem with at most k mismatches can be solved in time $O(m + k^{1.5})$. The LCPS (Longest Common Parameterized Subsequence) problem, equivalent to the parameterized pattern matching problem with insertions and deletions, is shown to be NP-hard in [26], which also provides an approximation algorithm. Those two different complexity classes for these problems raise the question of the complexity of the problem under the Levenshtein distance. This problem was left as an open question in the conclusion of [24].

Our paper establishes that this problem is NP-complete. Moreover, the result also extends to any possible edit distances obtained from deletion, insertion, and substitution as soon as substitution is not the only operation allowed, as summarized in Figure 1. Our main

d	$\emptyset$	D	I	DI
$\emptyset$	P [8]	NP-complete (Th. 12)	NP-complete (Cor. 14)	NP-complete [26]
S	P [24]	NP-complete (Cor. 14)	NP-complete (Cor. 14)	NP-complete (Th. 13)

Figure 1 Complexity of the variants of parameterized matching PM^d, depending on the kind of operations (D: deletion, I: insertion, S: substitution) allowed in the edit distance d.

proof also implies the main theorem of [26] with a different NP-completeness reduction. This contrasts with the problems of exact parameterized pattern matching which are all polynomial-time solvable, as well as all variants of the string matching problem with deletions, insertions or substitutions.

We also extend these results to function matching, which is the problem obtained by relaxing the 1-to-1 restriction in parameterized matching, as defined in [2]. This generalization, by breaking the symmetry of parameterized matching, actually gives rise to two close but different problems, depending of the order of operations that are considered. We study the links between all these problems and their computational complexity, and study two practical ways to solve them, parameterized complexity and the use of maxSAT solvers.

We also make a direct connection with the framework of instantiable words, more precisely with a natural problem of distance between languages. We show how instantiable word problems can be reduced to parameterized matching ones, under the right assumptions. This allows us to open new perspectives on the complexity of several language repair problems.

In Section 2, we give basic definitions and notations, and recall the existing formalism of parameterized matching and instantiable words. In Section 3 we discuss approximate parameterized matching and its various generalizations. We also link it to instantiable words. In Section 4, we first prove a collection of technical results that build up to the NP-completeness proofs for parameterized matching and function matching problems defined above. In Section 5, we study two approaches to solve one of the variants of parameterized matching in practice, a simple FPT algorithm parameterized by the alphabet size and a MaxSAT encoding. We show in Section 6 that these implementations can solve real instances of the problem, motivated by structure comparison of theater plays.

Finally, in Section 7, we conclude the paper and give a few perspectives on the notion of distance between parameterized languages.

2 Notations and Definitions

2.1 Basic Notations on Words and Editions

Words

An **alphabet** is a set of letters. A word on an alphabet A is a finite sequence of letters from A, indexed starting from 1. Let u be a word on A. Unless defined differently, we note u_i the i-th letter of u, and $|u|$ is the length of u. If $i \notin [1, |u|]$, u_i is defined as the empty word ε. If x is a letter from A, $|u|_x$ is the number of times x appears in u. Similarly, if X is a set of letters, $|u|_X = \sum\limits_{x \in X} |u|_x$ is the number of occurrences of letters of X in u. If f is a function defined on an alphabet A, we extend it to A^* in the usual way, so that $f(u) = f(u_1) \dots f(u_{|u|})$. If f is a function, we denote by $\mathcal{D}(f)$ the domain of f. Two functions f and g are said to be **compatible** if $f|_{\mathcal{D}(g) \cap \mathcal{D}(f)} = g|_{\mathcal{D}(g) \cap \mathcal{D}(f)}$. The identity function on D is defined as $Id_D(x) = x$ for all x in D.

Edit Operations

In this paper, we consider the three classical **edit operations** which are *deletion*, *substitution* and *insertion*. Let $u = u_1 \dots u_n$ be a word of size n. Let i be an integer between 0 and n and x be a letter of the alphabet, the **insertion** at position i is the operation that transforms u to $u_1 \dots u_i x u_{i+1} \dots u_n$ Let j be an integer between 1 and n, the **deletion** at position j is the operation that transforms u into $u_1 \dots u_{j-1} u_{j+1} \dots u_n$. Let y be a letter of the alphabet and $y \neq u_j$, the **substitution** to y at position j is the operation that transforms u into $u_1 \dots u_{j-1} y u_{j+1} \dots u_n$. A **sequence of operations** or **rewriting sequence** ρ is a sequence of edit operations. We denote by $\rho(u)$ the word obtained by applying the edit operations of ρ one after another, in the order defined by ρ, to u.

Distances

Given a set of edit operations E and two words u and v, the **edit distance** between u and v is defined as the length of a shortest sequence of operations of E changing u into v. We denote by D the distance obtained on words by allowing only deletion operations: that is to say $D(u, v) = k$ iff v can be obtained by deleting k letters from u. Similarly, we note I and S the distances obtained by allowing only insertions and substitutions respectively (note that S is the Hamming distance). We also combine these notations to define DI as the distance with insertions and deletions, and so on. We also denote the Levenshtein distance DIS by L. Note that some of these edit distances are not metrics, because they are not symmetrical. We emphasize this by calling **symmetric edit distances** the distances DI, S, and L.

2.2 Comparing Words with Parameters

Conceptually, a word with parameters is a word in which some letters are not yet determined. In order to distinguish the parameters from the constants, we split the alphabet into Σ, the alphabet of the constants and Π, the alphabet of the parameters. By definition, these alphabets are finite. A word with parameters can either be seen as representing a "word template" (i.e., a word with variable parts), or a set of words (determined by all possible affectations of its parameters). Depending on the definition chosen, comparing two words w_1 and w_2 is done in two different ways. In the first setting [6, 7, 8, 31, 2, 5, 24, 29, 26, 17], parameters of w_1 are renamed through a function f that maps the set of parameters to itself, and acts as identity on the set of constants. It is then possible to compare $f(w_1)$ and w_2, which are written on the same alphabet. In the second setting, constants are seen as the concrete values parameters can take [11]. Parameters are instantiated through two functions f_1 and f_2 that map constants to themselves, but also map parameters to constants. The words $f_1(w_1)$ and $f_2(w_2)$ are then made only of constants, and can be compared. Formally, this gives rise to the two following different definitions:

On the one hand, a **parameterized word** is a word on an alphabet $\Sigma \cup \Pi$. In all that follows, Σ and Π are two disjoint alphabets, respectively called the **alphabet of constants** and the **alphabet of parameters**. Alphabets Σ and Π are considered to be finite, unless specified otherwise.

Two parameterized words u and v are said to be in **function matching** if there exist $f_\Pi : \Pi \to \Pi$ and $f : \Pi \cup \Sigma \to \Pi \cup \Sigma$ such that $f|_\Pi = f_\Pi$, $f|_\Sigma = Id_\Sigma$, and $f(u) = v$. In the classical setting [6], f is also **constrained to be 1-to-1**, and this relationship is called **parameterized matching**. Note that parameterized matching is an equivalence relation on parameterized words. Testing if two words are parameterized matching can be achieved in linear time [7].

PM^{DIS}

$$\begin{tikzcd}
u = aabba \arrow[d, "L : (u_1 \to b, u_2 \to b)"] \\
u' = bbbba \arrow[d, "f : [b \to a, a \to b]"] \\
v = aaaab
\end{tikzcd}$$

FM_1^{DIS}

$$\begin{tikzcd}
u = aabba \arrow[d, "L : (u_1 \to b, u_2 \to b)"] \\
u' = bbbba \arrow[d, "f : [b \to a, a \to b]"] \\
v = aaaab
\end{tikzcd}$$

FM_2^{DIS}

$$\begin{tikzcd}
u = aabba \arrow[d, "f : [a \to a, b \to a]"] \\
v' = aaaaa \\
v = aaaab \arrow[u, "L : (v_5 \to a)"]
\end{tikzcd}$$

Figure 2 Side-by-side comparison of PM^{DIS}, FM_1^{DIS} and FM_2^{DIS}.

On the other hand, an **instantiable word** is a word on the alphabet $\Sigma \cup \Pi$. Given $f : \Pi \to \Sigma$, we extend it to constants by setting $f(x) = x$ for all $x \in \Sigma$, and we then define the **language of an instantiable word** u to be $L(u) = \{w \in \Sigma^* \mid \exists f : \Pi \to \Sigma, f(u) = w\}$. This definition is akin to the $L_\diamond$ semantic of a parameterized regular expression defined in [11], but restricted here to a single instantiable word. Two instantiable words w_1 and w_2 describe the same elements if their languages are equal, i.e. $L(w_1) = L(w_2)$.

3 Different Definitions for Different Semantics and Problems

In this section, we introduce various new approximate variants of parameterized matching, and compare them, highlighting their differences on examples.

3.1 Variants of Parameterized Matching

In parameterized matching, the function f renaming parameters is generally considered to be 1-to-1. In this paper, we also consider the **function matching problem**, which is the case where f is not constrained to be injective anymore, as defined in [2]. We also introduce multiple approximate variants of the parameterized matching problems, depending on several edit distances obtained by combining insertion, deletion and substitution operations.

3.1.1 Edit distances for parameterized matching between two strings: PM^d

▶ **Definition 1.** *If d is an edit distance, we denote by PM^d the parameterized matchingproblem under d, which is the following:*

- ***Input:*** *two parameterized words u, v, a parameter alphabet Π , an alphabet Σ of constants, and a natural number k.*
- ***Problem:*** *Does there exist u' such that $d(u, u') \leq k$ and u' and v are parameterized matching, i.e. there exists a 1-to-1 function $f : \Pi \cup \Sigma \to \Pi \cup \Sigma$ such that $f|_\Sigma = Id_\Sigma$, $f(\Pi) = \Pi$, and $f(u') = v$?*

In that case, we say that u' and f **realize** the matching between u and v. We sometimes write that only f or u' realize the matching if the other one is not relevant to a proof.

In cases where Σ and Π are already defined, we omit them and simply call $PM^d(u, v, k)$ the result of the decision problem. Furthermore, $PM^d(u, v)$ denotes the minimum integer k (potentially infinite) such that $PM^d(u, v, k)$ is true.

We can note that this problem can be solved in polynomial time adapting the classical dynamic programming algorithm [33, 37] when the alphabet sizes are fixed.

3.1.2 Edit distances for function matching between 2 strings: FM_i^d

To denote **function matching problems**, we use FM instead of PM: FM^D denotes the function matching problems with deletions.

Furthermore, if $\mathcal{P}$ is one of the problems defined above, we note $\mathcal{P}_1$ the problem where edit operations are only applied to the first argument, and $\mathcal{P}_2$ the one where they are only applied to the second argument.

▶ **Definition 2.** *The FM_1^d and FM_2^d problems are defined as follows. For both problems, the input is the following:*

- ***Input:** two parameterized words u, v, a parameter alphabet Π, a constant alphabet Σ, and a natural number k.*

The problems are then:

- ***Problem FM_1^d:** $\exists u'$ such that $d(u, u') \leq k$ and u' and v are function matching?*
- ***Problem FM_2^d:** $\exists v'$ such that $d(v, v') \leq k$ and u and v' are in function matching?*

Note that the renaming function f is always applied to one input only. These definitions are illustrated on an example in Figure 2.

3.2 Comparing Variants of PM

In this subsection, we compare the different variants of our problem.

Regarding the one-to-one parameterized matching PM, note that the definition we give above is designed to be easily extended to the different variants when we drop the one-to-one restriction. In [24], the authors consider that the "correct way for defining the edit distance problem" is "to allow the operations and then apply the edit distance". By extending the definition of FM_1^d and FM_2^d to define PM_1^d and PM_2^d in the case of one-to-one matching, we see that it is actually possible to switch the order of operations, and to reverse them (deletions then become insertions and vice versa, and the renaming function f^{-1} is well-defined), in this case. This makes our definition consistent with the quote from [24] above. Formally, this gives the following equalities, for all parameterized words u and v: $PM_1^I(u, v) = PM_1^D(v, u) = PM_2^D(u, v) = PM_2^I(v, u)$.

More generally, it holds that for every edit distance d, $PM_1^d(u, v) = PM_1^{d^{-1}}(v, u) = PM_2^{d^{-1}}(u, v) = PM_2^d(v, u)$, where d^{-1} denotes the converse distance of d, i.e. d^{-1} contains deletions if d contains insertions, insertions if d contains deletions, and substitutions if d contains substitutions.

However, for function matching, we only have the following equalities: $FM_1^I(u, v) = FM_2^D(u, v)$ and $FM_1^D(u, v) = FM_2^I(u, v)$.

By taking $u = ab$ and $v = cc$, we can notice that $FM_1^I(u, v) = 0$ and $FM_1^D(v, u) = \infty$, so the equality $FM_1^I(u, v) = FM_1^D(v, u)$ does not hold.

Finally, note the following inequalities:

▶ **Proposition 3.** *Let u and v be parameterized words over $\Sigma \cup \Pi$. Then:*

1. *$FM_1^d(u, v) \leq PM^d(u, v)$;*
2. *If d is a symmetric edit distance, $FM_2^d(u, v) \leq FM_1^d(u, v)$.*

Proof. The first point comes from the fact that any solution to PM^d is also a solution to FM_1^d. For the second point, let $k = FM_1^d(u, v)$, and let u' and f realize $FM_1^d(u, v)$. We construct a word v', obtained by applying to v the same operations applied to u to obtain u', but "mirrored". That is to say, for every operation used in u, we apply an operation in v, in the following way:

- If a letter a is inserted in u, there exists a position i in u' such that $u'_i = a$, and $f(u'_i) = v_i$. Hence, we delete v_i in v.
- Similarly, if a letter is substituted for another letter a' in u, there exists i such that $u'_i = a$, and we substitute v_i to $f(a)$.
- If a letter a is deleted in u at position i, we insert $f(a)$ in v at position i instead.

It then holds that $f(u) = v'$, and hence $PM_2^d(u, v) \leq k$. ◀

Note that the above proof does not work to prove the converse inequality between FM_1^d and FM_2^d, as it would require to consider an element of $f^{-1}(a)$, which might be empty. This is illustrated in the following example, on the alphabet $\Pi = \{a, b\}$:

▶ **Example 4.** Let $N \in \mathbb{N}$ and consider $u = a^N b^N b$ and $v = a^N a^N b$. u and v are not in parameterized matching, hence $FM_1^{DIS}(u, v) > 0$ and $FM_2^{DIS}(u, v) > 0$. By substituting the last b in v for a a, and picking a function f such that $f(a) = f(b) = a$, we get $FM_2^{DIS}(u, v) = 1$ (see Figure 2 for an example with $N = 2$). For FM_1^{DIS}, since b appears in v, it holds that for any function f realizing FM_1^{DIS}, $f(a) = b$ or $f(b) = b$. Hence, at least N occurrences of b appear in $f(u)$. Since there is only one occurrence of b in v, it is clear that $FM_1^{DIS}(u, v) \geq N - 1$.

The difference between FM_1^d and FM_2^d comes from the fact that Π is fixed in the input. In the case where Π could be extended, both problems can be shown equivalent (for example if we allow a new letter c in the example of Figure 2, we also get $FM_1^{DIS}(u, v) = 1$ by setting $u_5 \to c$ and $f : [a \to a, b \to a, c \to b]$), by using the same proof as Proposition 3.

3.3 Instantiable Words versus Parameterized Words

The parameterized word formalism and the instantiable word formalism give rise to two different definitions of distances between words. Given an edit distance d on words, there are two ways to extend it to words with parameters. Let w_1 and w_2 be two words over $\Sigma \cup \Pi$. The two possible extensions are the following:

- The distance between w_1 and w_2 is defined as $d(w_1, w_2) = PM^d(w_1, w_2)$. Alternatively, the function distance between w_1 and w_2 is defined as $FM_1^d(w_1, w_2)$.
- The distance between w_1 and w_2 is the distance between their respective languages seen as sets, that is to say $d(w_1, w_2) = d(L(w_1), L(w_2)) = \sup_{u \in L(w_1)} \inf_{v \in L(w_2)} d(u, v)$. Equivalently, $d(w_1, w_2) \leq k$ if and only if for all $f_1 : \Pi \to \Sigma$, there exists $f_2 : \Pi \to \Sigma$ such that $d(f_1(w_1), f_2(w_2)) \leq k$.

This second definition stems from the definition of distance between languages, as defined and studied in [12, 13, 14].

▶ **Example 5.** Consider the words $u = axyb$ and $v = xbby$, on the alphabets $\Sigma = \{a, b\}$ and $\Pi = \{x, y\}$, and consider the distance S. On the one hand, $FM_1^S(u, v) = 4$, because regardless of the matching chosen, every letter of $f(u)$ has to be substituted. On the other hand, for any function $f_1 : \Pi \to \Sigma$, choosing f_2 such that $f_2(x) = a$ and $f_2(y) = b$ yields a distance $d(f_1(u), f_2(v))$ of at most 2, by substituting the 2 middle letters.

Given a big enough alphabet, those two definitions can in fact be shown equivalent:

▶ **Proposition 6.** *Let w_1 and w_2 be words over $\Sigma \cup \Pi$, and let d be a symmetric edit distance on $\Sigma \cup \Pi$. Suppose $|\Sigma| \geq |w_1| + |w_2|$, and let k be an integer. Then, the following are equivalent:*

1. $FM_1^d(w_2, w_1) \leq k$
2. $d(L(w_1), L(w_2)) \leq k$

Notice how w_1 and w_2 change position between the two distances. This is not benign, as FM_1^d is not symmetric.

Proof. Suppose $FM_1^d(w_2, w_1) \leq k$. There exists $f : \Pi \to \Pi$ such that $d(f(w_2), w_1) \leq k$. For this proof, we will use the characterization of the distance betweeen languages with f_1 and f_2. Let $f_1 : \Pi \to \Sigma$. Define $f_2 = f_1 \circ f$. Since $d(w_1, f(w_2)) \leq k$, we have $d(f_1(w_1), f_1 \circ f(w_2)) \leq k$, by following the same edit operations. Hence $d(f_1(w_1), f_2(w_2)) \leq k$.

Suppose now $d(L(w_1), L(w_2)) \leq k$. Let $f_1 : \Pi \to \Sigma$ be a 1-to-1 function such that for all parameters x in w_1, $f(x)$ does not appear in w_1 or w_2. This is possible since Σ is large enough. There exists $f_2 : \Pi \to \Sigma$ such that $d(f_1(w_1), f_2(w_2)) \leq k$. Let $h : \Sigma \to \Pi \cup \Sigma$ be such that if $x \in \Pi$, $h(f_1(x)) = x$, and if $x \notin f_1(\Pi)$, $h(x) = x$. We then have $h \circ f_1 = Id$. What is more, since h is injective, $d(f_1(w_1), f_2(w_2)) = d(h(f_1(w_1)), h(f_2(w_2)) = d(h(f_2(w_2)), w_1)$. Hence, $FM_1^d(w_2, w_1) \leq k$. ◀

4 Hardness Results for Approximate Parameterized Matching

In this section, we study the complexity of the various parameterized matching problems. We show the NP-completeness of the simplest problems using only deletions, which will be sufficient to show the NP-completeness of all the other problems. We start by proving some practical lemmas, and then proceed to the reductions.

4.1 "Block by block" Lemmas

In this section, we regroup a few useful technical lemmas. We start of by stating two simple results on distance and words, for which the proofs can be found in Appendix A. We then turn to block lemmas, which will later be useful in the proofs of Theorems 12,17 and 15, to combine the various gadgets defined during the reduction.

This lemma simply states a commutativity result between the application of a matching f and the rewriting steps.

▶ **Lemma 7.** *Let d be a distance, k an integer and u, v two parameterized words such that $PM^d(u, v) \leq k$, and let f realize this parameterized match. Then: $d(f(u), v) \leq k$. The same result holds for $FM_1^d(u, v)$.*

Proof Idea. The proof is done by induction on k. We discuss whether the $(k+1)$-th operation is an insertion, a deletion, or a substitution, and show that a corresponding operation can be used in $f(u)$. ◀

▶ **Lemma 8.** *Let z, u and v be (parameterized) words, and let d be a distance. Then $d(zu, zv) = d(u, v)$.*

Proof Idea. We show that we can consider every rewriting operation to be applied in u only: if z is modified during an optimal rewriting sequence, the words have some redundancy, and the same operations could have been carried in u instead. We proceed again by induction, and focus on the base case by studying the 3 possible cases, one for each type of operation. ◀

Next, we turn to prove "block by block" matching lemmas. Those results state that it is possible to encode multiple parameterized matching instances into a single one. They hold for every type of problems considered here, but their proofs vary slightly; we present them in order of increasing complexity. Note that all the constructions given can be achieved in polynomial time.

▶ **Lemma 9.** *Let $u_1, \dots u_n$ and $v_1, \dots v_n$ be parameterized words over $\Sigma \cup \Pi$ such that for $1 \le i \le n$, $k_i = |u_i| - |v_i| \ge 0$, and $k = \sum_{i=1}^{n} k_i$. There exist u and v two parameterized words over $\{\#\} \cup \Sigma \cup \Pi$, where $\#$ is a fresh variable, such that the following are equivalent:*

1. *$PM^D(u, v) = k$*
2. *For all $1 \le i \le n$, $PM^D(u_i, v_i) = k_i$ and the applications f_i realizing those matchings are all compatible.*

Proof. The idea behind this proof and all the following ones is that we can introduce a separator $\#$ to concatenate all the words, and that this separator will never be touched by any deletions or applications of f.

Let $\#$ be a fresh constant. We define $u = u_1\#u_2\# \dots \#u_n$, and $v = v_1\#v_2\# \dots \#v_n$.

2. $\implies$ 1.: For every $1 \le i \le n$, take u'_i and f_i to realize the matchings. We can obtain $u' = u'_1\#u'_2 \dots \#u'_n$ from u by applying the same deletions. Taking f to be the smallest function extending all the f_i, we get $PM^D(u, v) \le k$.

1. $\implies$ 2.: Assume $PM^D(u, v) \le k$. Let u' and f realize this parameterized match. Since the $\#$ symbols are constants, we have $f(\#) = \#$. Since u' is obtained from u by deletions, we have $|u'|_\# \le |u|_\#$. Since f is injective and $f(\#) = \#$, $|f(u')|_\# \le |f(u)|_\#$. Hence, it holds that $|v|_\# = |f(u')|_\# \le |f(u)|_\# = |u|_\#$. Since $|u|_\# = |v|_\#$, this is an equality, and $|f(u')|_\# = |f(u)|_\#$. Hence $|u'|_\# = |u|_\#$, and no $\#$ character is deleted. The word u' is then of the form $u'_1\#u'_2\# \dots \#u'_n$, where $|u'_i|_\# = 0$ and $D(u_i, u'_i) = k_i$ for all i. Thus, $f(u') = f(u'_1)\#f(u'_2)\# \dots \#f(u'_n) = v_1\#v_2\# \dots {}_\# v_n$. Since no other $\#$ letter appear in any $f(u'_i)$ and v_i, we can deduce that $f(u'_i) = v_i$ for all i. Finally, this yields $PM^D(u_i, v_i) = k$, and taking all the $f_i = f$ gives all compatible functions, which concludes the proof. ◀

In this proof, we used a constant $\#$. However, it can also be conducted without using a constant alphabet; indeed, constants can be encoded with parameters, as shown in Appendix B.

Lemma 9 is still valid if PM^D is replaced by FM_2^D. This time, we conduct this proof without resorting to the use of constants. This result will be used twice: once for the proof of theorem 17, and again to show that we can once more encode constants into Π using Lemma 25 in Appendix B.

▶ **Lemma 10.** *Let $u_1, \dots u_n$ and $v_1, \dots v_n$ be parameterized words over Π such that $k_i = |v_i| - |u_i| \ge 0$, and $k = \sum_{i=1}^{n} k_i$. Then there exist u and v, two parameterized words over $\Pi \cup \{\#\}$, where $\#$ is a fresh variable, such that the following are equivalent:*

1. *$FM_2^D(u, v) \le k$*
2. *For all $1 \le i \le n$, $FM_2^D(u_i, v_i) \le k_i$, and the applications f_i realizing those matchings are all compatible.*

Proof Idea. The same technique as Lemma 9 is used but u and v are defined as $u = \#^{k+1}u_1\#u_2\# \dots \#u_n$ and $v = \#^{k+1}v_1\#v_2\# \dots \#v_n$ where $\#^{k+1}$ denotes $k+1$ repetitions of the character $\#$. The full proof can be found in Appendix A. ◀

Finally, the same block result holds for FM_1^D, and will be used in the proof of theorem 15.

▶ **Lemma 11.** *Let $u_1, \dots u_n$ and $v_1, \dots v_n$ be parameterized words over Π such that for every $1 \le i \le n$, $k_i = |u_i| - |v_i| \ge 0$, and $k = \sum_{i=1}^{n} k_i$. Then there exist u and v two parameterized words over $\Pi \cup \{\#\}$, where $\#$ is a fresh variable, such that the following are equivalent:*

1. *$FM_1^D(u, v) \le k$*
2. *For all $1 \le i \le n$, $FM_1^D(u_i, v_i) \le k_i$, and the applications f_i realizing those matchings are all compatible.*

Proof Idea. The difference with Lemma 10 is that some # symbols might be deleted, while some base letters could be mapped to #. To ensure this does not happen, we define $u = \#^N u_1 \#^N u_2 \dots \#^N u_n \#^N$ and $v = \#^N v_1 \#^N v_2 \dots \#^N v_n \#^N$. The full proof can be found in Appendix A. ◀

The technique of block-by-block matching will be used in all the reductions, to encode multiple constraints in a single PM or FM instance.

4.2 1-to-1 Parameterized Matching PM

We now focus on the complexity of the PM^d problems. These problems, as well as function matching problems, are all clearly in NP: given the list of deletion, insertion or substitution operations to do and the matching to apply, it is easy to check that the solution is correct.

For the reductions in this paper, we always assume that words are written without constants, that is to say $\Sigma = \emptyset$, since this is sufficient for NP-completeness results. This choice is also motivated by the results of Appendix B, which show that Σ can in most cases be coded into Π.

▶ **Theorem 12.** *The 1-to-1 Parameterized Matching with deletions PM^D is* NP*-complete.*

The proof is a reduction from the NP-complete problem **3-coloring**[20]. Given an instance G of **3-coloring**, we will construct two words u and v. The word v will represent the list of vertices and edges of G, while the word u will list the color of each vertex, and the possible coloring of each pair of vertices joined by an edge. By deleting characters from u, we make a choice for the coloring of each vertex, and thus each edge. The function f answering the parameterized matching problem will assign a choice of color to each vertex. The instance that we define should be positive iff G is 3-colorable. More formally:

Proof. The 3-Coloring problem is defined as follows:

- **Input:** $G = (V, E)$ a graph with vertices V and edges E
- **Output:** A coloring $c : V \to \{c_1, c_2, c_3\}$ such that for all $\{u, v\} \in E$, $c(u) \neq c(v)$

Let $G = (V, E)$ be an instance of **3-Coloring**, and let $V = \{x_1, \dots, x_n\}$ be the set of its n vertices, and $E = \{e_1, \dots, e_m\}$ be the set of its edges. The parameter alphabet Π, of polynomial size in $O(|G|)$ will contain:

- $x_1, \dots x_n$, corresponding to the vertices of G;
- n copies of the parameters corresponding to the colors c_1, c_2 and c_3: c^i_1, c^i_2, c^i_3 for $1 \leq i \leq n$;
- for every edge e, the delimiters Y^e and $\square^e_1, \dots \square^e_6$;
- $2n$ bottom symbols, $\perp^i_1$, $\perp^i_2$ for $1 \leq i \leq n$, which will be used to fix the image of some parameters.

First, we define words that will encode the constraint that each vertex is colored, and we make sure that the unused color variables cannot be assigned elsewhere. For $1 \leq i \leq n$, $u^i_1 = u^i_\perp = c^i_1 c^i_2 c^i_3$, $v^i_1 = x_i$ and $v^i_\perp = \perp^i_1 \perp^i_2$. We then define words that include all possible colorings of each edge, and we make sure to use enough brackets. For every edge $e = \{x_i, x_j\}$, we define $u^e_2 = \square^e_1 c^i_1 c^j_2 \square^e_1 \; \square^e_2 c^i_1 c^j_3 \square^e_2 \; \square^e_3 c^i_2 c^j_1 \square^e_3 \; \square^e_4 c^i_2 c^j_3 \square^e_4 \; \square^e_5 c^i_3 c^j_1 \square^e_5 \; \square^e_6 c^i_3 c^j_2 \square^e_6$ and $v^e_2 = Y^e x_i x_j Y^e$.

Applying Lemma 9 to $u^1_1, \dots u^n_1, u^1_\perp, \dots u^n_\perp, u^{e_1}_2, \dots u^{e_m}_2$ and $v^1_1 \dots v^n_1, v^1_\perp, \dots v^n_\perp, v^{e_1}_2, \dots v^{e_m}_2$, we obtain u and v. Let $k = |u| - |v| = 3n + 20m$. We now show that G is 3-colorable $\Leftrightarrow PM^D(u, v) \leq k$.

$\Rightarrow$: Suppose G is 3-colorable. Let $c : V \to \{c_1, c_2, c_3\}$ be a 3-coloring of G. We define f in the following way, for $1 \le y \le 3$:

$$f(c_y^i) = \begin{cases} x_i & \text{if } c(x_i) = c_y, \\ \perp_1^i & \text{if } y \text{ is the smallest integer in } \{1,2,3\} \text{ such that } c(x_i) \neq c_y, \\ \perp_2^i & \text{otherwise.} \end{cases}$$

For every edge $e = \{x_i, x_j\} \in E$, since c is a valid coloring, and since every allowed arrangements of the colors is in u_2^e, there exists a unique factor of the form $\Box_y^e f^{-1}(x_i) f^{-1}(x_j) \Box_y^e$ in u_2^e, for some $1 \le y \le n$. Hence, we define $f(\Box_y^e) = Y^e$. The function f can then be extended in any way to be 1-to-1 (the remaining characters whose image under f are not yet defined will all be deleted in what follows, so their image doesn't matter).

By using f defined in this way:

- For $1 \le i \le n$, $PM^D(u_1^i, v_1^i) \le 2$, by deleting the 2 colors not matching the color of x_i;
- For $1 \le i \le n$, $PM^D(u_\perp^i, v_\perp^i) \le 1$;
- For every edge $e \in E$, $PM^D(u_2^e, v_2^e) \le 20$, by keeping only the factor delimited by the $\Box_y^e$ symbols defined above.

Thus Lemma 9 yields $PM^D(u, v) \le k$.

$\Leftarrow$: We now suppose u and v are a parameterized match with k deletions. The following can then be derived about f:

1. Since the u_1^i and v_1^i are matching for $1 \le i \le n$, there exists an element $c \in \{c_1^i, c_2^i, c_3^i\}$ such that $f(c) = x_i$. Each of these matchings is done with exactly 2 deletions, for a total of $2n$.
2. Since the $u_\perp^i$ and $v_\perp^i$ are in matching, the two other colors that are not sent to x_i are sent to $\perp_1^i$ and $\perp_2^i$. Each of these matchings is done with exactly one deletion, for a total of n.
3. For every edge $e \in E$, u_2^e and v_2^e are in matching. Let $u_2^{e\prime}$ realize this matching. For every $1 \le i \le n$ and $1 \le i' \le 3$ the colors $c_{i'}^i$ have images that are different from Y^e, so there necessarily exists $1 \le y \le 6$ such that $f(\Box_y^e) = Y^e$. Furthermore, since f is injective, $|v_2^e|_{Y^e} = |u_2^{e\prime}|_{\Box_y^e}$. Since $|v_2^e|_{Y^e} = |u_2^e|_{\Box_y^e} = 2$, no $\Box_y^e$ is deleted from u. Since there are two characters between the Y^e in v_2^e and none outside, $u_2^{e\prime}$ has the same structure, and all other $\Box_{y'}^e$ for $y' \neq y$ and all other colors are deleted from u_2^e.
   Finally, $u_2^{e\prime}$ is of the form $\Box_y^e c_t c_{t'} \Box_y^e$, where $t \neq t'$ are elements of $\{1, 2, 3\}$. Each of these matchings is done with exactly 20 deletions, for a total of $20m$.

The function f then implies a coloring of G. Formally, we define $col(c_y^i) = c_y$ for $1 \le i \le n$ and $1 \le y \le 3$. We can then define $c : V \to \{c_1, c_2, c_3\}$ such that $c(x_i) = col(f^{-1}(x_i))$. Point 1 above ensures that this function definition is correct. Furthermore, for every edge $e = \{x_i, x_j\}$, point 3 ensures that $c(x_i) \neq c(x_j)$, and thus c is a valid coloring of G. ◀

This first NP-completeness results yields a few immediate corollary results, and in particular, the NP-completeness of the problem under the Levenshtein distance:

▶ **Theorem 13.** *The problem PM^{DIS} of parameterized matching under the Levenshtein distance is* NP*-complete.*

Proof. We do a simple reduction from PM^D. Let u, v, k be an instance of PM^D. If the instance is trivially false (that is to say, $k \neq |u| - |v|$), answer negatively. Else, consider u, v, k as an instance of PM^{DIS}. If this is a negative instance for PM^{DIS}, it is also negative for PM^D. Furthermore, if it is a positive instance for PM^{DIS}, exactly k deletions should be applied, and so no substitution or insertion are used in that solution. Hence, that solution is also a solution to PM^D, and the reduction holds. ◀

The same result in fact holds for all other distances, and in particular the longest common sub-word distance ID. This proves once again the result shown in [26]:

▶ **Corollary 14.** *The problems PM^I, PM^{DI}, PM^{IS}, PM^{DS} are all* NP*-complete.*

Proof. From Section 3.2, PM^I and PM^D are equivalent in the 1-to-1 case. For the other problem, we do an immediate reduction from PM^I or PM^D analog to the proof of Theorem 13. ◀

We now turn to proofs of NP-completeness without the restriction asking f to be injective.

4.3 Function Matching FM_1^d

The problem considered in this section is the one where both deletions and f are applied to the first word. A reduction very similar to the one given for PM^D is used.

▶ **Theorem 15.** *FM_1^D is* NP*-complete.*

Proof Idea. The reduction follows the same idea as in Theorem 12. Since the function f realizing the matchings is not injective in this version, it will be used to send every vertex to its color. Moreover, we add more “sink” $\perp$ letters to force the image of every unused letter. The full proof can be found in Appendix A. ◀

This again ensures the NP-completeness of the problem for all edit distances, using the same proof as for Theorem 13.

▶ **Corollary 16.** *The problem FM_1^{DIS} of function matching under the Levenshtein distance is* NP*-complete. The problems FM_1^I, FM_1^{ID}, FM_1^{IS}, FM_1^{DS} are all* NP*-complete too.*

We can notice that the problem FM_1^S, where substitution is the only operation allowed, is polynomial-time solvable. Intuitively, for each parameter, consider the possible parameters that it could be mapped to, and their respective number of occurrences. Then, choose the letter with the highest number of occurrences for the mapping. The remaining letters are then substituted.

4.4 Function Matching FM_2^d

The problem considered in this section is the one where deletions are applied to the second word, while f is applied to the first word.

▶ **Theorem 17.** *FM_2^D is* NP*-complete.*

Proof Idea. The proof is very similar to the previous case, but the bracketing has to be adapted. Separators Y^e are duplicated enough times to ensure that no vertex variable is mapped to them. The full proof can be found in Appendix A. ◀

▶ **Corollary 18.** *FM_1^I, FM_2^{DI}, and FM_2^L are all* NP*-complete.*

Proof. FM_1^I is equivalent to FM_2^D. For the two other problems, we use a reduction from FM_2^D exactly like in Corollary 14. ◀

This last result completes the picture of NP-completeness proofs, and indicates that computing the distances between parameterized words defined in Section 3.3 is in general an NP-complete problem.

Similarly to FM_1^S, FM_2^S is also polynomial-time solvable.

5 Approaches to Solve Parameterized Matching

In this section, we discuss two ways to get around the difficulty of the parameterized matching problems. The first one is to design an FPT algorithm in the alphabet size, and the second one is to translate the problem into a SAT formalism, with the intent of using a SAT-solver.

5.1 An FPT Algorithm in the Alphabet Size

The fact that Σ and Π are part of the input is what makes the various parameterized matching problems NP-hard. When the alphabet size is considered fixed, a simple polynomial algorithm can be used, which generalizes the "naïve" brute force algorithm of Theorem 1 of [26]:

Algorithm 1 Simple FPT algorithm for FM^d.

```
$m \leftarrow 0$
for all functions $f : \Pi \to \Pi$ do
    $dist \leftarrow d(f(u), v)$
    if $dist \leq m$ then
        $m \leftarrow dist$
    end if
end for
```

▶ **Theorem 19.** *Let d be a distance. Algorithm 1 computes $FM^d(u,v)$ in time $O(|\Pi|^{|\Pi|}|u||v|)$*

Proof. Algorithm 1 uses an exhaustive search and finds $\min_{f:\Pi\to\Pi} d(f(u),v)$, which is the definition of $FM^d(u,v)$. Furthermore, there are $|\Pi|^{|\Pi|}$ functions from Π to Π, and computing $d(f(u),v)$ is done in time $O(|f(u)||v|) = O(|u||v|)$, hence a total running time in $O(|\Pi|^{|\Pi|}|u||v|)$. ◀

Note that this also leads to a similar algorithm for PM^d by iterating over injective functions rather than all functions from Π to Π.

5.2 A MaxSat Formulation of Parameterized Matching

In this section, we encode PM^d problems into SAT problems, with the intent of solving them with a SAT solver. More precisely, we will use the weighted max-SAT variant of SAT, which is defined in the following way:

- **Input:** a set V of literals, a formula $\varphi = \bigwedge_{i=1}^{n} \varphi_i$ on V in conjunctive normal form (CNF), a weight function $w : [\![1,n]\!] \to \mathbb{N}$.
- **Output:** a valuation $\nu : V \to \{0,1\}$ such that $\sum_{\nu \models \varphi_i} w(i)$ is maximal.

Moreover, we will sometimes use a partially weighted variant of Max-SAT, which is defined in the following way:

- **Input:** a set V of literals, a satisfiable formula φ_c on V in CNF, a formula $\varphi_w = \bigwedge_{i=1}^{n} \varphi_i$ on V in CNF and a weight function $w : [\![1, n]\!] \to \mathbb{N}$.
- **Output:** a valuation $\nu : V \to \{0, 1\}$ such that $\nu \vDash \varphi_c$ and $\sum_{\nu \vDash \varphi_i} w(i)$ is maximal.

In that case, clauses of φ_c are called "hard" clauses while clauses of φ_w are called "soft clauses". We give a proof of the equivalence in Proposition 26 of Appendix C.

We will define an encoding of an instance (u, v) of PM^d such that an assignment of the variables of V will define an alignment between u and v. First, we make a link between the ID edit distance and the length of the optimal alignment between two strings.

▶ **Definition 20.** *Let u and v be two words on Π, such that $p = |u|$ and $p' = |v|$. A set $A \subset [\![1, |u|]\!] \times [\![1, |v|]\!]$ is an alignment between u and v iff the following are true:*

1. *Each position of u appears at most once: For all $1 \leq i \leq p$ and $1 \leq j, j' \leq p'$, if $(i, j) \in A$ and $(i, j') \in A$, then $j = j'$.*
2. *Each position of v appears at most once: For all $1 \leq j \leq p'$ and $1 \leq i, i' \leq p$, if $(i, j) \in A$ and $(i', j) \in A$, then $i = i'$.*
3. *There are no crossings: if $(i, j) \in A, (i', j') \in A$, and $i' > i$, then $j' > j$.*
4. *Aligned positions match in u and v: if $(i, j) \in A$, then $u_i = v_j$*

An alignment relates to the insertion/deletion distance ID in the following way:

▶ **Theorem 21.** *Let u, v be words on Π and $k \leq |u| + |v|$ be an integer. The following are equivalent:*

1. *There exists an alignment A such that $2|A| = |u| + |v| - k$*
2. *$ID(u, v) \leq k$.*

Proof. The proof, which works by induction, can be found in Appendix C. ◀

We now turn to the max-SAT encoding of our problem.

▶ **Theorem 22.** *Let u and v be two words over Π. There exists a formula $\varphi_{u,v} = \varphi_c \wedge \varphi_w$ and a weight function w, instance of the partially weighted Max-SAT problem such that the following are equivalent:*

- *ν is a solution to this partially weighted Max-SAT instance and satisfies k clauses of φ_w*
- *There exists a function $f : \Pi \to \Pi$ and an alignment between $f(u)$ and v of size k.*

The formula φ uses $|m||p| + |\Pi|^2$ variables and is of size $O(m^2p^2)$, where $m = |u|$ and $p = |v|$. Moreover, there exists φ^{inj} of size $O(|\Pi|^3)$ such that the above result is true for f injective by replacing φ_c with $\varphi'_c = \varphi_c \wedge \varphi^{inj}$.

In particular, finding the valuation maximizing k gives a maximal alignment between u and v, and with Theorem 21, the distance $ID(u, v)$.

Proof. For this proof, we fix an ordering on the alphabet $\Pi = \{a_1, \ldots, a_n\}$.
We define the set of literals V as $V = \{x_{i,j} \mid 1 \leq i \leq |u|, 1 \leq j \leq |v|\} \cup \{y_{a,b} \mid a \in \Pi, b \in \Pi\}$. Intuitively, $x_{i,j}$ represents a match between position i and j in the alignment, and $y_{a,b}$ will represent the fact that $f(a) = b$. We define the following sets of formulas, where all indices i are taken between 1 and m and all j between 1 and p, and a and b are taken in Π:

$$\begin{array}{llr}
\forall i\ \forall j' \neq j, & \varphi^{A_1}_{i,j,j'} \equiv x_{i,j} \implies \neg x_{i,j'} & \text{(NoDouble i)}\\
\forall j\ \forall i' \neq i, & \varphi^{A_2}_{i,i',j} \equiv x_{i,j} \implies \neg x_{i',j} & \text{(NoDouble j)}\\
\forall i' > i\ \forall j' < j, & \varphi^{C}_{i',i,j,j'} \equiv x_{i,j} \implies \neg x_{i',j'} & \text{(NoCrossing)}\\
\forall a \forall b \neq b', & \varphi^{f}_{a,b,b'} \equiv y_{a,b} \implies \neg y_{a,b'} & \text{(Function)}\\
\forall a \neq a' \forall \neq b, & \varphi^{inj}_{a,a',b} \equiv y_{a,b} \implies \neg y_{a',b} & \text{(Injectivity)}\\
\forall i \forall j, & \varphi^{M}_{i,j} \equiv x_{i,j} \implies y_{u_i,v_j} & \text{(Match)}\\
\forall i, & \varphi^{\exists}_{i} \equiv \bigvee_{1\leq j\leq p} x_{i,j} & \text{(ExistsMatch)}
\end{array}$$

We then define φ_c as the conjunction of all the formulas (NoDouble i), (NoDouble j), (NoCrossing), (Function), and (Match). Furthermore, we define φ^{inj} as the conjunction of all the (Injectivity) formulas. Lastly, we define $\varphi_w = \bigwedge_{1\leq i\leq m} \varphi^{\exists}_i$, and set $w(C) = 1$ for every clause C of φ_w.

There are $m\binom{p}{2}$ (NoDouble i) formulas, $p\binom{m}{2}$ (NoDouble j), $\binom{m}{2}\binom{p}{2}$ (NoCrossing), $n\binom{n}{2}$ (Function) and (Injectivity) formulas, pm (Match) formulas and n (ExistsMatch) formulas.

We now prove both implications of the theorem. Suppose ν is a valuation satisfying φ_c and k clauses of φ_w. We define, for all $a, b \in Pi$, $f(a) = b$ if and only if $\nu(y_{a,b}) = \top$. Since ν satisfies all the (Function) formulas, this is a correct definition of a (partial) function. We define $A = \{(i,j) \mid \nu(x_{i,j)=\top}\}$. A is an alignment between $f(u)$ and v. Indeed: (NoDouble i) and (NoDouble j) ensures point 1. and 2. of Definition 20, (NoCrossing) ensures point 3., and Match ensures point 4. The size of A is the number of $x_{i,j}$ instantiated to $\top$, which is exactly the number of clauses of φ_c satisfied, i.e., k.
Suppose now that there exists a function $\Pi \to \Pi$ and an alignment A between $f(u)$ and v. Similarly, we define $\nu(y_{a,b}) = \top$ if and only if $f(a) = b$, and $\nu(x_{i,j}) = \top$ if and only if $(i,j) \in A$. Since A is an alignment, ν satisfies (NoDouble i),(NoDouble j), and (NoCrossing). Since f is a function, (Function) is satisfied. Finally, if $\nu(x_{i,j}) = \top$, then $(i,j) \in A$, and since A is a matching, $f(u)_i = f(u_i) = v_j$ and $\nu(y_{u_i,v_j}) = \top$.

The proof for φ^b is the same, and (Injectivity) ensures the injectivity of f. ◀

What is more, this proof can be adapted to change the ID distance to the Levenshtein distance, simply by choosing to consider all the (Match) formulas as soft clauses.

6 Experiments

The two approaches presented in Section 5 were implemented in Python to solve PM^{ID}. They are available under the GPL license at `https://github.com/AaronFive/paramatch`. The FPT algorithm of Section 5.1 is implemented in the function `parameterizedAlignment` of file `fpt_alphabet_size.py`. The MaxSAT-reduction of Section 5.2 is implemented in the function `make_sat_instance` of file `sat_instance.py`. The MaxHS solver [18] available at `http://www.maxhs.org` is used by our script to solve the MaxSAT instances derived from the PM^{ID} instances.

Our initial motivation to introduce parameterized matching under various distances is theater play comparison. To represent the structure of a theater play, we represent each character by a letter of the alphabet, and create the parameterized word obtained by considering the succession of all consecutive speakers. To check their adequacy with real data, we use a corpus of theater plays in which each character is represented by one letter of the alphabet, and each act of the play is represented by a string corresponding to the sequence of speaking characters. A letter may be duplicated in this string if the corresponding characters has lines in the end of a scene and in the beginning of the next one. Therefore, the edit distance between two parameter words representing acts will be small if both acts have a similar structure in terms of succession of speaking characters. We selected a corpus of 10 pairs of plays where one inspired the other, and performed 47 comparisons between pairs of acts. Among those comparisons, 26 were solved by the maxSAT algorithm and all by the FPT algorithm (detailed results are presented in the supplementary material available at `https://github.com/AaronFive/paramatch/tree/main/corpus10pairs`), with a 800 second timeout. The computation times are obtained on a XMG laptop running on Windows, with a 2.60 Ghz processor and 16 Gb RAM. Only the running time of MaxHS is provided, the encoding into a MaxSAT formula usually runs in approximately 1 second. Note that all instances are solved faster by the FPT algorithm than by the MaxSAT approach. The analysis of running times depending on the product of the lengths of the input strings (see supplementary material) shows that the MaxSAT approach may be relevant for strings with more than 10 distinct characters, but where the product of the length of input strings may not exceed 2000.

7 Conclusion

In this paper, we studied the complexity of several variants of the edit distance problem between parameterized words. We proved the NP-completeness of all previously unsolved cases, including the Levenshtein distance left open in [24], and provided practical approaches to solve real instances of those problems. We also studied similar problems for various definitions of words with parameters, namely parameter words and parameterized expressions, proving some relationships with parameterized word problems.

As future work, we will study the restrictions introduced in [21, 22] for a pattern matching problem with patterns in the parameter, in order to obtain polynomial time algorithms for the edit distance between parameterized words. Moreover, we will explore the question of distance between sets of words, in particular when they are defined through generalizations of automata. These problems are variants of the notion of distance between regular languages as defined in [12]. In this context, we can notice that different notions of automata can be considered: either automata generating parameterized words, or automata using parameters to define languages over classical words, with two different semantics as defined in [11].

References

1 Rakesh Agrawal, Christos Faloutsos, and Arun Swami. Efficient similarity search in sequence databases. In *International conference on foundations of data organization and algorithms*, pages 69–84. Springer, 1993.

2 Amihood Amir, Yonatan Aumann, Richard Cole, Moshe Lewenstein, and Ely Porat. Function matching: Algorithms, applications, and a lower bound. In *Proceedings of the 30th International Conference on Automata, Languages and Programming*, pages 929–942, 2003. `doi:10.1007/3-540-45061-0_72`.

3 Amihood Amir, Martin Farach, and S. Muthukrishnan. Alphabet dependence in parameterized matching. *Information Processing Letters*, 49(3):111–115, 1994. `doi:10.1016/0020-0190(94)90086-8`.

4 Dana Angluin. Finding patterns common to a set of strings. *J. Comput. Syst. Sci.*, 21(1):46–62, 1980. `doi:10.1016/0022-0000(80)90041-0`.

5 Alberto Apostolico, Péter L. Erdős, and Moshe Lewenstein. Parameterized matching with mismatches. *Journal of Discrete Algorithms*, 5(1):135–140, 2007. doi:10.1016/j.jda.2006.03.014.

6 Brenda S. Baker. A theory of parameterized pattern matching: Algorithms and applications. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 71–80, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/167088.167115.

7 Brenda S. Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM Journal on Computing*, 26:1343–1362, 1997.

8 Brenda S. Baker. Parameterized diff. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, pages 854–855, USA, 1999. Society for Industrial and Applied Mathematics.

9 Pablo Barceló, Leonid Libkin, and Juan L. Reutter. Querying graph patterns. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2011)*, pages 199–210. ACM, 2011. doi:10.1145/1989284.1989307.

10 Pablo Barceló, Juan Reutter, and Leonid Libkin. Parameterized regular expressions and their languages. *Theoretical Computer Science*, 474:21–45, 2013. doi:10.4230/LIPIcs.FSTTCS.2011.351.

11 Pablo Barceló, Leonid Libkin, and Juan Reutter. Parameterized regular expressions and their languages. *Theoretical Computer Science*, 474:21–45, 2011. doi:10.1016/j.tcs.2012.12.036.

12 Michael Benedikt, Gabriele Puppis, and Cristian Riveros. The cost of traveling between languages. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, pages 234–245, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

13 Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Regular repair of specifications. In *2011 IEEE 26th Annual Symposium on Logic in Computer Science*, pages 335–344, 2011. doi:10.1109/LICS.2011.43.

14 Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Bounded repairability of word languages. *Journal of Computer and System Sciences*, 79(8):1302–1321, 2013. doi:10.1016/j.jcss.2013.06.001.

15 Francine Blanchet-Sadri. *Algorithmic Combinatorics on Partial Words (Discrete Mathematics and Its Applications)*. Chapman, Hall/CRC, 2007.

16 William W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. *SIGMOD Rec.*, 27(2):201–212, 1998. doi:10.1145/276305.276323.

17 Richard Cole, Carmit Hazay, Moshe Lewenstein, and Dekel Tsur. Two-dimensional parameterized matching. *ACM Trans. Algorithms*, 11(2), October 2014. doi:10.1145/2650220.

18 Jessica Davies. *Solving MAXSAT by Decoupling Optimization and Satisfaction.* PhD thesis, University of Toronto, Canada, 2014. URL: http://hdl.handle.net/1807/43539.

19 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. Pbwt: Achieving succinct data structures for parameterized pattern matching and related problems. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 397–407, 2017. doi:10.1137/1.9781611974782.25.

20 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, first edition edition, 1979.

21 Pawel Gawrychowski, Florin Manea, and Stefan Siemer. Matching patterns with variables under hamming distance. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 48:1–48:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.48.

22 Pawel Gawrychowski, Florin Manea, and Stefan Siemer. Matching patterns with variables under edit distance. In Diego Arroyuelo and Barbara Poblete, editors, *String Processing and Information Retrieval - 29th International Symposium, SPIRE 2022, Concepción, Chile, November 8-10, 2022, Proceedings*, volume 13617 of *Lecture Notes in Computer Science*, pages 275–289. Springer, 2022. doi:10.1007/978-3-031-20643-6_20.

23 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. URL: https://www.wikidata.org/entity/Q55980413.

24 Carmit Hazay, Moshe Lewenstein, and Dina Sokol. Approximate parameterized matching. *ACM Trans. Algorithms*, 3(3):29–es, 2007. doi:10.1145/1273340.1273345.

25 Wilbert Jan Heeringa. *Measuring dialect pronunciation differences using Levenshtein distance*. PhD thesis, University of Groningen, 2004.

26 Orgad Keller, Tsvi Kopelowitz, and Moshe Lewenstein. On the longest common parameterized subsequence. *Theoretical Computer Science*, 410(51):5347–5353, 2009. doi:10.1016/j.tcs.2009.09.011.

27 Lillian Jane Lee. *Similarity-based approaches to natural language processing*. PhD thesis, Harvard University, 1997.

28 Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10(8), pages 707–710. Soviet Union, 1966.

29 Moshe Lewenstein. *Parameterized Matching*, pages 635–638. Springer US, Boston, MA, 2008. doi:10.1007/978-0-387-30162-4_282.

30 Florin Manea and Markus L. Schmid. Matching patterns with variables. In Robert Mercas and Daniel Reidenbach, editors, *Combinatorics on Words - 12th International Conference, WORDS 2019, Loughborough, UK, September 9-13, 2019, Proceedings*, volume 11682 of *Lecture Notes in Computer Science*, pages 1–27. Springer, 2019. doi:10.1007/978-3-030-28796-2_1.

31 Juan Mendivelso, Sharma V. Thankachan, and Yoan Pinzón. A brief history of parameterized matching problems. *Discrete Applied Mathematics*, 274:103–115, 2020. Stringology Algorithms. doi:10.1016/j.dam.2018.07.017.

32 Mehryar Mohri. Edit-distance of weighted automata: General definitions and algorithms. *International Journal of Foundations of Computer Science*, 14(06):957–982, 2003.

33 Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

34 Sascha Schimke, Claus Vielhauer, and Jana Dittmann. Using adapted levenshtein distance for on-line signature authentication. In *Proceedings of the 17th International Conference on Pattern Recognition (ICPR 2004)*, volume 2, pages 931–934. IEEE, 2004.

35 T. Shibuya. Generalization of a suffix tree for RNA structural pattern matching. *Algorithmica (New York)*, 39(1):1–19, 2004. doi:10.1007/s00453-003-1067-9.

36 Bernd Voigt. The partition problem for finite abelian groups. *Journal of Combinatorial Theory, Series A*, 28(3):257–271, 1980.

37 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974. doi:10.1145/321796.321811.

38 Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

A Details of the proofs

Proof of Lemma 7. We proceed by induction on k. If $k = 0$, then u and v are parameterized matching, and $f(u) = v$, thus $d(f(u), v) = 0$. Suppose the result holds until a fixed k. Suppose $PM^d(u, v) = k + 1$. There exist f, u'' and u' such that $d(u, u'') = 1$, $d(u'', u') = k$, and $f(u') = v$. Hence $PM^d(u'', v) \leq k$, and by induction hypothesis $d(f(u''), v) \leq k$. Moreover, since $d(u, u'') = 1$, we get u'' from u by applying only one operation. We prove that regardless of this operation, $d(f(u), f(u'')) = 1$, and thus $d(f(u), v) \leq d(f(u), f(u''))+d(f(u''), v) \leq k+1$ which will conclude the proof. There are 3 cases to consider:

- If the operation is a deletion, $u = v_1xv_2$ and $u'' = v_1v_2$ for some words v_1 and v_2 and some letter x. Then $f(u) = f(v_1)f(x)f(v_2)$ and we can obtain $f(v_1)f(v_2) = f(u'')$ by deleting $f(x)$.
- If it is an insertion, $u = v_1v_2$ and $u'' = v_1xv_2$, and we can similarly go from $f(u)$ to $f(u'')$ by inserting $f(x)$.
- If it is a substitution, $u = v_1xv_2$ and $u'' = v_1yv_2$, and we can go from $f(u)$ to $f(u'')$ by replacing $f(x)$ with $f(y)$.

Hence $d(f(u), f(u'')) = 1$, which concludes the proof for PM^d.

Since this proof does not use the fact that f is 1-to-1, it also stands for FM_1^d. ◀

Proof of Lemma 8. It is obvious that $d(zu, zv) \leq d(u, v)$, so we only prove $d(u, v) \leq d(zu, zv)$. We prove that any rewriting sequence from zu to zv can be modified such that no edit operation is applied in z. This will be enough to prove the result, as the edit sequence obtained can be seen as an edit sequence between u and v. We proceed by induction on the size of z. Suppose $|z| = 1$. Then $z = a \in \Sigma \cup \Pi$. We can consider that no character is modified twice in an edit sequence (i.e. no character is inserted and then deleted, or inserted and then substituted etc.), as that is always sub-optimal. Suppose z is modified. There are 3 possible cases:

1. There is an insertion in z, hence a word w ends up being inserted before a. Since $zv = av$ starts with a, w must start with an a, hence $w = aw'$. We insert $w'a$ to the right of z instead with the same operations. If z should be deleted or substituted, we apply the same operation to the new a instead. These operations yield the same result, and do not modify z.
2. There is a deletion in z, and hence a is deleted. Since this an optimal rewriting sequence, no a is created at that position through insertion or substitution afterwards. Since av starts with an a, u must be of the form $u = sau'$, where all the characters in s are deleted, and a isn't. Deleting sa instead of as yields the same result, and doesn't modify z.
3. There is a substitution in z, hence a is modified into a character $b \neq a$, that will not be further modified. Since av starts with a, an a has to be inserted in z, which is handled in case 1.

Hence, we can consider that every edit operations is done in u, and $d(au, av) = d(u, v)$. Suppose now that the result is proven for $|z| = k$, and let $z = az'$, with $|z'| = k$. Using the base case and the case for $|z| = k$, we have $d(zu, zv) = d(azu, azv) = d(zu, zv) = d(u, v)$, which concludes the proof. ◀

Proof of Lemma 10. Let $\#$ be a fresh parameterized letter. Let then $u = \#^{k+1}u_1\#u_2\# \ldots \#u_n$ and $v = \#^{k+1}v_1\#v_2\# \ldots \#v_n$, where $\#^{k+1}$ denotes $k+1$ repetitions of the character $\#$. The proof of the reverse direction is the same as in Lemma 9, so we only prove the other one.

Assume $FM_2^D(u, v) \leq k$. Let v' and f realize this parameterized match.

We prove that $f(\#) = \#$, and that no other character is sent to $\#$ by f. Indeed, v starts with $k + 1$ symbols $\#$, which ensure that v' starts with the letter $\#$. Since u starts with $\#$ and $f(u) = v'$, $f(\#) = \#$. Furthermore, this implies that since $|u|_\# = k + n$, $|f(u)|_\# = |v'|_\# \geq k+n$. Since v' is obtained from v by deletions, we have $|v'|_\# \leq |v|_\# = k+n$. Hence $|v'|_\# = k + n$ and all those inequalities are equalities, which is only the case when no $\#$ symbols is deleted from v, and that for all $x \neq \#$, $f(x) \neq \#$.

Since all the # symbols are left untouched, the rest of the proof is the same as in Lemma 9, and all of the factors u_i and v_i are parameterized matching. ◀

Proof of Lemma 11. Let # be a fresh parameterized letter, and $N = k + 2$.

Let then $u = \#^N u_1 \#^N u_2 \ldots \#^N u_n \#^N$ and $v = \#^N v_1 \#^N v_2 \ldots \#^N v_n \#^N$. Once again, we only prove the non-trivial implication.

Suppose $FM_1^D(u, v) \leq k$, and let f and u' realize this matching. Since u starts with $k+1$ copies of #, u' starts with #. Since v starts with # too, $f(\#) = \#$.
We now prove that we can consider that for all $x \neq \#$, $f(x) \neq \#$. This will also imply that no # symbol is deleted from u. Let $S = \{a \in \Pi \mid f(a) = \#\}$ be the set of symbols (different from #) sent to #. Since $|u|_\# = |v|_\#$, the number of deleted # symbols from u is exactly $|u|_S$, hence $|u|_S \leq k$. Let us now consider the leftmost occurrence of an element of S in u', that we denote by a. The letter a appears in u in a factor of the form $\#^N w_1 a w_2 \#^N$. Since all # in v appear in blocks of size N, a must contribute to such a block, after deletions and application of f. We distinguish two cases:

1. The entirety of the word w_1 is deleted. In this case, at least one symbol # from the left $\#^N$ block is deleted; otherwise $f(\#^N)f(a) = \#^{N+1}$ would be a factor of v, which is impossible. Thus, choosing not to delete # and to delete a instead yields the same result.
2. w_1 is not deleted. Since no character from S appears to the left of a, $f(a)$ is the start of a $\#^N$ block. Furthermore, since $|u|_S \leq k$, it is not possible to form $\#^N$ with only a and w_2, and characters from the right $\#^N$ contribute to it. Hence, at least one # symbol from this right block is deleted. Like before, the same result can be obtained by not deleting it, and deleting a instead.

Either way, we can repeat this process to eliminate all occurrences of characters of S and of deletions of #, which proves that we can consider that for all $x \neq \#$, $f(x) \neq \#$. Once again, we are taken back to the conditions of Lemma 9, and the rest of the proof follows. ◀

Proof of Theorem 15. We define Π like in Theorem 12, and we add the letters $\perp_1, \perp_2, \perp_3, \perp_4$ and $\perp_5$. Similarly, we define $u_1^i, v_1^i, u_\perp^i, v_\perp^i, u_2^e$, and v_2^e just like in Theorem 12. Additionally, we define for every edge e,

$$u_\perp^e = \Box_1^e \Box_2^e \Box_3^e \Box_4^e \Box_5^e \Box_6^e \text{ and } v_\perp^e = \perp_1 \perp_2 \perp_3 \perp_4 \perp_5.$$

We then apply Lemma 11 with

$$u_1^1, \ldots u_1^n, u_\perp^1, \ldots u_\perp^n, u_2^{e_1} \ldots u_2^{e_m}, u_\perp^{e_1}, \ldots u_\perp^{e_m}$$

and

$$v_1^1, \ldots v_1^n, u_\perp^1, \ldots v_\perp^n, v_2^{e_1} \ldots v_2^{e_m}, v_\perp^{e_1}, \ldots v_\perp^{e_m}$$

to obtain u, v, and k. We show that G is 3-colorable $\Leftrightarrow$ $FM_1^D(u, v) \leq k$.

$\Rightarrow$ Suppose G is 3 colorable. Define f like in Theorem 12 on the c_y^i and $\Box_y^e$. Let e be an edge and $k_e \in [1, 6]$ be the integer such that $f(\Box_{k_e}^e)$ is defined. We map every remaining $\Box_y^e$ in the following way:

$$f(\Box_i^e) = \begin{cases} \perp_i & \text{if } i < k_e, \\ Y^e & \text{if } i = k_e, \\ \perp_{i-1} & \text{if } i > k_e. \end{cases} \tag{1}$$

It is then easy to check that $d(f(u), v) = k$, and thus $FM_1^D(u, v) \leq k$.

$\Leftarrow$ Suppose $FM_1^D(u,v) \leq k$, and let f and u' realize it. We define a coloring of G based on f. We note, for $1 \leq i \leq n$ and $1 \leq t \leq 3$, $col(c_t^i) = c_t$. If x_i is a vertex of G, define $c(x_i)$ to be $col(c_k^i)$, where c_k^i is the only element such that $f(c_k^i) = x_i$. We show in what follows that (1) this function definition is correct and (2) it is a valid coloring, i.e. if $e = \{x_i, x_j\}$ is an edge, $c(x_i) \neq c(x_j)$.

(1): The same points 1. and 2. from the proof of Theorem 12 apply, hence for every $1 \leq i \leq n$, exactly one element from $\{c_1^i, c_2^i, c_3^i\}$ is sent to x_i, while the two others are sent to $\perp_1^i$ and $\perp_2^i$, hence the result.

(2): Let e be an edge. The words $u_\perp^e$ and $v_\perp^e$ are in matching, which is done with exactly one deletion. Hence, there exists k_e such that

$$f(\square_i^e) = \begin{cases} \perp_i & \text{if } i < k_e, \\ \perp_{i-1} & \text{if } i > k_e. \end{cases} \tag{2}$$

Moreover, u_2^e and v_2^e are in matching. Since Y^e appears in v_2^e and all the characters in u_2^e apart from $\square_{k_e}^e$ have an image different from Y^e, $f(\square_{k_e}^e) = Y^e$. Hence, the only characters that are not suppressed from u_2^e are the two characters between the $\square_{k_e}^e$. Denoting them by c and c', the construction of the word ensures that $col(c) \neq col(c')$. Hence, if $e = \{x_i, x_j\}$, we have proven $c(x_i) \neq c(x_j)$, which is (2).

The coloring c is therefore valid, which concludes the proof. ◀

Proof of Theorem 17. Let $G = (V, E)$, with $V = \{x_1, \ldots, x_n\}$ and $\{e_1, \ldots, e_m\}$. Like in the 1-to-1 case, we construct factors u_i and v_i to encode vertex coloring. The parameter alphabet contains:

- $x_1, \ldots x_n$, corresponding to V,
- the colors c_1, c_2, c_3,
- for every $e \in E$, the delimiters Y^e,
- for every $e \in E$ and every $1 \leq i, j \leq 3$, $i \neq j$, the delimiters $Y_{i,j}^e$.

We define for $1 \leq i \leq n$, $u_1^i = x_i$ and $v_1^i = c_1c_2c_3$. If e is an edge and c_i and c_j are two colors, we denote $w^e(c_i, c_j) = Y_{i,j}^e Y_{i,j}^e Y_{i,j}^e \ c_ic_j \ Y_{i,j}^e Y_{i,j}^e Y_{i,j}^e$. For every edge $e = \{x_i, x_j\}$, we now define $u_2^e = Y^eY^eY^e \ x_ix_j \ Y^eY^eY^e$ and $v_2^e = w^e(c_1,c_2)w^e(c_1,c_3)w^e(c_2,c_1)w^e(c_2,c_3)w^e(c_3,c_1)w^e(c_3,\, c_2)$.

We now apply Lemma 10 with $u_1^1, \ldots u_1^n, u_2^{e_1} \ldots u_2^{e_m}$, $v_1^1, \ldots v_1^n, v_2^{e_1} \ldots v_2^{e_m}$, to obtain u and v.

$\Rightarrow$ Suppose G is 3-colorable, and let $c : V \to \{c_1, c_2, c_3\}$ be a valid coloring. Define $f|_V = c$. For every edge $e = \{x_i, x_j\}$, let s and t be such that $c(x_i) = c_s$ and $c(x_j) = c_t$. We then define $f(Y^e) = Y_{s,t}^e$. It is easy to check now that $d(f(u), v) = k$, and hence $FM_2^D(u,v) \leq k$.

$\Leftarrow$ Suppose now that $FM_2^D(u,v) \leq k$. We will show that $f|_V$ defines a 3-coloring of G, by showing that (1) for all $x \in V$, $f(x) \in \{c_1, c_2, c_3\}$ and (2) If $\{x, y\} \in E$, then $f(x) \neq f(y)$.

- Lemma 10 ensures that the words u_i and v_i are in matching, which proves (1).
- Lemma 10 also ensures that the words u^e and v^e are in matching. Let $e \in E$, with $e = x_s, x_t$. We have $|u_2^e|_{Y^e} = 6$, hence $|f(u_2^e)|_{f(Y^e)} \geq 6$. Since c_1, c_2 and c_3 each occur exactly 4 times in v_2^e, they cannot occur 6 times after deletions, and $f(Y_e) \notin \{c_1, c_2, c_3\}$. Hence, there exist $i \neq j$ with $1 \leq i, j \leq 3$ such that $f(Y^e) = Y_{i,j}^e$. This implies that all but one of the w^e factors from v_2^e are suppressed, and that the remaining one is $w^e(c_i, c_j)$. Hence $f(x_s) = c_i$ and $f(x_t) = c_j$, which proves (2). ◀

B Encoding Constant Alphabet Σ in Π

We show why it is always possible to consider that $\Sigma = \emptyset$ for certain problems. These results use the lemmas proved in Section 4.1.

▶ **Lemma 23.** *Let d be a distance, k an integer and u and v be two parameterized words over the alphabet of constants Σ and the alphabet of parameters Π. There exist words $\tilde{u}$ and $\tilde{v}$ over the alphabet of constants $\emptyset$ and the alphabet of parameters $\Pi' = \Pi \uplus \Sigma$ such that the following are equivalent:*

- *$PM^d(u, v, k)$ is realized by f;*
- *$PM^d(\tilde{u}, \tilde{v}, k)$ is realized by f.*

In particular, this implies that if $PM^d(\tilde{u}, \tilde{v}) \leq k$, all functions f realizing this matching verify that for all $x \in \Sigma$, $f(x) = x$, and for all $x \in \Pi, f(x) \in \Pi$.

Proof. Let $N = k + 1$. If $\Sigma = \{a_1, \dots, a_n\}$, we define z to be $a_1^N a_2^N \dots a_n^N u$ and $\tilde{u} = zu$, $\tilde{v} = zv$. It is clear that if $PM^d(u, v) \leq k$ then $PM^d(\tilde{u}, \tilde{v}) \leq k$, by following the same operations, and applying the same renaming function.

Suppose now that $PM^d(\tilde{u}, \tilde{v}) \leq k$, and let f and u' realize it. Let $i \in [1, n]$. All the letters of u between position Ni and $N(i + 1)$ are a_i. At most k of these positions can be modified with an edit operation. Since $N > k$, at least one of these positions is not modified, and thus there exists $j \in [Ni, N(i + 1)]$ such that $u'_j = a_i$. Since all letters in v between position Ni and $N(i + 1)$ are a_i, in particular $v_j = a_i$, and hence $f(a_i) = a_i$. This proves that for all $x \in \Sigma$, $f(x) = x$, and thus $f(z) = z$. Since f is 1-to-1, this entails $f(\Pi) \subseteq \Pi$. By Lemma 7, $d(f(\tilde{u}), \tilde{v}) \leq k$. Hence $d(f(zu), zv) = d(zf(u), zv) \leq k$ and by Lemma 8, $d(f(u), v) \leq k$. Hence $PM^d(u, v) \leq k$. ◀

▶ Remark 24. Note that the words $\tilde{u}$ and $\tilde{v}$ have a size increased by $N\Sigma$. If less operations are considered, it is possible to reduce this overhead. For example, in the case of PM^D, we can take z to be of the form $a_1 \dots a_n z^N$, to reduce the overhead to $N + \Sigma$.

Similarly, constants can be encoded in Π in some FM problems. We prove this result for FM_2^D, with the help of the block decomposition allowed by Lemma 10.

▶ **Lemma 25.** *Let u and v be two parameterized words over the alphabet of constants Σ and the alphabet of parameters Π. There exist words $\tilde{u}$ and $\tilde{v}$ over the alphabet of constants $\emptyset$ and the alphabet of parameters $\Pi' = \Pi \uplus \Sigma$ such that the following are equivalent:*

- *$FM_2^D(u, v, |v| - |u|)$ is realized by f;*
- *$FM_2^D(\tilde{u}, \tilde{v}, |\tilde{v}| - |\tilde{u}|)$ is realized by f.*

Proof. We write $\Sigma = \{a_1, \dots a_n\}$ and $\Pi = \{b_1, \dots, b_m\}$. We define $z_\Sigma = a_1 \dots a_n$, and $z_\Pi = b_1 \dots b_m$. Let $\tilde{u}$ and $\tilde{v}$ be the words obtained by applying Lemma 10 to $z_\Sigma, b_1, b_2, \dots, b_m, u$ and $z_\Sigma, z_\Pi, z_\Pi, \dots, z_\Pi, v$. If $FM_2^D(u, v, k)$ is realized by a function f, it realizes $FM_2^D(\tilde{u}, \tilde{v}, |\tilde{v}| - |\tilde{u}|)$ too. Indeed, it is enough to apply the same operations in v, and to delete all the characters but $f(b_i)$ in the i-th copy of z_Π.

Suppose now that $FM_2^D(\tilde{u}, \tilde{v}) \leq k$, and let f realize it. Then, by Lemma 10, we have:

- $D(z, f(z)) = 0$, and hence $f(z) = z$, which implies that for all $x \in \Sigma$, $f(x) = x$.
- For every $1 \leq i \leq m$, $D(z_\Pi, f(b_i)) = |\Pi| - 1$. Hence $f(b_i)$ is a character of z_Π, which is some character $b_j \in \Pi$.
- $D(v, f(u)) \leq k$.

Hence f verifies $D(f(v), u) \leq k$ and respects the conditions on Π and Σ, which implies that is also realizes $FM_2^D(u, v, k)$. ◀

The overhead to pay for this transformation is $O(|\Sigma| + |\Pi|^2 + k)$, where the term in k comes from the proof of Lemma 10.

Transposing the technique used for Lemma 25 is not sufficient to get a similar result for FM_1^D. The question thus remains open in this context.

C Proofs Regarding the Max-SAT Encoding

Proof of theorem 21. We proceed by induction on $|u| + |v|$. If $|u| + |v| = 0$, both u and v are the empty string, and the equivalence is trivial. Fix $n \in \mathcal{N}$ and suppose now that the result holds up for all words u, v such that $|u| + |v| \leq n - 1$. Let u and v be two words such that $|u| + |v| \leq n$. Without loss of generality, consider $|u| \geq |v|$.
Suppose $ID(u, v) \leq k$. Let ρ be a rewriting sequence between u and v of length k. If there is no deletion in u in ρ, there are only insertions in v, and v is a sub-word of u, and there exists another rewriting sequence ρ' only deleting letters from u. Hence, we can consider that there is at least a deletion in u in ρ. Let p be a position at which such a deletion occur, and let $a = u_p$. The word u can be written as $u = u'au''$ for some words u' and u''. Define $w = u'u''$. It holds that $d(w, v) \leq k - 1$ and $|w| = |u| - 1$. By induction, there exists an alignment A between w and v such that $2|A| = |w| + |v| - (k - 1) = |u| + |v| - k$. We define $r(i) = \begin{cases} i \text{ if } i < p \\ i - 1 \text{ if } i > p \end{cases}$, and $B = \{(r(i), j) \mid (i, j) \in A\}$. Since A is an alignment, so is B: it satisfies conditions 1 to 3 of Definition 20, and since $w_r(i) = u_i$, it also satisfies condition 4. Finally, $|B| = |A|$, hence $2|B| = |u| + |v| - k$, hence the result.

Suppose now that there exists an alignment A such that $2|A| = |u| + |v| - k$. Similarly, consider p, a position in u such that there does not exist a j with $(p, j) \in A$. If no such position exist, since $|u| \geq |v|$, $u = v$ and the result is proven. Consider w the word obtained by deleting u_p from u. It then holds that $|w| = |u| - 1$ and that $2|A| = |u| + |v| - k = |w| + |v| - (k - 1)$. Defining B in the same way as above yields an alignment between w and v of the same size, and thus by induction, $d(w, v) \leq k - 1$, and since $d(u, w) = 1$, $d(u, v) \leq k$. ◀

▶ **Proposition 26.** *Weighted Max-SAT and partial weighted Max-SAT are equivalent.*

Proof. Encoding a weighted Max-SAT instance as a partially weighted Max-SAT instance is straightforward, as we just have to choose φ_c to be empty.
Conversely, given a satisfiable CNF formula φ_c, a CNF formula φ_w, and a weight function w on the clauses of φ_w, we can define a weighted Max-Sat instance in the following way:

- We define $\varphi = \varphi_c \land \varphi_w$
- We set $W = 1 + \sum_{C_i \text{clause of} \varphi_c} w(C_i)$, and extend w to clauses of φ_c such that $w(C_j) = W$ for all clauses C_j of φ_c

If ν is a valuation, we denote by $w(\nu)$ the sum of the weights of all clauses it satisfies $\sum_{\nu \vDash C_i} w(C_i)$.
Since φ_c is satisfiable, there exists a valuation ν_c such that $\nu_c \vDash \varphi_c$, and $w(\nu_c) \geq |\varphi_c|W$. Let now ν be a valuation no satisfying a clause of φ_c. Then $w(\nu_c) \leq (|\varphi_c| - 1)W + (W - 1) < w(\nu_c)$, hence nu_c is not maximal and cannot be a solution to the weighted Max-SAT instance. ◀