The Intractability of Computing the Hamming Distance [⋆]

Bodo Manthey [1,∗], Rüdiger Reischuk

Universität zu Lübeck
Institut für Theoretische Informatik
Ratzeburger Allee 160, 23538 Lübeck, Germany

Abstract

Given a string x and a language L, the Hamming distance of x to L is the minimum Hamming distance of x to any string in L. The edit distance of a string to a language is analogously defined.

First, we prove that there is a language in AC^0 such that both Hamming and edit distance to this language are hard to approximate; they cannot be approximated with factor $O(n^{\frac{1}{3}-\epsilon})$, for any $\epsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$ (n denotes the length of the input string).

Second, we show the parameterized intractability of computing the Hamming distance. We prove that for every $t \in \mathbb{N}$ there exists a language in AC^0 for which computing the Hamming distance is $\mathsf{W}[t]$-hard. Moreover, there is a language in P for which computing the Hamming distance is $\mathsf{W}[\mathsf{P}]$-hard.

Then we show that the problems of computing the Hamming distance and of computing the edit distance are in some sense equivalent by presenting approximation ratio preserving reductions from the former to the latter and vice versa.

Finally, we define HamP to be the class of languages to which the Hamming distance can efficiently, i.e. in polynomial time, be computed. We show some properties of the class HamP. On the other hand, we give evidence that a characterization in terms of automata or formal languages might be difficult.

Key words: Hamming distance, edit distance, computational complexity, parameterized complexity, inapproximability

1 Introduction

Given a language L and a string x, one can ask whether there is a string in L in the "neighborhood" of x and how to find such a string. On the other hand, one can ask for the minimum distance of x to any string in L. Hamming and edit distance are widely used for measuring the distance. One topic in which these problems arise is for example the field of error-correcting codes (see e.g. Spielman [32] and Vardy [34]). Another field is parsing theory. A main problem when designing a parser is recovery from syntax errors. This problem has been solved for context-free languages [2,22,25,30]. Furthermore, the problem of computing distances between strings has gained popularity in computational biology [9,16,29]. From the computational complexity point of view, it is interesting whether there are properties other than membership that can efficiently be computed for appropriate classes of languages. Hemachandra [19] examined what kind of operations, such as approximate membership queries, can efficiently be performed, even if the set considered does not allow efficient membership testing. Allender et al. [3] considered the so called maximal word function. This is the question of finding the lexicographically maximal word smaller than the input string. They present an algorithm for solving this problem for languages in $\mathsf{1NAuxPDA^p}$.

1.1 Previous results

Computing the Hamming distance of two strings is easy. The edit distance of two strings of length n and m, respectively, can be computed via dynamic programming in time $O(n \cdot m)$ [16] using linear space [20]. The time bound has been improved by Masek and Paterson [27] to $O(n \cdot \max\{1, m/\log n\})$ for $n \geq m$. Batu et al. and Bar-Yossef et al. gave sublinear time algorithms for approximating the edit distance [5,6]. Pighizzini [31] presented a language in $\mathsf{co\text{-}NTime}(\log)$, which is a subclass of AC^0, for which computing the edit distance is NP-hard. On the other hand, he showed that computing the edit distance to languages in $\mathsf{1NAuxPDA^p}$ can be done in polynomial time and even in AC^1. $\mathsf{1NAuxPDA^p}$ denotes the class of all languages that can be recognized by logarithmic space and polynomial time bounded nondeterministic Turing machines equipped with a one-way input tape and an auxiliary pushdown

* A preliminary version of this work was presented at the 14th International Symposium on Algorithms and Computation (ISAAC), Dec. 15–17, 2003, Kyoto, Japan [26].

* Corresponding author.

Email addresses: `manthey@tcs.uni-luebeck.de` (Bodo Manthey),
`reischuk@tcs.uni-luebeck.de` (Rüdiger Reischuk).

[1] Supported by DFG research grant RE 672/3.

$$\begin{tikzcd}[column sep=small]
 & & & & \mathsf{LOGCFL} = \mathsf{NAuxPDA^p} \arrow[dl, dash] \arrow[dr, dash] & & \\
 & & & \mathsf{NL} \arrow[dl, dash] \arrow[dr, dash] & & \mathbf{1NAuxPDA^p} \arrow[dl, dash] \arrow[dr, dash, "\neq"] & \\
 & & \mathsf{L} \arrow[dl, dash, "\neq"'] \arrow[dr, dash, "\neq"] & & \mathsf{1NL} \arrow[dl, dash, "\neq"] & & \mathsf{CFL} \\
 & \mathsf{AC^0} \arrow[dl, dash, "\neq"'] & & \mathsf{1L} & & & \\
 \mathbf{co\text{-}NTime(log)} & & & & & &
\end{tikzcd}$$

Fig. 1. The relationship between several complexity classes.

store [7]. Without restricting the input tape to be one-way, such machines characterize LOGCFL, the closure of context-free languages under log-space reductions [33]. If we further omit the bound on the running time, such machines characterize the class P, independent of whether they are deterministic or nondeterministic [10]. Figure 1 gives an overview over the relationships between the complexity classes examined in this paper.

1.2 Our results

We consider computing the Hamming distance and computing the edit distance from a complexity theory point of view. Intuitively, computing the edit distance seems to be harder than computing the Hamming distance. Thus, one might hope that Pighizzini's hardness result for computing the edit distance does not hold for computing the Hamming distance. However, we show that this is not the case and even improve the intractability bound. This will be done by showing that the problem is hard to approximate and intractable in the sense of parameterized complexity, even for languages in very small complexity classes.

To be more precise, we present a language in AC^0 with the property that the Hamming distance of strings of length n to this language cannot be approximated in polynomial time with factor $O(n^{\frac{1}{3}-\epsilon})$, unless $\mathsf{P} = \mathsf{NP}$ (Section 3).

Furthermore, for a language L, we consider the parameterized language where on input x we ask whether there is a string $y \in L$ within distance k. We present a language in P for which this is $\mathsf{W[P]}$-hard for both Hamming and edit distance (Section 4.1). Moreover, we prove that for every $t \in \mathbb{N}$ there is a language in AC^0 for which this is $\mathsf{W}[t]$-hard for the Hamming distance (Section 4.2).

Then we reduce the problem of computing the Hamming distance to the problem of computing the edit distance and vice versa (Section 5). Hence, both problems are in some sense equivalent with respect to their approximability.

Finally, we consider the problem of computing the Hamming distance from a more abstract point of view (Section 6). We introduce HamP as the class of languages to which Hamming distances can efficiently be computed and reveal some properties of this class. Furthermore, we give evidence that a complete characterization of HamP in terms of automata or formal languages might be difficult.

2 Preliminaries

Let Σ be a finite alphabet. The length of a string x over Σ will be denoted by $|x|$. For two strings x and y of equal length, let $h(x, y)$ be the *Hamming distance* of x and y, i.e. the number of positions where x and y differ [18]. The Hamming distance of a language L over Σ to a string $x \in \Sigma^\star$ is the minimum Hamming distance of x to an element of L, i.e.

$$h(x, L) = \min\{h(x, y) \mid y \in L \text{ and } |y| = |x|\} .$$

If $\Sigma^{|x|} \cap L = \emptyset$, i.e. if there is no string of length $|x|$ in L, we define $h(x, L) = \infty$.

Let $\Delta \notin \Sigma$ denote the gap symbol. An *alignment* of two strings x and y over Σ is a pair of strings $\tilde{x}$ and $\tilde{y}$ over $\Sigma \cup \{\Delta\}$ such that $|\tilde{x}| = |\tilde{y}|$ and $\tilde{x}$ and $\tilde{y}$ are obtained from x and y, respectively, by inserting gap symbols. We assume that at neither position both $\tilde{x}$ and $\tilde{y}$ have a gap. We define the *edit distance* $d(x, y)$ of two strings x and y as

$$d(x, y) = \min\{h(\tilde{x}, \tilde{y}) \mid (\tilde{x}, \tilde{y}) \text{ is an alignment of } (x, y)\} .$$

The edit distance of two strings x and y is the minimum number of insertions, deletions, and substitutions of characters in x necessary to obtain y. The notion of edit distance is due to Levenshtein [24]. In contrast to the Hamming distance, x and y do not have to be of the same length. In general, we can allow an arbitrary function that yields some penalty for each operation depending on the participating characters. See for example Gusfield [16] or Navarro [28] for a survey on computing edit distances between two or more sequences. To obtain the hardness results, it suffices to restrict ourselves to the simplest case where all insertions, deletions and substitutions are at unit costs.

According to Pighizzini [31], the edit distance of a string x to a language L is defined as

$$d(x, L) = \min\{d(x, y) \mid y \in L\} .$$

We consider the problem of computing the Hamming distance or the edit distance of a language and a string in two different ways, namely as an optimization problem and as a parameterized language.

Definition 1 (Optimization Problems) *Let $L \subseteq \{0,1\}^\star$ be a language. We define $\mathsf{OPT}_\mathsf{H}(L)$ to be the following optimization problem:*

(1) An instance of $\mathsf{OPT}_\mathsf{H}(L)$ is a string $x \in \{0,1\}^\star$.
(2) A solution to an instance x is a string $y \in L$ with $|y| = |x|$.
(3) The goal is to find a string in L with minimum Hamming distance to x.

$\mathsf{OPT}_\mathsf{E}(L)$ is similarly defined: We omit the length constraint, i.e. all $y \in L$ are feasible solutions, and we use the edit distance as measure.

Both $\mathsf{OPT}_\mathsf{H}(L)$ and $\mathsf{OPT}_\mathsf{E}(L)$ are NP-optimization problems (see e.g. Ausiello et al. [4]), if $L \in \mathsf{P}$.

Definition 2 (Hamming/Edit Closures) *Let $L \subseteq \{0,1\}^\star$ be a language. Then*

$$L_\mathsf{H} = \{(x,k) \mid \exists y \in L : |x| = |y| \wedge h(x,y) \leq k\} .$$

L_E is similarly defined: We replace h by d and omit the constraint $|x| = |y|$. L_H and L_E are called the Hamming and edit closure of L, respectively.

If $L \in \mathsf{NP}$, then both L_H and L_E are in NP as well. Throughout this work, we consider Hamming and edit closures as parameterized languages with k as parameter. We will also consider Hamming and edit closures corresponding to classical complexity classes.

Definition 3 (Classes of Hamming/Edit Closures) *Let C be a class of languages. Then the class C_H of Hamming closures of languages in C is defined as $\mathsf{C}_\mathsf{H} = \{L_\mathsf{H} \mid L \in \mathsf{C}\}$. Analogously, the class C_E of edit closures of languages in C is defined as $\mathsf{C}_\mathsf{E} = \{L_\mathsf{E} \mid L \in \mathsf{C}\}$.*

3 The hamming distance is hard to approximate

In this section, we prove that there is a language $L \in \mathsf{AC}^0$ such that the Hamming distance to L cannot be approximated with factor $O(n^{\frac{1}{3}-\epsilon})$, for any $\epsilon > 0$, for strings of length n, unless $\mathsf{P} = \mathsf{NP}$.

We reduce from the optimization problem *Minimum Independent Dominating Set* (MIDS). An instance of MIDS is an undirected graph $G = (V, E)$. A solution is a subset $\tilde{V} \subseteq V$ of vertices that is both an independent set and a dominating set. $\tilde{V}$ is an independent set of G, if for every edge $\{u, v\} \in E$ at most one of the vertices u and v is in $\tilde{V}$. $\tilde{V}$ is a dominating set of G, if for every vertex $u \in V \setminus \tilde{V}$ there exists a node $v \in \tilde{V}$ with $\{u, v\} \in E$. The goal is to minimize the size of $\tilde{V}$. Every graph possesses an independent dominating set, since every maximal independent set is also dominating: If there were a node not contained in a maximal independent set and not adjacent to any other node in the set, then we can add this node to the set, contradicting its maximality. The problem MIDS is therefore also known as *Minimum Maximal Independent Set*, since an independent dominating set is an independent set that cannot be extended. Halldórsson [17] showed that MIDS cannot be approximated with factor $O(|V|^{1-\epsilon})$, for any $\epsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$.

Consider the following language over the alphabet $\{0,1\}$:

$$\begin{aligned} L^{\mathsf{MIDS}} = \{G_1 \dots G_{m+1}\tilde{V} \mid\ & G_\ell \in \{0,1\}^{\binom{m}{2}},\ \tilde{V} \in \{0,1\}^m \text{ for some } m \in \mathbb{N}, \\ & G_1 = \dots = G_{m+1},\ \text{each } G_\ell \text{ is an encoding of the same } m\text{-vertex} \\ & \text{graph } G,\ \text{and } \tilde{V} \text{ encodes an independent dominating set of } G\} \end{aligned}$$

An encoding G_ℓ $(1 \le \ell \le m+1)$ consists of $\binom{m}{2}$ bits $(e^\ell_{i,j})_{1\le i<j\le m}$. (For simplicity, we set $e^\ell_{i,j} = e^\ell_{j,i}$ for $i > j$.) We have $e^\ell_{i,j} = 1$ if and only if $\{v_i, v_j\} \in E$. The set $\tilde{V}$ is encoded with m bits z_i $(1 \le i \le m)$ with $z_i = 1$ if and only if $v_i \in \tilde{V}$.

Let us first show that $L^{\mathsf{MIDS}} \in \mathsf{AC}^0$. We build the following circuit:

$$\begin{aligned} \text{DOM} &= \bigwedge_{i=1}^{m} \left(z_i \vee \bigvee_{j=1}^{m} (z_j \wedge e^1_{i,j}) \right) , \\ \text{IND} &= \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{m} \Big((z_i \wedge z_j) \to \neg e^1_{i,j} \Big) , \\ \text{EQU} &= \bigwedge_{i=1}^{m} \bigwedge_{j=i+1}^{m} \left(\bigwedge_{\ell=1}^{m+1} e^\ell_{i,j} \vee \bigwedge_{\ell=1}^{m+1} \neg e^\ell_{i,j} \right) , \text{ and} \\ \text{OUTPUT} &= \text{DOM} \wedge \text{IND} \wedge \text{EQU} . \end{aligned}$$

We have DOM $= 1$ if and only if $\tilde{V}$ is a dominating set and IND $= 1$ if and only if $\tilde{V}$ is an independent set of G. Furthermore, EQU $= 1$ if and only if the matrices $(e^\ell_{i,j})_{1\le i<j\le m}$ encode the same graph G for any $1 \le \ell \le m+1$. Hence, OUTPUT $= 1$ if and only if the input is in L^{MIDS}. The circuit family implementing the above formulas has constant depth and is logarithmic space uniform. Thus, $L^{\mathsf{MIDS}} \in \mathsf{AC}^0$.

Theorem 4 *For any $\epsilon > 0$, $\mathsf{OPT_H}(L^{\mathsf{MIDS}})$ cannot be approximated in polynomial time with factor $O(n^{\frac{1}{3}-\epsilon})$ for strings of length n, unless* $\mathsf{P} = \mathsf{NP}$.

PROOF. Let a graph $G = (V, E)$ with $|V| = m$ be given as an instance for MIDS. We create an input string x as an instance for $\mathsf{OPT_H}(L^{\mathsf{MIDS}})$ by encoding the graph G by $(e^{\ell}_{i,j})_{1\leq i<j\leq m}$ for $1 \leq \ell \leq m+1$ and setting $z_i = 0$ for all $1 \leq i \leq m$.

Since every graph has an independent dominating set, we have $h(x, L^{\mathsf{MIDS}}) \leq m$. Thus, there exists a string $y \in L^{\mathsf{MIDS}}$ with $|x| = |y|$ and $h(x, y) \leq m$. Since the encoding of the graph G consists of $m+1$ identical copies, all differences between x and such a y are within the encoding of $\tilde{V}$. Thus, y yields an independent dominating set of size $h(x, y)$ for G.

A factor $O(m^{1-\epsilon'})$ approximation algorithm for $\mathsf{OPT_H}(L^{\mathsf{MIDS}})$ would yield an $O(m^{1-\epsilon'})$ approximation for MIDS. The theorem follows by choosing the length of the instance for L^{MIDS} as $n = \frac{1}{2}\cdot(m^3+m)$, which corresponds to an encoding of a graph and an independent dominating set of a graph with m nodes. $\square$

Theorem 5 *For any $\epsilon > 0$, $\mathsf{OPT_E}(L^{\mathsf{MIDS}})$ cannot be approximated in polynomial time with factor $O(n^{\frac{1}{3}-\epsilon})$ for strings of length n, unless $\mathsf{P} = \mathsf{NP}$.*

PROOF. Let $y \in L^{\mathsf{MIDS}}$ be a string with minimum edit distance to x (x is given as in the proof of Theorem 4). If $|y| \neq |x|$, then $d(x, y) > m$. Thus, we can assume that x and y are of equal length. Any difference in the graph encoding part of x and y yields $d(x, y) > m$ as well. Thus, x and y differ only in the last m positions. Within that part, x has m zeros. The string y encodes an independent dominating set $\tilde{V}$ of G, thus contains $|\tilde{V}|$ many ones in the last m positions. Hence, we have $d(x, y) = |\tilde{V}|$. $\square$

Thus, even in the small class AC^0 there exists a language such that both Hamming and edit distance to this language are hard to approximate. This may not be surprising: From Fagin's characterization of NP in terms of existential second-order logic [14], we get that all languages in NP have logarithmic space bounded verifiers [21, Thm. 3.1, Prop. 7.6] and there are languages, like e.g. $\mathsf{3SAT}$, that have even AC^0 algorithms to verify their certificates.

4 Parameterized intractability of hamming closures

4.1 Parameterized intractability of $\mathsf{P_H}$

The aim of this section is to analyze the complexity of Hamming closures of languages in P. On the one hand, we prove that the Hamming closures of languages in P are in $\mathsf{W[P]}$. On the other hand, there exists a language in P the Hamming closure of which is $\mathsf{W[P]}$-hard.

According to Downey and Fellows [11,13], $\mathsf{W[P]}$ is the class of parameterized languages that can be reduced to $\mathsf{EW\text{-}Circ\text{-}SAT}$. This problem is defined as follows:

$$\begin{aligned}\mathsf{EW\text{-}Circ\text{-}SAT} = \{(C,k) \mid\, & C \text{ is a Boolean circuit and has a satisfying} \\ & \text{assignment with weight } \textit{exactly } k\}\ .\end{aligned}$$

(Here, EW stands for *Exactly Weighted.* Downey and Fellows called the problem *Weighted Circuit Satisfiability.*) The weight of an assignment is the number of variables to which the value 1 has been assigned. We consider the following variant of weighted circuit satisfiability:

$$\begin{aligned}\mathsf{W\text{-}Circ\text{-}SAT} = \{(C,k) \mid\, & C \text{ is a Boolean circuit and has a satisfying} \\ & \text{assignment with weight } \textit{at most } k\}\ .\end{aligned}$$

It follows easily from work of Abrahamson et al. [1] that $\mathsf{W\text{-}Circ\text{-}SAT}$ is $\mathsf{W[P]}$-hard as well.

Theorem 6 $\mathsf{P_H} \subseteq \mathsf{W[P]}$.

PROOF. Consider an arbitrary language $L \in \mathsf{P}$. We reduce L_H to $\mathsf{W\text{-}Circ\text{-}SAT}$ to show that $L_\mathsf{H} \in \mathsf{W[P]}$. Assume that $L \subseteq \Sigma^\star$ for some finite alphabet $\Sigma = \{\alpha_1, \alpha_2, \ldots, \alpha_\sigma\}$. Let $g : \Sigma^\star \to \{0,1\}^\star$ be a homomorphism with $g(\alpha_i) = 0^{i-1}10^{\sigma-i}$. Note that $i \neq j$ implies $g(\alpha_i) \neq g(\alpha_j)$. We consider the language $g(L) = \{g(x) \mid x \in L\}$. Clearly, $g(L) \in \mathsf{P}$. Furthermore, we have $(x,k) \in L_\mathsf{H}$ if and only if $(g(x), 2k) \in g(L)_\mathsf{H}$. Since $g(L) \in \mathsf{P}$, there is a logarithmic space uniform circuit family of polynomial size for deciding $g(L)$ (see e.g. Vollmer [35]). Let C_n be the circuit in this family for strings of length n. Assume that we have an input string $y = y_1 \ldots y_n$. We modify C_n as follows to obtain a circuit $C_{n,y}$. If $y_i = 0$, then we leave the ith input bit unchanged. If $y_i = 1$, then we negate the ith input bit by interposing a NOT gate. Now C_n accepts y if and only if $C_{n,y}$ accepts 0^n. Furthermore, C_n accepts a string $\hat{y}$ if and only if $C_{n,y}$ accepts z with $z_i = y_i \oplus \hat{y}_i$, i.e. $C_{n,y}(z) = C_n(\hat{y})$.

Overall, we have

$$(x,k) \in L_\mathsf{H} \;\Leftrightarrow\; (g(x), 2k) \in g(L)_\mathsf{H} \;\Leftrightarrow\; (C_{|x|\cdot\sigma, g(x)}, 2k) \in \mathsf{W\text{-}Circ\text{-}SAT}\ ,$$

which proves the theorem. $\square$

Now we prove that there is a language in P the Hamming closure of which is $\mathsf{W[P]}$-hard. Therefore, we consider the circuit value problem CVP, which is defined as

$$\mathsf{CVP} = \{(C, x) \mid C \text{ is a Boolean circuit that outputs 1 on input } x\} .$$

Ladner [23] (see also Vollmer [35]) proved that CVP is P-complete. We consider the following variant of CVP, which is P-complete as well:

$$\mathsf{CVP}' = \{(\underbrace{C\#C\#\ldots\#C}_{(n+1)\text{ times}}, x) \mid (C, x) \in \mathsf{CVP} \text{ and } C \text{ has } n \text{ input bits}\} .$$

Theorem 7 $\mathsf{CVP}'_{\mathsf{H}}$ *is* $\mathsf{W[P]}$*-complete.*

PROOF. Let (C, k) be an instance for $\mathsf{W\text{-}Circ\text{-}SAT}$, such that C has n input bits. W.l.o.g. we assume $k \le n$. Then $X = ((C\#C\#\ldots\#C, 0^n), k)$ is an instance of $\mathsf{CVP}'_{\mathsf{H}}$ with $(C, k) \in \mathsf{W\text{-}Circ\text{-}SAT}$ if and only if $X \in \mathsf{CVP}'_{\mathsf{H}}$. Hence, we have reduced $\mathsf{W\text{-}Circ\text{-}SAT}$ to $\mathsf{CVP}'_{\mathsf{H}}$.

$\mathsf{CVP}'_{\mathsf{H}}$ is in $\mathsf{W[P]}$ due to Theorem 6 $\square$

4.2 Parameterized intractability of $\mathsf{AC}^0_{\mathsf{H}}$

A Boolean formula is called t-normalized, if it has the form "AND-of-ORs-of-ANDs-of-…-of-Literals" with t alternations [11,13]. For example, CNF formulas are 2-normalized. Consider the parameterized language

$$\mathsf{W}\text{-}t\text{-}\mathsf{SAT} = \{(F, k) \mid F \text{ is a } t\text{-normalized Boolean formula and has a satisfying assignment with at most } k \text{ ones}\} .$$

$\mathsf{W}\text{-}t\text{-}\mathsf{SAT}$ is $\mathsf{W}[t]$-complete for all $t \ge 2$ [8,13], while $\mathsf{W\text{-}1\text{-}SAT}$ is fixed parameter tractable [13]. Let us now encode a t-normalized formula F over n variables into a binary string. Therefore, we view F as a rooted tree T with vertices arranged in levels $V_1 \cup V_2 \cup \ldots \cup V_t$. The vertices in level V_ℓ $(1 \le \ell \le t-1)$ are labeled with AND, if ℓ is odd, and with OR, if ℓ is even. Every vertex $v \in V_t$ is labeled with $\mathrm{lit}(v)$ which is either a variable or a negated variable. For every vertex $v \in V_\ell$ we have a set $\mathrm{Adj}(v) \subseteq V_{\ell+1}$ that contains all those vertices in $V_{\ell+1}$ that serve as input bits for v. Thus, we can write F as (assume that t is even, if t is odd, then the formula ends with an AND)

$$F = \bigwedge_{v_1 \in V_1} \bigvee_{v_2 \in \mathrm{Adj}(v_1)} \bigwedge_{v_3 \in \mathrm{Adj}(v_2)} \cdots \bigvee_{v_t \in \mathrm{Adj}(v_{t-1})} \mathrm{lit}(v_t) .$$

We have $|V_1| \le |V_2| \le \ldots \le |V_t|$, since T is a tree, and we can assume that $|V_t| \ge n$. Otherwise, there would be unused variables. We call $m = |V_t|$ the size of F. We can encode every subgraph connecting vertices in $V_{\ell+1}$ to vertices in V_ℓ by an $m \times m$-matrix $(e^\ell_{i,j})_{1\le i,j\le m}$. Hence, we can write F as

$$\begin{aligned}F &= \bigwedge_{i_1=1}^{m} \bigvee_{i_2=1}^{m} \bigwedge_{i_3=1}^{m} \cdots \bigvee_{i_t=1}^{m} \left(\left(\bigwedge_{\ell=1}^{t-1} e^\ell_{i_{\ell+1},i_\ell} \right) \to \mathrm{lit}(v_{i_t}) \right) \\ &= \bigwedge_{i_1=1}^{m} \bigvee_{i_2=1}^{m} \bigwedge_{i_3=1}^{m} \cdots \bigvee_{i_t=1}^{m} \left(\mathrm{lit}(v_{i_t}) \vee \bigvee_{\ell=1}^{t-1} \neg e^\ell_{i_{\ell+1},i_\ell} \right) .\end{aligned}$$

Similar to the reduction presented in Section 4.1, we can create $m+1$ copies of each of these matrices. Thus, for each t and m there exists a circuit of depth t and size polynomial in m and t that evaluates each t-normalized formula of size m. (W.l.o.g. we assume that we have m input variables. Otherwise, we add $m-n$ variables that are never used.) The circuit family obtained characterizes the language

$$\begin{aligned}t\text{-}\mathsf{VAL} = \{(M,x) \mid\ & M \text{ is an encoding of a } t\text{-normalized formula } F \text{ as} \\ & \text{described above and outputs 1 on input } x \in \{0,1\}^m\} .\end{aligned}$$

and has constant depth for constant t. (We have omitted mentioning the copies of the formula encoding. But the equality of all copies of the encodings can be implemented with a circuit of depth 2, thus we obtain circuits of depth t for all $t \geq 2$.) Clearly, it is also logarithmic space uniform, thus, t-$\mathsf{VAL} \in \mathsf{AC}^0$ for any $t \in \mathbb{N}$.

Lemma 8 *For every $t \geq 2$, t-VAL_H is $\mathsf{W}[t]$-hard.*

PROOF. Let (F,k) be an instance for W-t-SAT and m be the size of F. We construct a circuit as described above. The input X for the circuit is as follows: The first bits encode the formula F and the last m bits are set to 0. Assume that $(F,k) \in \mathsf{W}$-t-SAT. We derive Y from X by setting a bit representing an input bit to 1, if the corresponding bit in the satisfying assignment for F is set to 1. Thus, $h(X,Y) \leq k$ and the circuit constructed accepts Y. On the other hand, assume that there is a Y with $h(X,Y) \leq k \leq m$ that is accepted by the circuit. Then X and Y encode the same formula and Y yields a satisfying assignment for F with weight at most k. Hence, we have reduced W-t-SAT to t-VAL_H. $\square$

Thus, for every $t \in \mathbb{N}$ there is a language $L \in \mathsf{AC}^0$ such that L_H is $\mathsf{W}[t]$-hard.

On the other hand, for any $L \in \mathsf{AC}^0$, we have a logarithmic space uniform circuit family of constant depth and polynomial size. The languages t-VAL_H

have such circuits of depth t. Thus, their circuits have also weft bounded by t and are thus in $\mathsf{W}[t]$ [12]. (The weft of a circuit is defined to be the maximum number of gates with unbounded fan-in, i.e. fan-in exceeding some preagreed bound, on any path from the input variables to the output gate [13].) Together with the above lemma, we get the following result.

Theorem 9 *For every $t \geq 2$, t-VAL_H is $\mathsf{W}[t]$-complete.*

From the fact that all languages in AC^0 have circuits of bounded weft, we get that all languages in the Hamming closure of AC^0 are indeed in $\mathsf{W}[t]$ for some $t \in \mathbb{N}$.

Theorem 10 $\mathsf{AC}^0_\mathsf{H} \subseteq \bigcup_{t\in\mathbb{N}} \mathsf{W}[t]$.

Let $\mathsf{AC}^0[t]$ be the class of languages in AC^0 that have circuits of depth bounded by t. Then we have $\mathsf{AC}^0[t]_\mathsf{H} \subseteq \mathsf{W}[t]$. On the other hand, t-$\mathsf{VAL}_\mathsf{H} \in \mathsf{AC}^0[t]$ is $\mathsf{W}[t]$-complete. Thus, together with Theorems 6 and 7, we obtain the following alternative characterization of the W hierarchy.

Proposition 11 *Let $\mathsf{AC}^0[t]^\star_\mathsf{H}$ and $\mathsf{P}^\star_\mathsf{H}$ be the closure of $\mathsf{AC}^0[t]_\mathsf{H}$ and P_H, respectively, under parameterized reductions.*

Then $\mathsf{P}^\star_\mathsf{H} = \mathsf{W}[\mathsf{P}]$ and $\mathsf{AC}^0[t]^\star_\mathsf{H} = \mathsf{W}[t]$ for all $t \geq 2$.

Parameterized reductions are used for proving hardness results in parameterized complexity [13].

This alternative characterization looks natural since the weighted satisfiability problems, which are the canonical complete problems in parameterized complexity, are closely related to Hamming distance problems: They can be viewed as the problem of finding a satisfying assignment with Hamming distance at most (or exactly) k from the zero assignment.

5 Edit distance versus hamming distance

5.1 Reduction from hamming distance to edit distance

Let L be a language to which we want to compute the Hamming distance. For every $x \in \{0,1\}^n$, let $x' = 0^n1^nx_10^n1^n\ 0^n1^nx_20^n1^n \ldots 0^n1^nx_n0^n1^n$. We construct a language L' as

$$L' = \{x' \mid x = x_1x_2\ldots x_n \in L\} .$$

Thus, every string x of length n has a counterpart x' of length $(4 \cdot n + 1) \cdot n$. Consider the substring $0^n 1^n x_i 0^n 1^n$ of x'. We call the prefix and postfix $0^n 1^n$ the left and right block, respectively, of x_i.

Lemma 12 *For every string x fulfilling $h(x, L) < \infty$, we have $h(x, L) = h(x', L') = d(x', L')$.*

PROOF. Obviously, we have $h(x, L) = h(x', L')$ and $h(x', L') \geq d(x', L')$. Thus, it remains to show that $h(x', L') \leq d(x', L')$.

Let $|x| = n$. If $L \cap \{0,1\}^n = \emptyset$, we have $h(x, L) = \infty$. Thus, we assume that L contains at least one string of length n. Let $y' \in L'$ be a string with minimum edit distance to x'. Then

$$y' = 0^{n'} 1^{n'} y_1 0^{n'} 1^{n'} \ldots 0^{n'} 1^{n'} y_{n'} 0^{n'} 1^{n'}$$

for some $n' \in \mathbb{N}$. If $n' \neq n$, then the difference of $|x'|$ and $|y'|$ is more than n and therefore $d(x', y') > n$. Thus, we can assume that $n' = n$. Consider now an optimal alignment $(\tilde{x}', \tilde{y}')$ of (x', y'). We have $h(\tilde{x}', \tilde{y}') \leq n$. Thus, we can assume that in the alignment considered, x_i is at most n positions away from y_i, because otherwise too many 0's or 1's will match a gap.

Consider any pair x_i and y_i that do not match. We know that x_i is at most n positions away from y_i. Then either there is a character in the left or right block of x_i or y_i matching a gap in the other sequence or at least one 1 of the left or right block of x_i matches a 0 in the left or right block of y_i. In either case, we have costs of at least one, which we charge to x_i and y_i.

In this way, we have charged at least cost one to every x_i and y_i that do not match. Thus, we can realign $\tilde{x}'$ and $\tilde{y}'$ to obtain an alignment $(\tilde{x}'', \tilde{y}'')$ without gaps with less or equal score, i.e. $h(\tilde{x}'', \tilde{y}'') \leq h(\tilde{x}', \tilde{y}')$. Thus, we have

$$h(x', y') = h(\tilde{x}'', \tilde{y}'') \leq h(\tilde{x}', \tilde{y}') = d(x', y')$$

and therefore $h(x', y') \leq d(x', y')$, which completes the proof. □

Theorem 13 *Let L be a language such that $\mathsf{OPT}_{\mathsf{H}}(L)$ cannot be approximated with a factor $f(n)$ for strings of length n. Then $\mathsf{OPT}_{\mathsf{E}}(L')$ cannot be approximated with factor $f(n)$ for strings of length $4 \cdot n^2 + n$.*

PROOF. Due to Lemma 12, any algorithm that computes a factor $f(n)$ approximation for $\mathsf{OPT}_{\mathsf{E}}(L')$ for strings of length $4 \cdot n^2 + n$ can be used for approximating $\mathsf{OPT}_{\mathsf{H}}(L)$ for strings of length n. □

An immediate consequence of the reduction presented above is the following corollary.

Corollary 14 *There is a language $L \in \mathsf{P}$ such that L_{E} is $\mathsf{W}[\mathsf{P}]$-hard.*

5.2 Reduction from edit distance to hamming distance

Let $L \subseteq \{0,1\}^\star$ be a language for which we want to compute the edit distance. We construct another language L' as follows:

$$L' = \{y \mid \exists x \in L : y \text{ is obtained from } x \text{ by inserting gaps}\} .$$

For a string x of length n we define $x' = \Delta^n x_1 \Delta^n \dots \Delta^n x_n \Delta^n$.

Lemma 15 *For every $x \in \{0,1\}^\star$ we have $d(x, L) = h(x', L')$.*

PROOF. We start with $d(x, L) \geq h(x', L')$. Let $y \in L$ be a string with $d(x, y) = d(x, L)$. Let $(\tilde{x}, \tilde{y})$ be an optimal alignment of x and y. We can assume that to the left of x_1, between x_i and x_{i+1} (for $1 \leq i \leq n-1$), and to the right of x_n there are always at most n gap symbols in $\tilde{x}$. Thus, in $\tilde{x}$ we can insert gaps to obtain x' as defined above and in the same places in $\tilde{y}$ to obtain some $\hat{y}$. Clearly, $d(x, L) = h(x', \hat{y}) \geq h(x', L')$. It remains to show $d(x, L) \leq h(x', L')$. Assume that we have a $y' \in L'$ with $h(x', y') = h(x', L')$. Then (x', y') is an alignment of (x, y), where y is obtained from y' by deleting all gaps. Thus, $d(x, L) \leq d(x, y) \leq h(x', y') = h(x', L')$. □

Theorem 16 *Let L be a language such that $\mathsf{OPT}_{\mathsf{E}}(L)$ cannot be approximated with a factor $f(n)$ for strings of length n. Then $\mathsf{OPT}_{\mathsf{H}}(L')$ cannot be approximated with factor $f(n)$ for strings of length $n^2 + 2 \cdot n$.*

PROOF. Due to Lemma 15, any algorithm that computes a factor $f(n)$ approximation for $\mathsf{OPT}_{\mathsf{H}}(L')$ for strings of length $n^2 + 2 \cdot n$ can be used for approximating $\mathsf{OPT}_{\mathsf{E}}(L)$ for strings of length n. □

We can extend the above results to languages over alphabets of size two using a homomorphism g mapping 0, 1, and Δ to 001, 010, and 100, respectively. Then we have $2 \cdot h(x', L') = h(g(x'), g(L'))$. Thus, if the Hamming distance to $g(L')$ cannot be approximated with a factor $f(n)$ for strings of length $3n$, then the Hamming distance to L' cannot be approximated with a factor $f(n)$ for strings of length n. Unfortunately, it might happen that $g(L') \notin \mathsf{AC}^0$ for some $L \in \mathsf{AC}^0$. Consider for example $L = \{x \mid x \in \{0,1\}^\star \text{ and } |x| \text{ is even}\}$. Then

L' and also $g(L')$ are essentially parity, which is known to be not in AC^0 [15]. Thus, there are languages $L \in \mathsf{AC}^0$ such that $g(L') \notin \mathsf{AC}^0$.

From the reduction presented we immediately obtain the following corollary as a counterpart of Corollary 14.

Corollary 17 $\mathsf{P_E} \subseteq \mathsf{W[P]}$.

PROOF. If $L \in \mathsf{P}$, then $L' \in \mathsf{P}$ and, by Theorem 6, $L'_{\mathsf{H}} \in \mathsf{W[P]}$. Since we have reduced L_{E} to L'_{H}, we have $L_{\mathsf{E}} \in \mathsf{W[P]}$. □

6 Towards a characterization of HamP

We define $\mathsf{HamP} = \{L \mid d(\cdot, L) \in \mathsf{FP}\}$ to be the class of languages to which the Hamming distance can efficiently be computed. Clearly, we have $\mathsf{1NAuxPDA^p} \subseteq \mathsf{HamP}$ due to Pighizzini's results [31] and $\mathsf{HamP} \subseteq \mathsf{P}$, since computing the Hamming distance is at least as hard as deciding membership.

Analogously to P, the class of languages for which membership is efficiently decidable, HamP is the class of languages to which the Hamming distance can efficiently be computed. We are not yet able to satisfactorily characterize the class HamP. But we are able to prove some basic properties of the class.

Theorem 18 HamP *is closed under union, Kleene closure, and concatenation.* HamP *is not closed under complementation and intersection, unless* $\mathsf{P} = \mathsf{NP}$.

PROOF. Let $L, L' \in \mathsf{HamP}$. We have $h(x, L \cup L') = \min\{h(x, L), h(x, L')\}$, thus $L \cup L' \in \mathsf{HamP}$. Furthermore, $\tilde{L} = \{yz \mid y \in L \wedge z \in L'\} \in \mathsf{HamP}$, since $h(x, \tilde{L}) = \min_{yz=x} h(y, L) + h(z, L')$. Let $L^\star = \{\epsilon\} \cup \{xy \mid x \in L \wedge y \in L^\star\}$ be the Kleene closure of L. Then $d(x, L^\star) = \min_{yz=x, y \neq \epsilon} d(x, L) + d(y, L^\star)$, which can efficiently be computed using dynamic-programming given that $d(\cdot, L)$ can efficiently be computed.

Due to Lemma 19, HamP is not closed under intersection, unless $\mathsf{P} = \mathsf{NP}$. Since HamP is closed under union, it cannot be closed under complementation, unless $\mathsf{P} = \mathsf{NP}$. □

Lemma 19 HamP *is not closed under intersection, unless* $\mathsf{P} = \mathsf{NP}$.

PROOF. The proof is very similar to Pighizzini's proof that there is a language in $\mathsf{co\text{-}NTime}(\log)$ to which Hamming distance computation is NP-hard. Let L_{NP} be any NP-complete language over some alphabet Σ and M be a polynomial time-bounded nondeterministic Turing machine with $L(M) = L_{\mathsf{NP}}$. Let p be the time-bound (and space-bound) of M.

We construct a language L that happens to be the intersection of two languages L_1 and L_2 in $\mathsf{1NAuxPDA^p}$. Let

$$L \subseteq \bigcup_{x\in\Sigma^\star} \{x\#w_0\#w_1\#\ldots\#w_p(|x|) \mid \forall i \in \{1,\ldots,p(|x|)\} : |w_i| = p(|x|)\}$$

be such that a string $x\#w_0\#w_1\#\ldots\#w_p(|x|)$ is in L if and only if the following conditions hold:

(1) If i is even, then w_i encodes a configuration c_i if M. If i is odd, then w_i^{R} (which denotes w_i read backwards) encodes a configuration c_i.
(2) c_0 is the initial configuration of M on input x.
(3) For all $i \in \{1, 2, \ldots, p(|x|) - 1\}$, c_{i+1} is one of M's possible successor configuration of c_i.
(4) $c_{p(|x|)}$ is the (unique) accepting configuration of M. (We can w.l.o.g. assume that M accepts with its tape empty and its head at the left-most positions.)

Given a string x for which we want to decide whether $x \in L_{\mathsf{NP}}$, we generate a string

$$y = x\,\underbrace{\#\beta^{p(|x|)}\#\beta^{p(|x|)}\#\ldots\#\beta^{p(|x|)}}_{p(|x|)\text{ times}}\,,$$

where β denotes some blank symbol that is not part of M's alphabet. Then $x \in L_{\mathsf{NP}}$ if and only if $h(y, L) \leq p(|x|)^2$. Thus, $L \notin \mathsf{HamP}$, unless $\mathsf{P} = \mathsf{NP}$.

Let L_{odd} be defined similarly to L, except that we demand Item (3) only for odd i. Analogously, L_{even} is defined like L, except that we demand Item (3) only for even i. Thus $L_{\text{odd}} \cap L_{\text{even}} = L$.

It remains to prove that both L_{odd} and L_{even} are in $\mathsf{1NAuxPDA^p}$ and thus in HamP. We restrict ourselves to proving $L_{\text{odd}} \in \mathsf{1NAuxPDA^p}$; L_{even} follows immediately. Assume that the head of the input tape is currently at the beginning of w_i for some odd i. Then we put w_i into the pushdown store while verifying that $|w_i| = p(|x|)$. When we are at the beginning of w_{i+1} we can step-by-step read w_i from the pushdown store and verify, whether c_{i+1} is a successor configuration of c_i. This can be done, since either w_i or w_{i+1} is reversed in the input string. $\square$

An immediate consequence of the lemma above is that if we allow a polynomial time and logarithmic space bounded Turing machine with an auxiliary pushdown store to scan the input twice (in contrast to one-way machines that can do this only once), then already such a Turing machine is able to accept a language to which computing the Hamming distance is NP-hard, since the language constructed in the proof is also contained in that class. We call the class of languages accepted by such machines $\mathsf{2NAuxPDA^p}$.

Up to now, one might suspect that $\mathsf{1NAuxPDA^p} = \mathsf{HamP}$. Unfortunately, this is not the case: The language

$$\mathsf{COPY} = \{ww \mid w \in \{0,1\}^\star\}$$

is not contained in $\mathsf{1NAuxPDA^p}$ [7]. Nevertheless, COPY is in HamP, since the Hamming distance of a string to COPY is simply the edit distance between its first and second half.

But things are worse. Consider the following variant of the circuit value problem of some alphabet Σ:

$$\mathsf{CVP}_1 = \Sigma^\star \setminus \{C\#x \mid C(x) = 1\}\ .$$

This variant is P-complete as well: It is simply CVP and additionally contains all syntactically incorrect inputs, i.e. inputs where either the x does not fit the C or the C does not encode a valid circuit. But the Hamming distance to CVP_1 can efficiently be computed: Either they are 0, i.e. the input string is actually contained in CVP_1, or it is 1, since then we can modify the input string to get a syntactically incorrect circuit, which is contained in CVP_1. Thus computing the Hamming distance to CVP_1 is essentially deciding membership.

Using brute-force enumeration, we can immediately generalize the observation to languages, where the maximum Hamming distance is bounded.

Proposition 20 *Let $L \in \mathsf{P}$ such that for all $x \in \Sigma^\star$, we have $h(x, L) \in O(1)$. Then $L \in \mathsf{HamP}$.*

Let us consider now another machine model. What happens if we replace the auxiliary stack of our Turing machine with a queue? We call that class $\mathsf{1NAuxQDA^p}$. We have $\mathsf{1NAuxQDA^p} \not\subseteq \mathsf{HamP}$, unless $\mathsf{P} = \mathsf{NP}$: We modify the language L defined in the proof of Lemma 19, such that all w_i encode configurations read forward. Then we can check whether w_i encodes a successor configuration of c_{i-1} while simultaneously putting w_i into the queue to allow the comparison with w_{i+1}.

On the one hand, there are (very dense) P-complete languages, to which computing the Hamming distance is easy, i.e. possible in polynomial time. On the other hand, even in very small complexity classes like $\mathsf{co\text{-}NTime}(\log)$, 1NAuxQDA$^{\mathsf{p}}$, or 2NAuxPDA$^{\mathsf{p}}$, there are languages to which computing the Hamming distance is NP-hard.

7 Open problems

On the one hand, algorithms for approximating the Hamming or the edit distance are clearly of interest. On the other hand, we conjecture that significantly stronger lower bounds for the approximability of these problems hold.

The reduction from the problem of computing the Hamming distance to the one of computing the edit distance preserves the size of the alphabet. Furthermore, if the language to which we want to compute the Hamming distance is in AC^0, then so is the one constructed. In the reduction from the latter to the former, we used a third symbol (which could be avoided by an appropriate encoding), and the language constructed is not necessarily in AC^0, even if the original language is. Another question is whether there is a reduction avoiding this.

The most important open problem indeed is characterizing the class HamP in terms of automata, formal languages, or complexity theory. Although we have shown some properties of HamP, a complete characterization seems to be difficult.

References

[1] K. A. Abrahamson, R. G. Downey, M. R. Fellows, Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogs, Annals of Pure and Applied Logic 73 (3) (1995) 235–276.

[2] A. V. Aho, T. G. Peterson, A minimum distance error-correcting parser for context-free languages, SIAM Journal on Computing 1 (4) (1972) 305–312.

[3] E. Allender, D. Bruschi, G. Pighizzini, The complexity of computing maximal word functions, Computational Complexity 3 (1993) 368–391.

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties, Springer, 1999.

[5] Z. Bar-Yossef, T. S. Jayram, R. Krauthgamer, R. Kumar, Approximating edit distance efficiently, in: Proc. of the 45th Ann. IEEE Symp. on Foundations of Computer Science (FOCS), IEEE Computer Society, 2004, pp. 550–559.

[6] T. Batu, F. Ergün, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, R. Sami, A sublinear algorithm for weakly approximating edit distance, in: Proc. of the 35th Ann. ACM Symp. on Theory of Computing (STOC), ACM Press, 2003, pp. 316–324.

[7] F.-J. Brandenburg, On one-way auxiliary pushdown automata, in: H. Tzschach, H. Waldschmidt, H. K.-G. Walter (Eds.), Proc. of the 3rd GI-Conference on Theoretical Computer Science, Vol. 48 of Lecture Notes in Computer Science, Springer, 1977, pp. 132–144.

[8] L. Cai, J. Chen, On the amount of nondeterminism and the power of verifying, SIAM Journal on Computing 26 (3) (1997) 733–750.

[9] S. C. Chan, A. K. C. Wong, D. K. Y. Chiu, A survey of multiple sequence comparison methods, Bulletin of Mathematical Biology 54 (4) (1992) 563–598.

[10] S. A. Cook, Characterizations of pushdown machines in terms of time-bounded computers, Journal of the ACM 18 (1) (1971) 4–18.

[11] R. G. Downey, M. R. Fellows, Fixed-parameter tractability and completeness I: Basic results, SIAM Journal on Computing 24 (4) (1995) 873–921.

[12] R. G. Downey, M. R. Fellows, Fixed-parameter tractability and completeness II: On completeness for W[1], Theoretical Computer Science 141 (1–2) (1995) 109–131.

[13] R. G. Downey, M. R. Fellows, Parameterized Complexity, Monographs in Computer Science, Springer, 1999.

[14] R. Fagin, Generalized first-order spectra and polynomial-time recognizable sets, in: R. M. Karp (Ed.), Complexity of Computation, Vol. 7 of SIAM-AMS Proceedings, 1974, pp. 43–73.

[15] M. Furst, J. B. Saxe, M. Sipser, Parity, circuits, and the polynomial-time hierarchy, Mathematical Systems Theory 17 (1) (1984) 13–27.

[16] D. M. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997.

[17] M. M. Halldórsson, Approximating the minimum maximal independence number, Information Processing Letters 46 (4) (1993) 169–172.

[18] R. W. Hamming, Error detecting and error correcting codes, Bell System Technical Journal 29 (2) (1950) 147–160.

[19] L. A. Hemachandra, Algorithms from complexity theory: Polynomial-time operations for complex sets, in: T. Asano, T. Ibaraki, H. Imai, T. Nishizeki (Eds.), Proc. of the SIGAL Int. Symp. on Algorithms, Vol. 450 of Lecture Notes in Computer Science, Springer, 1990, pp. 221–231.

[20] D. S. Hirschberg, A linear space algorithm for computing maximal common subsequences, Communications of the ACM 18 (6) (1975) 341–343.

[21] N. Immerman, Descriptive Complexity, Graduate Texts in Computer Science, Springer, 1998.

[22] E. T. Irons, An error-correcting parse algorithm, Communications of the ACM 6 (11) (1963) 669–673.

[23] R. E. Ladner, The circuit value problem is log space complete for P, SIGACT News 7 (1) (1975) 18–20.

[24] V. I. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals., Soviet Physics Doklady 10 (8) (1966) 707–710.

[25] G. Lyon, Syntax-directed least-errors analysis for context-free languages: A practical approach, Communications of the ACM 17 (1) (1974) 3–14.

[26] B. Manthey, R. Reischuk, The intractability of computing the hamming distance, in: T. Ibaraki, N. Katoh, H. Ono (Eds.), Proc. of the 14th Ann. Int. Symp. on Algorithms and Computation (ISAAC), Vol. 2906 of Lecture Notes in Computer Science, Springer, 2003, pp. 88–97.

[27] W. J. Masek, M. S. Paterson, A faster algorithm computing string edit distances, Journal of Computer and System Sciences 20 (1) (1980) 18–31.

[28] G. Navarro, A guided tour to approximate string matching, ACM Computing Surveys 33 (1) (2001) 31–88.

[29] P. A. Pevzner, Computational Molecular Biology: An Algorithmic Approach, MIT Press, 2000.

[30] G. Pighizzini, A parallel minimum distance error-correcting context-free parser, in: A. Marchetti-Spaccamela, P. Mentrasti, M. V. Zilli (Eds.), Proc. of the 4th Italian Conference on Theoretical Computer Science, World Scientific, 1992, pp. 305–316.

[31] G. Pighizzini, How hard is computing the edit distance?, Information and Computation 165 (1) (2001) 1–13.

[32] D. A. Spielman, The complexity of error-correcting codes, in: B. S. Chlebus, L. Czaja (Eds.), Proc. of the 11th Int. Symp. on Fundamentals of Computation Theory (FCT), Vol. 1279 of Lecture Notes in Computer Science, Springer, 1997, pp. 67–84.

[33] I. H. Sudborough, On the tape complexity of deterministic context-free languages, Journal of the ACM 25 (3) (1978) 405–414.

[34] A. Vardy, Algorithmic complexity in coding theory and the minimum distance problem, in: Proc. of the 29th Ann. ACM Symp. on Theory of Computing (STOC), ACM Press, 1997, pp. 92–109.

[35] H. Vollmer, Introduction to Circuit Complexity — A Uniform Approach, Texts in Theoretical Computer Science. An EATCS Series., Springer, 1999.