The Intractability of Computing the Hamming Distance

Bodo Manthey* and Rüdiger Reischuk

Universität zu Lübeck, Institut für Theoretische Informatik
Wallstraße 40, 23560 Lübeck, Germany
manthey/reischuk@tcs.uni-luebeck.de

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 a 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.
Finally, we show that the problems of computing the Hamming distance and of computing the edit distance are in some sense equivalent by presenting reductions from the former to the latter and vice versa.

1 Introduction

Given a language L and a string x, one can ask whether there is a string in L in the "neighbourhood" of x and how to find such a string. On the other hand, one can ask for the minimum distance of any string in L to x. 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 [15]). 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 [1, 8, 11]. Furthermore, the problem of computing distances between strings has gained popularity in computational biology [5, 9, 13]. 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 [7].

Computing the Hamming distance of two strings is easy. Computing the edit distance of two strings can be done via dynamic programming. Pighizzini [14] presented a language in $\mathsf{co\text{-}NTime}(\log)$ (a subclass of AC^0) for which computing

* Supported by DFG research grant Re 672/3.

the edit distance is NP-hard. On the other hand, he showed that computing the edit distance to languages in $\mathsf{1NAuxPDA}^{\mathsf{p}}$ can be done in polynomial time and even in AC^1. $\mathsf{1NAuxPDA}^{\mathsf{p}}$ denotes the class of all languages that can be recognized by logarithmic space and polynomial time bounded nondeterministic Turing machines equipped a one-way input tape and an auxiliary pushdown store.

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.

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 a factor $O(n^{\frac{1}{3}-\epsilon})$ unless $\mathsf{P} = \mathsf{NP}$. 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 prove that for every $t \in \mathbb{N}$ there is a language in AC^0 for which this is $\mathsf{W}[t]$-hard. Moreover, we present a language in P for which this is $\mathsf{W}[\mathsf{P}]$-hard. Thus, it turns out that computing the Hamming distance is hard even for languages in small complexity classes. Finally, we reduce the problem of computing the Hamming distance to the problem of computing the edit distance and vice versa. Hence, both problems are in some sense equivalent with respect to their approximability.

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. 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 and define $\Sigma' = \Sigma \cup \{\Delta\}$. An *alignment* of two strings x and y over Σ is a pair of strings $\tilde{x}$ and $\tilde{y}$ over Σ' 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. 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 [5] or Navarro [12] 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 have unit costs.

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. Then $\mathsf{OPT}_\mathsf{H}(L)$ is 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 measure is the Hamming distance between x and y, i.e. $h(x, y)$.*
4. *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.

Definition 2 (Hamming/Edit Closure). *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. Next, we define classes of Hamming and edit closures corresponding to classical complexity classes.

Definition 3 (Complexity 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 $\mathsf{C}_\mathsf{E} = \{L_\mathsf{E} \mid L \in \mathsf{C}\}$.*

The paper is organized as follows. In the next section we prove that the Hamming distance is hard to approximate. In Section 4 we focus our attention on Hamming closures. We show the intractability of Hamming closures in the sense of parameterized complexity. In Section 5 we present reductions from the problem of computing the Hamming distance to the one of computing the edit distance and vice versa. Finally, we raise some open problems in Section 6.

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 a factor $O(n^{\frac{1}{3}-\epsilon})$, for any $\epsilon > 0$, for strings of length n unless $\mathsf{P} = \mathsf{NP}$.

We consider 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 $\tilde{V}$. The problem MIDS is also known as *Minimum Maximal Independent Set*, since an independent dominating set is an independent set that cannot be extended. Halldórsson [6] showed that MIDS cannot be approximated with a factor $O(|V|^{1-\epsilon})$, for any $\epsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$.

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

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

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

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

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

We have DOM $= 1$ iff $\tilde{V}$ is a dominating set and IND $= 1$ iff $\tilde{V}$ is an independent set of G. Furthermore, EQU $= 1$ iff the matrices $(e^\ell_{i,j})_{1 \leq i < j \leq m}$ encode the same graph G for any $1 \leq \ell \leq m+1$. Hence, OUTPUT $= 1$ iff the input is in L^{MIDS}. The circuit implementing the above formulas has constant depth and the circuit family is logarithmic space uniform. Thus, $L^{\mathsf{MIDS}} \in \mathsf{AC}^0$. Let $n = \frac{m^3+3m+2}{2}$ be the length of an input string encoding a graph with m vertices.

Theorem 1. *For any $\epsilon > 0$, $\mathsf{OPT}_\mathsf{H}(L^{\mathsf{MIDS}})$ cannot be approximated in polynomial time with a 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 $\mathsf{OPT}_\mathsf{H}(L^{\mathsf{MIDS}})$ 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 $\mathsf{OPT}_\mathsf{H}(L^{\mathsf{MIDS}})$ would yield an $O(m^{1-\epsilon})$ approximation for MIDS. From $m \in \Theta(n^{\frac{1}{3}})$, the theorem follows. □

Theorem 2. *For any $\epsilon > 0$, $\mathsf{OPT}_{\mathsf{E}}(L^{\mathsf{MIDS}})$ cannot be approximated in polynomial time with a factor $O(n^{\frac{1}{3}-\epsilon})$ for strings of length n unless $\mathsf{P} = \mathsf{NP}$.*

Proof. Consider the input string x created in the proof of Theorem 1. Any string $y \in L^{\mathsf{MIDS}}$ with $|y| \neq |x|$ fulfils $d(x, y) > m$. Thus, any string in L^{MIDS} with minimum edit distance to x has length n. A change within the graph encoding again causes at least a difference of $m + 1$. Hence, if $y \in L^{\mathsf{MIDS}}$ has minimum edit distance to x, then x and y differ only within the encoding of $\tilde{V}$. Let z_y be the encoding of $\tilde{V}$ in y. The edit distance of x and y is at least the number of 1's in z_y — the theorem is proved. □

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.

4 Parameterized Intractability of Hamming Closures

4.1 Parameterized Intractability of 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 W[P]. On the other hand, there is a language in P the Hamming closure of which is W[P]-hard.

Downey and Fellows [2] defined W[P] to be the class of parameterized languages that can be reduced to EW-Circ-SAT. (Here, EW stands for *Exactly Weighted*. Downey and Fellows called the problem *Weighted Circuit Satisfiability*.)

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

The weight of an assignment is the number of variables to which the value 1 has been assigned. We also consider the following variant of weighted circuit satisfiability:

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

Lemma 1. W-Circ-SAT *is* W[P]*-complete.*

Proof. First, we reduce W-Circ-SAT to EW-Circ-SAT to prove W-Circ-SAT $\in$ W[P]. Let (C, k) be an instance for W-Circ-SAT. Assume that C has n input bits. We add k new input bits $z_1, \ldots, z_k$ and construct the circuit C' as $C \wedge \bigvee_{i=1}^{k}(z_i \vee \overline{z_i})$. If C has a satisfying assignment with weight $k' \leq k$, then C' will be satisfied by the same assignment together with $z_1 = \ldots = z_{k'} = 0$ and $z_{k'+1} = \ldots = z_k = 1$. The assignment obtained has weight k and hence $(C', k) \in$ EW-Circ-SAT. On the other hand, any satisfying assignment for C' with weight k yields a satisfying assignment for C with weight at most k.

Second, we reduce EW-Circ-SAT to W-Circ-SAT to prove the W[P]-hardness of W-Circ-SAT. Let (C, k) be an instance for EW-Circ-SAT. We construct another circuit $C_{n,k}$ that on input x outputs 1 if the number of ones in x is exactly k. The circuit $C_{n,k}$ has size polynomial in n. Then $(C, k) \in$ EW-Circ-SAT iff $(C \wedge C_{n,k}, k) \in$ W-Circ-SAT — we have reduced EW-Circ-SAT to W-Circ-SAT. □

Theorem 3. $\mathsf{P}_\mathsf{H} \subseteq \mathsf{W}[\mathsf{P}]$.

Proof. Consider an arbitrary language $L \in \mathsf{P}$. We reduce L_H to W-Circ-SAT to show that $L_\mathsf{H} \in \mathsf{W}[\mathsf{P}]$. 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 $g(\alpha_i) = 0^{i-1}10^{\sigma-i}$. We consider the language $g(L) = \{g(x) \mid x \in L\}$. Clearly, $g(L) \in \mathsf{P}$. Furthermore, we have $(x, k) \in L_\mathsf{H}$ iff $(g(x), 2k) \in g(L)_\mathsf{H}$. Since $g(L) \in \mathsf{P}$, there is a logarithmic space uniform circuit family of polynomial size for deciding $g(L)$ (see e.g. Greenlaw et al. [4]). 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 slightly 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 replace the ith input bit by itself followed by a NOT gate. Now C_n accepts y iff $C_{n,y}$ accepts 0^n. Furthermore, C_n accepts a string $\hat{y}$ iff $C_{n,y}$ accepts z with $z_i = y_i \oplus \hat{y}_i$, i.e. $C_{n,y}(z) = 1$ iff $C_n(\hat{y}) = 1$.

To summarize the above deliberations, we have $(x, k) \in L_\mathsf{H}$ iff $(g(x), 2k) \in g(L)_\mathsf{H}$ iff $(C_{|x|\cdot\sigma, g(x)}, 2k) \in$ W-Circ-SAT. Thus, the theorem is proved. □

Now we prove that there is a language in P the Hamming closure of which is W[P]-hard. Therefore, we consider the circuit value problem:

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

Ladner [10] 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 4. CVP'_H *is* W[P]*-hard.*

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

4.2 Parameterized Intractability of AC^0_H

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

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

W-t-SAT is W[t]-complete for all $t \geq 2$ while W-1-SAT is fixed parameter tractable [2]. 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 $(1 \leq \ell \leq t-1)$ are labelled with AND, if ℓ is odd, and with OR, if ℓ is even. Every vertex $v \in V_t$ is labelled 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 we have one more AND gate)

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

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

$$F = \bigwedge_{i_1=1}^{m} \bigvee_{i_2=1}^{m} \bigwedge_{i_3=1}^{m} \cdots \bigvee_{i_t=1}^{m} \Big(\big(\bigwedge_{\ell=1}^{t-1} e^\ell_{i_{\ell+1}, i_\ell} \big) \rightarrow \mathrm{lit}(v_t) \Big) \,.$$

Similar to the reduction presented in Section 4.1, we can create $m+1$ copies of each of these matrices. Thus, each t-normalized formula of size m can be evaluated by a circuit of depth $t+O(1)$ and polynomial size with $t \cdot m^2 \cdot (m+1)+m$ input variables. (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 (which is logarithmic space uniform) characterizes the language

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

Theorem 5. *For every $t \geq 2$, t-VAL$_\mathsf{H}$ is 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 with $t \cdot m^2 \cdot (m+1)+m$ input bits. The input X for the circuit is as follows. The first $t \cdot m^2 \cdot (m+1)$ bits encode the formula F. The last m bits are set to 0. Assume that $(F, k) \in$ 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$_\mathsf{H}$. □

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

5 Edit Distance versus Hamming Distance

5.1 Reduction from Computing the Hamming Distance to Computing the Edit Distance

Let L be a language to which we want to compute the Hamming distance. For every $x \in \{0,1\}^n$, let $x' = 0^n1^nx_11^n0^n0^n1^nx_21^n0^n \dots 0^n1^nx_n1^n0^n$. We construct a language L' as

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

Thus, every string x of length n has a counterpart x' of length $(4 \cdot n + 1) \cdot n$. Consider the substring $0^n1^nx_i1^n0^n$ of x'. We call the substrings 0^n1^n and 1^n0^n the left and right block, respectively, of x_i.

Lemma 2. *For every string x with $h(x, L) < \infty$, 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 $|x| = n$. We can assume $L \cap \{0,1\}^n \neq \emptyset$, since $h(x, L) = \infty$ otherwise. Let $y' \in L'$ be a string with minimum edit distance to x'. Then $y' = 0^{n'}1^{n'}y_11^{n'}0^{n'} \dots 0^{n'}1^{n'}y_{n'}1^{n'}0^{n'}$ 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. Assume that x_i and y_i do not match, but x_i is at most n position away from y_i. Then either parts of the left block of x_i match parts of the right block of y_i or parts of the right block of x_i match parts of the left block of y_i. Thus, either there are a lot of 0's matching 1's in the other string or there are a lot of gaps both in $\tilde{x}'$ and $\tilde{y}'$. Due to the structure of x' and y', we can modify $\tilde{x}'$ and $\tilde{y}'$ to obtain an alignment with less or equal score. This way, we iteratively obtain a new alignment $(\tilde{x}'', \tilde{y}'')$ that contains no gaps. Since $\tilde{x}', \tilde{y}'$ is an optimal alignment we have $h(x', L') = h(\tilde{x}'', \tilde{y}'') = h(\tilde{x}', \tilde{y}') = d(x', y') = d(x', L')$. □

Theorem 6. *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 a factor $f(n)$ for strings of length $4 \cdot n^2 + n$.*

Proof. Due to Lemma 2, any algorithm that computes an $f(n)$-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 in this section is the following corollary.

Corollary 1. *There is a language $L \in \mathsf{P}$ such that L_E is $\mathsf{W}[\mathsf{P}]$-hard.* □

5.2 Reduction from Computing the Edit Distance to Computing the 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 3. *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 that $d(x,L) \leq h(x',L')$. 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 gap symbols. Thus, we have $d(x,L) \leq d(x,y) \leq h(x',y') = h(x',L')$. □

Theorem 7. *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 a factor $f(n)$ for strings of length $n^2 + 2 \cdot n$.*

Proof. Due to Lemma 3, any algorithm that computes an $f(n)$-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 [3]. 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.

Corollary 2. $\mathsf{P}_\mathsf{E} \subseteq \mathsf{W}[\mathsf{P}]$.

Proof. If $L \in \mathsf{P}$, then $L' \in \mathsf{P}$ and, by Theorem 3, $L'_\mathsf{H} \in \mathsf{W}[\mathsf{P}]$. Since we have reduced L_E to L'_H, we have $L_\mathsf{E} \in \mathsf{W}[\mathsf{P}]$. □

6 Open Problems

An obvious open question is to find algorithms for approximating the Hamming or edit distance. On the other hand, we conjecture that significantly stronger lower bounds hold for the approximability of these problems.

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.

References

1. Alfred V. Aho and Thomas G. Petersen. A minimum distance error-correcting parser for context-free languages. *SIAM Journal on Computing*, 1(4):305–312, 1972.
2. Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.
3. Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.
4. Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, 1995.
5. Daniel M. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
6. Magnús M. Halldórsson. Approximating the minimum maximal independence number. *Information Processing Letters*, 46(4):169–172, 1993.
7. Lane A. Hemachandra. Algorithms from complexity theory: Polynomial-time operations for complex sets. In *Proc. of the SIGAL Int. Symp. on Algorithms*, volume 450 of *Lecture Notes in Computer Science*, pages 221–231. Springer, 1990.
8. Edgar T. Irons. An error-correcting parse algorithm. *Communications of the ACM*, 6(11):669–673, 1963.
9. Richard M. Karp. Mapping the genome: Some combinatorial problems arising in molecular biology. In *Proc. of the 25th Ann. ACM Symp. on Theory of Computing (STOC)*, pages 278–285, 1993.
10. Richard E. Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):18–20, 1975.
11. Gordon Lyon. Syntax-directed least-errors analysis for context-free languages: A practical approach. *Communications of the ACM*, 17(1):3–14, 1974.
12. Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.
13. Pavel A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. MIT Press, 2000.
14. Giovanni Pighizzini. How hard is computing the edit distance? *Information and Computation*, 165(1):1–13, 2001.
15. Daniel A. Spielman. The complexity of error-correcting codes. In *Proc. of the 11th Int. Symp. on Fundamentals of Computation Theory (FCT)*, volume 1279 of *Lecture Notes in Computer Science*, pages 67–84. Springer, 1997.