SIIM-TR-A-**02-17**

SIIM Technical Report

The Intractability of Computing the Hamming Distance

by

Bodo Manthey and Rüdiger Reischuk

Schriftenreihe der Institute für

Informatik/Mathematik

Serie A

October 31, 2002

Universität zu Lübeck
Technisch-Naturwissenschaftliche Fakultät

Email: manthey@tcs.mu-luebeck.de

Phone: +49-451-7030-414
Fax: +49-451-7030-438

The Intractability of Computing the Hamming Distance

Bodo Manthey* and Rüdiger Reischuk

October 31, 2002

Abstract

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

First, we prove that the Hamming distance to languages in AC^0 is hard to approximate; it cannot be approximated within a factor $O\left(n^{\frac{1}{3}-\epsilon}\right)$, for any $\epsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$ (n denotes the length of the input string).

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

Furthermore, 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 problem and vice versa.

1 Introduction

Given a language L and a certain string x, one can ask whether there is a string in L in the "neighbourhood" of x and to find such a string. On the other hand, one can ask for the minimum distance of any string in L to x. These problems arise for example in the area of self-correcting codes. To measure the distance one typically uses the Hamming distance or the edit distance.

Computing the Hamming distance of two strings is easy. Computing the edit distance of two strings can be done by dynamic programming (see e.g. Gusfield [4]). Pighizzini [11] has presented a language in $\mathsf{co\text{-}NTime}(\log)$, which is a subclass of AC^0, such that computing the edit distance of a string to this language is NP-hard. On the other hand, he has shown that computing the edit distance of strings to languages in $\mathsf{1NAuxPDA^p}$ can be done in polynomial time and even in AC^1. $\mathsf{1NAuxPDA^p}$ is the

*Birth name: Bodo Siebert. Supported by DFG research grant Re 672/3.

class of all languages that can be accepted by a logarithmic space and polynomial time bounded nondeterministic Turing machine with one-way input tape and equipped with an auxiliary pushdown store.

Intuitively, it seems that computing the edit distance is 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 parameterised complexity.

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

Furthermore, for a given language L we consider the parameterised 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 question is $\mathsf{W}[t]$-hard. Moreover, there is a language in P for which this question is $\mathsf{W}[\mathsf{P}]$-hard. Thus, it turns out that computing the Hamming distance to a language 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

See Downey and Fellows [2] for a survey on parameterised complexity and for a definition of the W-hierarchy, Wegener [12] for a survey about Boolean circuits, and Ausiello et al. [1] for a survey on approximation of optimisation problems.

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. there is no string of length $|x|$ in L, then we define $h(L, x) = \infty$.

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, only by inserting gap symbols.

We define the edit distance $d(x, y)$ of two strings x and y as

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

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. Note that, different from the Hamming distance, x and y do not have to be of equal length.

In general, we may allow a function that yields some penalty for each operation depending on the participating characters. See for example Gusfield [4], Pevzner [10], or Navarro [9] for a survey on computing edit distances (or alignments) 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 cost.

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 view the problem of computing the Hamming distance or the edit distance of a language and a string in two different ways, namely as optimisation problem and as parameterised language.

First, we consider the optimisation variant.

Definition 2.1 (Optimisation Problems) *Let $L \subseteq \{0,1\}^\star$ be a language. Then $\mathsf{OPT}_\mathsf{H}(L)$ is the following optimisation 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 just omit the length constraint, i.e. we allow all $y \in L$ as feasible solution, and use the edit distance as measure.

We say that $\mathsf{OPT}_\mathsf{H}(L)$ or $\mathsf{OPT}_\mathsf{E}(L)$ can be approximated within a factor f if there is a polynomial time algorithm that for all strings x computes a string $y \in L$ with $h(x, y) \leq f(|x|) \cdot h(x, L)$ or $d(x, y) \leq f(|x|) \cdot d(x, L)$, respectively.

Second, for a language L we consider the parameterised languages L_H and L_E.

Definition 2.2 (Hamming/Edit Closure of a Language) *Let $L \subseteq \{0,1\}^\star$ be a language. Then*

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

L_E is defined similarly. We simply replace h by d in the definition and omit the constraint $|x| = |y|$.

We call L_H and L_E the Hamming and edit closure of L, respectively.

Obviously, if $L \in \mathsf{NP}$, then L_{H} and L_{E} are also in NP.

Throughout this work, we consider Hamming and edit closures as parameterised languages with k as parameter.

With this definition, we can define classes of Hamming and edit closures corresponding to classical complexity classes.

Definition 2.3 (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 $\mathsf{C}_{\mathsf{H}} = \{L_{\mathsf{H}} \mid L \in \mathsf{C}\}$.*

Similarly, the class C_{H} of edit closures of languages in C is defined as $\mathsf{C}_{\mathsf{H}} = \{L_{\mathsf{E}} \mid L \in \mathsf{C}\}$.

The paper is organised 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 parameterised 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 conclude and 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, for arbitrary $\epsilon > 0$, the Hamming distance of L cannot be approximated within a factor $O\big(n^{\frac{1}{3}-\epsilon}\big)$ for strings of length n, unless $\mathsf{P} = \mathsf{NP}$.

For this purpose, consider the optimisation 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 $\{u, v\} \in E$ there is at most one of the vertices u and v in $\tilde{V}$. $\tilde{V}$ is a dominating set of G, if for every vertex $u \in V \setminus \tilde{V}$ there exists an edge $\{u, v\} \in E$ and $v \in \tilde{V}$. The goal is to minimise the size of $\tilde{V}$. The problem MIDS is also known as "Minimum Maximal Independent Set Problem", since an independent dominating set is an independent set that cannot be extended. It cannot be approximated within any fixed constant, even if G has to be bipartite [7]. Halldórsson [5] has shown that MIDS cannot be approximated within a factor $O\big(|V|^{1-\epsilon}\big)$, for any $\epsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$.

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

$$\begin{aligned} L^{\mathsf{MIDS}} = \Big\{ G_1 \# G_2 \# \ldots \# G_{m+1} \# \tilde{V} \;\Big|\; & G_i \in \{0,1\}^{\binom{m}{2}},\ \tilde{V} \in \{0,1\}^m \text{ for some } m \in \mathbb{N}, \\ & G_1 = \ldots = G_{m+1}, \text{each } G_i \text{ is an encoding of an} \\ & m\text{-vertex graph } G, \text{ and } \tilde{V} \text{ encodes an} \\ & \text{independent dominating set of } G. \Big\} \end{aligned}$$

We assume that G is encoded by $(m+1) \cdot \binom{m}{2}$ bits $e^{\ell}_{i,j}$. (For simplicity, $e^{\ell}_{i,j} = e^{\ell}_{j,i}$.) Here, $\left(e^{\ell}_{i,j}\right)_{1 \le i < j \le m}$ encodes the ℓth copy of G $(1 \le \ell \le m+1)$. 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:

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

One can see that DOM $= 1$ if and only if $\tilde{V}$ is a dominating set of G and IND $= 1$ if and only if $\tilde{V}$ is an independent set of G. Furthermore, EQUALITY $= 1$ if and only if the matrices $\left(e^{\ell}_{i,j}\right)_{1 \le i < j \le m}$ encode the same graph G for all $\ell \in \{1, 2, \ldots, m+1\}$. Hence, OUTPUT $= 1$ if and only if $\left(G, \tilde{V}\right) \in L^{\mathsf{MIDS}}$. Clearly, the circuit implementing the above formulas has constant depth and the circuit family is logarithmic space uniform. Hence, $L_{\mathsf{MIDS}} \in \mathsf{AC}^0$.

Let $n = \frac{m^3+m}{2}$ be the length of an input string encoding a graph with m vertices.

Theorem 3.1 *For any $\epsilon > 0$, $\mathsf{OPT_H}(L^{\mathsf{MIDS}})$ cannot be approximated in polynomial time within a factor $O\left(n^{\frac{1}{3}-\epsilon}\right)$ 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 $\left(\left(e^{\ell}_{i,j}\right)_{1 \le i < j \le m}\right)_{1 \le \ell \le m+1}$ and setting $z_i = 0$ for all $1 \le i \le m$.

Since every graph has an independent dominating set, we have $h\left(L^{\mathsf{MIDS}}, x\right) \le m$. Thus, there exists a string $y \in L^{\mathsf{MIDS}}$ with $|x| = |y|$ and $h(x, y) \le m$. 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.

Any approximation algorithm for $\mathsf{OPT_H}(L^{\mathsf{MIDS}})$ with approximation performance $O\left(m^{1-\epsilon}\right)$ yields an approximation algorithm for MIDS with approximation performance $O\left(m^{1-\epsilon}\right)$. Since $m \in \Theta\left(n^{\frac{1}{3}}\right)$, any approximation algorithm for $\mathsf{OPT_H}(L^{\mathsf{MIDS}})$ with approximation ratio $O\left(n^{\frac{1}{3}-\epsilon}\right)$ for some $\epsilon > 0$ yields an approximation algorithm for MIDS with approximation ratio $O\left(m^{1-\epsilon'}\right)$ for some $\epsilon' > 0$. This would imply $\mathsf{P} = \mathsf{NP}$ according to Halldórsson [5]. □

Our construction even shows that the Hamming distance of strings to a language L is hard to approximate even if L belongs to a very small complexity class.

Corollary 3.2 *There is a language $L \in \mathsf{AC}^0$ such that $\mathsf{OPT}_\mathsf{H}(L)$ cannot be approximated within a factor $O\big(n^{\frac{1}{3}-\epsilon}\big)$ for strings of length n, for every $\epsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$.*

4 Parameterised Intractability of Hamming Closures

In this section, we are concerned with Hamming closures. We present a language $L \in \mathsf{P}$ such that L_H is $\mathsf{W}[\mathsf{P}]$-hard. On the other hand, for every $L \in \mathsf{P}$ we have $L_\mathsf{H} \in \mathsf{W}[\mathsf{P}]$, thus $\mathsf{P}_\mathsf{H} \subseteq \mathsf{W}[\mathsf{P}]$.

Furthermore, we prove that for every $t \in \mathbb{N}$ there is a language in AC^0 such that its Hamming closure is $\mathsf{W}[t]$-hard.

4.1 Parameterised Intractability of P_H

The aim of this section is to analyse 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 the class $\mathsf{W}[\mathsf{P}]$. On the other hand, there is a language in P whose Hamming closure is $\mathsf{W}[\mathsf{P}]$-hard.

Downey and Fellows [2] have defined $\mathsf{W}[\mathsf{P}]$ to be the class of those parameterised language that can be reduced to $\mathsf{EW\text{-}Circ\text{-}SAT}$. (Here, EW stands for *Exact Weighted*. Downey and Fellows call the problem "Weighted Circuit Satisfiability".)

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

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 $\mathsf{EW\text{-}Circ\text{-}SAT}$:

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

Lemma 4.1 $\mathsf{W\text{-}Circ\text{-}SAT}$ *is* $\mathsf{W}[\mathsf{P}]$*-complete.*

Proof: First, we reduce $\mathsf{W\text{-}Circ\text{-}SAT}$ to $\mathsf{EW\text{-}Circ\text{-}SAT}$ to prove $\mathsf{W\text{-}Circ\text{-}SAT} \in \mathsf{W}[\mathsf{P}]$. Let (C,k) be an instance for $\mathsf{W\text{-}Circ\text{-}SAT}$. 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 and we may simply set $z_1 = \ldots = z_{k'} = 0$ and $z_{k'+1} = \ldots = z_k = 1$. The assignment obtained has weight k and hence $(C', k) \in \mathsf{EW\text{-}Circ\text{-}SAT}$. 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 prove that W-Circ-SAT is $\mathsf{W}[\mathsf{P}]$-hard by reduction from EW-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 if and only if $(C \wedge C_{n,k}, k) \in$ W-Circ-SAT. Thus, we have reduced EW-Circ-SAT to W-Circ-SAT. □

Theorem 4.2 $\mathsf{P}_\mathsf{H} \subseteq \mathsf{W}[\mathsf{P}]$.

Proof: Consider an arbitrary language $L \in \mathsf{P}$. We reduce L_H to W-Circ-SATto 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 $(x, k) \in L_\mathsf{H}$ 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 for example Greenlaw et al. [3]). Let C_n be the circuit in this family for strings of length n. Assume that we have an input string $x = x_1 \ldots x_n$. We modify C_n a little bit as follows to obtain a circuit $C_{n,x}$. If $x_i = 0$, then we leave the ith input bit unchanged. If $x_i = 1$, then we replace the ith input bit by itself followed by a NOT gate. One can see that C_n accepts x if and only if $C_{n,x}$ accepts 0^n. Furthermore, C_n accepts an input string y if and only if $C_{n,x}$ accepts z with $z_i = 1$ if x_i and y_i differ and $z_i = 0$ otherwise, i.e.

$$C_{n,x}(z) = 1 \iff C_n(x \oplus z) = 1 \, .$$

To summarise the above deliberations, we have

$$(x, k) \in L_\mathsf{H} \iff (g(x), 2k) \in g(L)_\mathsf{H} \iff (C_{n,x}, 2k) \in \text{W-Circ-SAT} \, .$$

Thus, the theorem is proved. □

To prove that there is a language in P such that its Hamming closure is $\mathsf{W}[\mathsf{P}]$-hard, we consider the circuit value problem:

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

Ladner [8] has shown that CVP is P-complete (see also Greenlaw et al. [3]). For our purposes, we consider the following variant of CVP:

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

Since CVP is P-complete, CVP' is P-complete as well.

Theorem 4.3 CVP'_H *is* $\mathsf{W}[\mathsf{P}]$*-hard.*

Proof: Let (C, k) be an instance for $\mathsf{W\text{-}Circ\text{-}SAT}$, where C has n input bits. Without loss of generality we can assume that $k \leq n$. Then $X = \big((C\#C\#\ldots\#C, 0^n), k\big)$ is an instance for CVP'_H with

$$(C, k) \in \mathsf{W\text{-}Circ\text{-}SAT} \iff X \in \mathsf{CVP}'_\mathsf{H} .$$

Hence, we have reduced $\mathsf{W\text{-}Circ\text{-}SAT}$ to CVP'_H. □

4.2 Parameterised Intractability of AC^0_H

A Boolean formula is called t-normalised, if it is of the form "AND-of-ORs-of-ANDs-of-... -of-Literals" with t alternations [2]. CNF formulas are, for example, 2-normalised.

Consider the parameterised language

$$\begin{aligned}\mathsf{W\text{-}}t\mathsf{\text{-}SAT} \;=\; \big\{(F, k) \mid\ & F \text{ is a } t\text{-normalised Boolean formula and has a}\\ & \text{satisfying assignment with at most } k \text{ ones}\big\} .\end{aligned}$$

$\mathsf{W\text{-}}t\mathsf{\text{-}SAT}$ is $\mathsf{W}[t]$-complete for all $t \geq 2$ and $\mathsf{W\text{-}1\text{-}SAT}$ is fixed parameter tractable [2].

Let us now encode a t-normalised 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 labeled with AND, if ℓ is odd, or 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 we have just one more OR 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. Furthermore, we can assume that $|V_t| \geq n$. Otherwise, there would be unused variables. Let $\mathrm{size}(F) = |V_t|$ and $m = \mathrm{size}(F)$, for short. We can encode each bipartite graph that is 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

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

Similar to the reduction presented in section 4.1, we can create $m+1$ copies of each of these matrices. Thus, all t-normalised formula of size m can evaluated by a circuit of depth $t + O(1)$ and polynomial size with $t \cdot m^2 \cdot (m+1) + m$ input variables. (Without loss of generality we assume that we have m input variables. Otherwise we add $m - n$ variables that are never used.) The circuit family obtained decides the language

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

Clearly, the circuit family is logarithmic space uniform.

Theorem 4.4 *For every $t \geq 2$, t-$\mathsf{VALUE}_\mathsf{H}$ is $\mathsf{W}[t]$-hard.*

Proof: Let (F, k) be an instance for W-t-SAT. Let $m = \mathrm{size}(F)$. Then 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 $(F, k) \in \mathsf{W}$-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 agree on the formula encoding part. Hence, they encode the same formula and Y yields a satisfying assignment for F with weight at most k. Thus, we have reduced W-t-SAT to t-$\mathsf{VALUE}_\mathsf{H}$. □

We immediately obtain the following corollary.

Corollary 4.5 *For every $t \in \mathbb{N}$ there is a language $L \in \mathsf{AC}^0$ such that L_H is $\mathsf{W}[t]$-hard. L can be decided by a circuit of depth $t + O(1)$.*

5 Edit Distance versus Hamming Distance

The aim of this section is to compare the complexity of computing the Hamming distance with the complexity of computing the edit distance. We reduce the former problem to the latter and vice versa. Thus, we show that both problems are in some sense equivalent with respect to their approximability.

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. We construct another language L' as follows:

$$L' = \{0^n 1^n x_1 1^n 0^n \; 0^n 1^n x_2 1^n 0^n \ldots 0^n 1^n x_n 1^n 0^n \mid x = x_1 x_2 \ldots x_n \in L\} \ .$$

For every string $x \in \{0,1\}^n$, we define $x' = 0^n 1^n x_1 1^n 0^n \; 0^n 1^n x_2 1^n 0^n \ldots 0^n 1^n x_n 1^n 0^n$. Thus, every string x of length n has a counterpart x' of length $\big(4 \cdot n + 1\big) \cdot n$. Consider the substring $0^n 1^n x_i 1^n 0^n$ of x'. We call the substrings $0^n 1^n$ and $1^n 0^n$ the right and left block, respectively, of x_i.

The following lemma is crucial for reducing the problem of computing the Hamming distance to the one of computing the edit distance.

Lemma 5.1 *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 $h(x', L') \leq d(x', L')$.

Assume that x has length n. Let us first assume that $h(x, L) \leq n$, i.e. $L \cap \{0,1\}^n \neq \emptyset$. Let $y' \in L'$ be the string in L' with minimum edit distance to x'. Then y' is of the form $0^{n'} 1^{n'} y_1 1^{n'} 0^{n'} \ldots 0^{n'} 1^{n'} y_{n'} 1^{n'} 0^{n'}$ 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' and 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 $h(x', L') = h(\tilde{x}'', \tilde{y}'') = h(\tilde{x}', \tilde{y}') = d(x', y') = d(x', L')$.

It remains to consider the case that $L \cap \{0,1\}^n = \emptyset$. But then $h(x, L) = \infty$ and $d(x', L') > n$. □

From the previous lemma we obtain the following theorem.

Theorem 5.2 *Let L be a language such that $\mathsf{OPT_H}(L)$ cannot be approximated within a factor $f(n)$ for strings of length n. Then $\mathsf{OPT_E}(L')$ cannot be approximated within a factor $f(n)$ for strings of length $4 \cdot n^2 + n$.*

Proof: Due to Lemma 5.1, any approximation algorithm that computes an $f(n)$-approximation for $\mathsf{OPT_E}(L')$ for strings of length $4 \cdot n^2 + n$ can be used as an approximation algorithm for $\mathsf{OPT_H}(L)$ for strings of length n. □

An immediate consequence of the previous theorem is the following corollary.

Corollary 5.3 *There exists a language $L \in \mathsf{AC}^0$ such that $\mathsf{OPT_E}(L)$ cannot be approximated within a factor $O\big(n^{\frac{1}{6}-\epsilon}\big)$ for strings of length n, for any $\epsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$.*

Proof: If $L \in \mathsf{AC}^0$, then $L' \in \mathsf{AC}^0$ as well. We choose $f(n) = n^{\frac{1}{3}-\epsilon}$ and apply Theorem 5.2. □

Furthermore, the reductions described yield intractability results, in the sense of parameterised complexity, for edit closures.

Corollary 5.4 *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 to which we want to compute the edit distance. We construct another language L' as follows:

$$L' = \{y \mid \exists x \in L \text{ such that } y \text{ is obtained from } x \text{ by inserting "}\Delta\text{"s}\} \ .$$

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

We show that computing the Hamming distance to L' is at least as hard as computing the Hamming distance to L.

Lemma 5.5 *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 $\tilde{y}$ 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 we have an $y' \in L'$ with $h(x', y') = h(x', L')$. Then x' and y' are an alignment of the strings x and y, where y is obtained from y' by deleting all gap symbols. Thus we have $d(x, L) \leq d(x, y) \leq h(x', y') = h(x', L')$. □

Using the previous lemma, we can prove the following theorem.

Theorem 5.6 *Let L be a language such that $\mathsf{OPT}_\mathsf{E}(L)$ cannot be approximated within a factor $f(n)$ for strings of length n. Then $\mathsf{OPT}_\mathsf{H}(L')$ cannot be approximated within a factor $f(n)$ for strings of length $n^2 + 2 \cdot n$.*

Proof: Due to Lemma 5.5, any approximation 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 as an approximation algorithm for $\mathsf{OPT}_\mathsf{E}(L)$ 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

$$2 \cdot h\left(x', L'\right) = h\left(g(x'), g(L')\right) \ .$$

Thus, if the Hamming distance to $g(L')$ cannot be approximated within a factor f, then the Hamming distance to $g(L')$ cannot be approximated within a factor $2 \cdot f$. 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 we can use $g(L')$ for computing the parity function, which is known to be not in AC^0 [6]. Thus, there are languages $L \in \mathsf{AC}^0$ such that for deciding $g(L')$ we need circuits with non-constant depth.

As a counterpart to Corollary 5.4 we obtain the following corollary from the reduction presented in this section.

Corollary 5.7 $\mathsf{P}_{\mathsf{E}} \subseteq \mathsf{W}[\mathsf{P}]$.

Proof: If $L \in \mathsf{P}$, then $L' \in \mathsf{P}$ and, by Theorem 4.2, $L'_{\mathsf{H}} \in \mathsf{W}[\mathsf{P}]$. Since we have reduced L_{E} to L'_{H}, we have $L_{\mathsf{E}} \in \mathsf{W}[\mathsf{P}]$. □

6 Conclusions

In this paper, we have shown the intractability of computing the Hamming distance of strings to a given language L, even in the case that L is restricted to be in a very small complexity class. We have shown the intractability of this problem both in sense of non-approximability and parameterised intractability. Furthermore, we have shown that approximating the Hamming distance and approximating the edit distance is in some sense equivalent by reducing the former to the latter and vice versa.

An obvious open question is to find nontrivial algorithms for approximating the Hamming or edit distance.

On the other hand, the lower bound for the approximability of computing the edit distance is weaker than the one for computing the Hamming distance. It is simply obtained by the reduction presented in Section 5. We conjecture that a significantly stronger lower bound holds for the approximability of computing the edit distance.

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 which we constructed. In the reduction from the latter to the former problem, we used a third symbol (which could be avoided by an appropriate encoding), and the language constructed is not in AC^0, even if the language to which we want to compute the edit distance is. Another question is whether there is a reduction avoiding this.

References

[1] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* Springer, 1999.

[2] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Springer, 1999.

[3] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press, 1995.

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

[5] Magnús M. Halldórsson. Approximating the minimum maximal independence number. *Inform. Process. Lett.*, 46(4):169–172, 1993.

[6] Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proc. of the 18th Ann. Symp. on Theory of Comput. (STOC)*, pages 6–20. ACM, 1986.

[7] Robert W. Irving. On approximating the minimum independent dominating set. *Inform. Process. Lett.*, 37(4):197–200, 1991.

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

[9] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[10] Pavel A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach.* MIT Press, 2000.

[11] Giovanni Pighizzini. How hard is computing the edit distance? *Inform. and Comput.*, 165(1):1–13, 2001.

[12] Ingo Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner, 1987.