Hard instances of algorithms and proof systems

YIJIA CHEN, Shanghai Jiaotong University, China
JÖRG FLUM, Universität Freiburg, Germany
MORITZ MÜLLER, Kurt Gödel Research Center, Universität Wien, Austria

If the class TAUT of tautologies of propositional logic has no almost optimal algorithm, then every algorithm $\mathbb{A}$ deciding TAUT has a hard sequence, i.e., a polynomial time computable sequence witnessing that $\mathbb{A}$ is not almost optimal. We show that this result extends to every Π_t^p-complete problem with $t \geq 1$; however, assuming the Measure Hypothesis, there is a problem which has no almost optimal algorithm but is decided by an algorithm without hard sequences. For problems Q with an almost optimal algorithm, we analyze whether every algorithm deciding Q, which is not almost optimal, has a hard sequence.

Categories and Subject Descriptors: F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes—*Complexity hierarchies*; *Relations among complexity classes*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Alternation and nondeterminism*

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: Hard Sequences, Optimal Algorithms, Optimal Proof Systems, Measure Hypothesis

ACM Reference Format:
ACM V, N, Article A (January YYYY), 22 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. Introduction

A SAT-solver is an algorithm that on satisfiable propositional formulas α as input yields a satisfying assignment and that does not stop on unsatisfiable formulas. By a result due to Levin [1973] (and presented as Theorem 6.4 in this paper) we know that there is an optimal SAT-solver, that is, a SAT-solver whose running time on satisfiable formulas is polynomially bounded in the running time of any SAT-solver.

In computational complexity, we are more often interested in *decision* algorithms which always halt and give yes or no answers. Let $\mathbb{A}$ be such an algorithm deciding SAT. Then, for every SAT-solver $\mathbb{S}$, we obtain an algorithm $\mathbb{S}^{\text{dec}}$ deciding SAT essentially by running $\mathbb{A}$ and $\mathbb{S}$ in parallel. On satisfiable formulas α the running time $t_{\mathbb{S}^{\text{dec}}}(\alpha)$ of $\mathbb{S}^{\text{dec}}$ on input α is $O(t_{\mathbb{S}}(\alpha))$ (for an algorithm $\mathbb{B}$ and a string x we denote by $t_{\mathbb{B}}(x)$ the number of steps the algorithm $\mathbb{B}$ takes on input x). For an optimal SAT-solver $\mathbb{O}$, what kind of optimality does $\mathbb{O}^{\text{dec}}$ inherit from $\mathbb{O}$? Schnorr [1976] and Verbitzky [1979] showed that $\mathbb{O}^{\text{dec}}$ is length-optimal, that is, for every algorithm $\mathbb{B}$ deciding SAT and all $\alpha \in \text{SAT}$,

$$t_{\mathbb{O}^{\text{dec}}}(\alpha) \leq \big(\max\{t_{\mathbb{B}}(\alpha') \mid \alpha' \in \text{SAT and } |\alpha'| \leq |\alpha|\} + |\alpha|\big)^{O(1)}.$$

However, the algorithm $\mathbb{O}^{\text{dec}}$ is not an almost optimal algorithm for SAT unless NP $\cap$ coNP = P (see [Chen and Flum 2014]). By definition an algorithm $\mathbb{A}$ is *almost optimal* for a problem Q if for

Yijia Chen, Department of Computer Science, Shanghai Jiaotong University, Dongchuan Road 800, 200240 Shanghai, China; Jörg Flum, Mathematisches Institut, Albert-Ludwigs-Universität Freiburg, Eckerstr. 1, 79104 Freiburg, Germany; Moritz Müller, Kurt Gödel Research Center for Mathematical Logic, Währinger Straße 25, 1090 Wien, Austria.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

any other algorithm $\mathbb{B}$ deciding Q and all $x \in Q$ (note that nothing is required for strings not in Q),

$$t_{\mathbb{A}}(x) \leq \big(t_{\mathbb{B}}(x) + |x|\big)^{O(1)}.$$

The concept of optimality just defined was first considered in [Krajíček and Pudlák 1989] for algorithms deciding the set TAUT of tautologies of propositional logic. It is not known whether TAUT has an almost optimal algorithm. Every problem Q in P (polynomial time) has an almost optimal algorithm. Indeed every polynomial-time bounded algorithm deciding Q is almost optimal. As shown in [Messner 2000] there are problems in E $\setminus$ P with almost optimal algorithms $\big($where E := DTIME$(2^{O(n)})\big)$.[1] To the best of our knowledge it is still not known whether there is a problem in NP $\setminus$ P having an almost optimal algorithm (even assuming P $\neq$ NP). In particular, this question is open for SAT.

Let $\mathbb{A}$ be an algorithm deciding a problem Q. Assume that $\mathbb{A}$ is not almost optimal and that $\mathbb{B}$ is an algorithm deciding Q that witnesses this nonoptimality of $\mathbb{A}$. Then, for every $s \in \mathbb{N}$ there is a string $x_s \in Q$ with $t_{\mathbb{A}}(x_s) > (|x_s| + t_{\mathbb{B}}(x_s))^s$. Can we generate such a sequence $(x_s)_{s\in\mathbb{N}}$ efficiently, that is, in time polynomially bounded in s? If so, then the algorithm $\mathbb{A}$ can be speeded up on those instances. We define the notion of a hard sequence for $\mathbb{A}$ without reference to a further algorithm deciding Q (as the algorithm $\mathbb{B}$ above): The sequence $(x_s)_{s\in\mathbb{N}}$ of strings in Q is *hard for* $\mathbb{A}$ if it is computable in time polynomially bounded in s but the sequence $\big(t_{\mathbb{A}}(x_s)\big)_{s\in\mathbb{N}}$ is not polynomially bounded in s.[2]

Clearly, if $\mathbb{A}$ is a polynomial-time bounded algorithm, then $\mathbb{A}$ has no hard sequences. Furthermore, an almost optimal algorithm for Q has no hard sequences either. In fact, if $(x_s)_{s\in\mathbb{N}}$ is a hard sequence for an algorithm, then one can superpolynomially speed it up on $\{x_s \mid s \in \mathbb{N}\}$, so it cannot be almost optimal (cf. Lemma 4.1). We say that the problem Q *has hard sequences for algorithms* if every algorithm deciding Q has a hard sequence.

Central to this paper is the question: To what extent can we show that algorithms which are not almost optimal have hard sequences? Our starting point is the following result (more or less explicit in [Krajíček and Pudlák 1989; Krajíček 1995; Monroe 2011; Chen and Flum 2010]):

> TAUT *has no almost optimal algorithm if and only if* TAUT *has hard sequences for algorithms.*

First, we generalize this result from the Π_1^p-complete problem TAUT to all problems which are Π_t^p-complete for some $t \geq 1$ $\big($cf. Theorem 4.3 (a)$\big)$:

(i) *A Π_t^p-complete problem Q has no almost optimal algorithm if and only if Q has hard sequences for algorithms.*

Apparently there are some limitations when trying to show the result for *all* problems Q as we prove (cf. Theorem 7.10):

(ii) *If the Measure Hypothesis holds, then there is a problem which has no almost optimal algorithm but is decided by an algorithm without hard sequences.*

Perhaps one would expect to be able to strengthen (i) by showing that even if a Π_t^p-complete problem Q has an almost optimal algorithm, then every algorithm, which is not almost optimal and decides Q, has a hard sequence. However (cf. Theorem 7.5):

---

[1] Here, as usual, given a class F of total functions from $\mathbb{N}$ to $\mathbb{N}$ we denote by DTIME(F) the class of problems decidable by an algorithm $\mathbb{A}$ with

$$t_{\mathbb{A}}(|x|) \leq O(f(|x|))$$

for some $f \in F$.

[2] Let us mention that for SAT-solvers a different, weaker notion of hard sequence has been considered (e.g., in [Gutfreund et al. 2007; Krajíček 2012]).

> *If the Measure Hypothesis holds, then* every *problem with padding and with an almost optimal algorithm is decided by an algorithm which is not almost optimal but has no hard sequences.*

In particular, if the Measure Hypothesis holds and TAUT has an almost optimal algorithm, then it is decided by an algorithm which is not almost optimal but has no hard sequences.

As an algorithm deciding a problem Q which is not almost optimal can be superpolynomially speeded up on an infinite subset of Q, by (ii) we see that, assuming the Measure Hypothesis, this notion of speeding up (e.g. considered in [Stockmeyer 1974]) is weaker than our notion of the existence of a hard sequence.

Assume that $Q :=$ TAUT (or any Π_t^p-complete Q) has no almost optimal algorithm; thus, by (i), every algorithm deciding Q has a hard sequence. Can we even effectively assign to every algorithm deciding Q a hard sequence? We believe that under reasonable complexity-theoretic assumptions one should be able to show that such an effective procedure or at least a polynomial time procedure does not exist, but we were not able to show it.[3]

By results of Stockmeyer [1974] and Berman [1976] and rediscovered by Messner [1999] we know:

> *For every* EXP*-hard (or, equivalently,* E*-hard) problem Q there is a polynomial-time bounded effective procedure assigning to every algorithm solving Q a hard sequence.*

Hence, if EXP = Π_t^p, then for every Π_t^p-hard problem Q there is a polynomial-time bounded effective procedure assigning a hard sequence to every algorithm deciding Q.

Our proof of (i) generalizes to nondeterministic algorithms $\big($cf. Theorem 4.3 (b)$\big)$. This "nondeterministic statement" yields a version for Π_t^p-complete problems of a result that Krajíček [1995, Theorem 14.2.2] derived for propositional proof systems: TAUT has no optimal proof system if and only if for every propositional proof system $\mathbb{P}$ there is a polynomial time computable sequence $(\alpha_s)_{s\in\mathbb{N}}$ of propositional tautologies α_s which only have superpolynomial $\mathbb{P}$-proofs; moreover, he showed that the α_s can be chosen with $s \leq |\alpha_s|$. While it is well-known that for any problem Q *nondeterministic* algorithms deciding Q and proof systems for Q are more or less the same, the relationship between *deterministic* algorithms and proof systems is more subtle. Nevertheless, we are able to use (i) to derive a statement on hard sequences for proof systems of Π_t^p-complete problems Q without a *polynomially* optimal proof system (cf. Theorem 6.7).

As a byproduct of results mentioned so far, we obtain results in "classical terms" (that is, not referring to hard sequences). For example, we get for $t \geq 1$ the following statements (cf. Corollary 4.9 and Theorem 5.5) previously only known for $t = 1$:

(iii) *If some Π_t^p-complete problem has no almost optimal algorithm, then every Π_t^p-hard problem has no almost optimal algorithm.*

(iv) *Let Q be Π_t^p-complete. Then, Q has a polynomially optimal proof system if and only if Q has an almost optimal algorithm.*

It is still open whether there exist problems outside of NP with optimal proof systems. Krajíček and Pudlák [1989] proved that E = NE implies that TAUT has an optimal proof system (see [Ben-David and Gringauze 1998; Köbler and Messner 1998; Köbler et al. 2003] for subsequent improvements of this result). We show their existence (in NE) assuming the Measure Hypothesis (cf. Theorem 7.4).

We discuss the relationship between our notions and results with previously known ones. Our focus is on hard sequences while related previous work concentrates on hard sets and on strongly hard sequences. For an algorithm $\mathbb{A}$ deciding a problem Q a set X of strings is *hard for* $\mathbb{A}$ if $X \subseteq Q$, $X \in$ P, and $\mathbb{A}$ is not polynomial-time bounded on X.

[3] As pointed out by one of the reviewers of our paper, in the meantime Krajíček [2014] obtained some conditional negative results concerning a related question for proof systems.

A hard sequence $(x_s)_{s\in\mathbb{N}}$ for $\mathbb{A}$ is *strongly hard* if $s \leq |x_s|$. Clearly, a strongly hard sequence $(x_s)_{s\in\mathbb{N}}$ yields the hard set $\{x_s \mid s \in \mathbb{N}\}$. Furthermore, one can show (Proposition 3.2):

(v) *Assume Q has padding. If Q has hard sequences for algorithms, then Q has strongly hard sequences (and hence, hard sets) for algorithms.*

We present a problem Q such that every algorithm that decides Q and is not almost optimal has a hard sequence and a hard set but none has a strongly hard sequence (see Example 4.2).

On the other hand, we prove (Theorem 7.10):

> *If the Measure Hypothesis holds, there is a problem which has hard sets for algorithms but has algorithms without hard sequences*

For Q with padding, by (v) we can replace the conclusion in (i), namely "Q has hard sequences for algorithms", by "Q has hard sets for algorithms." This version of (i) has been proven by Messner [2000, Theorem 3.4] for *all* paddable problems:

(vi) *If Q has padding, then Q has no almost optimal algorithm if and only if Q has hard sets for algorithms.*

(Messner assumes a property even weaker than padding.) Together with (i) and (v) this yields (Corollary 5.6):

> *Assume Q is Π_t^p-complete and has padding. Then Q has hard sequences for algorithms if and only if Q has hard sets for algorithms.*

The property "Q has hard sequences for algorithms" is preserved under polynomial time reductions, that is, if $Q \leq_p Q'$ and Q has hard sequences for algorithms, then so does Q'. It is this property of hard sequences we use to derive the results (iii) and (iv), the results in "classical terms." In the proofs we cannot replace hard sequences by hard sets. In fact, we only know that the property "Q has hard sets for algorithms" is preserved under polynomial time reductions *among paddable problems*.

We prove (cf. Theorem 3.5) for arbitrary Q that the existence of hard sets for all algorithms is equivalent to the existence of an effective enumeration of all polynomial time decidable subsets of Q, a property which has turned out to be useful in various contexts (cf. [Sadowski 2002; 2007; Chen and Flum 2010; 2011; Beyersdorff and Sadowski 2011]).

By (vi), the satisfiability problem SAT for propositional logic has hard sets for algorithms unless it has an almost optimal algorithm. We do not know whether a similar result holds for hard sequences.

The rest of the paper is organized as follows. In Section 2 we recall some concepts. We introduce and compare the notions of hard sequence, strongly hard sequence, and hard set in Section 3. We obtain our results concerning the existence of hard sequences for algorithms in Section 4 and for proof systems in Section 6. We derive some consequences of our results concerning hard sequences of algorithms in Section 5. Section 7 contains the results and the examples of problems with special properties obtained assuming that the Measure Hypothesis holds. Finally Section 8 gives an effective procedure yielding hard sequences for nondeterministic algorithms for coNEXP-hard problems.

2. Preliminaries

By $n^{O(1)}$ we denote the class of polynomially bounded functions on the natural numbers. We let Σ be the alphabet $\{0,1\}$ and $|x|$ the length of a string $x \in \Sigma^*$. We identify problems with subsets of Σ^*. *In this paper we always assume that Q denotes a decidable and nonempty problem.*

If $\mathbb{A}$ is a deterministic or nondeterministic algorithm and $\mathbb{A}$ accepts the string x, then we denote by $t_{\mathbb{A}}(x)$ the minimum number of steps of an accepting run of $\mathbb{A}$ on x; if $\mathbb{A}$ does not accept x, then $t_{\mathbb{A}}(x)$ is not defined. By $L(\mathbb{A})$ we denote the language accepted by $\mathbb{A}$. We use deterministic and nondeterministic Turing machines as our basic computational model for algorithms (and we often use the notions "algorithm" and "Turing machine" synonymously). Unless necessary, we will not distinguish between a Turing machine and its code, a string in Σ^*. *By default, algorithms are deterministic*. If an algorithm $\mathbb{A}$ on input x eventually halts and outputs a value, we denote it by $\mathbb{A}(x)$.

We assume familiarity with the classes P (polynomial time), NP (nondeterministic polynomial time), and the classes Π_t^p for $t \geq 1$ (the "universal" class of the tth level of the polynomial hierarchy). In particular, Π_1^p = coNP. For a function $t : \mathbb{N} \to \mathbb{N}$ we denote by $\text{DTIME}(t)$ the class of problems decidable by an algorithm $\mathbb{A}$ with $t_{\mathbb{A}}(x) \leq c{\cdot}t(|x|)$ for all $x \in \Sigma^*$ and some constant $c \in \mathbb{N}$. The nondeterministic class $\text{NTIME}(t)$ is defined accordingly. Recall the classes $\text{E} := \bigcup_{d\in\mathbb{N}} \text{DTIME}(2^{d\cdot n})$ (exponential time with linear exponent) and $\text{NE} = \bigcup_{d\in\mathbb{N}} \text{NTIME}(2^{d\cdot n})$ (nondeterministic exponential time with linear exponent).

The *Measure Hypothesis* [Lutz 1997b] is the assumption

NP does not have measure 0 in E.

For the corresponding notion of measure we refer to [Mayordomo 1994]. The Measure Hypothesis has been used extensively in complexity theory [Lutz 1997a; Buhrman et al. 1997].

A problem $Q \subseteq \Sigma^*$ *has padding* (or, *is paddable*) if there is a function $pad : \Sigma^* \times \Sigma^* \to \Sigma^*$ computable in polynomial time having the following properties:

- For any $x, y \in \Sigma^*$, $|pad(x, y)| > |x| + |y|$ and $\big(pad(x, y) \in Q \iff x \in Q\big)$.
- There is a polynomial time algorithm which, given $pad(x, y)$ recovers y.

By $\langle \ldots, \ldots \rangle$ we denote some standard polynomial time computable tupling function with polynomial time computable inverses.

If Q and Q' are problems, we write $Q \leq_p Q'$ if there is polynomial time (many-one) reduction from Q to Q'.

3. Hard sequences and hard sets for algorithms

In this section we introduce or recall the notions of hard sequence, of strongly hard sequence, and of hard set for algorithms. Furthermore, we derive some simple results comparing these notions.

Definition 3.1. Let $Q \subseteq \Sigma^*$.

(a) Let $\mathbb{A}$ be a deterministic (nondeterministic) algorithm deciding (accepting) Q.
  - A sequence $(x_s)_{s\in\mathbb{N}}$ is *hard for* $\mathbb{A}$ if $\{x_s \mid s \in \mathbb{N}\} \subseteq Q$, the function $1^s \mapsto x_s$ is computable in polynomial time, and $t_{\mathbb{A}}(x_s)$ is not polynomially bounded in s.
  - If in addition $s \leq |x_s|$ holds for all s, then $(x_s)_{s\in\mathbb{N}}$ is *strongly hard for* $\mathbb{A}$
  - A subset X of Q is *hard for* $\mathbb{A}$ if $X \in$ P ($X \in$ NP if $\mathbb{A}$ is nondeterministic) and $\mathbb{A}$ is not polynomial-time bounded on X.

(b) The problem Q *has hard sequences for algorithms* if every algorithm deciding Q has a hard sequence.

(c) The problem Q *has hard sequences for nondeterministic algorithms* if every nondeterministic algorithm accepting Q has a hard sequence.

(d) Similarly we define the notions of Q *has strongly hard sequences for algorithms*, Q *has hard sets for algorithms*, ...

Clearly, a strongly hard sequence $(x_s)_{s\in\mathbb{N}}$ yields the hard set $\{x_s \mid s \in \mathbb{N}\}$. In Example 4.2 we show that there are algorithms with hard sequences but without strongly hard sequences. On the other hand, we observe:

PROPOSITION 3.2. *Assume Q has padding. If Q has hard sequences for (nondeterministic) algorithms, then Q has strongly hard sequences and hence, hard sets for (nondeterministic) algorithms.*

PROOF. Let *pad* be a padding function for Q and let $\mathbb{A}$ be any algorithm deciding Q. We show that $\mathbb{A}$ has a strongly hard sequence. The algorithm $\mathbb{B}$ decides Q as follows: on input x the algorithm $\mathbb{B}$ on its ith step performs the ith step of the computation of $\mathbb{A}$ on $pad(x, 1^0)$, the $(i-1)$th step of the computation of $\mathbb{A}$ on $pad(x, 1^1)$,..., the first step of of the computation $\mathbb{A}$ on $pad(x, 1^{i-1})$.

If any of these computations of $\mathbb{A}$ stops, then $\mathbb{B}$ answers accordingly. There is a polynomial $p(u, v)$ such that for all strings x and $s \in \mathbb{N}$ we have

$$t_{\mathbb{B}}(x) \leq p(s, t_{\mathbb{A}}(pad(x, 1^s))).$$

Hence if $(x_s)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{B}$, then $\big(pad(x_s, 1^s)\big)_{s\in\mathbb{N}}$ is a strongly hard sequence for $\mathbb{A}$. □

As already mentioned in Section 1, we will show (see Theorem 7.10) that, assuming the Measure Hypothesis, there are problems having hard sets for algorithms but which have algorithms without hard sequences. We do not know whether there are problems Q with hard sequences which do not have hard sets (by the previous proposition such Q's can not be paddable). However, the following example shows that there are problems Q that have algorithms with hard sequences but without hard sets.

Example 3.3. Let Q be P-immune,[4] that is, Q is infinite yet has no infinite P-subsets. In particular, no algorithm deciding Q has a hard set. But Q (as every infinite problem) has an algorithm with a hard sequence: Let $\mathbb{E}$ be an algorithm that enumerates all elements in Q,

$$e_0, e_1, \ldots,$$

such that $|e_s| \leq |e_{s+1}|$ for every $s \in \mathbb{N}$. Moreover let t_0 be the number of steps which $\mathbb{E}$ needs to output the first element e_0. Let $\mathbb{S}$ be a polynomial time algorithm which on input 1^s outputs the last element enumerated by $\mathbb{E}$ in $s + t_0$ steps. We denote this element by x_s.

We consider the algorithm $\mathbb{A}$ which on input x simulates $\mathbb{S}$ on inputs 1^0, 1^1, 1^2,... outputting x_0, x_1, x_2,... until we get the first x_s with $|x| < |x_s|$. Then $\mathbb{A}$ checks whether x is among $\{x_0, \ldots, x_{s-1}\}$. If so, the algorithm $\mathbb{A}$ makes 2^m additional dummy steps and accepts, where m is the number of steps done so far by $\mathbb{A}$. Note that $m \geq s$. Otherwise, $\mathbb{A}$ rejects. Then $(x_s)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{A}$. ⊣

Our main tool will be hard sequences. In comparison with hard sets, they have the advantage (besides the fact that they can be generated in polynomial time) that they are "preserved upward" under polynomial time reductions, while this is known for hard sets only among paddable problems:

LEMMA 3.4.

(a) Assume that $\mathbb{S}$ is a polynomial time reduction from Q to Q' and let $\mathbb{B}$ be a (nondeterministic) algorithm deciding (accepting) Q'. If $(x_s)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{B} \circ \mathbb{S}$, then $\big(\mathbb{S}(x_s)\big)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{B}$.
Therefore, if $Q \leq_p Q'$ and Q has hard sequences for (nondeterministic) algorithms, then so does Q'.

(b) If $Q \leq_p Q'$, Q' is paddable, and Q has hard sets for (nondeterministic) algorithms, then so does Q'.

We leave the straightforward proof of this lemma to the reader. We close this section showing that for an arbitrary problem Q the existence of hard sets is equivalent to a (non-)listing property. We introduce this property.

Let C be the complexity class P or NP. A set X is a C-subset of Q if $X \subseteq Q$ and $X \in$ C. We write List(C, Q) and say that there is a *listing of the* C*-subsets of* Q *by* C*-machines* if there is an algorithm that, once having been started, lists Turing machines $\mathbb{M}_1, \mathbb{M}_2, \ldots$ of type C such that

$$\{L(\mathbb{M}_i) \mid i \geq 1\} = \{X \subseteq Q \mid X \in \text{C}\}.$$

[4] As observed in [Ko and Moore 1981], Berman and Hartmanis [1977] show that E contains a P-immune (even a P-bi-immune) set.

A weaker notion of listing is considered in [Beyersdorff and Sadowski 2011] and called there "recursive P-presentation (resp. NP-presentation)." For Q with padding the equivalences in the following proposition were known [Sadowski 2002].

THEOREM 3.5.

(a) Q has hard sets for algorithms if and only if List(P, Q) *does not hold.*
(b) Q has hard sets for nondeterministic algorithms if and only if List(NP, Q) *does not hold.*

PROOF. We only prove the first claim as the second one can be obtained along the same lines. First we show the implication from right to left. For a contradiction assume that $\mathbb{A}$ is an algorithm deciding Q without hard sets. For $d \in \mathbb{N}$, by $\mathbb{A}(d)$ we denote the algorithm that on input x simulates $\mathbb{A}$ on input x but rejects if the simulation exceeds time $|x|^d$. We fix an effective enumeration $\mathbb{D}_1, \mathbb{D}_2, \ldots$ of all polynomial time Turing machines. Then $\big(\mathbb{D}_i(\mathbb{A}(j))\big)_{i,j\geq 1}$ is a listing of the P-subsets of Q, where $\mathbb{D}_i(\mathbb{A}(j))$ on input x, first simulates $\mathbb{A}(j)$ on x and if this algorithm accepts, then it simulates $\mathbb{D}_i$ on input x and answers accordingly. As $\mathbb{A}(j)$ has to accept x, we have $L(\mathbb{D}_i(\mathbb{A}(j))) \subseteq Q$. Let X be any P-subset of Q accepted, say, by $\mathbb{D}_i$. As $\mathbb{A}$ has no hard set, there is a $d \in \mathbb{N}$ such that $X \subseteq L(\mathbb{A}(d))$. Then $L(\mathbb{D}_i(\mathbb{A}(d))) = X$.

Conversely, assume that Q has hard sets for algorithms. By contradiction assume that $\mathbb{L}$ is a listing witnessing List(P, Q). Let $\mathbb{Q}$ be an algorithm deciding Q. Consider the algorithm $\mathbb{A}$ that on input x simulates $\mathbb{Q}$ on x and in parallel for $i = 1, 2, \ldots$ does the following:

– performs the ith step of $\mathbb{L}$;
– if $\mathbb{M}_1, \ldots, \mathbb{M}_s$ are the machines listed by $\mathbb{L}$ so far, it performs an additional step of each of the $\mathbb{M}_j$s on x; if one of these accepts, it accepts.

If $\mathbb{Q}$ halts first, it answers accordingly.

It should be clear that $\mathbb{A}$ accepts Q. By assumption, there is a set X hard for $\mathbb{A}$. Let $\mathbb{M}_{i_0}$ accept X. By definition of $\mathbb{A}$ it should be clear that $\mathbb{A}$ is polynomial on X, a contradiction. □

4. Almost optimal algorithms and hard sequences

First, we recall the notion of almost optimal algorithm. Then we derive results concerning the existence of hard sequences for Π_t^p-complete problems and draw some consequences.

Let $Q \subseteq \Sigma^*$. A deterministic (nondeterministic) algorithm $\mathbb{A}$ deciding (accepting) Q is *almost optimal* if for every deterministic (nondeterministic) algorithm $\mathbb{B}$ deciding (accepting) Q we have

$$t_{\mathbb{A}}(x) \leq \big(t_{\mathbb{B}}(x) + |x|\big)^{O(1)}$$

for all $x \in Q$. Note that nothing is required for $x \notin Q$.

Clearly, for a problem Q in P (in NP) every polynomial time (nondeterministic polynomial time) algorithm deciding (accepting) Q is an almost optimal algorithm (an almost optimal nondeterministic algorithm). There are problems outside P with an almost optimal algorithm (see Messner [2000, Corollary 3.33]; we slightly improve his result in Theorem 7.1 of Section 7). However, it is not known whether there are problems outside NP having an almost optimal nondeterministic algorithm and it is not known whether there are problems with padding outside P having an almost optimal algorithm. We show in Theorem 7.4 of Section 7 that the former is true if the Measure Hypothesis holds.

The following lemma is well-known and its proof straightforward. It shows that if $(x_s)_{s\in\mathbb{N}}$ is hard for an algorithm $\mathbb{A}$, then $\mathbb{A}$ can be superpolynomially speeded up on $\{x_s \mid s \in \mathbb{N}\}$; thus $\mathbb{A}$ can't be almost optimal.

LEMMA 4.1. *Let $\mathbb{A}$ be a deterministic (nondeterministic) algorithm deciding (accepting) Q. If $\mathbb{A}$ has a hard sequence or a hard set, then $\mathbb{A}$ is not almost optimal.*

PROOF. We prove the deterministic case for hard sequences, the other cases are obtained by the obvious modifications. So assume that the algorithm $\mathbb{A}$ decides Q and has a hard sequence $(x_s)_{s\in\mathbb{N}}$; in particular,

$$t_{\mathbb{A}}(x_s) \text{ is not polynomially bounded in } s. \tag{1}$$

Let $\mathbb{G}$ be a polynomial time algorithm computing the function $1^s \mapsto x_s$. The following algorithm $\mathbb{G}^*$ accepts the set $\{x_s \mid s \in \mathbb{N}\}$ and for $x = x_s$ runs in time polynomial in s.

```
G*   // x ∈ Σ*
1.  ℓ ← 0
2.  for s = 0 to ℓ
3.        simulate the (ℓ − s)th step of G on 1^s
4.        if this simulation outputs y and y = x then accept and halt
5.  ℓ ← ℓ + 1
6.  goto 2.
```

We consider the algorithm $\mathbb{A}\|\mathbb{G}^*$ that on input x runs $\mathbb{A}$ and $\mathbb{G}^*$ in parallel, both on input x, and halts, when the first of these algorithms halts, then answering in the same way. Hence, $\mathbb{A}\|\mathbb{G}^*$ accepts Q and $t_{\mathbb{A}\|\mathbb{G}^*}(x_s)$ is polynomially bounded in s. As $|x_s| \leq s^{O(1)}$, by (1) we see that $t_{\mathbb{A}}(x_s)$ is not polynomially bounded in $t_{\mathbb{A}\|\mathbb{G}^*}(x_s) + |x_s|$; thus $\mathbb{A}\|\mathbb{G}^*$ witnesses that $\mathbb{A}$ is not an almost optimal algorithm. □

If an algorithm is not almost optimal, does it have hard sequences? We show that it may not have strongly hard sequences.

Example 4.2. We set $r_0 := 0$ and $r_{i+1} := 2^{r_i}$ for $i \geq 1$ and define Q by

$$Q := \{1^{r_i} \mid i \in \mathbb{N}\}.$$

So $Q \in$ P and thus Q itself is a hard set of every algorithm deciding Q which is not almost optimal. We show:

(a) There is no sequence $(x_s)_{s\in\mathbb{N}}$ of elements of Q with $s \leq |x_s|$ such that the function $1^s \mapsto x_s$ is computable in polynomial time and $s \leq |x_s|$ for all $s \geq 0$. In particular, no algorithm deciding Q has a strongly hard sequence.
(b) The sequence $(x_s)_{s\in\mathbb{N}}$ defined by

$$x_s := 1^{r_i}, \text{ if } r_i \leq s < r_{i+1} \tag{2}$$

is hard for every algorithm which decides Q and is not almost optimal.

To show (a), assume for a contradiction that the sequence $(x_s)_{s\in\mathbb{N}}$ is "strongly hard." Then, $r_i + 1 \leq |x_{r_i+1}|$ and thus, $2^{r_i} \leq |x_{r_i+1}|$ since $x_{r_i+1} \in Q$. This shows that the function $1^s \mapsto x_s$ is not computable in polynomial time.

To show (b) let $\mathbb{A}$ be an algorithm that decides Q and is not almost optimal. Then, for every $i \in \mathbb{N}$ there is $y_i \in Q$ such that

$$t_{\mathbb{A}}(y_i) > |y_i|^i.$$

As $y_i \in Q$, we know that for some $j_i \in \mathbb{N}$ we have $y_i = 1^{r_{j_i}} = x_{r_{j_i}}$ $\big($the last equality holding by (2)$\big)$. Thus, for all $i \in \mathbb{N}$,

$$t_{\mathbb{A}}(x_{r_{j_i}}) = t_{\mathbb{A}}(y_i) > |y_i|^i = (r_{j_i})^i.$$

Thus, $t_{\mathbb{A}}(x_s)$ is not polynomially bounded in s. ⊣

By Lemma 4.1, if a problem Q has hard sequences (or hard sets), then it has no almost optimal algorithm. Does the converse hold? For sequences we show the converse for problems Q that are Π_t^p-complete for some t. As partly mentioned in the Introduction (cf. (vi) on page 4), Messner [Messner 2000, Theorem 3.4 and Theorem 3.19] proved the corresponding result for hard sets. He doesn't need the assumption of Π_t^p-completeness of Q but must assume that Q satisfies a property which holds for all paddable Q.

THEOREM 4.3. *Let Q be a Π_t^p-complete problem for some $t \geq 1$. Then:*

(a) Q has no almost optimal algorithm if and only if Q has hard sequences for algorithms.
(b) Q has no almost optimal nondeterministic algorithm if and only if Q has hard sequences for nondeterministic algorithms.

Remark 4.4. For Q = TAUT (or for a halting problem polynomially isomorphic to TAUT) the previous result is implicit in [Krajíček and Pudlák 1989; Krajíček 1995; Monroe 2011; Chen and Flum 2010]. In Remark 5.4 we show how this can directly be extended to every coNP-complete problem using known results relating almost optimal algorithms and proof systems. ⊣

Lemma 4.1 yields the implications from right to left in Theorem 4.3. The following considerations will yield a proof of the converse direction. For a nondeterministic algorithm $\mathbb{A}$ and $s \in \mathbb{N}$ let $\mathbb{A}^s$ be the algorithm that rejects all $x \in \Sigma^*$ with $|x| > s$. If $|x| \leq s$, then it simulates s steps of $\mathbb{A}$ on input x; if this simulation halts and accepts, then $\mathbb{A}^s$ accepts; otherwise it rejects.

Recall that by $L(\mathbb{A})$ we denote the language accepted by $\mathbb{A}$. For $Q \subseteq \Sigma^*$ we consider the *deterministic algorithm subset problem* DAS(Q) and the *nondeterministic algorithm subset problem* NAS(Q).

DAS(Q)	
Instance:	A deterministic algorithm $\mathbb{A}$ and 1^s with $s \in \mathbb{N}$.
Question:	$L(\mathbb{A}^s) \subseteq Q$?

NAS(Q)	
Instance:	A nondeterministic algorithm $\mathbb{A}$ and 1^s with $s \in \mathbb{N}$.
Question:	$L(\mathbb{A}^s) \subseteq Q$?

The following two lemmas relate the equivalent statements in Theorem 4.3 (a) $\big($in Theorem 4.3 (b)$\big)$ to a statement concerning the complexity of DAS(Q) $\big($of NAS(Q)$\big)$.

LEMMA 4.5.

(a) If $\langle \mathbb{A}, 1^s \rangle \in$ DAS(Q) is solvable in time $s^{f(\mathbb{A})}$ for some function f, then Q has an almost optimal algorithm.
(b) If there is a nondeterministic algorithm $\mathbb{V}$ accepting NAS(Q) *such that for all $\langle \mathbb{A}, 1^s \rangle \in$ NAS(Q) we have $t_{\mathbb{V}}(\langle \mathbb{A}, 1^s \rangle) \leq s^{f(\mathbb{A})}$ for some function f, then Q has an almost optimal nondeterministic algorithm.*

PROOF. Again we only prove (a). Let $\mathbb{V}$ be an algorithm deciding $\langle \mathbb{A}, 1^s \rangle \in$ DAS(Q) in time $s^{f(\mathbb{A})}$ for some function f. Further let $\mathbb{Q}$ be an algorithm deciding Q and let $\mathbb{A}_0, \mathbb{A}_1, \ldots$ be an effective enumeration of all algorithms. Consider the following algorithm $\mathbb{A}$ deciding Q.

```
𝔸   // x ∈ Σ*
1.  simulate ℚ on x and in parallel do the following
2.      for i = 0 to |x| do in parallel
3.          simulate 𝔸_i on x
4.          if 𝔸_i accepts then
5.              s ← max{|x|, t_{𝔸_i}(x)} [5]
6.              if 𝕍 accepts ⟨𝔸_i, 1^s⟩ then accept and halt
7.              else never halt
8.          else never halt
9.  if ℚ stops first then answer accordingly and halt.
```

It is easy to see that $\mathbb{A}$ decides Q. We show it is almost optimal. Let $\mathbb{B}$ be any algorithm deciding Q. We choose $i_{\mathbb{B}} \in \mathbb{N}$ such that $\mathbb{B} = \mathbb{A}_{i_{\mathbb{B}}}$. Note that $\mathbb{V}$ accepts $\langle \mathbb{B}, 1^s \rangle$ for all s. Hence for inputs $x \in Q$ with $|x| \geq i_{\mathbb{B}}$ the algorithm $\mathbb{A}$, for $i = i_{\mathbb{B}}$, accepts x in Line 6 if it was not already accepted earlier. Thus, $t_{\mathbb{A}}(x)$ is polynomially bounded in

$$|x| + t_{\mathbb{B}}(x) + t_{\mathbb{V}}\left(\langle \mathbb{B}, 1^{\max\{|x|, t_{\mathbb{B}}(x)\}} \rangle\right).$$

Hence, by the assumption on $\text{DAS}(Q)$, it is polynomially bounded in $\max\{|x|, t_{\mathbb{B}}(x)\}^{f(\mathbb{B})}$. Altogether, $t_{\mathbb{A}}(x) \leq \left(|x| + t_{\mathbb{B}}(x)\right)^{O(1)}$. □

If Q is in Π_l^p, then the problem $\text{NAS}(Q)$ and hence the problem $\text{DAS}(Q)$ are in Π_l^p, too (this is the reason why 1^s and not just s is part of the input of $\text{NAS}(Q)$ and of $\text{DAS}(Q)$). Thus, together with Lemma 4.5 the following lemma yields the remaining claims of Theorem 4.3.

LEMMA 4.6.

(a) Assume that $\text{DAS}(Q) \leq_p Q$*, that is, that* $\text{DAS}(Q)$ *is polynomial time reducible to* Q*. If for every function* f*,* $\langle \mathbb{A}, 1^s \rangle \in \text{DAS}(Q)$ *is not solvable in time* $s^{f(\mathbb{A})}$*, then* Q *has hard sequences for algorithms.*

(b) Assume that $\text{NAS}(Q) \leq_p Q$*. If there is no nondeterministic algorithm* $\mathbb{V}$ *accepting* $\text{NAS}(Q)$ *such that for all* $\langle \mathbb{A}, 1^s \rangle \in \text{NAS}(Q)$ *we have* $t_{\mathbb{V}}(\langle \mathbb{A}, 1^s \rangle) \leq s^{f(\mathbb{A})}$ *for some function* f*, then* Q *has hard sequences for nondeterministic algorithms.*

PROOF. Again we only prove part (a).

Claim. Assume that $\langle \mathbb{A}, 1^s \rangle \in \text{DAS}(Q)$ is not solvable in time $s^{f(\mathbb{A})}$ for some function f. Then, for every algorithm $\mathbb{W}$ deciding $\text{DAS}(Q)$, there exists an algorithm $\mathbb{A}$ with $L(\mathbb{A}) \subseteq Q$ and $t_{\mathbb{W}}\big(\langle \mathbb{A}, 1^s \rangle\big)$ is not polynomially-bounded in s.

Proof of the Claim. By contradiction, assume that $\mathbb{W}$ decides $\text{DAS}(Q)$ and that for all algorithms $\mathbb{A}$ with $L(\mathbb{A}) \subseteq Q$ there is a $c_{\mathbb{A}} \in \mathbb{N}$ such that for all $s \in \mathbb{N}$ we have $t_{\mathbb{W}}\big(\langle \mathbb{A}, 1^s \rangle\big) \leq s^{c_{\mathbb{A}}}$.

Let $\mathbb{V}$ be the algorithm that, on an arbitrary input $\langle \mathbb{A}, 1^s \rangle$, in parallel runs $\mathbb{W}$ on $\langle \mathbb{A}, 1^s \rangle$ and computes

$$r_{\mathbb{A}} := \text{the least } r \text{ such that } L(\mathbb{A}^r) \not\subseteq Q$$

by systematically checking for $r = 0, 1, \ldots$ whether $L(\mathbb{A}^r) \not\subseteq Q$ (this is done by running for all x with $|x| \leq r$ the algorithm $\mathbb{A}$ at most r steps on input x and a decision procedure for Q on x). Note that $r_{\mathbb{A}}$ is not defined if $L(\mathbb{A}) \subseteq Q$. If $\mathbb{W}$ stops first, $\mathbb{V}$ answers accordingly; if $r_{\mathbb{A}}$ is obtained first, then $\mathbb{V}$ accepts if $s < r_{\mathbb{A}}$ and otherwise it rejects. It should be clear that the algorithm $\mathbb{V}$ decides $\langle \mathbb{A}, 1^s \rangle \in \text{DAS}(Q)$ in $\leq s^{f(\mathbb{A})}$ steps for some function f. ⊣

[5] In the nondeterministic case, we replace Line 3 by "simulate $\mathbb{A}_i$ on x and count the number $n_{\mathbb{A}_i}(x)$ of steps" and Line 5 by "$s \leftarrow \max\{|x|, n_{\mathbb{A}_i}(x)\}$."

By assumption, there is a polynomial time reduction $\mathbb{S}$ from $\textsc{Das}(Q)$ to Q. Let $\mathbb{B}$ be an arbitrary algorithm deciding Q. Then the algorithm $\mathbb{B} \circ \mathbb{S}$, which on input x first simulates $\mathbb{S}$ on x and then $\mathbb{B}$ on $\mathbb{S}(x)$, decides $\textsc{Das}(Q)$. Hence, by the Claim, there exists an algorithm $\mathbb{A}$ with $L(\mathbb{A}) \subseteq Q$ such that $t_{\mathbb{B}\circ\mathbb{S}}\big(\langle \mathbb{A}, 1^s\rangle\big)$ is not polynomially bounded in s. For $s \in \mathbb{N}$ we set $x_s := \mathbb{S}(\langle \mathbb{A}, 1^s\rangle)$. Then $x_s \in Q$ for all s and the function $1^s \mapsto x_s$ is polynomial time computable. Furthermore

$$t_{\mathbb{B}\circ\mathbb{S}}(\langle \mathbb{A}, 1^s\rangle) \leq O\Big(t_{\mathbb{S}}(\langle \mathbb{A}, 1^s\rangle) + t_{\mathbb{B}}(\mathbb{S}(\langle \mathbb{A}, 1^s\rangle))\Big) \leq s^{O(1)} + O\big(t_{\mathbb{B}}(x_s)\big).$$

As the left-hand side is not polynomially bounded in s, neither is $t_{\mathbb{B}}(x_s)$. Hence $(x_s)_{s\in\mathbb{N}}$ is hard for $\mathbb{B}$. □

Remark 4.7. In the proof of Theorem 4.3 we used the assumption that Q is Π_t^p-complete only to ensure that $\textsc{Nas}(Q) \leq_p Q$ (cf. Lemma 4.6). This condition is also fulfilled for every Q complete, say, for one of the classes E or PSPACE. Thus the statements of Theorem 4.3 hold for such a Q.

We also get $\textsc{Nas}(Q) \leq_p Q$ if Q is $\forall$-closed, a property considered by Messner [2000]. He shows in [Messner 2000, Theorem 3.48 and Theorem 3.49] the following (we only state the result for the deterministic case):

> *Let Q be $\forall$-closed problem, which has padding (again, a weaker notion than padding suffices). Then Q has no almost optimal algorithm if and only if Q has* supersparse *hard sets for algorithms.*

Recall that a set X of strings is *supersparse* if for some constant c and all $n \in \mathbb{N}$ it contains at most $c + \log\log n$ strings of length $\leq n$. Note that even a supersparse set X which is hard for a given algorithm $\mathbb{A}$ does not seem to yield a hard sequence for $\mathbb{A}$; it may be that $t_{\mathbb{A}}(x_s)$ is polynomially bounded in s for every polynomial time enumeration $(x_s)_{s\in\mathbb{N}}$ of X. ⊣

Remark 4.8. Assume that Q is Π_t^p-complete and has padding (for $t = 1$, the set TAUT is an example of such a Q). If Q has no almost optimal algorithm, then every algorithm $\mathbb{B}$ deciding Q has a strongly hard sequence (as already mentioned, for $Q =$ TAUT this was proven in [Krajíček 1995, Theorem 14.2.3]). In fact, it is well-known that for Q with padding we can replace any polynomial time reduction to Q by a length-increasing one. Hence, then in the proof of Lemma 4.6 we may assume that $\mathbb{S}$ is length-increasing and therefore $s \leq |x_s|$. ⊣

We derive a consequence of Theorem 4.3 which does not mention hard sequences.

COROLLARY 4.9. *Let $t \geq 1$ and assume that some Π_t^p-complete problem has no almost optimal algorithm. Then every Π_t^p-hard problem has no almost optimal algorithm.*

PROOF. Assume that the Π_t^p-complete problem Q has no almost optimal algorithm. Then, by Theorem 4.3, the problem Q has hard sequences for algorithms and so does every Π_t^p-hard problem by Lemma 3.4. Now the claim follows from Lemma 4.1. □

5. Proof systems

In this section we recall the notion of proof system and use some well-known results to obtain further consequences of Theorem 4.3.

A *proof system for Q* is a polynomial time algorithm $\mathbb{P}$ computing a function from Σ^* onto Q. If $\mathbb{P}(w) = x$, we say that w is a $\mathbb{P}$-*proof* of x. Often we introduce proof systems implicitly by defining the corresponding function; then the definition of this function will suggest an algorithm.

Definition 5.1. Let $\mathbb{P}$ and $\mathbb{P}'$ be proof systems for Q. An algorithm $\mathbb{T}$ is a *translation from $\mathbb{P}'$ into $\mathbb{P}$* if $\mathbb{P}(\mathbb{T}(w')) = \mathbb{P}'(w')$ for every $w' \in \Sigma^*$. Note that translations always exist. A translation is *polynomial* if it runs in polynomial time.

A proof system $\mathbb{P}$ for Q is *p-optimal* or *polynomially optimal* if for every proof system $\mathbb{P}'$ for Q there is a polynomial translation from $\mathbb{P}'$ into $\mathbb{P}$. A proof system $\mathbb{P}$ for Q is *optimal* if for every proof system $\mathbb{P}'$ for Q and every $w' \in \Sigma^*$ there is a $w \in \Sigma^*$ such that $\mathbb{P}(w) = \mathbb{P}'(w')$ and $|w| \leq |w'|^{O(1)}$. Clearly, every p-optimal proof system is optimal.

We often will make use of the following relationship between the optimality notions for algorithms and that for proof systems (see [Krajíček and Pudlák 1989; Messner 2000]).

THEOREM 5.2.

(a) For every Q we have (i) $\Rightarrow$ (ii) and (ii) $\Rightarrow$ (iii); moreover (i), (ii), and (iii) are all equivalent if Q has padding. Here
- *(i) Q has a p-optimal proof system.*
- *(ii) Q has an almost optimal algorithm.*
- *(iii) There is an algorithm that decides Q and has no hard sets.*

(b) For every Q we have (i) $\iff$ (ii), (ii) $\Rightarrow$ (iii), and (iii) $\Rightarrow$ (iv); moreover (i)–(iv) are all equivalent if Q has padding. Here
- *(i) Q has an optimal proof system.*
- *(ii) Q has an almost optimal nondeterministic algorithm.*
- *(iii) There is a nondeterministic algorithm that accepts Q and has no hard sets.*
- *(iv) There is a nondeterministic algorithm that accepts Q and runs in polynomial time on every subset X of Q with $X \in$ P.*

In the following Remark 5.4, we combine the "upward preservation" of hard sequences $\big($cf. Lemma 3.4 (a)$\big)$ with the "downward preservation" of optimal proof systems, that is, with the following statement:

LEMMA 5.3. *([Köbler et al. 2003, Lemma 2.1]) If $Q \leq_p Q'$ and Q' has a p-optimal (optimal) proof system, so does Q.*

Remark 5.4. Using the previous theorem, we get a simple direct proof of

> *if Q is* coNP*-complete and has no almost optimal (nondeterministic) algorithm, then Q has hard sequences for (nondeterministic) algorithms*

using the result for $Q =$ TAUT (that was already known). In fact, assume that Q has no almost optimal algorithm (the proof for the nondeterministic case follows analogously). Then TAUT has no almost optimal algorithm; otherwise, TAUT would have a p-optimal proof system by the equivalence of (i) and (ii) in part (a) of the previous theorem (TAUT has padding!). As $Q \leq_p$ TAUT, then Q would have a p-optimal proof system, too (by Lemma 5.3). Hence, again by the previous theorem, Q would have an almost optimal algorithm, a contradiction. So, TAUT has no almost optimal algorithm and thus, TAUT has hard sequences for algorithms. As TAUT $\leq_p Q$, the problem Q has hard sequences for algorithms, too (by Lemma 3.4). ⊣

We use the results of Section 3 to get further consequences of the equivalence of Theorem 5.2. First, we derive the equivalence of (i) and (ii) in Theorem 5.2 (a) for every Π_t^p-complete problem.

THEOREM 5.5. *Let Q be a Π_t^p-complete problem for some $t \geq 1$. Then:*

> *Q has a p-optimal proof system if and only if Q has an almost optimal algorithm.*

PROOF. By Theorem 5.2 (a) the left-hand side implies the right hand side. Now assume that Q is Π_t^p-complete and has an almost optimal algorithm. As $Q \times \Sigma^*$ is Π_t^p-complete too, it has an almost optimal algorithm (by Corollary 4.9). As $Q \times \Sigma^*$ has padding, it has a p-optimal proof system $\mathbb{P}$ (cf. Theorem 5.2 (a)). Now it is routine to show that the algorithm $\mathbb{P}'$ that on input w computes $\mathbb{P}(w)$ and outputs its first component is a p-optimal proof system for Q. □

Furthermore, we get:

COROLLARY 5.6. *Assume Q has padding and is Π_t^p-complete. Then,*

Q has hard sequences for (nondeterministic) algorithms
if and only if Q has hard sets for (nondeterministic) algorithms.

PROOF. The direction from left to right was already shown in Proposition 3.2. If Q has hard sets for algorithms, then, by the equivalence of (ii) and (iii) in Theorem 5.2 (Q has padding!), Q has no almost optimal algorithm. Thus, by Theorem 4.3, Q has hard sequences for algorithms. □

6. Hard sequences for proof systems

We already mentioned in the Introduction that for every $Q \subseteq \Sigma^*$ there is a well-known and straightforward correspondence between proof systems and nondeterministic algorithms preserving the optimality notions, so that the proof of the equivalence between (i) and (ii) in Theorem 5.2 (b) is immediate. In fact, if $\mathbb{P}$ is a proof system for Q, then the nondeterministic algorithm $\mathbb{A}(\mathbb{P})$ accepts Q, where $\mathbb{A}(\mathbb{P})$ on input $x \in \Sigma^*$ guesses a string w and accepts if $\mathbb{P}(w) = x$.

Conversely, we assign to every nondeterministic algorithm a proof system in a canonical way. This assignment works for nondeterministic as well as for deterministic algorithms. So let $\mathbb{A}$ be a deterministic (nondeterministic) algorithm deciding (accepting) Q. Then for every fixed $x_0 \in Q$ the algorithm $\mathbb{A}$ induces a proof system $\mathbb{P}_\mathbb{A}$ for Q defined by

$$\mathbb{P}_\mathbb{A}(w) := \begin{cases} x, & \text{if } w \text{ is a computation of } \mathbb{A} \text{ accepting } x \\ x_0, & \text{otherwise.} \end{cases}$$

We introduce the notions of hard sequences, strongly hard sequences, and hard sets for proof systems such that for proof systems of the form $\mathbb{P}_\mathbb{A}$ they correspond with the notions for the algorithm $\mathbb{A}$ (see Lemma 6.2). Note that, by definition, a proof system is always a deterministic algorithm.

Definition 6.1. Let $\mathbb{P}$ be a proof system for Q.

(a) A sequence $(x_s)_{s\in\mathbb{N}}$ is *hard for* $\mathbb{P}$ if $\{x_s \mid s \in \mathbb{N}\} \subseteq Q$, the function $1^s \mapsto x_s$ is computable in polynomial time, and there is no polynomial time algorithm $\mathbb{W}$ with $\mathbb{P}(\mathbb{W}(1^s)) = x_s$.
(b) The sequence $(x_s)_{s\in\mathbb{N}}$ is *length-hard for* $\mathbb{P}$ if $\{x_s \mid s \in \mathbb{N}\} \subseteq Q$, the function $1^s \mapsto x_s$ is computable in polynomial time, and there is no sequence $(w_s)_{s\in\mathbb{N}}$ such that $\mathbb{P}(w_s) = x_s$ and the length of w_s is polynomially-bounded in s (i.e., $|w_s| \leq |s|^{O(1)}$).
(c) A subset X of Q is *hard for* $\mathbb{P}$ if $X \in$ P and there is no polynomial time algorithm $\mathbb{W}$ such that $\mathbb{P}(\mathbb{W}(x)) = x$ for all $x \in X$.
(d) A subset X of Q is *length-hard for* $\mathbb{P}$ if $X \in$ NP and there is no family $(w_x)_{x\in X}$ with $|w_x| \leq |x|^{O(1)}$ such that $\mathbb{P}(w_x) = x$ for all $x \in X$.

It should be clear how we define strongly hard sequences for $\mathbb{P}$ and strongly length-hard sequences for $\mathbb{P}$. Furthermore, it should be clear how we define for a problem Q the notions *Q has hard sequences for proof systems*, *Q has length-hard sequences for proof systems*, ...

The verification of the following lemma is straightforward.

LEMMA 6.2. *Let $\mathbb{A}$ be a deterministic (nondeterministic) algorithm deciding (accepting) Q and $\mathbb{P}_\mathbb{A}$ be the induced proof system.*

(a) If $\mathbb{A}$ is deterministic, then a sequence $(x_s)_{s\in\mathbb{N}}$ (a set X) is hard for $\mathbb{A}$ if and only if it is hard for $\mathbb{P}_\mathbb{A}$.
(b) If $\mathbb{A}$ is nondeterministic, then a sequence $(x_s)_{s\in\mathbb{N}}$ (a set X) is hard for $\mathbb{A}$ if and only if it is length-hard for $\mathbb{P}_\mathbb{A}$.

For a problem Q every hardness property for algorithms is equivalent to this property for proof systems:

LEMMA 6.3.

(a) Q has hard sequences, strongly hard sequences, or hard sets for algorithms if and only Q it has hard sequences, strongly hard sequences, or hard sets for proof systems, respectively.
(b) Q has hard sequences, strongly hard sequences, or hard sets for nondeterministic algorithms if and only if Q has length-hard sequences, strongly length-hard sequences, or length-hard sets for proof systems, respectively.

Due to the relationship between proof systems and nondeterministic algorithms the verification of (b) is immediate. As a deterministic algorithm $\mathbb{A}$ induces the proof system $\mathbb{P}_{\mathbb{A}}$, also the implication from right to left in (a) is easy. Our proof of the reverse implication is based on a theorem due to Levin on inverters. We recall it.

Let $\mathbb{F}$ be an algorithm computing a function from Σ^* to Σ^*. An *inverter of* $\mathbb{F}$ is an algorithm $\mathbb{I}$ that given y in the range of $\mathbb{F}$ halts with some output $\mathbb{I}(y)$ such that $\mathbb{F}(\mathbb{I}(y)) = y$. On inputs not in the range of $\mathbb{F}$, the algorithm $\mathbb{I}$ may do whatever it wants. Levin [1973] proved the following result.

THEOREM 6.4. *Let $\mathbb{F}$ be an algorithm computing a function from Σ^* into Σ^*. Then there is an optimal inverter that is, an inverter $\mathbb{O}_{\mathbb{F}}$ of $\mathbb{F}$ such that for every inverter $\mathbb{I}$ of $\mathbb{F}$ and all y in the range of $\mathbb{F}$ we have*

$$t_{\mathbb{O}_{\mathbb{F}}}(y) \leq \big(t_{\mathbb{I}}(y) + t_{\mathbb{F}}(\mathbb{I}(y)) + |y|\big)^{O(1)}.$$

Furthermore, $\mathbb{O}_{\mathbb{F}}$ does not halt on inputs y not in the range of $\mathbb{F}$.

PROOF OF THE IMPLICATION FROM LEFT TO RIGHT OF LEMMA 6.3. Assume, say, that Q has hard sequences for algorithms. Let $\mathbb{P}$ be any proof system for Q. By Theorem 6.4, we have an inverter $\mathbb{O}_{\mathbb{P}}$ of $\mathbb{P}$ which is optimal, that is, for every inverter $\mathbb{I}$ of $\mathbb{P}$ and $x \in Q$ we have

$$t_{\mathbb{O}_{\mathbb{P}}}(x) \leq \big(t_{\mathbb{I}}(x) + t_{\mathbb{P}}(\mathbb{I}(x)) + |x|\big)^{O(1)} \leq (t_{\mathbb{I}}(x) + |x|)^{O(1)}, \tag{3}$$

where the second inequality holds as $t_{\mathbb{P}}(w) \leq |w|^{O(1)}$ and hence $t_{\mathbb{P}}(\mathbb{I}(x)) \leq |\mathbb{I}(x))|^{O(1)} \leq t_{\mathbb{I}}(x)^{O(1)}$. Moreover, for $x \notin Q$ the algorithm $\mathbb{O}_{\mathbb{P}}$ will not halt on input x.

We choose an arbitrary algorithm $\mathbb{Q}$ that decides Q and consider the algorithm $\mathbb{S}$ that on input x in parallel simulates $\mathbb{Q}$ and $\mathbb{O}_{\mathbb{P}}$, both on input x. If $\mathbb{Q}$ halts first, then it answers accordingly and if $\mathbb{O}_{\mathbb{P}}$ halts first, then it accepts. Obviously $\mathbb{S}$ decides Q and for every $x \in Q$ we have

$$t_{\mathbb{S}}(x) \leq O\big(t_{\mathbb{O}_{\mathbb{P}}}(x)\big). \tag{4}$$

As Q has hard sequences for algorithms, there is a polynomial time computable algorithm $\mathbb{G}$ generating a hard sequence $(x_s)_{s\in\mathbb{N}}$ for $\mathbb{S}$; that is, $\mathbb{G}$ on input 1^s computes $x_s \in Q$ in polynomial time such that

$$t_{\mathbb{S}}(x_s) \text{ is not polynomially bounded in } s. \tag{5}$$

We show that $(x_s)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{P}$. For this purpose let $\mathbb{G}^+$ be the variant of the algorithm $\mathbb{G}^*$ in the proof of Lemma 4.1 obtained by replacing Line 4 by

if this simulation outputs y and $y = x$ **then** output 1^s and halt.

Of course, on input $x = x_s$ the algorithm $\mathbb{G}^+$ runs in time polynomial in s. Assume for a contradiction that $(x_s)_{s\in\mathbb{N}}$ is not a hard sequence for $\mathbb{P}$. Then there is a polynomial time algorithm $\mathbb{W}$ with $\mathbb{P}(\mathbb{W}(1^s)) = x_s$ for all $s \in \mathbb{N}$. We consider the inverter $\mathbb{I}$ of $\mathbb{P}$ that on input x in parallel simulates $\mathbb{O}_{\mathbb{P}}$ and $\mathbb{G}^+$, both on input x. If $\mathbb{O}_{\mathbb{P}}$ halts, then it outputs the output of $\mathbb{O}_{\mathbb{P}}$ and halts; if $\mathbb{G}^+$ halts, then it simulates $\mathbb{W}$ on $\mathbb{G}^+(x)$, outputs $\mathbb{W}(\mathbb{G}^+(x))$, and halts.

By definition of $\mathbb{G}^+$ the algorithm $\mathbb{I}$ runs, on input x_s, in time polynomial in s, hence so does $\mathbb{O}_{\mathbb{P}}$ by (3) as $|x_s| \leq s^{O(1)}$. But then by (4), the same holds for the algorithm $\mathbb{S}$ contradicting (5).

Clearly, if $(x_s)_{s\in\mathbb{N}}$ was a strongly hard sequence for $\mathbb{S}$, then it is a strongly hard sequence for $\mathbb{P}$. The verification for hard sets (instead of hard sequences) is similar, even easier. □

Now we can show for proof systems the result corresponding to Proposition 3.2.

COROLLARY 6.5. *Assume Q has padding. If Q has hard (length-hard) sequences for proof systems, then Q has strongly hard (strongly length-hard) sequences for proof systems. In particular, then Q has hard sets (length-hard) sets.*

PROOF. Assume Q has hard sequences for proof systems. By Lemma 6.3, Q has hard sequences for algorithms and thus, by Proposition 3.2, strongly hard sequences for algorithms. Again applying Lemma 6.3 we see that Q has strongly hard sequences for proof systems. □

Similarly one can prove the analogue of Corollary 5.6 for proof systems:

COROLLARY 6.6. *Assume Q has padding and is Π_t^p-complete. Then,*

Q has hard (length-hard) sequences for proof systems
if and only if Q has hard (length-hard) sets for proof systems.

As already remarked in the Introduction, for Q = TAUT the following result is known and goes back to Krajíček and Pudlák [1989] (see [Krajíček 1995, Theorem 14.2.2]).

THEOREM 6.7. *Let Q be a Π_t^p-complete problem for some $t \geq 1$. Then:*

(a) Q has no p-optimal proof system if and only if Q has hard sequences for proof systems.
(b) Q has no optimal proof system if and only if Q has length-hard sequences for proof systems.

The implications from right to left hold for all problems Q.

PROOF. First, we present a proof of the directions from right to left, say, for (b). Let $\mathbb{P}$ be any proof system for Q. We show that $\mathbb{P}$ is not optimal. By our assumption on Q there is a length-hard sequence $(x_s)_{s\in\mathbb{N}}$ for $\mathbb{P}$. We consider the proof system $\mathbb{P}'$ for Q given by

$$\mathbb{P}'(w') := \mathbb{P}(w), \text{ if } w' = 0w; \qquad \mathbb{P}'(w') := x_s, \text{ if } w' = 1^s;$$

and $\mathbb{P}'(w') := z_0$ for some fixed element z_0 of Q, otherwise. Let $\mathbb{T}$ be any translation from $\mathbb{P}'$ into $\mathbb{P}$. In particular,

$$\mathbb{P}(\mathbb{T}(1^s)) = \mathbb{P}'(1^s) = x_s$$

for all $s \in \mathbb{N}$. As $(x_s)_{s\in\mathbb{N}}$ is a length-hard sequence for $\mathbb{P}$, the length of $\mathbb{T}(1^s)$ is not polynomially bounded in s. Hence, $\mathbb{T}$ is not a polynomial translation; therefore, $\mathbb{P}$ is not optimal.

Now we present a proof of the direction from left to right, say, for (a). So, assume that Q has no p-optimal proof system. By Theorem 5.5, Q has no almost optimal algorithm and hence has hard sequences for algorithms by Theorem 4.3. Now the claim follows from Lemma 6.3. □

Remark 6.8. If in the previous theorem we assume that Q, in addition, has padding, then (by Corollary 6.6) we can replace the right hand sides in (a) and (b) by "Q has hard sets for proof systems" and by "Q has length-hard sets for proof systems," respectively.

For hard sets Messner [2000, Theorem 3.4 and Theorem 3.19] proved the results corresponding to Theorem 6.7 . He doesn't need the assumption of Π_t^p-completeness of Q but must assume that Q satisfies a property which is weaker than padding. Again with the hypothesis that Q is $\forall$-closed (cf. Remark 4.7), Messner [2000, Theorem 3.4 and Theorem 3.49] shows that one can require the hard sets to be supersparse. ⊣

7. Assuming the Measure Hypothesis

In this section we present some examples of problems with special properties, some yield limitations to possible extensions of results mentioned in this paper. Most are proven assuming the Measure Hypothesis, that is, the statement "NP does not have measure 0 in E," where $\mathrm{E} := \bigcup_{d\in\mathbb{N}} \mathrm{DTIME}(2^{d\cdot n})$.

Recall that an algorithm $\mathbb{A}$ deciding Q is *optimal* if for every algorithm $\mathbb{B}$ deciding Q we have

$$t_{\mathbb{A}}(x) \leq (t_{\mathbb{B}}(x) + |x|)^{O(1)}$$

for *all* $x \in \Sigma^*$. Clearly, every problem in P has an optimal algorithm.

THEOREM 7.1.

(a) There exist problems in E $\setminus$ P *with optimal algorithms.*
(b) If the Measure Hypothesis holds, then there exist problems in NP $\setminus$ P *with optimal algorithms.*

Messner [2000, Theorem 3.32] showed the existence of problems in E $\setminus$ P with *almost* optimal algorithms.

The following considerations will finally lead to a proof of Theorem 7.1. Let C be a class of problems. Recall that a problem Q is C-*immune* if no infinite subset of Q is in C; and it is C-*bi-immune* if Q and its complement $\Sigma^* \setminus Q$ are C-immune.

LEMMA 7.2.

(a) If $Q \in$ E *is a* DTIME$(2^{\ell \cdot n})$*-bi-immune problem for some* $\ell \geq 1$*, then every algorithm witnessing that* $Q \in$ E *is optimal.*
(b) If $Q \in$ NE *is an* NTIME$(2^{\ell \cdot n})$*-immune problem for some* $\ell \geq 1$*, then every nondeterministic algorithm witnessing that* $Q \in$ NE *is optimal.*

PROOF. We prove (a); part (b) is obtained by the obvious modifications. Assume that the Turing machine $\mathbb{M}$ decides the DTIME$(2^{\ell \cdot n})$-bi-immune problem Q in time $c \cdot 2^{d \cdot n}$ for some $c, d \in \mathbb{N}$. We claim that $\mathbb{M}$ is optimal.

Assume otherwise, then there is a machine $\mathbb{M}'$ deciding Q and witnessing that $\mathbb{M}$ is not optimal. Then for every $i \in \mathbb{N}$ there exists an x_i such that

$$t_{\mathbb{M}}(x_i) > \left(t_{\mathbb{M}'}(x_i) + |x_i|\right)^i.$$

It follows that, for every $i \in \mathbb{N}$,

$$c \cdot 2^{d \cdot |x_i|} \geq t_{\mathbb{M}}(x_i) > t_{\mathbb{M}'}(x_i)^i.$$

Thus, $t_{\mathbb{M}'}(x_i) \leq 2^{\ell \cdot |x_i|/2}$ for all sufficiently large $i \in \mathbb{N}$. Of course, infinitely many of these x_i's are in Q or they are in $\Sigma^* \setminus Q$. In the first case consider the following machine:

```
M''      // x ∈ Σ*
1. simulate M' on x for at most 2^{ℓ·|x|/2} steps
2.          if the simulation halts and accepts then accept else reject.
```

It accepts an infinite subset of Q in time $2^{\ell \cdot n}$. This contradicts our immunity assumption. The second case is handled similarly. □

Part (a) of the previous lemma and the following result due to Mayordomo [1994] immediately yields the statements of Theorem 7.1.

THEOREM 7.3. *Let* $\ell \geq 1$.

(a) The class of DTIME$(2^{\ell \cdot n})$*-bi-immune problems has measure 1 in* E*. In particular, the class* E *contains* DTIME$(2^{\ell \cdot n})$*-bi-immune problems.*
(b) If the Measure Hypothesis holds, then NP $\cap$ E *contains* DTIME$(2^{\ell \cdot n})$*-bi-immune problems.*

The question whether there are sets outside of NP with optimal proof systems was stated by Krajíček and Pudlák [1989] and is still open. As already mentioned, they proved that TAUT has an optimal proof system if E = NE. We are able to show:

THEOREM 7.4. *If the Measure Hypothesis holds, then there exist problems in* NE $\setminus$ NP *with optimal proof systems (or, equivalently, with almost optimal nondeterministic algorithms).*

PROOF. It suffices to show that there is a $Q \in$ NE which is NTIME(2^n)-immune. Then, by Lemma 7.2 (b), such a Q has an almost optimal nondeterministic algorithm and hence, an optimal proof system by Theorem 5.2.

By Theorem 7.3 (b) there is a $Q_0 \in$ NP which is DTIME(2^n)-bi-immune problem. We choose $d \geq 1$ such that $Q_0 \in$ NTIME(n^d). We set

$$Q := \left\{ 1^m \mid m \in \mathbb{N} \text{ and } 1^{2^{2m}} \in Q_0 \right\}.$$

Then $Q \in$ NE. Furthermore, Q is infinite as otherwise the set $\{1^{2^m} \mid m \in \mathbb{N} \text{ and } 1^{2^{2m}} \notin Q_0\}$ would be an infinite subset of $\Sigma^* \setminus Q_0$ in P contradicting the bi-immunity property of Q_0. Finally we show that Q is NTIME(2^n)-immune. By contradiction assume that there is an infinite $S \subseteq Q$ accepted by a nondeterministic algorithm $\mathbb{S}$ in time 2^{2n}. Then the set

$$S^* := \{1^n \mid n = 2^{2m} \text{ for some } m \in \mathbb{N} \text{ and } 1^m \in S\}$$

is an infinite subset of Q_0. The algorithm $\mathbb{S}^*$ that first computes m from 1^n and then deterministically simulates all possible runs of $\mathbb{S}$ on 1^m decides S^* and runs in time

$$n^{O(1)} + O\left(2^{2^{2m}}\right) = n^{O(1)} + O(2^n) = O(2^n).$$

This contradicts the DTIME(2^n)-immunity of Q_0. □

Concerning algorithms which are not almost optimal but do not have hard sequences, we derive the following results.

THEOREM 7.5. *Let Q be a problem with padding and with an almost optimal algorithm. If the Measure Hypothesis holds, then there is an algorithm deciding Q, which is not almost optimal and has hard sets but does not have hard sequences.*

The proof of this theorem (and that of Theorem 7.10) are based on the following proposition.

PROPOSITION 7.6. *If the Measure Hypothesis holds, then there is a problem $Q_0 \in$ P such that*

(a) there is an algorithm $\mathbb{B}$ deciding Q_0 which is not almost optimal (or, equivalently, is not polynomial time) but has no hard sequences;
(b) every algorithm $\mathbb{A}$ deciding Q_0 with

$$t_{\mathbb{A}}(x) \leq 2^{e \cdot (\log |x|)^2}$$

for every $x \in \Sigma^*$ and some constant $e \geq 1$ has no hard sequences;
(c) there is a proof system for Q_0 which is not optimal but has no hard sequences.

To get the statements of this proposition we first show:

LEMMA 7.7. *Let $\mathbb{A}$ be an algorithm deciding a problem with*

$$t_{\mathbb{A}}(x) \leq 2^{e \cdot (\log |x|)^2} \tag{6}$$

for all $x \in \Sigma^*$ and some $e \geq 1$. Assume that $(x_s)_{s \in \mathbb{N}}$ is a hard sequence for $\mathbb{A}$. Then there is a sequence $s_0 < s_1 < s_2 < \ldots$ such that

$$\lim_{i \to \infty} \frac{\log s_i}{(\log |x_{s_i}|)^2} = 0 \quad \textit{i.e.,} \quad s_i = 2^{o((\log |x_{s_i}|)^2)}.$$

In particular, the set $\{x_{s_i} \mid i \in \mathbb{N}\}$ is infinite.

PROOF. Assume otherwise that for some $\varepsilon > 0$ and some $n \in \mathbb{N}$ and all $s \geq n$

$$\frac{\log s}{(\log |x_s|)^2} \geq \varepsilon,$$

or equivalently, $s \geq 2^{\varepsilon \cdot (\log |x_s|)^2}$; then $s \geq t_{\mathbb{A}}(x_s)^{\varepsilon/e}$ by assumption (6). This contradicts the hardness of $(x_s)_{s\in\mathbb{N}}$. □

PROOF OF PROPOSITION 7.6. (a) and (b): Assume the Measure Hypothesis. Then, by Theorem 7.3 (b), there is a DTIME(2^n)-bi-immune $Q_1 \in$ NP. In particular, there exists a nondeterministic Turing machine $\mathbb{M}$ with binary nondeterminism and a $d \in \mathbb{N}$ such that for all $y \in \Sigma^*$ (with $|y| \geq 2$) the machine $\mathbb{M}$ decides whether $y \in Q_1$ in $\leq |y|^d$ steps. Thus for $y \in \Sigma^*$ every string $x \in \{0,1\}^{|y|^d}$ determines a unique run of $\mathbb{M}$ on y. We set

$$Q_0 := \Big\{ x \in \{0,1\}^* \;\Big|\; \text{for some } n \in \mathbb{N} \text{ we have } |x| = n^d \text{ and } x \text{ determines an accepting run of } \mathbb{M} \text{ on input } 1^n \Big\}.$$

Then Q_0 is infinite, as otherwise the set $\{1^n \in Q_1 \mid n \in \mathbb{N}\}$ would be finite contradicting the DTIME(2^n)-bi-immunity of Q_1. Clearly $Q_0 \in$ P. Let $\mathbb{A}_0$ be an algorithm deciding Q_0 in polynomial time and let $\mathbb{B}$ be the algorithm deciding Q_0 by first simulating $\mathbb{A}_0$, and then making an appropriate number of dummy steps such that for some $e \geq 1$ and all $y \in \Sigma^*$

$$t_{\mathbb{B}}(y) = 2^{e\cdot(\log |y|)^2}. \tag{7}$$

Then $\mathbb{A}_0$ witnesses that $\mathbb{B}$ is not almost optimal.

We finish our proof by showing that for every algorithm $\mathbb{A}$ deciding Q_0 such that for some $e \geq 1$ and all $y \in \Sigma^*$

$$t_{\mathbb{A}}(y) \leq 2^{e\cdot(\log |y|)^2}$$

has no hard sequences. Towards a contradiction assume $\mathbb{A}$ has a hard sequence $(x_s)_{s\in\mathbb{N}}$. We set

$$L_0 := \{1^n \mid \text{for some } s \in \mathbb{N},\ |x_s| = n^d \text{ and } x_s \text{ determines an accepting run of } \mathbb{M} \text{ on } 1^n\}.$$

Clearly, $L_0 \subseteq Q_1$. We choose a polynomial time algorithm $\mathbb{G}$ computing the function $1^s \mapsto x_s$. The following algorithm $\mathbb{C}$ accepts L_0.

```
ℂ      // y ∈ Σ*
1. n ← |y|
2. if y ≠ 1^n then reject
3. ℓ ← 0
4. for s = 0 to ℓ
5.         simulate the (ℓ − s)th step of 𝔾 on 1^s
6.         if the simulation outputs x with |x| = n^d then accept
7. ℓ ← ℓ + 1
8. goto 3.
```

By (7) we can apply Lemma 7.7 to $\mathbb{A}$ and get a sequence $s_0 < s_1 < s_2 < \ldots$. For $i \in \mathbb{N}$ we let

$$n_i := \sqrt[d]{|x_{s_i}|}. \tag{8}$$

Hence, x_{s_i} is an accepting run of $\mathbb{M}$ on input 1^{n_i}. We show that

$$t_{\mathbb{C}}(1^{n_i}) = 2^{o((\log n_i)^2)}. \tag{9}$$

In fact, as $\mathbb{G}$ runs in polynomial time, we have $|x_{s_i}| \leq s_i^{O(1)}$, and by (8) therefore, $n_i \leq s_i^{O(1)}$. Now one easily sees that $\mathbb{C}$ accepts 1^{n_i} in time polynomial in s_i, too. By Lemma 7.7

$$s_i = 2^{o\left((\log |x_{s_i}|)^2\right)}.$$

Thus (8) implies that

$$s_i = 2^{o\left((\log n_i)^2\right)}.$$

Hence, we get (9).

Finally, we consider the algorithm $\mathbb{C}^*$ that on input y simulates $\mathbb{C}$ for $2^{|y|}$ steps and accepts if the simulation accepts. By (9), $\mathbb{C}^*$ accepts an infinite subset of L_0. As $L_0 \subseteq Q_1$, this contradicts the DTIME(2^n)-bi-immunity of Q_1.

(c) Let Q_0 and $\mathbb{B}$ be as in part (a). We leave it to the reader to show that the following proof system $\mathbb{P}$ for Q_0 is not optimal but has no hard sequence. For $w \in \Sigma^*$ let

$$\mathbb{P}(w) := x, \quad \text{if } w \text{ is a computation of } \mathbb{B} \text{ accepting } x$$

and $\mathbb{P}(w) := z_0$ for some fixed $z_0 \in Q_0$ otherwise. □

PROOF OF THEOREM 7.5. Let Q be a problem with a padding function *pad* and with an almost optimal algorithm $\mathbb{O}$. With Proposition 7.6 (a) choose a $Q_0 \in$ P and an algorithm $\mathbb{B}$ deciding Q_0 which is not almost optimal but has no hard sequences. Fix $z_0 \in Q$ and let $\mathbb{A}$ be the algorithm deciding Q that on input x first checks in polynomial time whether $x \in \{pad(z_0, y) \mid y \in Q_0\}$ (using the properties of the padding function and a polynomial time algorithm deciding Q_0); if $x = pad(z_0, y)$ with $y \in Q_0$, it simulates $\mathbb{B}$ on y and accepts; otherwise it simulates $\mathbb{O}$ on x and answers accordingly.

Clearly, $\mathbb{A}$ is not almost optimal as it can be speeded up on the set $\{pad(z_0, y) \mid y \in Q_0\}$, a hard set of $\mathbb{A}$. By contradiction, assume $(x_s)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{A}$ and let $y_0 \in Q_0$. For $s \geq 1$ we set

$$y_s := \begin{cases} y, & \text{if } y \in Q_0 \text{ and } x_s = pad(z_0, y) \\ y_{s-1}, & \text{otherwise} \end{cases}$$

and

$$z_s := \begin{cases} x_s, & \text{if } x_s \notin \{pad(z_0, y) \mid y \in Q_0\} \\ z_{s-1}, & \text{otherwise.} \end{cases}$$

Then either $(y_s)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{B}$ or $(z_s)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{O}$, in both cases a contradiction. □

The following example shows that the padding hypothesis in Theorem 7.5 cannot be dropped.

Example 7.8. Let $Q := \{1^n \mid n \in \mathbb{N}\}$. As $Q \in$ P, it has an almost optimal algorithm. However, the set Q itself is a hard set and $(1^s)_{s\in\mathbb{N}}$ a hard sequence for every non-optimal (that is, for every superpolynomial) algorithm deciding Q. ⊣

COROLLARY 7.9. *If the Measure Hypothesis holds, then the following are equivalent for $t \geq 1$:*

(i) No Π_t^p-complete problem has an almost optimal algorithm.
(ii) Every non-almost optimal algorithm deciding a Π_t^p-complete problem has hard sequences.

PROOF. We already know that (i) implies (ii) by Theorem 4.3 (a). Assume (ii) and by contradiction, suppose that Q is a Π_t^p-complete problem with an almost optimal algorithm. By Corollary 4.9, we may assume that Q has padding. Then, by the previous theorem, there is a non-almost optimal algorithm deciding Q without hard sequences, contradicting (ii). □

THEOREM 7.10. *If the Measure Hypothesis holds, there is a problem which has hard sets for algorithms (and hence has no almost optimal algorithm) but has algorithms without hard sequences.*

PROOF. Let $Q_0 \in \mathrm{P}$ be a problem with the properties stated in Proposition 7.6. We fix an effective enumeration

$$\mathbb{A}_0, \mathbb{A}_1, \ldots, \tag{10}$$

of all algorithms such that there is a universal algorithm $\mathbb{U}$ which on every input $\langle 1^i, x\rangle$ simulates the algorithm $\mathbb{A}_i$ on input $\langle 1^i, x\rangle$ in such a way that

$$t_{\mathbb{U}}\big(\langle 1^i, x\rangle\big) \leq (i+1) \cdot t_{\mathbb{A}_i}(\langle i, x\rangle)^2. \tag{11}$$

For every $i \in \mathbb{N}$ we let

$$S_i := \Big\{ \langle 1^i, x\rangle \;\Big|\; x \in Q_0 \text{ and } \mathbb{A}_i \text{ does } \textit{not} \text{ accept } \langle 1^i, x\rangle \text{ in } \leq 2^{(\log |x|)^2} \text{ steps} \Big\}. \tag{12}$$

Finally, we set

$$Q := \bigcup_{i \in \mathbb{N}} S_i.$$

and show that Q is a problem with the properties mentioned in the theorem.

Claim 1. Let $k \in \mathbb{N}$. If $\mathbb{A}_k$ $\big($see (10)$\big)$ decides Q, then $S_k = \{\langle 1^k, x\rangle \mid x \in Q_0\}$.

Proof of Claim 1. Otherwise, there exists an $x_0 \in Q_0$ with $\langle 1^k, x_0\rangle \notin S_k$. It follows that

$$\begin{aligned}
x_0 \in Q_0 \text{ with } \langle 1^k, x_0\rangle \notin S_k &\implies \mathbb{A}_k \text{ accepts } \langle 1^k, x_0\rangle \text{ in } \leq 2^{(\log |x|)^2} \text{ steps} \qquad \text{(by (12))} \\
&\implies \mathbb{A}_k \text{ accepts } \langle 1^k, x_0\rangle \\
&\implies \langle 1^k, x_0\rangle \in Q \qquad\qquad \text{(as } \mathbb{A}_k \text{ decides } Q\text{)} \\
&\implies \langle 1^k, x_0\rangle \in S_k \qquad\quad \text{(since all } S_i\text{'s are disjoint).}
\end{aligned}$$

This is a contraction. ⊣

Claim 2. Q has hard sets for algorithms.

Proof of Claim 2. Assume that $\mathbb{A}_k$ decides Q. By Claim 1, $S_k = \{\langle 1^k, x\rangle \mid x \in Q_0\}$ and by (12) for every $x \in Q_0$,

$$t_{\mathbb{A}_k}\big(\langle 1^k, x\rangle\big) > 2^{(\log |x|)^2}.$$

As $Q_0 \in \mathrm{P}$, thus S_k is a hard set for $\mathbb{A}_k$. ⊣

Claim 3. For all sufficiently large $d \in \mathbb{N}$ there is an algorithm $\mathbb{Q}_d$ deciding Q such that

$$t_{\mathbb{Q}_d}\big(\langle 1^i, x\rangle\big) = (i+1) \cdot 2^{d \cdot (\log |x|)^2}$$

for every $i \in \mathbb{N}$ and $x \in \Sigma^*$.

Proof of Claim 3. By (11) and (12) as $Q_0 \in \mathrm{P}$. ⊣

Now we choose a sufficiently large $d \in \mathbb{N}$ and consider the algorithm $\mathbb{Q}_d$ of Claim 3. Assume that $\mathbb{Q}_d$ has a hard sequence

$$\big(\langle 1^{i_s}, x_s\rangle\big)_{s \in \mathbb{N}}.$$

By (12) every x_s is in Q_0 and by hardness,

$$t_{\mathbb{Q}_d}\big(\langle 1^{i_s}, x_s\rangle\big) = (i_s+1) \cdot 2^{d \cdot (\log |x_s|)^2}$$

is superpolynomial in s. Since the mapping $1^s \mapsto \langle 1^{i_s}, x_s\rangle$ is computable in polynomial time, we have $|i_s| \leq |s|^{O(1)}$. Therefore,

$$2^{d \cdot (\log |x_s|)^2} \text{ is superpolynomial in } s. \tag{13}$$

As Q_0 is decidable in polynomial time and d is sufficiently large, we have by Claim 3 an algorithm $\mathbb{A}$ deciding Q_0 such that $t_{\mathbb{A}}(x) = 2^{d\cdot(\log|x|)^2}$ on every instance $x \in \Sigma^*$. Then (13) implies that $(x_s)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{A}$, which contradicts Proposition 7.6 (b). □

8. Getting hard sequences in an effective way

We have mentioned in the Introduction that Stockmeyer [1974] has shown that for every EXP-hard problem Q there is a polynomial time procedure assigning to every algorithm deciding Q a hard sequence. Based on his proof, we derive a "nondeterministic" version.

THEOREM 8.1. *Let Q be a* coNEXP*-hard problem. Then there is a polynomial time computable function $g : \Sigma^* \times \{1\}^* \to \Sigma^*$ such that for every nondeterministic algorithm $\mathbb{A}$ accepting Q the sequence $\big(g(\mathbb{A}, 1^s)\big)_{s\in\mathbb{N}}$ is hard for $\mathbb{A}$.*

PROOF. Consider the problem

Q_0	
Instance:	A nondeterministic algorithm $\mathbb{A}$.
Question:	Is it true that $\mathbb{A}$ does not accept $\mathbb{A}$ in at most $2^{\lvert\mathbb{A}\rvert}$ steps?

Claim 1. If $\mathbb{B}$ is a nondeterministic algorithm accepting Q_0, then $\mathbb{B} \in Q_0$ and therefore, $t_{\mathbb{B}}(\mathbb{B}) > 2^{|\mathbb{B}|}$.

Proof of Claim 1. Assume that $\mathbb{B} \notin Q_0$. Therefore, $\mathbb{B}$ does not accept $\mathbb{B}$. Then, by the definition of Q_0, we have $\mathbb{B} \in Q_0$, a contradiction. ⊣

To every nondeterministic algorithm $\mathbb{A}$ and every $s \in \mathbb{N}$ we can assign in time polynomial in $\mathbb{A}$ and s a nondeterministic algorithm $\mathbb{A}_s$ with

$$|\mathbb{A}_s| \geq s, \quad L(\mathbb{A}_s) = L(\mathbb{A}), \quad \text{and} \quad t_{\mathbb{A}_s} = t_{\mathbb{A}} \tag{14}$$

(say, by adding s new "dummy" states).

Claim 2. If $\mathbb{A}$ is a nondeterministic algorithm accepting Q_0, then $(\mathbb{A}_s)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{A}$.

Proof of Claim 2. It suffices to verify for all $s \in \mathbb{N}$,

$$\mathbb{A}_s \in Q_0 \tag{15}$$

$$t_{\mathbb{A}}(\mathbb{A}_s) > 2^s. \tag{16}$$

By (14) we know that $L(\mathbb{A}_s) = L(\mathbb{A})$. Hence, (15) holds by Claim 1, which also shows the first inequality in

$$t_{\mathbb{A}}(\mathbb{A}_s) = t_{\mathbb{A}_s}(\mathbb{A}_s) > 2^{|\mathbb{A}_s|} \geq 2^s,$$

the second one and the equality holding by (14). ⊣

Now let Q be coNEXP-hard. Since $Q_0 \in$ coNEXP there is a polynomial time reduction $\mathbb{S}$ from Q_0 to Q. Again, for a nondeterministic algorithm $\mathbb{A}$ let $\mathbb{A} \circ \mathbb{S}$ be the nondeterministic algorithm that on input $x \in \Sigma^*$ first runs $\mathbb{S}$ on x and then runs $\mathbb{A}$ on $\mathbb{S}(x)$.

For a nondeterministic algorithm $\mathbb{A}$ and $s \in \mathbb{N}$ we define

$$g(\mathbb{A}, 1^s) := \mathbb{S}((\mathbb{A} \circ \mathbb{S})_s).$$

Clearly, g is polynomial time computable. If $\mathbb{A}$ decides Q, then $\mathbb{A} \circ \mathbb{S}$ decides Q_0; therefore, $((\mathbb{A} \circ \mathbb{S})_s)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{A} \circ \mathbb{S}$ by Claim 2. Hence, $\big(g(\mathbb{A}, 1^s)\big)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{A}$ by Lemma 3.4. □

ACKNOWLEDGMENT

We thank the anonymous referees for their detailed comments.

REFERENCES

S. Ben-David and A. Gringauze. 1998. On the existence of propositional proof systems and oracle-relativized propositional logic. *Electronic Colloquium on Computational Complexity (ECCC), TR98-021* (1998).

L. Berman. 1976. On the structure of complete sets: almost everywhere complexity and infinitely often speedup. In *Proceedings of the 17th IEEE Symposium on Foundations of Computer Science (FOCS'76)*. 76–80.

L. Berman and J. Hartmanis. 1977. On isomorphisms and density of NP and other complete sets. *SIAM J. Comput.* 6, 2 (1977), 305–322.

O. Beyersdorff and Z. Sadowski. 2011. Do there exist complete sets for promise classes? *Mathematical Logic Quarterly* 57, 6 (2011), 535–550.

H. Buhrman, S. A. Fenner, and L. Fortnow. 1997. Results on resource-bounded measure. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming, (ICALP'97) (Lecture Notes in Computer Science)*, Vol. 1256. Springer, 188–194.

Y. Chen and J. Flum. 2010. On p-optimal proof systems and logics for PTIME. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP'10, Track B) (Lecture Notes in Computer Science 6199)*, Vol. 6199. Springer, 321–332.

Y. Chen and J. Flum. 2011. Listings and logics. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS'11)*. IEEE Computer Society, 165–174.

Y. Chen and J. Flum. 2014. On optimal inverters. *Bulletin of Symbolic Logic* (2014). To appear.

D. Gutfreund, R. Shaltiel, and A. Ta-Shma. 2007. If NP languages are hard on the worst-Case, then it is easy to find their hard instances. *Computational Complexity* 16, 4 (2007), 412–441.

K-I Ko and D. J. Moore. 1981. Completeness, approximation and density. *SIAM J. Comput.* 10, 4 (1981), 787–796.

J. Köbler and J. Messner. 1998. Complete problems for promise classes by optimal proof systems for test sets. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity (CCC'98)*. Springer, 132–140.

J. Köbler, J. Messner, and J. Torán. 2003. Optimal proof systems imply complete sets for promise classes. *Information and Computation* 184, 1 (2003), 71–92.

J. Krajíček. 1995. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*. Cambridge University Press.

J. Krajíček. 2012. A note on SAT algorithms and proof complexity. *Inf. Process. Lett.* 112, 12 (2012), 490–493.

J. Krajíček. 2014. On the computational complexity of finding hard tautologies. *Bulletin of the London Mathematical Society* 46, 1 (2014), 111–125.

J. Krajíček and P. Pudlák. 1989. Propositional proof systems, the consistency of first order theories and the complexity of computations. *The Journal of Symbolic Logic* 54, 3 (1989), 1063–1079.

L. Levin. 1973. Universal sequential search problems. *Problems of Information Transmission* 9, 3 (1973), 265–266.

J. H. Lutz. 1997a. Observations on measure and lowness for Δ_2^p. *Theory Comput. Syst.* 30, 4 (1997), 429–442.

J. H. Lutz. 1997b. The quantitative structure of exponential time. *Complexity Theory Retrospective II* (1997), 225–254.

E. Mayordomo. 1994. Almost every set in exponential time is P-bi-Immune. *Theoretical Computer Science* 136, 2 (1994), 487–506.

J. Messner. 1999. On optimal algorithms and optimal proof systems. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science, (STACS'99) (Lecture Notes in Computer Science)*. Springer, 541–550.

J. Messner. 2000. *On the Simulation Order of Proof Systems*. Ph.D. Dissertation. University of Ulm.

H. Monroe. 2011. Speedup for natural problems and noncomputability. *Theor. Comput. Sci.* 412, 4-5 (2011), 478–481.

Z. Sadowski. 2002. On an optimal propositional proof system and the structure of easy subsets. *Theoretical Computer Science* 288, 1 (2002), 181–193.

Z. Sadowski. 2007. Optimal proof systems, optimal acceptors and recursive presentability. *Fundamenta. Informaticae* 79, 1-2 (2007), 169–185.

C. P. Schnorr. 1976. Optimal algorithms for self-reducible problems. In *Proceedings of the 3rd International Colloquium on Automata, Languages and Programming, (ICALP'76)*. Edinburgh University Press, 322–337.

L. Stockmeyer. 1974. *The Complexity of Decision Problems in Automata Theory*. Ph.D. Dissertation. MIT.

O. V. Verbitsky. 1979. Optimal algorithms for coNP-sets and the problem EXP=NEXP. *Matematicheskie zametki* 50, 2 (1979), 37–46.