Strong inapproximability of the shortest reset word*

Paweł Gawrychowski[†1] and Damian Straszak[‡2]

[1]Institute of Informatics, University of Warsaw, Poland
[2]EPFL, Switzerland

September 30, 2021

Abstract

The Černý conjecture states that every n-state synchronizing automaton has a reset word of length at most $(n-1)^2$. We study the hardness of finding short reset words. It is known that the exact version of the problem, i.e., finding the shortest reset word, is NP-hard and coNP-hard, and complete for the DP class, and that approximating the length of the shortest reset word within a factor of $O(\log n)$ is NP-hard [Gerbush and Heeringa, CIAA'10], even for the binary alphabet [Berlinkov, DLT'13]. We significantly improve on these results by showing that, for every $\varepsilon > 0$, it is NP-hard to approximate the length of the shortest reset word within a factor of $n^{1-\varepsilon}$. This is essentially tight since a simple $O(n)$-approximation algorithm exists.

1 Introduction

Let $A = (Q, \Sigma, \delta)$ be a deterministic finite automaton. We say that $w \in \Sigma^*$ resets (or synchronizes) A if $|\delta(Q, w)| = 1$, meaning that the state of A after reading w does not depend on the choice of the starting state. If at least one such w exists, A is called synchronizing. In 1964 Černý conjectured that every synchronizing n-state automaton admits a reset word of length $(n-1)^2$. The problem remains open as of today. It is known that an $\frac{n^3-n}{6}$ bound holds [20] and that there are automata requiring words of length $(n-1)^2$. The conjecture was proved for various special classes of automata [1, 8, 12, 17, 21, 22]. For a thorough discussion of the Černý conjecture see [23].

Computational problems related to synchronizing automata were also studied. It is known that finding the shortest reset word is both NP-hard and coNP-hard [8]. Moreover, it was shown to be DP-complete [19].

In this paper, rather than looking at the exact version, we consider the problem of finding *s*hort reset words for automata, or to put it differently, the question of approximating the length of the shortest reset word. For a given n-state synchronizing automaton, we want to find a reset word which is at most α times longer than the shortest one, where α can be either a constant or a function of n. There is a simple polynomial time algorithm achieving $O(n)$-approximation [11].

*Supported by the *NCN* grant 2011/01/D/ST6/07164.

†Currently holding a post-doctoral position at Warsaw Center of Mathematics and Computer Science.

‡Part of the work was carried out while the author was a student at Institute of Computer Science, University of Wrocław, Poland.

1.1 Previous work and our results.

Berlinkov showed that finding an $O(1)$-approximation is NP-hard by giving a combinatorial reduction from SAT [6]. Later, Gerbush and Heeringa [11] used the $\log n-$approximation hardness of SetCover [9] to prove that $O(\log n)$-approximation of the shortest reset word is NP-hard. Finally, Berlinkov [7] extended their result to hold even for the binary alphabet, and conjectured that a polynomial time $O(\log n)$-approximation algorithm exists. We refute the conjecture by showing that, for every constant $\varepsilon > 0$, no polynomial time $n^{1-\varepsilon}-$approximation is possible unless P = NP. This together with the simple $O(n)-$approximation algorithm gives a sharp threshold result for the shortest reset word problem.

The mathematical motivation and its algorithmic version considered in this paper are closely connected, although not in a very formal sense. All known methods for proving bounds on the length of the shortest reset word are actually based on explicitly computing a short reset word (in polynomial time). The best known method constructs a reset word of length $\frac{n^3-n}{6}$, while the (most likely) true upper bound is just $(n-1)^2$, which is smaller by a factor of roughly $\frac{n}{6}$. Similarly, the best known (polynomial time) approximation algorithm achieves $O(n)$-approximation. Hence it is reasonable to believe that an $o(n)$-approximation algorithm could be used to significantly improve the upper bound on the length of the shortest synchronizing word to $o(n^3)$. In this context, our result suggests that improving the bound on the length of the shortest synchronizing word to $O(n^{3-\varepsilon})$ requires non-constructive tools.

The main insight is to start with the PCP theorem. We recall the notion of constraint satisfaction problems, and using the result of Håstad and Zuckerman provide a class of hard instances of such problems with specific properties tailored to our particular application. Then, we show how to appropriately translate such a problem into a synchronizing automaton.

1.2 Organization of the paper.

We provide the necessary definitions and the background on finite automata in the preliminaries. We also introduce the notion of probabilistically checkable proofs and state the PCP theorem, then define constraint satisfaction problems and their basic parameters.

In the next three sections we gradually move towards the main result. In Section 3 we prove that $(2-\varepsilon)$-approximation of the shortest reset word is NP-hard. In Section 4 we strengthen this by showing that, for a small fixed $\varepsilon > 0$, $n^{\varepsilon}-$approximation is also NP-hard. Finally, in Section 5, we provide more background on probabilistically checkable proofs and free bit complexity, and prove that, for every $\varepsilon > 0$, even $n^{1-\varepsilon}$-approximation is NP-hard. Even though the final result subsumes Sections 3 and 4, this allows us to gradually introduce the new components.

In the Appendix we sketch how deduce the subconstant error PCP theorem from the classical version.

2 Preliminaries

2.1 DFA.

A deterministic finite automaton (in short, an automaton) is a triple $A = (Q, \Sigma, \delta)$, where Q is a nonempty finite set of states, Σ is a nonempty finite alphabet, and δ is a transition function $\delta : Q \times \Sigma \to Q$. In the usual definition one includes additionally a starting state and a set of

accepting states, which are irrelevant in our setting. Equivalently, we can treat an automaton as a collection of $|\Sigma|$ transformations of a finite set Q. We consider words over Σ, which are finite sequences of letters (elements of Σ). The empty word is denoted by ε, the set of words of length n by Σ^n, and the set of all words by Σ^*. For $w \in \Sigma^*$, $|w|$ stands for the length of w and w_i is the i-th letter of w, for any $i \in \{1, 2, \ldots, |w|\}$.

If $A = (Q, \Sigma, \delta)$ is an automaton, then we naturally extend δ from single letters to whole words by defining $\delta(q, \varepsilon) = q$ and $\delta(q, wa) = \delta(\delta(q, w), a)$. For $P \subseteq Q$ we denote by $\delta(P, w)$ the image of P under $\delta(\cdot, w)$.

2.2 Synchronizing Automata.

An automaton $A = (Q, \Sigma, \delta)$ is synchronizing if there exists a word w for which $|\delta(Q, w)| = 1$. Such w is then called a synchronizing (or reset) word and the length of a shortest such word is denoted by $\mathrm{Syn}(A)$. One can check if an automaton is synchronizing in polynomial time by verifying that every pair of states can be synchronized to a single state.

SynAppx(Σ, α)
Given a synchronizing n-state automaton A over an alphabet Σ, find a word of length at most $\alpha \cdot \mathrm{Syn}(A)$ synchronizing A. Here both α and $|\Sigma|$ can be a function of n.

We are interested in solving SynAppx(Σ, α) in polynomial time, with α as small as possible.

2.3 $O(n)-$Approximation.

It is known [11] that for any fixed k the problem SynAppx$(\Sigma, \frac{n}{k})$ can be solved in $O(n^{k+1})$ time (we assume that Σ is of constant size). The basic idea is that, for a given automaton $A = (Q, \Sigma, \delta)$, we construct a graph G with the vertex set $V = \{S \subseteq Q : |S| \leq k+1\}$ and a directed edge $S \to \delta(S, a)$ labeled with a for every $S \in V$ and $a \in \Sigma$. Then for a given $S \in V$ the shortest word synchronizing S to a single state corresponds to the shortest path connecting S to some singleton set $\{q\} \in V$. Each such word is of length at most $\mathrm{Syn}(A)$. The algorithm works in $\lceil \frac{n}{k} \rceil$ phases. We start with the full set of states to reset $R := Q$ and with an empty word $w := \varepsilon$, and in each phase we will decrease the size of R by k, while assuring that $\delta(Q, w) = R$. In a single phase we take any subset S of R of size $k+1$ (if possible) and find the shortest word w' resetting S to a single state (note that $|w'| \leq \mathrm{Syn}(A)$). We set $w := ww'$, $R := \delta(R, w')$ and continue. One can easily see that in the end we obtain a synchronizing word w of length at most $\lceil \frac{n}{k} \rceil \cdot \mathrm{Syn}(A)$.

2.4 Cubic Bound for the Černý Conjecture.

Setting $k = 1$ in the reasoning from Section 2.3, we obtain an upper bound for $\mathrm{Syn}(A)$. This follows from the fact that the graph G has $O(n^2)$ vertices and consequently every shortest path has length $O(n^2)$. In the end we have $\mathrm{Syn}(A) \leq |w| = (n-1) \cdot O(n^2) = O(n^3)$. In contrast, the famous Černý conjecture states that for every synchronizing automaton A it holds $\mathrm{Syn}(A) \leq (n-1)^2$. Interestingly, the best bound known up to now is $\frac{n^3-n}{6}$ [20], which is also cubic. Any $o(n^3)$ upper bound would be a very interesting result for this problem.

2.5 Alphabet Size.

In the general case, the size of the alphabet can be arbitrary. Our construction will use $\Sigma = \{0, 1, 2\}$, which can be then reduced to the binary alphabet using the method of Berlinkov [7]. It is based on encoding every letter in binary and adding some intermediate states. For completeness we state the appropriate lemma and sketch its proof.

Lemma 2.1 (Lemma 7 of [7]) *Suppose* $\textsc{SynAppx}(\{0,1\}, n^\alpha)$ *can be solved in polynomial time for some* $\alpha \in (0, 1)$*, then so can be* $\textsc{SynAppx}(\Sigma, O(n^\alpha))$ *for any* Σ *of constant size.*

Proof: As shown in Lemma 7 of [7], given an n-state automaton A over an alphabet Σ, one can efficiently construct an automaton B on $\tilde{n} := 2|\Sigma|n$ states over the binary alphabet, such that $\mathrm{Syn}(A)t \leq \mathrm{Syn}(B) \leq t(1 + \mathrm{Syn}(A))$, where $t = \lceil \log_2 |\Sigma| \rceil + 1$. Then, if we can approximate $\mathrm{Syn}(B)$ within a factor of $\tilde{n}^\alpha$, we can compute in polynomial time an x such that $\mathrm{Syn}(B) \leq x \leq \mathrm{Syn}(B)\tilde{n}^\alpha$. Then $t \cdot \mathrm{Syn}(A) \leq x$ and $x \leq t(1+\mathrm{Syn}(A))\tilde{n}^\alpha \leq 2t\mathrm{Syn}(A)\tilde{n}^\alpha$. Therefore, $\mathrm{Syn}(A) \leq \frac{x}{t} \leq 2\mathrm{Syn}(A)\tilde{n}^\alpha$, so $\frac{x}{t}$ approximates $\mathrm{Syn}(A)$ within a factor of $2\tilde{n}^\alpha = O(n^\alpha)$. □

2.6 PCP Theorems.

We briefly introduce the notion of *Probabilistically Checkable Proofs* (PCPs). For a comprehensive treatment refer to [5] or [2].

A polynomial-time probabilistic machine V is called a $(p(n), r(n), q(n))$-PCP verifier for a language $L \subseteq \{0,1\}^*$ if:

- for an input x of length n, given random access to a "proof" $\pi \in \{0,1\}^*$, V uses at most $r(n)$ random bits, accesses at most $q(n)$ locations of π, and outputs 0 or 1 (meaning "reject" or "accept" respectively),
- if $x \in L$ then there is a proof π, such that $\Pr[V(x, \pi) = 1] = 1$,
- if $x \notin L$ then for every proof π, $\Pr[V(x, \pi) = 1] \leq p(n)$.

We consider only *nonadaptive* verifiers, meaning that the subsequently accessed locations depend only on the input and the random bits, and not on the previous answers, hence we can think that V specifies at most $q(n)$ locations and then receives a sequence of bits encoding all the answers. $p(n)$ from the above definition is often called the *soundness* or the *error probability*. In some cases, also the *proof length* is important. For a fixed input x of length n the proof length is the total number of distinct locations queried by V over all possible $2^{r(n)}$ runs of V (on different sequences of $r(n)$ random bits). The proof length is always at most $q(n) \cdot 2^{r(n)}$, and such a bound is typically sufficient for applications, however in some cases we desire PCP-verifiers with smaller proof length.

The set of languages for which there exists a (p, r, q)-PCP verifier is denoted by $\mathrm{PCP}_p[r, q]$.

Theorem 2.2 (PCP Theorem [3, 4]) $\mathrm{NP} = \mathrm{PCP}_{1/2}[O(\log n), O(1)]$.

2.7 Constraint Satisfaction Problems.

We consider *Constraint Satisfaction Problems* (CSPs) over boolean variables. An instance of a general CSP over N boolean variables $x_1, x_2, \ldots, x_N$ is a collection of M boolean constraints $\phi = (C_1, C_2, \ldots, C_M)$, where a boolean constraint is just a function $C : \{0,1\}^N \to \{0,1\}$. A boolean assignment $v : \{0,1\}^N \to \{0,1\}$ satisfies a constraint C if $C(v) = 1$, and ϕ is satisfiable if there exists an assignment $v : \{0,1\}^N \to \{0,1\}$ such that $C_i(v) = 1$ for all $i = 1, 2, \ldots, M$. We define $\mathrm{Val}(\phi)$ to be the maximum fraction of constraints in ϕ which can be satisfied by a single assignment. In particular $\mathrm{Val}(\phi) = 1$ iff ϕ is satisfiable.

We consider computational properties of CSPs. We are mainly interested in CSPs, where every N-variable constraint has description of size $\mathrm{poly}(N)$ (as opposed to the naive representation using 2^N bits). A natural class of such CSPs are CNF-formulas, where every constraint is a clause being a disjunction of N literals, thus described in $O(N)$ space. Another important class are qCSPs, where every constraint *depends* only on at most q variables. Such a constraint can be described using $\mathrm{poly}(N, 2^q)$ space, which is polynomial whenever $q = O(\log N)$. Formally, we say that a clause C depends on variable x_i if there exists an assignment $v \in \{0,1\}^N$ such that $C(v)$ changes after modifying the value of x_i and keeping the remaining variables intact. We define V_C to be the set of all such variables. It is easy to see that $\phi(C)$ is determined as soon as we assign the values to all variables in V_C. Finally, the following class will be of interest to us.

Definition 2.3 *Let C be an N-variable constraint and let V_C be the set of variables on which C depends. Consider all $2^{|V_C|}$ assignments $\{0,1\}^{V_C} \to \{0,1\}$. If only K of such assignments satisfy C, we write* $\mathrm{Fsat}(C) \leq K$. $\mathrm{Fsat}(\phi) \leq K$ *if* $\mathrm{Fsat}(C) \leq K$ *for every constraint C in ϕ.*

According to the above definition, if ϕ is a qCSP instance then $\mathrm{Fsat}(\phi) \leq 2^q$. A constraint C such that $\mathrm{Fsat}(C) \leq K$ can be described by its set V_C and a list of at most K assignments to the variables in V_C satisfying C. Thus the description is polynomial in N and K. We will consider CSPs ϕ with $\mathrm{Fsat}(\phi) \leq \mathrm{poly}(N)$ and always assume that they are represented as just described.

3 Simple Hardness Result

We start with a simple introductory result, which is that for any fixed constant $\varepsilon > 0$, it is NP-hard to find for a given n-state synchronizing automaton A a synchronizing word w such that $|w| \leq (2-\varepsilon)\cdot\mathrm{Syn}(A)$. The final goal is to prove a much strong result, but the basic construction presented in this section is the core idea further developed in the subsequent sections. The construction is not the simplest possible, nor the most efficient in the number of states of the resulting automaton, but it provides good intuitions for the further proofs. For a simpler construction in this spirit see [6].

Theorem 3.1 *For every constant $\varepsilon > 0$,* $\textsc{SynAppx}(\{0,1,2\}, 2-\varepsilon)$ *is not solvable in polynomial time, unless* $\mathrm{P} = \mathrm{NP}$.

3.1 Idea.

Fix $\varepsilon > 0$. We will reduce 3-SAT to our problem, that is, show that an algorithm solving $\textsc{SynAppx}(\{0,1,2\}, 2-\varepsilon)$ can be used to decide satisfiability of 3-CNF formulas. This will stem from the following reduction. For a given N-variable 3-CNF formula ϕ consisting of M clauses we can build in polynomial time a synchronizing automaton A_ϕ such that:

1. if ϕ is satisfiable then $\mathrm{Syn}(A_\phi) \approx N$,
2. if ϕ is not satisfiable then $\mathrm{Syn}(A_\phi) \geq 2N$.

This implies Theorem 3.1, since applying an $(2-\varepsilon)$-approximation algorithm to A_ϕ allows us to find out whether ϕ is satisfiable or not.

3.2 Construction.

Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_M$ be a 3-CNF formula with N variables $x_1, x_2, \ldots, x_N$ and M clauses. We want to build an automaton $A_\phi = (\{0,1,2\}, Q, \delta)$ with properties as described above. A_ϕ consists of M gadgets, one for each clause in ϕ, and a single sink state s. All letters leave s intact, that is, $\delta(s,0) = \delta(s,1) = \delta(s,2) = s$. We describe now a gadget for a fixed clause C.

The gadget built for a clause C can be essentially seen as a tree with 8 leaves. Each leaf corresponds to one of the assignments to 3 variables appearing in C. First we introduce the uncompressed version of the gadget. Take all possible 2^N assignments and form a full binary tree of height N. Every edge in the tree is directed from a parent to its child and has a label from $\{0,1\}$. Every assignment naturally corresponds to a leaf in the tree. We could potentially use such a tree as the gadget, except that its size is exponential. We will fix this by merging isomorphic subtrees to obtain a tree of size linear in N.

Let us denote by $L_0, L_1, \ldots, L_N$ the vertices at levels $0, 1, \ldots, N$, respectively, so that $L_0 = \{r\}$, where r is the root, and L_N is the set of leaves.

Suppose that the variable x_k does not occur in C. Take any vertex $v \in L_{k-1}$ and denote the subtrees rooted at its children by T_0 and T_1. It is easy to see that T_0 and T_1 are isomorphic and can be merged, so that we have two edges outgoing from v, labeled by 0 and 1, respectively, and both leading to the same vertex v', which is the root of T_0. We continue the merging until there are no more such vertices, which can be seen as "compressing" the tree.

Let us now formalize the construction. The set of vertices at level j (where $0 \leq j \leq N$) is $L_j = \{q_j^w : w \in \{0,1\}^d\}$, where d is the number of variables x_i occurring in C with $i \leq j$. For example L_0 is simply $\{q_0^\varepsilon\}$. Given q_j^w, one should think of $w \in \{0,1\}^d$ as some boolean assignment to variables $x_{j_1}, x_{j_2}, \ldots, x_{j_d}$ appearing in C (where $j_1 \leq j_2 \leq \ldots \leq j_d \leq j$). Let us now describe the edges. Take any variable x_k and a vertex $q_{k-1}^w \in L_{k-1}$, then:

- if x_k occurs in C, then q_{k-1}^w has two distinct children $\delta(q_{k-1}^w, 0) = q_k^{w0}$ and $\delta(q_{k-1}^w, 1) = q_k^{w1}$,
- if x_k does not occur in C, then q_{k-1}^w has one child $\delta(q_{k-1}^w, 0) = \delta(q_{k-1}^w, 1) = q_k^w$.

We have already defined the edges outgoing from levels $0, 1, \ldots, N-1$. This justifies the name tree-gadget. It remains to define level N, where intuitively the "synchronization" or "rejection" happens. Let q_N^w be a vertex on the last level, then:

- if w corresponds to a satisfying assignment of C then $\delta(q_N^w, 0) = \delta(q_N^w, 1) = s$,
- otherwise $\delta(q_N^w, 0) = \delta(q_N^w, 1) = q_0^\varepsilon$.

The above defined *tree-gadget* will be further denoted by T_C, and its root q_0^ε will be usually referred to as r. To complete the definition, we set $\delta(q, 2) = r$ for every $q \in T_C$. See Figure 1 for an example.

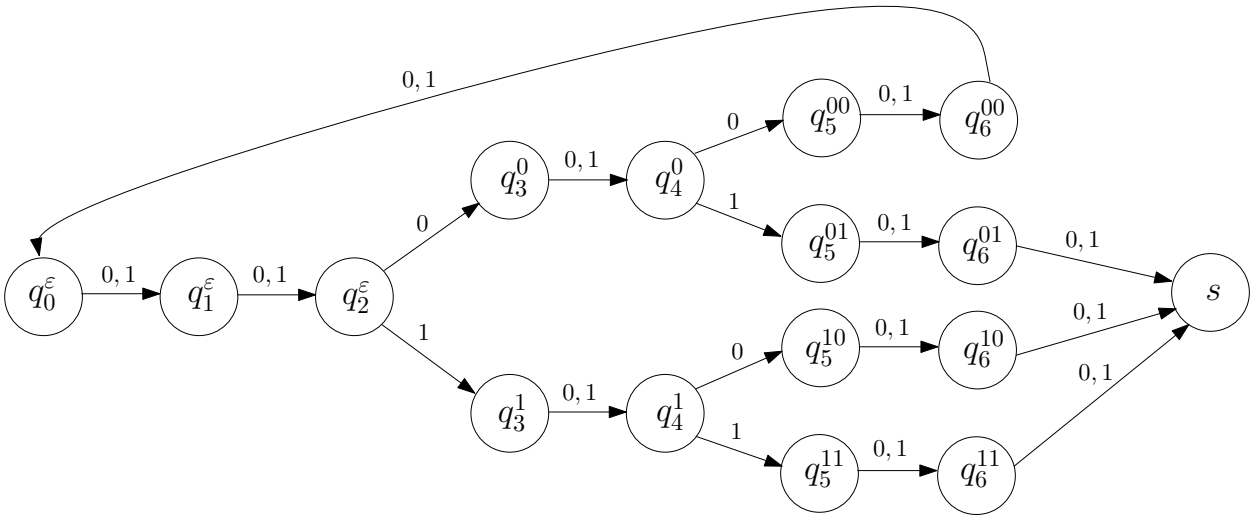

Figure 1: Tree-gadget T_C constructed for a clause $C = x_3 \vee x_5$ and $N = 6$.

The automaton A_ϕ consists of M disjoint tree-gadgets $T_{C_1}, T_{C_2}, \ldots, T_{C_M}$ and a single "sink state" s. Formally, its set of states is $Q = \sum_{i=1}^{M} T_{C_i} \cup \{s\}$ and the transition δ is defined above for every tree-gadget and the sink state s.

3.3 Properties of A_ϕ.

The following properties of A_ϕ can be established.

Proposition 3.2 *Consider a tree-gadget T_C with root r constructed for a clause C. If C depends only on variables $x_{j_1}, x_{j_2}, x_{j_3}$ then for any binary assignment $v \in \{0,1\}^N$ we have $\delta(r, v) = q_N^w$, where $w = v_{j_1} v_{j_2} v_{j_3}$.*

Proof: This follows immediately from the construction of T_C. We start in state q_0^ε. Whenever we meet a relevant variable, we concatenate the assigned bit to our "memory". Thus after reading the whole assignment, we end up with the 3 relevant bits. □

Proposition 3.2 immediately yields the following.

Corollary 3.3 *Consider a tree-gadget T_C constructed for a clause C, and let $w = vc$ be a binary word with $|v| = N$ and $c \in \{0,1\}$. If v is an assignment satisfying C then $\delta(r, w) = s$, otherwise $\delta(r, w) = r$.*

Since s is a sink state, synchronizing A_ϕ is equivalent to pushing all of its states into s. Actually, it is enough to consider how to synchronize the set $R = \{r_1, r_2, \ldots, r_M\}$, where r_i is the root of the i-th tree-gadget T_{C_i}. This is because $\delta(T_{C_i}, 2) = \{r_i\}$ for every i, hence one application of letter 2 "synchronizes" every gadget to its root and then it is enough to synchronize the roots.

It is already easy to see that A_ϕ is always synchronizing, because we can synchronize gadgets one by one. The following lemma says that in case when ϕ is satisfiable, we can synchronize A_ϕ very quickly.

Lemma 3.4 *If ϕ is satisfiable and $v \in \{0,1\}^N$ is a satisfying assignment, then the word $w = 2v0$ synchronizes A_ϕ. Therefore* $\mathrm{Syn}(A_\phi) \leq N + 2$.

Proof: Applying 2 to the states Q of A_ϕ yields $\delta(Q, 2) = R \cup \{s\}$. Then, for every $r_i \in R$, since C_i is satisfiable by v, we have by Corollary 3.3 $\delta(r_i, v0) = s$. Hence $\delta(Q, w) = \{s\}$. □

Our next goal is to show that whenever ϕ is not satisfiable, A_ϕ cannot by synchronized quickly. To this end, we prove the following.

Lemma 3.5 *Suppose there exists a binary word of length less than $2N+2$ synchronizing R to $\{s\}$. Then ϕ is satisfiable.*

Proof: First note that if for some tree-gadget T_C with root r and a word $v \in \{0,1\}^{N+1}$ we have $\delta(r, v) = r$, then for every binary word v' of length $d \leq N$ we have $\delta(r, vv') \in L_d$ (where L_d is the d-th level of the tree-gadget), in particular vv' does not push r to s. This implies that if there is a word w of length less than $2N + 2$, synchronizing A_ϕ, then there exists a word of length $N + 1$ synchronizing A_ϕ, because we can simply truncate it after the $(N + 1)$-th letter. By Corollary 3.3, such a synchronizing word of length $N + 1$ has then a prefix of length N, which is a satisfying assignment for ϕ. □

By Lemma 3.5, if there is no short binary word synchronizing R then ϕ is not satisfiable. Using letter 2 does not help at all in synchronizing R as shown below.

Lemma 3.6 *Suppose there exists a word $w \in \{0,1,2\}^*$ synchronizing R to $\{s\}$. Then there is a word $w' \in \{0,1\}^*$ of length at most $|w|$ synchronizing R to $\{s\}$.*

Proof: For convenience assume w ends with a 2. Decompose w as follows $w = v_1 2 v_2 2 \dots v_k 2$, where $v_1, v_2, \dots, v_k \in \{0,1\}^*$. Fix one part of the form $v_i 2$. If $|v_i| \leq N$, then for every $r \in R$, the empty word ε acts exactly the same on r as $v_i 2$. Hence we can replace the part $v_i 2$ by ε in w. More generally if v_i is of the form $v_i = vv'$, where v has length being a multiple of $N + 1$ and $|v'| \leq N$, then the words $v_i 2$ and v are again equivalent with respect to action on R. Thus every part $v_i 2$ can be replaced by a shorter binary word yielding a word w' with the same action on R as w. The lemma follows. □

We are now ready to prove Theorem 3.1.

Proof: [of Theorem 3.1] Fix $\varepsilon > 0$ and suppose we can solve $\textsc{SynAppx}(\{0,1,2\}, 2 - \varepsilon)$ in polynomial time. We will show that we can solve 3-SAT in polynomial time. Let ϕ be any 3-CNF formula. We construct A_ϕ and approximate its shortest reset word within a factor of $2 - \varepsilon$. By Lemma 3.4 if ϕ is satisfiable then $\mathrm{Syn}(A_\phi) \leq N + 2$, and if ϕ is not satisfiable then by Lemma 3.5 and Lemma 3.6 we have $\mathrm{Syn}(A_\phi) \geq 2N + 2$. Hence, $(2 - \varepsilon)$-approximation allows us to distinguish between those two cases in polynomial time. □

4 Hardness with Ratio n^ε

In this section we show that it is possible to achieve a stronger hardness result using essentially the same reduction, but from a different problem. The problem we reduce from is CSP with some specific parameters. Its hardness is proved by suitably amplifying the error probability in the classical PCP theorem. We provide the details below.

4.1 PCP, qCSP and Probability Amplification.

We want to obtain a hard boolean satisfaction problem, which allows us to perform more efficient reductions to SynAppx. The usual source of such problems are PCP theorems, the most basic one asserting that $\mathrm{NP} = \mathrm{PCP}_{1/2}[O(\log n), O(1)]$. By sequential repetition we can obtain verifiers erring with much lower probability, i.e., $\mathrm{NP} = \mathrm{PCP}_{\varepsilon}[O(\log n), O(1)]$ for any fixed $\varepsilon \in (0,1)$. Combining such a verifier with the construction of A_ϕ described in the previous section yields that it is NP-hard to approximate the shortest reset word within a factor of α, for any constant α. However, we aim for a stronger n^{ε}-hardness for some $\varepsilon > 0$. To this end we need to construct PCP verifiers with subconstant error.

Sequential repetition used to reduce the error probability as explained above has severe limitations. We want the error probability to be $\approx n^{-1}$. This requires $\Theta(\log n)$ repetitions, each consuming fresh $O(\log n)$ random bits, and results in a verifier with the error probability bounded by n^{-1} using $O(\log n)$ queries. The total number of used random bits is then $r = \Theta(\log^2 n)$, which is too much, since the size of the automaton polynomially depends on 2^r. Fortunately, the amount of used random bits can be reduced using the standard idea of a random walk on an expander, resulting in the following theorem. We explain the details and deduce the following theorem in Appendix A from Theorem 2.2.

Theorem 4.1 (Subconstant Error PCP) $\mathrm{NP} \subseteq \mathrm{PCP}_{1/n}[O(\log n), O(\log n)]$.

Now we can use Theorem 4.1 to prove the following.

Theorem 4.2 *There exists a polynomial time reduction f, which takes a 3-CNF n-variable formula ϕ and returns a qCSP instance $f(\phi)$ with $q = O(\log n)$, such that:*

- *if ϕ is satisfiable then* $\mathrm{Val}(f(\phi)) = 1$,
- *if ϕ is not satisfiable then* $\mathrm{Val}(f(\phi)) \leq \frac{1}{n}$.

Proof: Take the PCP verifier V for 3-SAT from Theorem 4.1. Assume it uses $r = O(\log n)$ random bits and queries the proof $q = O(\log n)$ times, for an n-variable formula ϕ. One can see that the proof length ℓ is polynomial in n (at most $q \cdot 2^r$). There will be ℓ variables in the resulting qCSP instance, one for every position in the proof. $f(\phi)$ consists of 2^r constraints, one for every possible sequence of random bits of length r. For a fixed sequence of random bits $s \in \{0,1\}^r$, we create a constraint C_s. Given an assignment $v \in \{0,1\}^l$, C_s evaluates to 1 if and only if V accepts a proof v (note that C_s depends on at most q variables). One can easily see that such a constraint satisfaction instance is satisfiable for satisfiable ϕ. If ϕ is not satisfiable, then for every proof the probability of acceptance is at most $\frac{1}{n}$. It means that for every assignment $v \in \{0,1\}^l$ at most $(\frac{1}{n})-$fraction of constraints can be satisfied by v. Finally, the reduction is polynomial time computable, because there are polynomially many constraints, each depending on $q = O(\log n)$ variables, thus one constraint can be described in $O(2^q) = \mathrm{poly}(n)$ time. □

4.2 Construction.

Let ϕ be an N-variables qCSP instance with M clauses and $q = O(\log N)$. We want to construct a synchronizing automaton A_ϕ, such that the length of its shortest reset word allows us to reconstruct $\mathrm{Val}(\phi)$ up to some error.

The construction of A_ϕ is exactly the same as the one given for 3-CNF instances in Section 3. For a q-constraint C we build a tree-gadget T_C with 2^q leaves, each corresponding to an assignment to the variables C depends on. As previously, the automaton has one sink state s and M tree-gadgets, one for every constraint. The construction still takes just polynomial time, the size of the automaton is polynomial in M, N and $2^q = \text{poly}(N)$.

4.3 Properties of A_ϕ.

Similarly as in the previous sections, the following properties of A_ϕ can be established.

Lemma 4.3 *Let ϕ be a N-variable qCSP instance. If ϕ is satisfiable and $v \in \{0,1\}^N$ is a satisfying assignment, then the word $w = 2v0$ synchronizes A_ϕ. Therefore* $\text{Syn}(A_\phi) \leq N + 2$.

For the case when ϕ is not satisfiable we need a stronger statement than the one from Lemma 3.5.

Lemma 4.4 *Let ϕ be a N-variable qCSP instance. If w synchronizes A_ϕ then $|w| \geq \frac{1}{\text{Val}(\phi)}(N+1)$.*

Proof: We prove a lower bound, thus we can focus on synchronizing a particular set of states. Let $R = \{r_1, r_2, \ldots, r_M\}$ be the set of roots of all tree-gadgets. Suppose w synchronizes R to $\{s\}$. By Lemma 3.6 we can assume w does not contain any occurrence of 2. Also, we can assume (as in the proof of Lemma 3.5) that the length of w is a multiple of $(N+1)$. If it is not then we can cut out the last $|w| \bmod (N+1)$ letters and the resulting word will still synchronize R to $\{s\}$.

Decompose w into the following parts: $w = v_1 c_1 v_2 c_2 \ldots v_k c_k$, where v_i is a binary word of length N and c_i is a single binary character, for $i = 1, 2, \ldots, k$. We claim that for every constraint C in ϕ, some assignment v_i (for $i \in \{1, 2, \ldots, k\}$) satisfies C. Here a binary string of length N is treated as a boolean assignment to the N variables. Suppose for the sake of contradiction that there is a constraint C in ϕ such that no v_i satisfies C. Suppose r is the root of the corresponding tree-gadget T_C. Using Corollary 3.3 we can reason by induction that if w' is a prefix of w of length d then $\delta(r, w') \in L_{d \bmod (N+1)}$. In particular $\delta(r, w) = r$.

Therefore we know that every constraint C in ϕ is satisfied by some v_i. However, one assignment can satisfy at most $\text{Val}(\phi)$ constraints, hence $k \geq \frac{1}{\text{Val}(\phi)}$. The lemma follows. □

Now we are ready to prove the main theorem of this section.

Theorem 4.5 *There exists a constant $\varepsilon > 0$, such that* $\textsc{SynAppx}(\{0,1,2\}, n^\varepsilon)$ *is not solvable in polynomial time, unless* $\text{P} = \text{NP}$.

Proof: We reduce 3-SAT to $\textsc{SynAppx}(\{0,1,2\}, n^\varepsilon)$, for some constant $\varepsilon > 0$. Let ϕ be an n-variable 3-CNF formula ϕ. We use Theorem 4.2 to obtain a qCSP instance $f(\phi)$ on N variables and then convert it into a G-state automaton $A_{f(\phi)}$. If ϕ is satisfiable, then by Lemma 4.3 $\text{Syn}(A_{f(\phi)}) \leq N + 2$. On the other hand, if ϕ is not satisfiable, then $\text{Val}(f(\phi)) \leq 1/n$, hence by Lemma 4.4 $\text{Syn}(A_{f(\phi)}) \geq n(N+1)$. The ratio between those two quantities is $\frac{n(N+1)}{N+2} = \Omega(n)$.

It remains to show that $n = \Omega(G^\varepsilon)$ for some constant $\varepsilon > 0$. In other words, we need to show that G is polynomial in n. This holds, because f is a polynomial time reduction, hence $N, M = \text{poly}(n)$ and the size of $A_{f(\phi)}$ is polynomial with respect to $N, M, 2^q$, but $q = O(\log N) = O(\log n)$ so $2^q = \text{poly}(n)$. □

Remark 4.6 *By keeping track of all the constants, one can obtain $n^\varepsilon-$hardness for $\varepsilon \approx 0.0095$, but this is anyway subsumed by the next section.*

5 Hardness with Ratio $n^{1-\varepsilon}$

In this section we prove the main result of the paper. It is not enough to use the reasoning from the previous section and simply optimize the constants. In fact, the strongest hardness result that we can possibly obtain by applying Theorem 4.1 is n^{ε} for some tiny constant $\varepsilon > 0$. This stems from the fact that in our reduction we require the number of queries q to be logarithmic in the size of the instance. If this is not the case, then the reduction takes superpolynomial time. However, the crucial observation is that the reduction can be modified so that we do not need the query complexity of the verifier to be logarithmic. It suffices that the free bit complexity (defined below) is logarithmic.

Theorem 5.1 *For every constant $\varepsilon > 0$,* SynAppx$(\{0,1,2\}, n^{1-\varepsilon})$ *is not solvable in polynomial time, unless* P = NP.

5.1 Free Bit Complexity and Stronger PCP Theorems.

Let us first briefly introduce the notion of free bit complexity. For a comprehensive discussion see [5].

Definition 5.2 *Consider a PCP verifier V using r random bits on any input x. For a fixed input x and a sequence of random bits $R \in \{0,1\}^r$, define $G(x,R)$ to be the set of sequences of answers to the questions asked by V, which result in an acceptance. We say that V has free bit complexity f if $|G(x,R)| \leq 2^f$ and there is a polynomial time algorithm which computes $G(x,R)$ for given x and R.*

The set of languages for which there exists a verifier with soundness p, free bit complexity f, and proof length ℓ is denoted by $\mathrm{FPCP}_p[r, f, \ell]$.

Håstad in his seminal work [13] proved that approximating the maximum clique within the factor $n^{1-\varepsilon}$ is hard, for every $\varepsilon > 0$. To obtain this result he constructs PCP verifiers with arbitrarily small *amortized free bit complexity*[1]. We state his result in a more recent and stronger version [14]:

Theorem 5.3 *For every $\varepsilon > 0$, there exist constants $t \in \mathbb{N}$ and $\alpha, \beta > 0$ such that* NP $\subseteq \mathrm{FPCP}_{2^{-t}}[\beta \log n, \varepsilon \cdot t, n^{\alpha}]$.

In the next step we need to amplify the error probability, as we did in the previous section. It turns out that the amplification using expander walks is too weak for our purpose. Håstad [13], following the approach of Bellare et al. [5], uses sequential repetition together with a technique to reduce the demand for random bits. (See Proposition 11.2, Corollary 11.3 in [5].) Unfortunately, this procedure involves randomization, so his MaxClique hardness result holds under the assumption that ZPP $\neq$ NP.

In his breakthrough paper Zuckerman [24] showed how to derandomize Håstad's MaxClique hardness result by giving a deterministic method for amplifying the error probability of PCP verifiers. He constructs very efficient randomness extractors, which then by known reductions allow to perform error amplification. One can conclude the following result from Theorem 5.3 and his result (see also Lemma 6.4 and Theorem 1.1 in [24]).

[1] Amortized free bit complexity is a parameter of a PCP verifier which essentially corresponds to the ratio between the free bit complexity and the logarithm of error probability.

Theorem 5.4 *For every $\varepsilon > 0$, there exist $c, \alpha > 0$ such that for $t = c \log n$ it holds that* NP $\subseteq$ $\mathrm{FPCP}_{2^{-t}}[(1+\varepsilon)t, \varepsilon t, n^{\alpha}]$.

Based on the above theorem, we can prove the following very strong analogue of Theorem 4.2.

Theorem 5.5 *For every $\varepsilon > 0$, there exists a polynomial time reduction f, which takes an n-variable 3-CNF formula ϕ and returns an N-variable CSP instance $f(\phi)$ with M constraints, such that:*

- $N \leq M^{\varepsilon}$,
- *if ϕ is satisfiable then* $\mathrm{Val}(f(\phi)) = 1$,
- *if ϕ is not satisfiable then* $\mathrm{Val}(f(\phi)) \leq \frac{1}{M^{1-\varepsilon}}$,
- $\mathrm{Fsat}(f(\phi)) \leq M^{\varepsilon}$.

Proof: Fix an $\varepsilon > 0$. We use Theorem 5.4 to conclude NP $\subseteq \mathrm{FPCP}_{2^{-t}}[(1+\varepsilon)t, \varepsilon t, n^{\alpha}]$ for some $\alpha, c > 0$ and $t = c \log n$. As a preliminary step, we make the proof length negligible (because α can be arbitrarily big) using the simple sequential repetition. By repeating the verification procedure k times we obtain NP $\subseteq \mathrm{FPCP}_{2^{-kt}}[(1+\varepsilon)kt, \varepsilon kt, n^{\alpha}]$. We take such verifier V for the 3-SAT language with k chosen large enough to make sure that $\alpha < \varepsilon kc$.

Consider any 3-CNF formula ϕ. For convenience denote $r := (1+\varepsilon)kt = (1+\varepsilon)kc \log n$. We construct $f(\phi)$ as in the proof of Theorem 4.2. The number of variables is $N = n^{\alpha}$. For every possible sequence $s \in \{0,1\}^r$ of r random bits we define a constraint C_s, such that given an assignment $v \in \{0,1\}^N$, C_s evaluates to 1 if and only if V accepts the proof v.

We have defined $f(\phi)$, now we prove that it satisfies all the claimed properties. $N = n^{\alpha}$, $M = n^{(1+\varepsilon)kc}$ and $\frac{\alpha}{kc} < \varepsilon$, hence $N \leq M^{\varepsilon}$. If ϕ is satisfiable then there is a proof which is accepted for all possible sequences of random bits, hence all the constraints can be satisfied simultaneously, so $\mathrm{Val}(f(\phi)) = 1$. Suppose ϕ is not satisfiable, then the probability of accepting a wrong proof is at most $2^{-kt} = n^{-kc} = M^{-1/(1+\varepsilon)} \leq \frac{1}{M^{1-\varepsilon}}$. Finally, because of the bound on the free bit complexity of V, for every sequence of random bits $s \in \{0,1\}^r$ there are at most $2^{\varepsilon kt} = n^{\varepsilon kc} \leq M^{\varepsilon}$ sequences of bits encoding the answers to the queries, which result in an acceptance. By the definition of free bit complexity, those sequences can be efficiently listed, hence the CSP instance $f(\phi)$ can be constructed in polynomial time. □

5.2 Construction.

Let ϕ be an N-variable CSP instance with M constraints such that $\mathrm{Fsat}(\phi) \leq K$ (for a parameter K to be chosen later). We want to construct an automaton $\hat{A}_{\phi}$ of size polynomial in K and the size of ϕ such that $\mathrm{Syn}(\hat{A}_{\phi}) \approx \frac{N}{\mathrm{Val}(\phi)}$.

Using A_{ϕ} as in the previous sections gives an automaton of superpolynomial size, so we need to tweak it. Take any constraint C and suppose it depends on q variables. Consider the tree-gadget T_C built for C. We cannot assume that $q = O(\log n)$ as in the Section 4. In consequence, T_C can be of exponential size, because the only possible bound on its number of leaves is 2^q. However, we have a bound K on the number of essentially satisfying assignments. We will modify the definition of T_C, so that its size depends polynomially on K rather than 2^q.

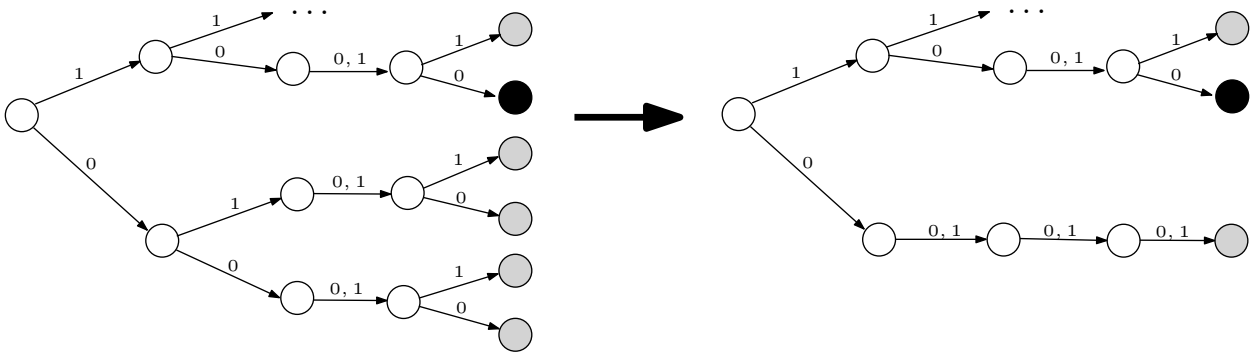

Figure 2: Compressing the tree. The grey and black leaves correspond to non-satisfying and satisfying assignments, respectively.

Observe that in the original construction, at most K out of 2^q leaves of T_C correspond to satisfying assignments (think of K much smaller than 2^q). Imagine for a moment a subtree of T_C corresponding to the at most K satisfying assignments. The size of such subtree is at most NK, but it is not yet a good candidate for our gadget, because some transitions are not well defined. However, this is not difficult to fix to obtain an equivalent *compressed tree-gadget* $\hat{T}_C$ as follows. Take a variable x_k (such that C depends on x_k) and a node q_k^w with $w = vc$ (then basically $c \in \{0,1\}$ corresponds to the boolean value assigned to x_k). Suppose that every leaf in the subtree rooted at q_k^w corresponds to a non-satisfying assignment. Suppose further that there is some satisfying assignment in the subtree q_{k-1}^v (then the leaf with a satisfying assignment must necessarily lie in the subtree rooted at $q_k^{v\bar{c}}$). In such a case there is nothing interesting happening in the subtree rooted at q_k^w, it has height $h = N - k$ and all of its leaves have two edges (labeled 0 and 1) going back to the root r of T_C. We remove this subtree rooted at q_k^w and instead attach a path of length h to q_k^w, we also add transitions from the endpoint of the path to r.

For an illustration of the compression procedure refer to Figure 2. The leftmost vertex in the figure is q_{k-1}^v, his two children are q_k^{v0} and q_k^{v1}, the subtree rooted at q_k^{v0} gets compressed. In a tree-gadget the black leaf in the picture has transitions (labeled by 0 and 1) to the sink vertex s and all the grey vertices have transitions back to the root r. We obtain the compressed tree-gadget $\hat{T}_C$ by applying the above transformation to every relevant node q_k^w.

The automaton $\hat{A}_\phi$ is built analogously to A_ϕ, with the crucial difference that we use $\hat{T}_C$ instead of T_C. One can see that $\hat{T}_C$ can be constructed in time polynomial in its size by proceeding from the root down to the leaves. Furthermore, the resulting automaton $\hat{A}_\phi$ is small.

Lemma 5.6 *Suppose ϕ is an N-variable CSP instance with M constraints and* $\mathrm{Fsat}(\phi) \leq K$. *Then the size of $\hat{A}_\phi$ is $O(MN^2K)$.*

Proof: The automaton consists of M gadgets and 1 additional state s. It suffices to prove that every gadget has size $O(N^2K)$. Fix any constraint C in ϕ and consider the compressed tree-gadget $\hat{T}_C$. Suppose we remove from $\hat{T}_C$ all leaves corresponding to non-satisfying assignments together with the paths created in the compression procedure. What remains is a subtree T_0 of height N with at most K leaves, each corresponding to the satisfying assignments of C. The size of T_0 is $O(NK)$. To get $\hat{T}_C$ back from T_0, we need to attach paths (of length at most N) to some vertices

of T_0 (at most one path per vertex). We add at most $O(NK)$ paths consisting of at most N nodes each, hence the total size of $\hat{T}_C$ is $O(N^2K)$. □

5.3 Properties of $\hat{A}_\phi$.

The lemma below summarizes the properties of $\hat{A}_\phi$. Its proof is very similar to the proofs of the lemmas summarizing the properties of A_ϕ and hence skipped.

Lemma 5.7 *Let ϕ be an N-variable CSP instance with M constraints and* $\mathrm{Fsat}(\phi) \leq K$. *Then $\hat{A}_\phi$ is a synchronizing automaton of size $O(MN^2K)$, which can be constructed in polynomial time. Furthermore, if ϕ is satisfiable then* $\mathrm{Syn}(\hat{A}_\phi) \leq N+2$ *and otherwise* $\mathrm{Syn}(\hat{A}_\phi) \geq \frac{N+1}{\mathrm{Val}(\phi)}$.

Proof: [of Theorem 5.1] Fix any $\varepsilon > 0$. We reduce 3-SAT to SynAppx$(\{0,1,2\}, n^\varepsilon)$. Let ϕ be an n-variable 3-CNF formula. Then by Theorem 5.5 we can construct an N-variable CSP instance $f(\phi)$ with $M = \mathrm{poly}(n)$ constraints and $\mathrm{Fsat}(f(\phi)) \leq K = M^\varepsilon$, where $N \leq M^\varepsilon$. We know that if ϕ is satisfiable then $f(\phi)$ is satisfiable as well and if ϕ is not satisfiable then $\mathrm{Val}(f(\phi)) \leq \frac{1}{M^{1-\varepsilon}}$. Then by Theorem 5.7 we can construct $\hat{A}_{f(\phi)}$, which is an automaton of size $O(MKN^2) = O(M^{1+3\varepsilon})$. If ϕ is satisfiable, $\mathrm{Syn}(\hat{A}_{f(\phi)}) \leq N+2$ and if ϕ is not satisfiable then $\mathrm{Syn}(\hat{A}_{f(\phi)}) \geq \frac{N+1}{\mathrm{Val}(f(\phi))}$. The ratio of those two bounds is:

$$\frac{N+1}{(N+2)\mathrm{Val}(f(\phi))} = \Theta\left(\frac{1}{\mathrm{Val}(f(\phi))}\right) = \Theta\left(M^{1-\varepsilon}\right)$$

The size of the automaton $\hat{A}_{f(\phi)}$ is $G = O(M^{1+3\varepsilon})$, so the above ratio can be related to the size of the automaton as $\Omega\left(G^{\frac{1-\varepsilon}{1+3\varepsilon}}\right) = \Omega\left(G^{1-4\varepsilon}\right)$. Hence assuming P $\neq$ NP, approximating the shortest reset word within ratio $G^{1-4\varepsilon}$ in polynomial time is not possible. □

References

[1] Ananichev, D.S., Volkov, M.V.: Synchronizing generalized monotonic automata. Theor. Comput. Sci. 330(1), 3–13 (Jan 2005)

[2] Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, 1st edn. (2009)

[3] Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. J. ACM 45(3), 501–555 (1998)

[4] Arora, S., Safra, S.: Probabilistic checking of proofs: A new characterization of NP. J. ACM 45(1), 70–122 (1998)

[5] Bellare, M., Goldreich, O., Sudan, M.: Free bits, PCPs, and nonapproximability—towards tight results. SIAM J. Comput. 27, 804–915 (1998)

[6] Berlinkov, M.V.: Approximating the minimum length of synchronizing words is hard. Theor. Comp. Sys. 54(2), 211–223 (Feb 2014)

[7] Berlinkov, M.: On two algorithmic problems about synchronizing automata. In: Shur, A.M., Volkov, M.V. (eds.) Developments in Language Theory, Lecture Notes in Computer Science, vol. 8633, pp. 61–67. Springer International Publishing (2014)

[8] Eppstein, D.: Reset sequences for monotonic automata. SIAM J. Comput. pp. 500–510 (1990)

[9] Feige, U.: A threshold of $\ln n$ for approximating set cover. J. ACM 45(4), 634–652 (1998)

[10] Gabber, O., Galil, Z.: Explicit constructions of linear-sized superconcentrators. J. Comput. Syst. Sci. 22(3), 407–420 (1981)

[11] Gerbush, M., Heeringa, B.: Approximating minimum reset sequences. In: Implementation and Application of Automata, Lecture Notes in Computer Science, vol. 6482, pp. 154–162. Springer Berlin Heidelberg (2011)

[12] Grech, M., Kisielewicz, A.: The Černý conjecture for automata respecting intervals of a directed graph. Discrete Mathematics & Theoretical Computer Science pp. 61–72 (2013)

[13] Hastad, J.: Clique is hard to approximate within $n^{1-\varepsilon}$. In: Proceedings of the 37th Annual Symposium on Foundations of Computer Science. pp. 627–636. FOCS '96 (1996)

[14] Hastad, J., Khot, S.: Query efficient PCPs with perfect completeness. In: Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on. pp. 610–619 (Oct 2001)

[15] Impagliazzo, R., Zuckerman, D.: How to recycle random bits. In: Proceedings of the 30th Annual Symposium on Foundations of Computer Science. pp. 248–253. SFCS '89 (1989)

[16] Jimbo, S., Maruoka, A.: Expanders obtained from affine transformations. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing. pp. 88–97. STOC '85, ACM (1985)

[17] Kari, J.: Synchronizing finite automata on eulerian digraphs. Theor. Comput. Sci. 295, 223–232 (2003)

[18] Margulis, G.A.: Explicit constructions of concentrators. Probl. Peredachi Inf. 9(4), 71–80 (1973)

[19] Olschewski, J., Ummels, M.: The complexity of finding reset words in finite automata. In: Proceedings of the 35th International Conference on Mathematical Foundations of Computer Science, pp. 568–579. MFCS'10, Springer-Verlag (2010)

[20] Pin, J.: On two combinatorial problems arising from automata theory. In: Combinatorial Mathematics Proceedings of the International Colloquium on Graph Theory and Combinatorics, vol. 75, pp. 535 – 548. North-Holland (1983)

[21] Rystsov, I.: Reset words for commutative and solvable automata. Theoretical Computer Science 172(1–2), 273 – 279 (1997)

[22] Steinberg, B.: The Černý conjecture for one-cluster automata with prime length cycle. Theoretical Computer Science 412(39), 5487 – 5491 (2011)

[23] Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers, Lecture Notes in Computer Science, vol. 5196, pp. 11–27. Springer (2008)

[24] Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. In: Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing. pp. 681–690. STOC '06 (2006)

A Expanders and Error Reduction

In this section we show how to deduce a subconstant error PCP theorem from the standard version, i.e., Theorem 2.2. Thus, we prove the following.

Theorem 4.1 (Subconstant Error PCP) $\mathrm{NP} \subseteq \mathrm{PCP}_{1/n}[O(\log n), O(\log n)]$.

To this end we need a method of reusing random bits when repeating a random experiment introduced by Impagliazzo and Zuckerman [15]. The idea is that, instead of generating fresh random bits for each repetitions of the experiment, we use a random walk on an expander to construct a pseudorandom sequence of bits, which is then used in subsequent repetitions. Because of expanding properties of such graphs, which we summarize below, this is enough to significantly decrease the error probability. A good reference for expander graphs and pseudorandom constructions is [2]. For completeness we include all essential definitions, but we refer to the book for the proofs.

Whenever G appears in the following text, it denotes an undirected d-regular graph on n vertices, possibly containing loops and parallel edges.

Definition A.1 ($\lambda(G)$) *Let A_G be the random walk matrix of G (that is, A_G is the adjacency matrix of G with each entry scaled by $\frac{1}{d}$). Let $\lambda_1, \lambda_2, \ldots, \lambda_n \in [-1, 1]$ be the eigenvalues of A_G, sorted so that $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$. We define $\lambda(G)$ to be $|\lambda_2|$.*

Definition A.2 ((n, d, λ)-expander) *If G is an n-vertex d-regular multigraph with $\lambda(G) \leq \lambda < 1$, then we say that G is an (n, d, λ)-graph.*

It turns out that constructing a family of (n, d, λ)-graphs for some fixed $\lambda > 0$ is a pretty simple task, since a random d-regular graph is an expander with high confidence. However, a true challenge is to construct expanders explicitly and without any use of random bits. A beautiful example of such a construction was given by Margulis [18], its analysis was later improved and simplified first by Gabber and Galil [10], and then by Jimbo and Maruoka [16], to yield the following.

Theorem A.3 *Let G_{n^2} be the 8-regular graph on vertex set $\mathbb{Z}_n^2$, with edges defined as follows: (x, y) has neighbors $(x \pm 2y, y), (x \pm (2y+1), y), (x, y \pm 2x), (x, y \pm (2x+1))$ (addition is performed modulo n). Then G_{n^2} is an $(n^2, 8, \frac{5\sqrt{2}}{8})$-graph.*

The following theorem can be now used to reduce the error probability.

Theorem A.4 (Expander walks, 21.12 in [2]) *Let G be an (n, d, λ)-graph and let $B \subseteq [n]$ be a set satisfying $|B| \leq \beta n$ for some $\beta \in (0, 1)$. Let $X_1, X_2, \ldots, X_k$ be random variables denoting a $(k-1)$-step random walk in G, meaning that X_1 is chosen uniformly from $[n]$ and X_{i+1} is a uniform random neighbor of X_i. Then $P(X_1 \in B \wedge X_2 \in B \wedge \ldots \wedge X_k \in B) \leq ((1-\lambda)\sqrt{\beta} + \lambda)^{k-1}$.*

Theorem 4.1 (Subconstant Error PCP) $\mathrm{NP} \subseteq \mathrm{PCP}_{1/n}[O(\log n), O(\log n)]$.

Proof: Take any language $L \in \mathrm{NP}$, we would like to show that there exists a polynomial time $(\frac{1}{n}, O(\log n), O(\log n))-$verifier for L. By the basic Theorem 2.2 we know that there is a $(\frac{1}{2}, O(\log n), O(1))-$verifier for L. Suppose it consumes at most $r \log n$ random bits. We intend to define another verifier V', which makes more queries to the proof and needs more random bits, but its failure probability is at most $\frac{1}{n}$. Running V independently $\Omega(\log n)$ times and checking if there was at least one reject is not acceptable, as it increases the number of used random bits to $\Omega(\log^2 n)$ bits. For this reason, instead of making $\Omega(\log n)$ fully independent runs, we save some random bits using expanders. Let $k = c \log n$ be the number of repetitions. Fix an input x of length n. We construct an $(\mathcal{O}(2^{r \log n}), 8, \lambda)$-graph with $\lambda = \frac{5\sqrt{2}}{8}$ from Theorem A.3, select a random starting vertex X_1 there, and choose a random walk of length $k-1$ starting from X_1 obtaining vertices $X_2, X_3, \ldots, X_k$. Now run V k times using X_i as the required stream of $r \log n$ bits for the i-th run. Answer 1 if and only if all runs returned 1. To obtain the sequence $X_1, \ldots, X_k$ we use r random bits for X_1 and $(k-1) \cdot 3$ bits for $X_2, \ldots X_k$ (we need 3 bits to pick one neighbor out of five), $O(\log n)$ random bits in total. Let us now calculate the error probability. According to Theorem A.4 (where we choose B to be the set of length$-(r \log n)$ bitstrings which cause a false positive for x), it is at most:

$$\left((1-\lambda)\sqrt{\frac{1}{2}} + \lambda\right)^{k-1} = \left(\left(1 - \frac{5\sqrt{2}}{8}\right)\sqrt{\frac{1}{2}} + \frac{5\sqrt{2}}{8}\right)^{k-1} < (0.97)^k$$

Which is less then $\frac{1}{n}$ when we take $c = \frac{1}{\log 0.97}$. To finish, let us note that the query complexity of V' is $k \cdot O(1) = O(\log n)$ as claimed. $\square$