Available online at www.sciencedirect.com

ScienceDirect

Journal of Computer and System Sciences 73 (2007) 908–923

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.elsevier.com/locate/jcss

Minimizing nfa's and regular expressions ☆

Gregor Gramlich *,[1], Georg Schnitger [1]

Institut für Informatik, Johann Wolfgang Goethe-Universität Frankfurt, Robert-Mayer-Straße 11-15, 60054 Frankfurt am Main, Germany

Received 22 December 2004; received in revised form 27 November 2006

Available online 22 December 2006

Abstract

We show inapproximability results concerning minimization of nondeterministic finite automata (nfa's) as well as of regular expressions relative to given nfa's, regular expressions or deterministic finite automata (dfa's).

We show that it is impossible to efficiently minimize a given nfa or regular expression with n states, transitions, respectively symbols within the factor $o(n)$, unless $P = \text{PSPACE}$. For the unary case, we show that for any $\delta > 0$ it is impossible to efficiently construct an approximately minimal nfa or regular expression within the factor $n^{1-\delta}$, unless $P = NP$.

Our inapproximability results for a given dfa with n states are based on cryptographic assumptions and we show that any efficient algorithm will have an approximation factor of at least $\frac{n}{\text{poly}(\log n)}$. Our setup also allows us to analyze the minimum consistent dfa problem.

Keywords: Automata and formal languages; Computational complexity; Approximability

1. Introduction

Among the most basic objects of formal language theory are regular languages and their acceptance devices, finite automata and regular expressions. Regular expressions describe lexical tokens for syntactic specifications, textual patterns in text manipulation systems and they are the basis of standard utilities such as scanner generators, editors or programming languages (perl, awk, php). Internally regular expressions are converted to (nondeterministic) finite automata and the succinctness of this representation crucially determines the running time of the applied algorithms.

Contrary to the problem of minimizing dfa's, which is efficiently possible, it is well known that nfa or regular expression minimization is computationally hard, namely PSPACE-complete [11]. Jiang and Ravikumar [8] show moreover that the minimization problem for nfa's or regular expressions remains PSPACE-complete, even when specifying the regular language by a dfa.

☆ This paper is the final version of [G. Gramlich, G. Schnitger, Minimizing nfa's and regular expressions, in: V. Diekert, B. Durand (Eds.), STACS '05, in: Lecture Notes in Comput. Sci., vol. 3404, Springer-Verlag, 2005. [4]] and also includes results from [G. Gramlich, Probabilistic and nondeterministic unary automata, in: B. Rovan, P. Vojtás (Eds.), MFCS, in: Lecture Notes in Comput. Sci., vol. 2747, Springer-Verlag, 2003, pp. 460–469].

* Corresponding author. Fax: +49 69 798 28814.

E-mail addresses: gramlich@thi.informatik.uni-frankfurt.de (G. Gramlich), georg@thi.informatik.uni-frankfurt.de (G. Schnitger).

URL: http://www.thi.cs.uni-frankfurt.de/~gramlich (G. Gramlich).

[1] Partially supported by DFG project SCHN503/2-1.

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2006.11.002

We consider the problem of *approximating* a minimal nfa or a minimal regular expression. There are several approaches to nfa minimization [1,5,6,10] either without approximation guarantees or running in at least exponential time. This article explains why such guarantees cannot be expected for efficient algorithms.

We investigate the approximation problem in two scenarios. In the first scenario the language is specified by a dfa which makes proofs of inapproximability hard, since the input is not specified concisely and thus more time compared to concise inputs such as nfa's or regular expressions is available. Jiang and Ravikumar [8] ask to determine the approximation complexity of converting dfa's into nfa's, and in particular ask whether efficient approximation algorithms with a polynomial approximation factor exist. Corollary 13 shows that such an approximation is at least as hard as factoring Blum integers and therefore efficient approximation algorithms with polynomial approximation factor are unlikely.

We show in Theorem 10 that efficient approximation algorithms determine regular expressions of length at least $\frac{k}{\text{poly}(\log k)}$ for a given dfa of size k, even if optimal regular expressions of length $\text{poly}(\log k)$ exist. We have to assume however that strong pseudo-random functions exist in nonuniform NC^1. The concept of a strong pseudo-random function is introduced by Razborov and Rudich [15]. Naor and Reingold [12] show that strong pseudo-random functions exist even in TC^0, provided factoring Blum integers requires time $2^{\Omega(n^\varepsilon)}$ (for some $\varepsilon > 0$).

We show similar results for approximating nfa's in Corollary 13, but now relative to the weaker assumption that strong pseudo-random functions exist in nonuniform Logspace. We also apply our technique to the minimum consistent dfa problem [9,14] in which a dfa of minimum size, consistent with a set of classified inputs, is to be determined.

Thus in the first scenario we follow the cryptographic approach of Kearns and Valiant [9] when analyzing the complexity of approximation, but work with pseudo-random functions instead of one-way functions.

In the second scenario we assume that the language is specified by either an nfa or a regular expression. For the *unary* case we show in Theorem 18 a lower bound of $\frac{\sqrt{n}}{\ln n}$ for the approximation factor. This holds under the assumption $P \neq NP$ for given nfa's as well as for given regular expressions [3]. We improve the approximation factor in Theorem 22 to $n^{1-\delta}$ for every $\delta > 0$, provided $P \neq NP$ and provided we require the approximation algorithm to determine a small equivalent nfa or regular expression, opposed to just determining its size.

Furthermore we show a PSPACE-completeness result for approximating the minimal size of *general* nfa's or regular expressions. Specifically Theorem 24 shows that it is impossible to efficiently minimize a given nfa or regular expression with n states, n transitions, respectively n symbols within the factor $o(n)$, unless $P = \text{PSPACE}$. The proof of Theorem 24 is based on the PSPACE-completeness of the regular expression nonuniversality problem.

We introduce strong pseudo-random functions in Section 2 and investigate the complexity of approximating minimal regular expressions or nfa's, relative to a given dfa, in Sections 2.1 and 2.2. The minimum consistent dfa problem is considered in Section 2.3. Finally, relative to a given nfa or regular expression, the complexity of approximately minimizing unary nfa's or regular expressions is determined in Section 3.1, whereas general alphabets are treated in Section 3.2.

2. Pseudo-random functions and approximation

We consider the question of computing small equivalent nfa's or regular expressions for given dfa's. Inapproximability results seem to be hard to prove, since, intuitively, it takes large dfa's to specify hard inputs and consequently the allowed running time increases. We can only weakly utilize the dfa specification in comparison with a mere truth table specification and hence first concentrate on the truth table specification for functions $f : \{0,1\}^n \to \{0,1\}$.

Our goal is to utilize the natural proof setup of Razborov and Rudich [15] and, in particular, to conclude that any efficient approximation algorithm separates pseudo-random functions from random functions. However we then face the problem that nfa's or regular expressions are too weak to express pseudo-random functions with few states. Therefore we follow the approach of Pitt and Warmuth [13] and repeat inputs. Thus instead of approximating minimum nfa's for the language $L(f) = \{x \mid x \in \{0,1\}^n \wedge f(x) = 1\} \subseteq \{0,1\}^n$, we consider the approximation problem for (the complement of) the language

$$L_p(f) := \left\{x^p \;\middle|\; x \in \{0,1\}^n \wedge f(x) = 1\right\} \subseteq \{0,1\}^{n \cdot p}$$

for a suitable natural number p; x^p is the p-fold concatenation of x.

First we introduce the concept of strong pseudo-random functions [15], but replace circuits by probabilistic Turing machines and require only a constant probability of separating pseudo-randomness from true randomness. Obviously

strong pseudo-random functions exist in our setting, provided strong pseudo-random functions exist in the sense of Razborov and Rudich.

Definition 1. Let $f_n = (f_n^s)_{s\in S}$ be a function ensemble with functions $f_n^s : \{0,1\}^n \to \{0,1\}$ for a seed $s \in S$ and let $(r_n^i)_{i\in\{1,\ldots,2^{2^n}\}}$ be the ensemble of all n-bit boolean functions. We call f_n a strong pseudo-random ensemble with parameter ε iff for any randomized algorithm A

$$\left|\text{prob}\left[A(f_n) = 1\right] - \text{prob}\left[A(r_n) = 1\right]\right| < \frac{1}{3},$$

provided A runs in time $2^{O(n^\varepsilon)}$ and has access to f_n^s, respectively r_n^i, via a membership oracle. The probability is defined by the random choices of A and the uniform sampling of s from S, respectively the uniform sampling of i from $\{1,\ldots,2^{2^n}\}$.

It is widely believed that there is some $\varepsilon > 0$, such that any algorithm running in time $2^{O(n^\varepsilon)}$ cannot factor Blum integers well on average. Observe that we may assume $\varepsilon < 1$. Naor and Reingold [12] construct TC^0 functions which are strong pseudo-random functions, provided factoring Blum integers requires time $2^{\Omega(n^\varepsilon)}$ for some ε.

As already mentioned, we restrict our attention to approximating minimal nfa's or regular expressions for the languages $\overline{L_p(f)}$ for n-bit boolean functions $f : \{0,1\}^n \to \{0,1\}$. We interpret this approximation problem as the problem of approximating a functional $G(f)$, where $G(f)$ is either defined as the size of a minimum nfa or the minimal length of a regular expression for the language $\overline{L_p(f)}$.

Definition 2. B_n is the set of all n-bit boolean functions. We define the compression $k_m : B_n \to B_m$ for $m < n$ by $(k_m(f))(x) = f(0^{n-m}x)$ for $x \in \{0,1\}^m$.

We say, that a functional $G = (G_n)_n$ with $G_n : B_n \to \mathbb{N}$ separates a function class $\mathcal{C}$ from random functions with monotonically increasing thresholds $t_1(\cdot)$ and $t_2(\cdot)$ iff $G_n(f) < t_1(n)$ holds for every function $f \in \mathcal{C} \cap B_n$, whereas $G_n(h_n) > t_2(n)$ for most functions in B_n, i.e.,

$$\lim_{n\to\infty} \frac{|\{h_n \in B_n \mid G_n(h_n) \leqslant t_2(n)\}|}{|B_n|} = 0$$

holds. Moreover we require that $G_m(k_m(f)) \leqslant t_1(n)$ for any function $f \in \mathcal{C} \cap B_n$ and any $m < n$.

It is not surprising that a functional G, which separates a function class $\mathcal{C}$ containing pseudo-random functions from random functions, cannot be efficiently approximated. We even allow randomized approximation algorithms which may underestimate the minimum.

Definition 3. Let $|x|$ be the length of input x. We say that a randomized algorithm $App : X \to \mathbb{N}$ with approximation factor $\mu(|x|)$ for a minimization problem *opt* has overestimation error $\epsilon_+ = \sup_{x\in X} \text{prob}[App(x) > \mu(|x|) \cdot opt(x)]$ and underestimation error $\epsilon_- = \sup_{x\in X} \text{prob}[App(x) < opt(x)]$. The probabilities are defined by the random choices of *App*.

We state a generic lemma for approximation algorithms on compressed inputs allowing us to replace oracle access by truth table presentation.

A quick remark on our notation: we use r_n to denote an n-bit random ensemble, f_n to denote an n-bit pseudo-random ensemble and h_n to denote the input functions for which a small regular expression is to be found. The generic lemma separates $\mathcal{C}_n$ from n-bit random functions by applying an approximation algorithm on the compressed function $k_m(h_n)$.

Lemma 4. *Assume that the functional G separates $\mathcal{C}$ from random functions with thresholds t_1, t_2 and suppose that $\mathcal{C}$ contains a strong pseudo-random ensemble with parameter ε.*

Let App be a randomized approximation algorithm that approximately determines $G_m(h_m)$, when given the truth table of size $|h_m| = 2^m$ of a function $h_m \in B_m$. Then for all $l \geqslant 1$, if App runs in time $2^{O(m^l)}$ and achieves an approximation factor $\mu(2^m) < \frac{t_2(m)}{t_1(m^{l/\varepsilon})}$, then App must have errors $\epsilon_+ + \epsilon_- \geqslant \frac{2}{3}$.

Proof. By assumption $\mathcal{C}$ contains strong pseudo-random functions with parameter ε. For the sake of contradiction, let *App* be an algorithm which approximates $G_m(h_m)$ when given the truth table of h_m (with running time $2^{O(m^l)}$ for some $l \geqslant 1$, approximation factor $1 \leqslant \mu(2^m) < \frac{t_2(m)}{t_1(m^{l/\varepsilon})}$ and errors $\epsilon_+ + \epsilon_- < \frac{2}{3}$). We construct an algorithm A which uses *App* to distinguish n-bit functions in $\mathcal{C}$ from n-bit random functions. We set $m = \lceil n^{\varepsilon/l} \rceil$.

A has oracle access to the input $h_n \in B_n$ and builds the truth table for the restriction $k_m(h_n)$. Then A runs *App* on $k_m(h_n)$ and accepts (i.e. $A(h_n) = 1$), if $App(k_m(h_n)) \leqslant t_2(m)$, and rejects (i.e. $A(h_n) = 0$) otherwise. So

$$\bigl|\text{prob}\bigl[A(f_n) = 1\bigr] - \text{prob}\bigl[A(r_n) = 1\bigr]\bigr| = \bigl|\text{prob}\bigl[App\bigl(k_m(f_n)\bigr) \leqslant t_2(m)\bigr] - \text{prob}\bigl[App\bigl(k_m(r_n)\bigr) \leqslant t_2(m)\bigr]\bigr|$$

holds, where probabilities are defined by the probabilistic choices of *App* as well as the random sampling of seeds for f_n, respectively the uniform random sampling of functions $r_n \in B_n$.

G separates $\mathcal{C}$ from random functions and hence we have $G_m(k_m(f_n)) \leqslant t_1(n)$ for $f_n \subseteq \mathcal{C}$. Finally observe that $\mu(2^m) \cdot t_1(n) < t_2(m)$ holds by assumption on $\mu(2^m)$, and since t_1 is monotonically increasing. Thus

$$\begin{aligned}
\text{prob}\bigl[App\bigl(k_m(f_n)\bigr) \leqslant t_2(m)\bigr] &\geqslant \text{prob}\bigl[App\bigl(k_m(f_n)\bigr) \leqslant \mu\bigl(2^m\bigr) \cdot t_1(n)\bigr] \\
&= 1 - \text{prob}\bigl[App\bigl(k_m(f_n)\bigr) > \mu\bigl(2^m\bigr) \cdot t_1(n)\bigr] \\
&\geqslant 1 - \text{prob}\bigl[App\bigl(k_m(f_n)\bigr) > \mu\bigl(2^m\bigr) \cdot G_m\bigl(k_m(f_n)\bigr)\bigr] \\
&\geqslant 1 - \epsilon_+
\end{aligned}$$

holds. We utilize that the restriction of a uniformly sampled function r_n from B_n leads to a uniformly sampled random function r_m from B_m and obtain

$$\begin{aligned}
\text{prob}\bigl[App\bigl(k_m(r_n)\bigr) \leqslant t_2(m)\bigr] &\leqslant \text{prob}\bigl[G_m\bigl(k_m(r_n)\bigr) \leqslant t_2(m)\bigr] + \epsilon_- = \frac{|\{h_m \mid G_m(h_m) \leqslant t_2(m)\}|}{|B_m|} + \epsilon_- \\
&= \epsilon_- + o(1).
\end{aligned}$$

Thus $|\text{prob}[App(k_m(f_n)) \leqslant t_2(m)] - \text{prob}[App(k_m(r_n)) \leqslant t_2(m)]| \geqslant 1 - \epsilon_+ - \epsilon_- - o(1) > \frac{1}{3}$ holds for sufficiently large m. Since A runs in time $O(2^m) + 2^{O(m^l)} = 2^{O(n^\varepsilon)}$, this contradicts the assumption that $\mathcal{C}$ contains a strong pseudo-random ensemble with parameter ε. □

2.1. Regular expressions and logarithmic formula depth

Definition 5. A formula is a binary tree with $\wedge$ and $\vee$ gates as interior nodes; leaves are marked by labels from $\{x_1, \overline{x_1}, \ldots, x_i, \overline{x_i}, \ldots\}$. For a formula $\mathbf{f}$ let $\ell(\mathbf{f})$ be the length, i.e., the number of leaves of $\mathbf{f}$. The length $\ell(R)$ of a regular expression R is the number of symbols from the alphabet Σ appearing in R.

We later use a strong pseudo-random ensemble $\mathcal{C}_1 \subset NC^1$. Observe that any $f \in \mathcal{C}_1 \cap B_m$ has formula depth at most $c \cdot \log m$ and formula length at most $p_1(m) := m^c$.

We define the functional $G^{(1)}$ by setting $G^{(1)}_m(h_m)$ to equal the minimum length of a regular expression for the complement of $L_{p_1}(h_m) = \{x^{p_1} \mid h_m(x) = 1\}$.

We associate regular expressions with formulae and show that the length of the regular expression is exponentially related to the depth of the formula.

Definition 6. Let $\mathbf{f}$ be a formula for a function $f : \{0,1\}^m \to \{0,1\}$. We define the regular expression $R(\mathbf{f})$ recursively as follows:

- If $\mathbf{f} = x_i$, then $R(\mathbf{f}) := (0+1)^{i-1}1(0+1)^{m-i}$.
- If $\mathbf{f} = \overline{x_i}$, then $R(\mathbf{f}) := (0+1)^{i-1}0(0+1)^{m-i}$.
- If $\mathbf{f} = \mathbf{f}_1 \wedge \mathbf{f}_2$, then $R(\mathbf{f}) := R(\mathbf{f}_1) \circ R(\mathbf{f}_2)$.
- If $\mathbf{f} = \mathbf{f}_1 \vee \mathbf{f}_2$, then $R(\mathbf{f}) := R(\mathbf{f}_1) \circ (0+1)^{m \cdot \ell(\mathbf{f}_2)} + (0+1)^{m \cdot \ell(\mathbf{f}_1)} \circ R(\mathbf{f}_2)$.

Lemma 7. *Let* $W = \{w \mid \exists x \in \{0,1\}^m \wedge w \in \{x\}^*\}$ *be the language of repeated inputs of length* m.

(a) $L(R(\mathbf{f})) \cap W = \{x^{\ell(\mathbf{f})} \mid f(x) = 1\} = L_{\ell(\mathbf{f})}(f)$.
(b) *If* $\mathbf{f}$ *is a formula of depth at most* k, *then the regular expression* $R(\mathbf{f})$ *has length at most* $2 \cdot 4^k m$.
(c) *For a given formula* $\mathbf{f}$ *of depth* k *there is a regular expression* $\overline{R_{\mathbf{f}}}$ *of length at most* $O(4^k m)$ *which describes the complement of* $L(R(\mathbf{f})) \cap W$.
(d) *In particular,* $\overline{L_{p_1}(f_m)}$ *has regular expressions of length at most* $t_1^{(1)} = \Theta(m^{2c+1})$ *for any* $f_m \in \mathcal{C}_1 \cap B_m$.

Proof. (a) can be shown by an induction on the structure of formula $\mathbf{f}$: Obviously $L(R(\mathbf{f})) \subseteq \{0,1\}^{m \cdot \ell(\mathbf{f})}$. If $\mathbf{f} = x_i$, then $R(\mathbf{f}) = (0+1)^{i-1}\ 1\ (0+1)^{m-i}$ and thus

$$L\big(R(\mathbf{f})\big) \cap W = \big\{x \bigm| x \in \{0,1\}^m \wedge x_i = 1\big\} = \big\{x \bigm| f(x) = 1\big\}.$$

The case $\mathbf{f} = \overline{x_i}$ follows analogously. If $\mathbf{f} = \mathbf{f}_1 \wedge \mathbf{f}_2$, then $R(\mathbf{f}) = R(\mathbf{f}_1) \circ R(\mathbf{f}_2)$ and thus

$$\begin{aligned} L\big(R(\mathbf{f})\big) \cap W &= \big(L\big(R(\mathbf{f}_1)\big) \circ L\big(R(\mathbf{f}_2)\big)\big) \cap W = \big(\big(L\big(R(\mathbf{f}_1)\big) \cap W\big) \circ \big(L\big(R(\mathbf{f}_2)\big) \cap W\big)\big) \cap W \\ &= \big(\big\{x^{\ell(\mathbf{f}_1)} \bigm| f_1(x) = 1\big\} \circ \big\{x^{\ell(\mathbf{f}_2)} \bigm| f_2(x) = 1\big\}\big) \cap W = \big\{x^{\ell(\mathbf{f}_1)+\ell(\mathbf{f}_2)} \bigm| f_1(x) = 1 \wedge f_2(x) = 1\big\} \\ &= \big\{x^{\ell(\mathbf{f})} \bigm| f(x) = 1\big\}. \end{aligned}$$

If $\mathbf{f} = \mathbf{f}_1 \vee \mathbf{f}_2$, then $R(\mathbf{f}) = R(\mathbf{f}_1) \circ (0+1)^{m \cdot \ell(\mathbf{f}_2)} + (0+1)^{m \cdot \ell(\mathbf{f}_1)} \circ R(\mathbf{f}_2)$ and thus

$$\begin{aligned} L\big(R(\mathbf{f})\big) \cap W &= \big(L\big(R(\mathbf{f}_1)\big) \circ \{0,1\}^{m \cdot \ell(\mathbf{f}_2)} \cup \{0,1\}^{m \cdot \ell(\mathbf{f}_1)} \circ L\big(R(\mathbf{f}_2)\big)\big) \cap W \\ &= \big(\big(L\big(R(\mathbf{f}_1)\big) \circ \{0,1\}^{m \cdot \ell(\mathbf{f}_2)}\big) \cap W\big) \cup \big(\big(\{0,1\}^{m \cdot \ell(\mathbf{f}_1)} \circ L\big(R(\mathbf{f}_2)\big)\big) \cap W\big) \\ &= \big\{x^{\ell(\mathbf{f}_1)+\ell(\mathbf{f}_2)} \bigm| f_1(x) = 1\big\} \cup \big\{x^{\ell(\mathbf{f}_1)+\ell(\mathbf{f}_2)} \bigm| f_2(x) = 1\big\} = \big\{x^{\ell(\mathbf{f}_1)+\ell(\mathbf{f}_2)} \bigm| f_1(x) = 1 \vee f_2(x) = 1\big\} \\ &= \big\{x^{\ell(\mathbf{f})} \bigm| f(x) = 1\big\}. \end{aligned}$$

(b) Let $\ell(k)$ be the maximal length of the regular expression $R(\mathbf{f})$ for a formula $\mathbf{f}$ with depth k. We show recursively that $\ell(k) \leqslant 2 \cdot 4^k m$. For $k = 0$ we have $\ell(0) \leqslant 2m = 2 \cdot 4^k m$.

For formulae $\mathbf{f}_1$ and $\mathbf{f}_2$ of depth at most k the regular expression $R(\mathbf{f}_1 \wedge \mathbf{f}_2)$ has length at most $2\ell(k) \leqslant 2 \cdot 2 \cdot 4^k m = 4^{k+1} m$, and the regular expression $R(\mathbf{f}_1 \vee \mathbf{f}_2)$ has length at most

$$\begin{aligned} 2\ell(k) + 2m\big(\ell(\mathbf{f}_1) + \ell(\mathbf{f}_2)\big) &\leqslant 2 \cdot 2 \cdot 4^k m + 2m \cdot \big(2^k + 2^k\big) \leqslant 2m \cdot \big(2^{2k+1} + 2^{k+1}\big) \leqslant 2m \cdot \big(2^{k+1} \cdot \big(2^k + 1\big)\big) \\ &\leqslant 2m \cdot \big(2^{k+1}\big)^2 \leqslant 2 \cdot 4^{k+1} m. \end{aligned}$$

(c) We want a small regular expression for $\overline{L_{\ell(\mathbf{f})}(f)}$ and first observe that if we negate $\mathbf{f}$ with DeMorgan, then depth does not increase. Hence $L(R(\bar{\mathbf{f}}))$ has a regular expression of length at most $2 \cdot 4^k m$. Assuming $\overline{L(R(\mathbf{f}))} \cap \{0,1\}^{m \cdot \ell(\mathbf{f})} = L(R(\bar{\mathbf{f}}))$ holds, as we will show next by induction over the structure of the formula, we obtain

$$\overline{L_{\ell(\mathbf{f})}(f)} = \overline{L\big(R(\mathbf{f})\big) \cap W} = \overline{L\big(R(\mathbf{f})\big)} \cup \overline{W} = L\big(R(\bar{\mathbf{f}})\big) \cup \big\{x \in \{0,1\}^*\colon\ |x| \neq m \cdot \ell(\mathbf{f})\big\} \cup \overline{W},$$

since $L(R(\mathbf{f})) \subseteq \{0,1\}^{m \cdot \ell(\mathbf{f})}$. We check whether the input does not consist of repetitions with the regular expression

$$\big((0+1)^*\ 1\ (0+1)^{m-1}\ 0\ (0+1)^*\big) + \big((0+1)^*\ 0\ (0+1)^{m-1}\ 1\ (0+1)^*\big)$$

and cover words of wrong length by $(0+1+\varepsilon)^{m \cdot \ell(\mathbf{f})-1}\ +\ (0+1)^{m \cdot \ell(\mathbf{f})+1}(0+1)^*$.

It remains to show that for any $w \in \{0,1\}^{m \cdot \ell(\mathbf{f})}$ either $w \in L(R(\mathbf{f}))$ or $w \in L(R(\bar{\mathbf{f}}))$. This is not obvious, since w might be a word that does not consist of repetitions.

If $\mathbf{f} = x_i$, then $R(\mathbf{f}) = (0+1)^{i-1}1(0+1)^{m-i}$. On the other hand $\bar{\mathbf{f}} = \overline{x_i}$ and thus $R(\bar{\mathbf{f}}) = (0+1)^{i-1}0(0+1)^{m-i}$. Now assume, that $w_1 \ldots w_m \in \{0,1\}^{m \cdot 1}$, then obviously $w \in L(R(\mathbf{f})) \Leftrightarrow w \notin L(R(\bar{\mathbf{f}}))$, since w_i is either 0 or 1.

Let the assumption hold for $\mathbf{f}_1$ and $\mathbf{f}_2$ and let $\mathbf{f} = \mathbf{f}_1 \vee \mathbf{f}_2$. Thus

$$R(\mathbf{f}) = R(\mathbf{f}_1) \circ (0+1)^{m \cdot \ell(\mathbf{f}_2)} + (0+1)^{m \cdot \ell(\mathbf{f}_1)} \circ R(\mathbf{f}_2)$$

and

$$R(\bar{\mathbf{f}}) = R(\overline{\mathbf{f}_1} \wedge \overline{\mathbf{f}_2}) = R(\overline{\mathbf{f}_1}) \circ R(\overline{\mathbf{f}_2})$$

hold. Let $w \in \{0,1\}^{m\cdot\ell(\mathbf{f})}$, then

$$\begin{aligned} w \in L\big(R(\mathbf{f})\big) \quad &\Leftrightarrow \quad w_1 \dots w_{m\cdot\ell(\mathbf{f}_1)} \in L\big(R(\mathbf{f}_1)\big) \vee w_{m\cdot\ell(\mathbf{f}_1)+1} \dots w_{m\cdot\ell(\mathbf{f})} \in L\big(R(\mathbf{f}_2)\big) \\ &\Leftrightarrow \quad w_1 \dots w_{m\cdot\ell(\mathbf{f}_1)} \notin L\big(R(\overline{\mathbf{f}_1})\big) \vee w_{m\cdot\ell(\mathbf{f}_1)+1} \dots w_{m\cdot\ell(\mathbf{f})} \notin L\big(R(\overline{\mathbf{f}_2})\big) \\ &\Leftrightarrow \quad w_1 \dots w_{m\cdot\ell(\mathbf{f}_1)} \circ w_{m\cdot\ell(\mathbf{f}_1)+1} \dots w_{m\cdot\ell(\mathbf{f})} \notin L\big(R(\overline{\mathbf{f}_1})\big) \circ L\big(R(\overline{\mathbf{f}_2})\big) \\ &\Leftrightarrow \quad w \notin L\big(R(\overline{\mathbf{f}_1}) \circ R(\overline{\mathbf{f}_2})\big) \\ &\Leftrightarrow \quad w \notin L\big(R(\tilde{\mathbf{f}})\big) \end{aligned}$$

holds and the proof for $\mathbf{f} = \mathbf{f}_1 \wedge \mathbf{f}_2$ is analogous.

(d) Since all functions in $\mathcal{C}_1 \cap B_m$ have formula depth at most $c \cdot \log m$, we may assume that all these functions have formulae of depth exactly $c \cdot \log m$ and length exactly $p_1(m) = m^c$. Thus with part (a) $L_{p_1}(f_m)$ coincides with $L(R(\mathbf{f})) \cap W$ and, with part (b), $\overline{L_{p_1}(f_m)}$ has regular expressions of length $O(4^{c\log m} m) = O(m^{2c+1})$. □

Naor and Reingold [12] show that NC^1 contains a strong pseudo-random ensemble for some parameter $\varepsilon > 0$, provided factoring Blum integers is sufficiently hard. More precisely there are some constant c and a hard pseudo-random ensemble $\mathcal{C}_1$ with formula depth at most $c \cdot \log m$ and formula length at most $p_1(m) = m^c$ for functions in $\mathcal{C}_1 \cap B_m$.

Thus we know that (strong pseudo-random) functions from $\mathcal{C}_1 \cap B_m$ have short regular expressions of length at most $t_1^{(1)}(m) = \text{poly}(m)$, whereas we show next that most m-bit functions have only regular expressions of length at least $\Omega(2^m)$.

Lemma 8. *The number of languages described by regular expressions of length at most $t_2^{(1)}(m) = \frac{2^m}{40}$ is bounded by $\sqrt{2^{2^m}} = o(|B_m|)$.*

Proof. We define the rpn-length of a regular expression R as the number of symbols from $\Sigma \cup \{+, \circ, {}^*, \varepsilon, \emptyset\}$ appearing in R, when R is written in reverse Polish notation. A regular expression of length at most t has rpn-length at most $6t$ [6]. At any position in the regular expression in reverse Polish notation there may be one of the seven distinct symbols $0, 1, +, \circ, {}^*, \varepsilon, \emptyset$. Thus we can have at most $\sum_{j \leqslant 6t} 7^j \leqslant 7^{7t} \leqslant 2^{20t}$ distinct regular expressions of rpn-length at most $6t$. The claim follows, since $2^{20 t_2^{(1)}(m)} = 2^{20\frac{2^m}{40}} = 2^{2^{m-1}}$. □

The functional $G^{(1)}$ measures the length of minimal regular expressions. To show that $G^{(1)}$ separates $\mathcal{C}_1$ from random functions, we also need to show, that the language $\overline{L_{p_1}(k_m(f_n))}$ for the restriction $k_m(f_n)$ does not need longer regular expressions than $\overline{L_{p_1}(f_n)}$. $G_m^{(1)}(k_m(f_n)) \leqslant t_1^{(1)}(n)$ holds for functions $f_n \in \mathcal{C}_1 \cap B_n$, because a formula $\mathbf{f}$ for f_n can be transformed into a formula for $k_m(f_n)$ of same depth: We show how to deal with the leaves that are constants after setting $x_1, \dots, x_{n-m}$ to zero. We have to replace these leaves and their parent gates, because our definition of formulae only allows variables as leaves. We consider the last level of $\wedge$ or $\vee$ gates in $\mathbf{f}$. If one of the children is a constant after restricting its value and the gate is equivalent to a variable (e.g. $1 \wedge x_i$), we replace the fixed child by the variable. If the value of the gate is fixed however (e.g. $0 \wedge x_i$), we replace the gate and its children by $x_n \wedge \overline{x_n}$, respectively $x_n \vee \overline{x_n}$.

Hence as a consequence of Lemmas 7 and 8, $G^{(1)}$ separates $\mathcal{C}_1$ from random functions with thresholds $t_1^{(1)}(m) = \Theta(m^{2c+1})$ and $t_2^{(1)}(m) = \frac{2^m}{40}$. Thus we may apply the generic Lemma 4 and obtain that efficient algorithms approximating the length of a shortest regular expression for $\overline{L_{p_1}(f)}$ do not exist. However we have to specify the input not by a truth table but by a dfa.

Proposition 9. *Let $h \in B_m$ and let p be some function of m, then there is a dfa $D_p(h)$ with $\Theta(2^m \cdot p)$ states that accepts $\overline{L_p(h)}$. Moreover $D_p(h)$ can be constructed in time* $\text{poly}(2^m \cdot p)$.

Proof. The dfa $D_p(h)$ consists of a binary tree of depth m rooted at the initial state. A leaf that corresponds to a word x with $h(x) = 0$ gets a self loop, a leaf that corresponds to a word x with $h(x) = 1$ is starting point of a path of length

$(p-1)m$ that can only be followed by inputs with $p-1$ repetitions of x. Each such path leads to a rejecting state and any wrong letter on this path, respectively any word longer than $(p-1)m$ (measured on the path only) leads to an accepting trap state. Each state is accepting, except for those already described as rejecting. The dfa $D_p(h)$ has $\Theta(2^m \cdot p)$ states. □

Observe that Proposition 9 is the only place, where we utilize the dfa specification to recognize $\overline{L_p(h)}$ with relatively few states.

As a consequence of Lemma 4, running time $2^{O(m^l)}$ is insufficient for good approximations and we have obtained our first main result.

Theorem 10. *Suppose that strong pseudo-random functions in B_m with parameter ε and formula depth bounded by $c \cdot \log m$ exist for some c.*

Let App be a randomized approximation algorithm that approximately determines the length of a shortest equivalent regular expression, when given a dfa with k states. Then there is a polynomial poly*, such that for all $l \geqslant 1$, if App runs in time $2^{O((\log k)^l)}$ and achieves an approximation factor $\mu(k) < \frac{k}{\mathrm{poly}((\log k)^{l/\varepsilon})}$, then App must have errors $\epsilon_+ + \epsilon_- \geqslant \frac{2}{3}$.*

Proof. We assume by way of contradiction that the claim is falsified by an approximation algorithm *App*. We show how to determine a good bound on the length of a regular expression for the language $\overline{L_{p_1}(h_m)}$. We then apply Lemma 4, since we obtain a too good approximation for the functional $G^{(1)}(h_m)$ which is defined as the length of a shortest regular expression for $\overline{L_{p_1}(h_m)}$.

In particular, given a truth table for h_m, we apply Proposition 9 and obtain a dfa D for $\overline{L_{p_1}(h_m)}$ in time $2^{O(m)}$. D has $\Theta(2^m \cdot p_1(m))$ states. In the next step, we apply *App* to D and obtain an approximation of $G^{(1)}(h_m)$ with factor $\mu(2^m \cdot p_1(m))$ where $p_1(m) = m^c$ holds. By assumption for any polynomial "poly," there is some $l \geqslant 1$, such that $\mu(2^m \cdot m^c) < \frac{2^m \cdot m^c}{\mathrm{poly}((\log(2^m \cdot m^c))^{l/\varepsilon})}$ holds. Thus the approximation factor is bounded by $\frac{2^m \cdot m^c}{\mathrm{poly}'(m^{l/\varepsilon})}$.

To apply Lemma 4, remember that $G^{(1)}$ separates $\mathcal{C}_1$ from random functions with thresholds $t_1^{(1)}(m) = \Theta(m^{2c+1})$ and $t_2^{(1)}(m) = \frac{2^m}{40}$. For a polynomial $\mathrm{poly}'(n) \geqslant n^{4c}$ and for sufficiently large m,

$$\mu\left(2^m \cdot m^c\right) < \frac{2^m \cdot m^c}{m^{(l/\varepsilon)\cdot 4c}} < \frac{2^m}{m^{(l/\varepsilon)\cdot 3c}} < \frac{t_2^{(1)}(m)}{t_1^{(1)}(m^{l/\varepsilon})}$$

holds, which is a better approximation than allowed by Lemma 4. □

Remark 11. We repeatedly apply the reasoning of Theorem 10. The only difference will be different values of t_1 and t_2. Observe that the approximation factor is then at least $\frac{t_2(m)}{t_1(m^{l/\varepsilon})}$.

The argument shows that there are always dfa's with optimal regular expressions of length $\mathrm{poly}(\log k)$, such that an "efficient" approximation algorithm can only determine regular expressions of length $\frac{k}{\mathrm{poly}(\log k)}$. Thus the original question of Jiang and Ravikumar [8] phrased for regular expressions instead of nfa's, namely whether it is possible to approximate within a polynomial, has a negative answer modulo cryptographic assumptions.

2.2. Nfa's and two-way automata of polynomial size

In this section we use the functionals $G^{(2)}$ and $G^{(3)}$ defined by $G_m^{(2)}(h_m)$, respectively $G_m^{(3)}(h_m)$. $G_m^{(2)}(h_m)$ equals the minimum number of states, respectively $G_m^{(3)}(h_m)$ equals the minimum number of transitions, of an nfa recognizing $\overline{L_{p_1}(h_m)}$. We choose $p_1(m) = m^c$ as defined in the previous section as an upper bound for the length of shortest formulae for functions in $\mathcal{C} \cap B_m$ and derive upper bounds $t_1^{(2)}$ for the number of states and $t_1^{(3)}$ for the number of transitions from the upper bound $t_1^{(1)} = \Theta(m^{2c+1})$ for the length of a shortest regular expression for $\overline{L_{p_1}(h_m)}$. We set $t_1^{(2)} = t_1^{(1)}$, $t_1^{(3)} = (t_1^{(1)})^2$ and observe that the number of states of a minimum nfa is not larger than the length ℓ of an equivalent regular expression and the number of transitions is at most quadratic in ℓ. Thus all functions in $\mathcal{C}_1$ have

nfa's of "size" at most $t_1^{(2)}$, respectively $t_1^{(3)}$. Moreover all but a negligible fraction of languages require nfa's with at least $t_2^{(2)}(m) = 2^{\frac{m}{2}-1}$ states, respectively $t_2^{(3)}(m) = \frac{2^m}{20m}$ transitions.

Lemma 12.

(a) *The number of languages accepted by nfa's with at most* $t_2^{(2)}(m) = 2^{\frac{m}{2}-1}$ *states is bounded by* $\sqrt{2^{m+2^m}} = o(|B_m|)$.
(b) *The number of languages accepted by nfa's with at most* $t_2^{(3)}(m) = \frac{2^m}{20m}$ *transitions is bounded by* $\sqrt{2^{2^m}} = o(|B_m|)$.

Proof. (a) Let $N(k)$ be the number of distinct languages accepted by nfa's with at most k states over a two-letter alphabet. Then $N(k) \leqslant 2k \cdot 2^{2 \cdot k^2}$ [2] and hence

$$N\bigl(t_2^{(2)}(m)\bigr) \leqslant \frac{2}{2} \cdot 2^{\frac{m}{2}} \cdot 2^{2 \cdot (2^{m/2}/2)^2} = 2^{\frac{m}{2}} \cdot 2^{\frac{2}{4} \cdot (2^{m/2})^2} = 2^{\frac{m}{2}} \cdot 2^{\frac{2^m}{2}} = \sqrt{2^{m+2^m}}.$$

(b) We show that there are at most $M(k) = k^{10k}$ languages accepted by nfa's with at most k transitions over a two-letter alphabet. This establishes the claim, if we set $t_2^{(3)}(m) = \frac{2^m}{20m}$, since

$$M\bigl(t_2^{(3)}(m)\bigr) = \left(\frac{2^m}{20m}\right)^{10\frac{2^m}{20m}} \leqslant 2^{10m \cdot \frac{2^m}{20m}} = \sqrt{2^{2^m}}.$$

For any nfa N with s states and k transitions there is an equivalent nfa N' with $s+1$ states, at most $2k$ transitions and exactly one final state. Just add a final state f, make every other state nonfinal and for every transition in N that leads to a final state in N, add a transition to f and keep every other transition.

There are at most $\binom{(s+1)^2}{2k}^2 \cdot s^2 \leqslant s^{8k+2}$ distinct languages over $\{0,1\}$ accepted by nfa's with s states and k transitions, since this is an upper bound for the number of possibilities to place $2k$ transitions for each letter of the alphabet $\{0,1\}$ and the number of choices for the initial and the final state.

We can assume that the number of states is bounded by the number of transitions and hence we have at most $k^{8k+2} \leqslant k^{10k}$ distinct languages. □

We apply Remark 11 with thresholds $t_1^{(2)} = \Theta(m^{2c+1})$ and $t_2^{(2)} = 2^{\frac{m}{2}-1}$ for state minimization and $t_1^{(3)} = \Theta(m^{4c+2})$ and $t_2^{(3)} = \frac{2^m}{20m}$ for transition minimization.

Corollary 13. *Suppose that strong pseudo-random functions in* B_m *with parameter* ε *and formula depth bounded by* $c \cdot \log m$ *exist for some* c.

Let App be a randomized approximation algorithm that approximately determines the number of states (*respectively number of transitions*) *of a minimum equivalent nfa, when given a dfa with* k *states. Then there is a polynomial* poly, *such that for all* $l \geqslant 1$, *if App runs in time* $2^{O((\log k)^l)}$ *and achieves an approximation factor* $\mu(k) < \frac{\sqrt{k}}{\mathrm{poly}((\log k)^{l/\varepsilon})}$ (*respectively* $\mu(k) < \frac{k}{\mathrm{poly}((\log k)^{l/\varepsilon})}$), *then App must have errors* $\epsilon_+ + \epsilon_- \geqslant \frac{2}{3}$.

We cannot expect better bounds with our cryptographic approach, since an arbitrary $h \in B_m$ always has an nfa for $\overline{L_p(h)}$ with $O(2^{\frac{m}{2}} + p \cdot m)$ states, as we show next.

We first construct an nfa $N_{\bar{h}}$ with ε-transitions and less than $4 \cdot 2^{\frac{m}{2}}$ states for $L_1(\bar{h})$. Our nfa consists of a state q_w and p_w for each $w \in \{0,1\}^*$ with $|w| \leqslant \frac{m}{2}$. q_ε is the initial state and p_ε is the only final state. There are transitions $\delta(q_w, \sigma) = \{q_{w\sigma}\}$ for $\sigma \in \{0,1\}$ and $|w| < \frac{m}{2}$. Thus we initially build a binary tree. For $|w| = \frac{m}{2}$, we add ε-transitions $\delta(q_w, \varepsilon) = \{p_x \mid x \in \{0,1\}^{\frac{m}{2}} \wedge h(wx) = 0\}$. The remaining transitions are $\delta(p_{\sigma w}, \sigma) = \{p_w\}$ for $\sigma \in \{0,1\}$ and $|w| < \frac{m}{2}$. There are no transitions for symbols that do not match the first letter of the state's index: $\delta(p_{\sigma w}, 1-\sigma) = \emptyset$.

A word y accepted by $N_{\bar{h}}$ must have length m and must evaluate to $h(y) = 0$, on the other hand for every input y with $h(y) = 0$, there is a path from the initial to the final state.

By adding an ε-transition from the final state to the initial state, we turn $N_{\bar{h}}$ into an nfa for $(L_1(\bar{h}))^*$. Additionally, we accept every word that is not a repetition of a word of length m with $2(m+1)$ states and accept every word that is

not of length $p \cdot m$ with $p \cdot m + 2$ states. The method for removing ε-transitions is well-known and does not increase the number of states.

We finally mention that the assumption of strong pseudo-random functions with small formula depth can be replaced by the weaker assumption of strong pseudo-random functions with two-way dfa's of polynomial size. (Observe that two-way dfa's of polynomial size have the power of nonuniform Logspace, which is at least as powerful as nonuniform NC^1.) We show that two-way dfa's can be simulated efficiently by nfa's after repeating the input suitably often.

Lemma 14. *Let $m, k \in \mathbb{N}$. Then there is a polynomial $p(m)$, such that for any two-way deterministic finite automaton A_m with at most m^k states, there is an nfa N_m with $O(p(m))$ states and transitions accepting the complement of*

$$L_p(A_m) := \left\{ x^{p(m)} \;\middle|\; x \in \{0,1\}^m \wedge A_m \text{ accepts } x \right\}.$$

Proof. Obviously A_m runs for at most $p(m) = m \cdot m^k$ steps on inputs $x \in \{0,1\}^m$, since no cell can be visited twice in the same state. As shown in [13], A_m on input $x \in \{0,1\}^m$ can be simulated by a dfa D_m with $p(m)$ states working on input $x^{p(m)}$. The nfa N_m decides nondeterministically to run D_m (with final and nonfinal states interchanged) or to check whether the input is syntactically incorrect, i.e., verifying inequality or incorrect length. N_m has $t_1(m) = \text{poly}(m)$ states, respectively transitions. □

We can rephrase Corollary 13 with the weaker assumption that pseudo-random functions in B_m exist which are computed by two-way dfa's with at most m^k states for some k. When applying Lemma 4, we have to first redefine the number of repetitions to make sure that a class $\mathcal{C}_2$ of pseudo-random functions can be recognized by two-way dfa's of size at most m^k. We therefore set $p_2(m) = m^{k+1}$ and are guaranteed to find an equivalent nfa recognizing $\overline{L_{p_2}(f_m)}$ (for $f_m \in \mathcal{C}_2 \cap B_m$) with $\Theta(p_2(m))$ states, respectively transitions. Thus $t_1^{(2)}(m)$ and $t_1^{(3)}(m)$ have to be reset accordingly.

$G_m(k_m(f)) \leqslant t_1(n)$ holds, since we can transform a two-way dfa A_n for $L_1(f) = \{x \mid x \in \{0,1\}^n \wedge f(x) = 1\}$ with state set Q into a two-way dfa A_m for $L_1(k_m(f)) = \{y \mid y \in \{0,1\}^m \wedge f(0^{n-m}y) = 1\}$ with the same state set Q. The new initial state q_0' is the state in which A_n enters position $n - m + 1$ for the first time. The transitions for A_m remain the same as for A_n except for the case, when A_m reads the left end marker $\dashv$. We define $\delta(q, \dashv)$ to move right and take the state that A_n reaches, when visiting position $n - m + 1$ for the first time, after starting in q on position $n - m$ with zeros only on positions $1, \ldots, n - m$.

2.3. The minimum consistent dfa problem

In the *minimum consistent dfa* problem, sets $POS, NEG \subseteq \{0,1\}^*$ with $POS \cap NEG = \emptyset$ are given. The goal is to determine the minimum size of a dfa D such that $POS \subseteq L(D)$ and $NEG \cap L(D) = \emptyset$.

Remember our assumption that the class $\mathcal{C}_2$ of functions computable by two-way dfa's with m^k states for inputs of length m contains strong pseudo-random functions. Since two-way dfa's of polynomial size have the power of nonuniform Logspace, this assumption is weaker than the assumption that NC^1 contains strong pseudo-random functions.

To make the transition from two-way to one-way dfa's, we repeat an input $p_2(m) = m^{k+1}$ times and define $G_m^{(4)}(h_m)$ as the minimum size of a dfa accepting $POS = \{x^{p_2} \mid h_m(x) = 1\}$ and rejecting $NEG = \{x^{p_2} \mid h_m(x) = 0\}$. Observe that for any function $f_m \in \mathcal{C}_2 \cap B_m$ we have $G_m^{(4)}(f_m) \leqslant t_1^{(4)}(m) := m^{k+1}$, since any two-way dfa with m^k states can be simulated by a dfa with m^{k+1} states, *provided* the input $x \in \{0,1\}^m$ is repeated $p_2(m) = m^{k+1}$ times. (See the proof of Lemma 14.)

Lemma 15. *$G_m^{(4)}(h_m) \leqslant t_2^{(4)}(m) = \frac{2^m}{6m}$ holds for at most $\sqrt{2^{2^m}} = o(|B_m|)$ functions in B_m.*

Proof. Let $K(s)$ be the number of distinct languages accepted by dfa's with at most s states over a two-letter alphabet. Then $K(s) \leqslant s^{3s}$ [2] and hence

$$K\left(t_2^{(4)}(m)\right) \leqslant \left(\frac{2^m}{6m}\right)^{3\frac{2^m}{6m}} \leqslant 2^{3m\frac{2^m}{6m}} = \sqrt{2^{2^m}}.$$

The claim holds, since different functions h_m have different consistent dfa's. □

Thus $G_m^{(4)}$ separates $\mathcal{C}_2$ from random functions with thresholds $t_1^{(4)}, t_2^{(4)}$ and we apply Remark 11. However, when given the function h_m we build the sets $POS = \{x^{p_2} \mid h_m(x) = 1\}$ and $NEG = \{x^{p_2} \mid h_m(x) = 0\}$ instead of building a dfa for $L_p(h_m)$. We obtain the following theorem.

Theorem 16. *Suppose that strong pseudo-random functions in B_m with parameter ε exist which are computed by two-way dfa's with at most m^k states for some k.*

Let App be a randomized approximation algorithm that approximately determines the number of states of a minimum consistent dfa and let $N = \sum_{x \in POS \cup NEG} |x|$ be the input length. Then there is a polynomial poly*, such that for all $l \geqslant 1$, if App runs in time $2^{O((\log N)^l)}$ and achieves an approximation factor $\mu(N) < \frac{N}{\text{poly}((\log N)^{l/\varepsilon})}$, then App must have errors $\epsilon_+ + \epsilon_- \geqslant \frac{2}{3}$.*

Thus, assuming that minimal consistent dfa's have size $opt = \text{poly}(\log N)$, efficient approximation algorithms are doomed to determine consistent dfa's of size at least $\frac{N}{\text{poly}(\log N)} \geqslant 2^{opt^{1/l}} \cdot d^{\beta}$, where $\beta < 1$, l is sufficiently large and $d \leqslant N$ is the number of classified examples. This result is stronger than the result of at least $opt^{\alpha} \cdot d^{\beta}$ due to Kearns and Valiant [9]. The stronger result is a consequence of our use of pseudo-random functions instead of one-way functions. (See also Naor and Reingold [12].)

3. Approximately minimizing nfa's or regular expressions

We now assume that the language is specified concisely, i.e., as an nfa or a regular expression and prove in this scenario strong inapproximability results.

3.1. Unary nfa's and regular expressions

We begin by investigating unary languages, i.e., languages over a one-letter alphabet. A unary regular language is recognized by a dfa that starts with a possibly empty path and ends in a nonempty cycle.

In our proofs, we only consider cyclic languages, i.e., languages that can be recognized by dfa's consisting of a cycle only. In particular, we say that a language $L \subseteq \{a\}^*$ is d-cyclic iff

$$\left(a^j \in L \;\Leftrightarrow\; a^{j+d} \in L\right)$$

holds for any $j \in \mathbb{N}$ and call d a period of L. A smallest period is called the minimal period and any period is a multiple of the minimal period.

Our first result shows that efficient approximations for state minimization within the factor $\frac{\sqrt{m}}{\ln m}$ for a given nfa with m states do *not* exist, if $P \neq NP$. This result remains true for the number of transitions (respectively number of symbols in regular expressions).

We can improve the inapproximability result, if we require the *construction* of a small nfa or regular expression. We show for this case, that for a given nfa or regular expression A of size m and any $\delta > 0$, no efficient algorithm can determine an nfa or regular expression A' equivalent to A of size at most $opt \cdot m^{1-\delta}$, if $P \neq NP$.

Stockmeyer and Meyer [16] show, that the nonuniversality problem $L(N) \neq \Sigma^*$ is *NP*-complete for regular expressions and nfa's N, if we consider only unary languages. Since our argument is based on their construction, we show the proof.

Fact 17. [16] *For a unary nfa N, it is NP-hard to decide, if $L(N) \neq \{a\}^*$.*

Proof. We reduce 3SAT to the universe problem for unary nfa's. Let Φ be a 3CNF-formula over n variables with m clauses. Let $p_1, \ldots, p_n$ be the first n primes and set $D := \prod_{i=1}^{n} p_i$. According to the Chinese remainder theorem, the function $\mu : \mathbb{N} \to \mathbb{N}^n$ with $\mu(x) = (x \bmod p_1, \ldots, x \bmod p_n)$ is injective, if we restrict the domain to $\{0, \ldots, D-1\}$. We call x a code for an assignment, if $\mu(x) \in \{0,1\}^n$.

We construct an nfa N_{Φ} that accepts $\{a\}^*$ iff Φ is not satisfiable. We first make sure, that $L_{0,\Phi} = \{a^k \mid k \text{ is not a code}\}$ is accepted. Therefore, for every prime p_i ($p_i > 2$) we construct a deterministic cycle that accepts the words a^j with $j \not\equiv 0 \pmod{p_i} \wedge j \not\equiv 1 \pmod{p_i}$. So there are 2 nonfinal states and $(p_i - 2)$ final states in the

cycle. For every clause C of Φ with variables $x_{i_1}, x_{i_2}, x_{i_3}$ we construct a deterministic cycle C^* of length $p_{i_1} p_{i_2} p_{i_3}$. C^* accepts

$$\left\{a^k \;\middle|\; \text{the assignment } k \bmod p_{i_j} \text{ for } x_{i_j}\ (j = 1, 2, 3) \text{ does not satisfy } C\right\}.$$

Since the falsifying assignment is unique for the three variables in question, exactly one state is accepting in C^*. We turn the set of disjoint cycles into an nfa by adding an initial state q_0 and transitions from q_0 to the second state of each cycle.

The construction can be done in time polynomial in the length of Φ. If there is a word $a^j \notin L(N_\Phi)$, then j is a code for a satisfying assignment. On the other hand every satisfying assignment has a code j and a^j is not accepted by N_Φ. □

Observe that the number of transitions is the same as the number of states $|Q_\Phi|$ plus the number of cycles. An equivalent regular expression R_Φ with length at most $2 \cdot |Q_\Phi|$ can obviously be constructed.

We set $L_\Phi = L(N_\Phi)$ for the automaton N_Φ constructed above. Observe that L_Φ is a union of cyclic languages and hence is cyclic itself. Obviously if $\Phi \notin 3SAT$, then a minimum nfa or regular expression for L_Φ has size 1. We show that, for $\Phi \in 3SAT$, every nfa accepting L_Φ must have at least $\sum_{i=2}^{n} p_i$ states, which implies Theorem 18.

Theorem 18. *Given an nfa or regular expression N of size n, it is impossible to efficiently approximate the minimal size of an nfa or regular expression for $L(N)$ within a factor of $\frac{\sqrt{n}}{\ln n}$ unless $P = NP$.*

We first determine a lower bound for the period of L_Φ.

Lemma 19. *For any given 3CNF-formula $\Phi \in 3SAT$ the minimal period of L_Φ is either $D := \prod_{i=1}^{n} p_i$ or $\frac{D}{2}$.*

Proof. L_Φ is D-cyclic, since D is the least common multiple of the cycle lengths of N_Φ. Assume that neither D nor $\frac{D}{2}$ is the minimal period of L_Φ. Then there is $i \geqslant 2$, such that $d = \frac{D}{2p_i}$ is a period of L_Φ. We know that $a^{qp_i+2} \in L_{0,\Phi}$ for every $q \in \mathbb{N}$, because $qp_i + 2$ does not represent a code. Since $L_{0,\Phi} \subseteq L_\Phi$ and we assume that L_Φ is d-cyclic, a^{qp_i+2+rd} belongs to L_Φ for every $r \in \mathbb{N}$ as well.

On the other hand, since $L_\Phi \neq \{a\}^*$, there is a word $a^l \notin L_\Phi$, and so $a^{l+td} \notin L_\Phi$ for every $t \in \mathbb{N}$. It is a contradiction, if we find $q, r, t \in \mathbb{N}$, so that $qp_i + 2 + rd = l + td$, since the corresponding word has to be in L_Φ because of the left-hand side of the equation and cannot be in L_Φ because of the right-hand side.

$$\begin{aligned}
\exists q, r, t\colon \quad qp_i + 2 + rd = l + td \quad &\Leftrightarrow \quad \exists q, r, t\colon \quad qp_i = l - 2 + (t - r)d \\
&\Leftrightarrow \quad \exists q\colon \quad qp_i \equiv l - 2 \pmod{d} \\
&\Leftrightarrow \quad \exists q\colon \quad q \equiv (l-2)\, p_i^{-1} \pmod{d}.
\end{aligned}$$

The multiplicative inverse of p_i modulo d exists, since $\gcd(p_i, d) = 1$, and we have obtained the desired contradiction. □

We need a linear relation between the number of clauses and variables in the CNF-formula. Hence we consider $E3SAT - E5$, the satisfiability problem for formulae with exactly 3 literals in every clause and every variable appearing in exactly 5 distinct clauses. It is well known that $E3SAT - E5$ is NP-complete.

The following lemma determines a lower bound for the size of an nfa equivalent to N_Φ, if Φ is satisfiable.

Lemma 20. *Let $\Phi \in E3SAT - E5$ and assume that Φ consists of m clauses. Then any nfa for L_Φ has at least $cm^2 \ln m$ states for some constant c.*

Proof. We know from Lemma 19, that $L(N_\Phi)$ is either minimally D-cyclic or $\frac{D}{2}$-cyclic with $D = \prod_{i=1}^{n} p_i$ where n is the number of variables in Φ. Jiang, McDowell and Ravikumar [7] show that any nfa accepting a unary cyclic language with a period $\frac{D}{2}$ that factorizes as $\prod_{i=2}^{n} p_i$ must have at least $\sum_{i=2}^{n} p_i$ states. We estimate the size of the

ith prime number p_i by $i \ln i \leqslant p_i \leqslant 2i \ln i$ (the lower bound holds for all $i \in \mathbb{N}$, the upper bound holds for $i \geqslant 3$). So every nfa for L_Φ must have at least

$$\sum_{i=2}^{n} p_i \geqslant \sum_{i=1}^{n} i \ln i \geqslant \int_1^n x \ln x \, dx \geqslant \frac{n^2}{4} \ln n$$

states. We have $5n = 3m$ and thus we can express the lower bound for the number of states in relation to the number of clauses in Φ: Every nfa for L_Φ must have at least $cm^2 \ln m$ states for some constant c. □

Finally we determine an upper bound for the size of the nfa N_Φ.

Lemma 21. *Let Φ be a 3CNF formula with m clauses and exactly 5 appearances of every variable. Then the nfa N_Φ as well as the regular expression R_Φ has size at most $O(m^4(\ln m)^3)$ and at least $\Omega(m^2 \ln m)$.*

Proof. The number of states in a cycle for a clause is a product of three primes. So there are at most $m \cdot p_n^3 = O(m(m \ln m)^3)$ states in all of these cycles. The cycles recognizing $L_{0,\Phi}$ have $\sum_{i=2}^{n} p_i = \Theta(n^2 \ln n)$ states, where n is the number of variables of Φ. Since $n = \Theta(m)$ the claim follows. Remember that the number of transitions in N_Φ and the number of symbols in R_Φ is linearly related to the number of states in N_Φ. □

Proof of Theorem 18. Assume that the polynomial time deterministic algorithm A approximates the minimum size of an equivalent nfa or regular expression for a given regular nfa or regular expression of size s within the factor $\frac{\sqrt{s}}{\ln s}$. We show that the satisfiability problem can be decided in polynomial time.

Let Φ be the given input for the $E3SAT - E5$ problem, where we assume that Φ has n variables and m clauses. We construct the nfa N_Φ or regular expression R_Φ as in Fact 17. If Φ is not satisfiable, then size 1 is the optimum, and according to Lemma 21 the algorithm *App* claims that an equivalent nfa or regular expression with size at most

$$\frac{\sqrt{s}}{\ln s} = \frac{\sqrt{(O(m^4(\ln m)^3))}}{\ln(\Omega(m^2(\ln m)))} = o\big(m^2 \ln m\big)$$

exists. Since, by Lemma 20 any satisfiable formula Ψ generates a language L_Ψ with nfa's of size $\Omega(m^2 \ln m)$, and thus regular expressions of size $\Omega(m^2 \ln m)$, the claimed size $(\sqrt{s}/\ln s)$ is asymptotically smaller than the minimum size for L_Ψ and hence with the help of *App*, we can decide if Φ is satisfiable within polynomial time. □

Now we consider approximation algorithms that construct small equivalent nfa's or regular expressions opposed to just determining the size and obtain an even stronger inapproximability result.

Theorem 22. *Let N be an arbitrary unary nfa or regular expression of size m. Let opt be the size of a minimal equivalent nfa, respectively regular expression. For any $\delta > 0$, if $P \neq NP$, then no efficient algorithm can determine an nfa or regular expression N' equivalent to N with size at most $opt \cdot m^{1-\delta}$.*

Proof. Let N be an nfa (regular expression) constructed in the proof of Fact 17. N has the property that either $opt = 1$ or $opt > \frac{\sqrt{m}}{\ln m}$ and it is NP-complete to distinguish the two cases.

Suppose that there is a constant $\delta > 0$ and an efficient algorithm A that computes an nfa, respectively a regular expression, $A(N)$ equivalent to N with $\text{size}(A(N)) \leqslant opt \cdot \text{size}(N)^{1-\delta}$. If we apply A on its output again, then

$$\text{size}\big(A\big(A(N)\big)\big) \leqslant opt \cdot \text{size}\big(A(N)\big)^{1-\delta} \leqslant opt^2 \cdot \text{size}(N)^{(1-\delta)^2}.$$

If we repeat this process k times, then $\text{size}(A^k(N)) \leqslant opt^k \cdot \text{size}(N)^{(1-\delta)^k}$. So for $k \geqslant \frac{-2}{\log(1-\delta)}$, we have $\text{size}(A^k(N)) \leqslant opt^k \cdot \text{size}(N)^{\frac{1}{4}}$, and hence for m large enough, $\text{size}(A^k(N)) \leqslant \frac{\sqrt{m}}{\ln m}$ follows, if $opt = 1$, respectively $\text{size}(A^k(N)) \geqslant opt > \frac{\sqrt{m}}{\ln m}$ holds otherwise. □

3.2. General nfa's or regular expressions

Our negative results for *general* alphabets are based on the well known proof [11] of the PSPACE-completeness of regular expression nonuniversality: Given a regular expression R, is $L(R) \neq \Sigma^*$? The PSPACE-completeness of regular expression nonuniversality implies the PSPACE-completeness of the exact minimization of nfa's and regular expressions.

The proof of [11] shows, that for an arbitrary language $L \in \text{PSPACE}$ there is a polynomial time transformation T such that $w \in L \Leftrightarrow L(T(w)) \neq \Sigma^*$, where $L(T(w))$ is the language described by the nfa, respectively regular expression $T(w)$. We restrict ourselves to languages $L \in \mathcal{L}$ where $\mathcal{L}$ is the class of languages that can be accepted by deterministic in-place Turing machines.[2] Our inapproximability result utilizes the following observation.

Lemma 23. *For any given language $L \in \mathcal{L}$ there is a deterministic in-place Turing machine M_L recognizing L with a single accepting state. M_L runs for at least 2^n steps on every input $w \in L$ of length n.*

Proof. Let M be some deterministic in-place Turing machine which accepts L and has only one accepting state q_f. We construct a Turing machine M_L that has all the states and transitions M has. However, whenever M_L enters q_f, it counts in binary from 0^n to 1^n, changes to a new state q'_f, when reaching 1^n, and stops. q'_f is the only state in which M_L accepts and q'_f causes M_L to stop. □

Assume that $M = (Q_M, \Sigma_M, \Gamma_M, \delta, q_0, \{q_f\})$ is a Turing machine with the properties stated in Lemma 23 which recognizes the PSPACE-complete language $L(M)$. (A padding argument shows that $\mathcal{L}$ contains PSPACE-complete languages.) We reduce the word problem for $L(M)$ to the minimization problem for regular expressions and nfa's. In particular for an input w of M, we construct a regular expression R_w, which describes exactly all words which are *not* concatenations of consecutive legal configurations starting from configuration $q_0 w$ leading to the unique accepting state q_f.

R_w is defined over the alphabet $\Sigma = (Q_M \times \Gamma_M) \cup \Gamma_M \cup \{\#\}$ which allows us to describe sequences of configurations of M separated by the new symbol #. Every legal configuration has length exactly $n = |w|$ and is a word in $\Gamma_M^* \cdot (Q_M \times \Gamma_M) \cdot \Gamma_M^*$. The symbol $[q, a] \in Q_M \times \Gamma_M$ represents the head position of M on a cell with contents a while M is in state q. R_w is a union of the regular expressions R_1, R_2, R_3 and R_4.

- R_1 describes all words which do not start with $\#[q_0, w_1]w_2 \dots w_n\#$, i.e.,

  $$R_1 = N_\# + \#\big(N_{[q_0,w_1]} + [q_0, w_1]\big(N_{w_2} + w_2\big(N_{w_3} + w_3\big(\dots(N_{w_n} + w_n N_\#)\dots\big)\big)\big)\big)$$

  with $N_a = \varepsilon + (\Sigma \setminus a)\Sigma^*$. Observe that we use the abbreviation

  $$\Sigma = \sum_{\sigma \in \Sigma} \sigma \quad \text{and} \quad (\Sigma \setminus a) = \sum_{\sigma \in \Sigma \setminus \{a\}} \sigma.$$

- R_2 describes all words which do not contain $[q_f, \gamma]$ for any $\gamma \in \Gamma_M$, i.e.,

  $$R_2 = \left(\sum_{a \in \Sigma \setminus (\{q_f\} \times \Gamma_M)} a \right)^*.$$

- $R_3 = \Sigma^*(\Sigma \setminus \#)$ describes all words which do not end with #.
- R_4 describes any illegal change on the tape between consecutive configurations: In a legal sequence y of configurations, for every triple $y_{i-1}y_i y_{i+1} \in \Sigma^3$ of consecutive letters, the new middle symbol y_{i+n+1} is a function of $y_{i-1}y_i y_{i+1}$. Thus for any illegal sequence x of configurations either there is a position i with $x_{i+n+1} \neq x_i$, if the head is not scanning x_i, or x_{i+n+1} is not updated correctly.
  In particular, for each $a_1, a_2, a_3 \in \Gamma_M \cup \{\#\}$ accept every word x which does not have a_2 at the corresponding middle position in the next configuration by the regular expression

  $$\Sigma^* \circ a_1 a_2 a_3 \circ \Sigma^{n-1} \circ (\Sigma \setminus a_2) \circ \Sigma^*.$$

[2] $\mathcal{L}$ coincides with DSPACE($O(n)$), but considering only Turing machines that work in-place simplifies the proof.

Finally, for each $a_1, a_2 \in \Gamma_M$ and each $[q,a] \in Q_M \times \Gamma_M$ with $\delta(q,a) = (q', b, \rightarrow)$ for some $q' \in Q_M$ and some $b \in \Gamma_M$ accept wrong sequences by

$$\Sigma^* \circ [q,a]a_1a_2 \circ \Sigma^{n-1} \circ \big(\Sigma \setminus [q', a_1]\big) \circ \Sigma^* + \Sigma^* \circ a_1[q,a]a_2 \circ \Sigma^{n-1} \circ (\Sigma \setminus b) \circ \Sigma^* + \Sigma^* \circ a_1a_2[q,a] \circ \Sigma^{n-1} \circ (\Sigma \setminus a_2) \circ \Sigma^*.$$

Treat those $[q,a] \in Q_M \times \Gamma_M$ with $\delta(q,a) = (q', b, \leftarrow)$ accordingly. R_4 is the union of all the regular expressions just described.

The regular expression R_w has $m \leqslant |w| \cdot 3 \cdot |\Sigma|^4 = O(|w|)$ symbols. Thus an equivalent nfa with m states can be constructed. It is easy to verify, that there is an equivalent nfa with $O(|w|)$ transitions.

If M rejects w, then $L(R_w)$ coincides with Σ^*. However, if M accepts w, then the configuration sequence y corresponding to the accepting computation is not covered by R_w and it is the only word not in $L(R_w)$.

Any accepting computation y has length at least $2^{|w|}$, since M is a Turing-Machine as described in Lemma 23. We show that $\Sigma^* \setminus \{y\}$ requires nfa's with at least $|w|$ states. Every dfa which excludes a single word of length at least $2^{|w|}$ needs at least $2^{|w|}$ states, thus every equivalent nfa needs at least $|w|$ states. Hence, if $L(R_w) = \Sigma^* \setminus \{y\}$ for some y with $|y| \geqslant 2^{|w|}$, then every nfa which accepts $L(R_w)$ needs at least $|w|$ states. Thus $|w|$ is a lower for the number of transitions in any equivalent nfa as well as for the size of any equivalent regular expression.

Thus, if $w \notin L(M)$, then $L(R_w)$ can be recognized by an nfa with one state or $|\Sigma|$ transitions, respectively a regular expression of size $|\Sigma|$, whereas for $w \in L(M)$, nfa's with at least $|w|$ states or transitions, respectively regular expressions of size at least $|w|$ are required.

Since we efficiently constructed R_w with $m = O(|w|)$ symbols and the efficient construction of an equivalent nfa with $O(|w|)$ states and transitions is possible as well, we have found the desired gap.

Theorem 24. *Unless $P =$ PSPACE, it is impossible to efficiently approximate the size of a minimal nfa or regular expression describing $L(A)$ within an approximation factor of $o(m)$ when given an nfa or a regular expression A with m states, transitions or symbols, respectively.*

Standard encoding arguments show that this PSPACE-completeness result is true for regular expressions or nfa's over any alphabet Σ with $|\Sigma| \geqslant 2$.

4. Conclusions and open problems

We have been able to verify inapproximability of nfa's or regular expressions either for given nfa's or regular expressions (utilizing $P \neq NP$, respectively $P \neq$ PSPACE) or for given dfa's (assuming the existence of strong pseudo-random functions in nonuniform NC^1, respectively nonuniform Logspace). Below we list our results.

NFA AND REGULAR EXPRESSION MINIMIZATION

INSTANCE: An nfa N with k states over a binary alphabet.
SOLUTION: The size of a smallest nfa equivalent with N.
MEASURE: Number of transitions or number of states.
BAD NEWS: Not approximable within $o(k)$.
ASSUMPTION: $P \neq$ PSPACE.
REMARK: More generally the same complexity result holds, if a given nfa or regular expression is to be transformed into an equivalent nfa or regular expression of minimal size.
REFERENCE: Theorem 24.

UNARY NFA OR REGULAR EXPRESSION MINIMIZATION

INSTANCE: An nfa N with k states over a unary alphabet.
SOLUTION: The size of a smallest nfa equivalent with N.

MEASURE: Number of transitions or number of states.
BAD NEWS: Not approximable within $\frac{\sqrt{k}}{\ln k}$.
ASSUMPTION: $P \neq NP$.
REMARK: More generally the same complexity result holds, if a given unary nfa or regular expression is to be transformed into an equivalent nfa or regular expression of minimal size.
REFERENCE: Theorem 18.

CONSTRUCTIVE UNARY NFA OR REGULAR EXPRESSION MINIMIZATION

INSTANCE: An nfa N with k states over a unary alphabet.
SOLUTION: A smallest nfa equivalent with N.
MEASURE: Number of transitions or number of states.
BAD NEWS: Not approximable within $k^{1-\delta}$ for any δ.
ASSUMPTION: $P \neq NP$.
REMARK: More generally the same complexity result holds, if a given unary nfa or regular expression is to be transformed into an equivalent nfa or regular expression of minimal size.
REFERENCE: Theorem 22.

DFA $\rightarrow$ NFA MINIMIZATION (STATES)

INSTANCE: A dfa D with k states over a binary alphabet.
SOLUTION: The size of a smallest nfa equivalent with D.
MEASURE: Number of states.
BAD NEWS: Not approximable within $\frac{\sqrt{k}}{\text{poly}(\log k)}$.
ASSUMPTION: Strong pseudo-random functions in Logspace.
REFERENCE: Corollary 13.

DFA $\rightarrow$ NFA MINIMIZATION (TRANSITIONS)

INSTANCE: A dfa D with k states over a binary alphabet.
SOLUTION: The size of a smallest nfa equivalent with D.
MEASURE: Number of transitions.
BAD NEWS: Not approximable within $\frac{k}{\text{poly}(\log k)}$.
ASSUMPTION: Strong pseudo-random functions in Logspace.
REFERENCE: Corollary 13.

DFA $\rightarrow$ REGULAR EXPRESSION MINIMIZATION

INSTANCE: A dfa D with k states over a binary alphabet.
SOLUTION: The size of a smallest regular expression equivalent with D.
MEASURE: Number of symbols.
BAD NEWS: Not approximable within $\frac{k}{\text{poly}(\log k)}$.
ASSUMPTION: Strong pseudo-random functions in NC^1.
REFERENCE: Theorem 10.

MINIMUM CONSISTENT DFA

INSTANCE: Two finite sets P, N of binary strings.
SOLUTION: The minimal size of a dfa accepting all strings in P and rejecting all strings in N.

MEASURE: Number of states in the automaton.
BAD NEWS: Not approximable within $\frac{|P|+|N|}{\text{poly}(\log(|P|+|N|))}$.
ASSUMPTION: Strong pseudo-random functions in Logspace.
REFERENCE: Theorem 16.

Our results for nfa or regular expression minimization, for given nfa's or regular expressions, are best possible and include the number of states, the number of transitions, respectively the length as resources to be minimized. The situation is different for a given dfa, since ideally we would like to have hardness results relative to the assumption $P \neq NP$. Moreover, when minimizing the number of states, our methods are only able to show approximation factors of size at least $\frac{\sqrt{k}}{\text{poly}(\log k)}$ for a given dfa of size k and sharper bounds are not excluded. Finally the complexity of nfa or regular expression minimization remains open, if a language $L_1(h_m)$ is specified by a truth table for h_m.

The exact complexity of the unary nfa or regular expression minimization problem remains open, since it is not excluded that efficient algorithms with approximation factor $\frac{n}{f(n)}$ exist for some function $f(n)$ growing slower than any root of n. Finally we mention the unary dfa $\rightarrow$ nfa minimization problem, whose exact status is also to be resolved.

UNARY DFA $\rightarrow$ NFA MINIMIZATION

INSTANCE: A dfa D with k states over a unary alphabet.
SOLUTION: The size of a smallest nfa equivalent with D.
MEASURE: Number of states or transitions.
BAD NEWS: Optimal solution cannot be determined efficiently.
ASSUMPTION: $NP \not\subseteq DTIME(n^{O(\log n)})$
REMARK: Cyclic case can be approximated within $1 + \ln k$.
REFERENCE: [7], [3].

References

[1] J.-M. Champarnaud, F. Coulon, Nfa reduction algorithms by means of regular inequalities, Theoret. Comput. Sci. 327 (3) (2004) 241–253.
[2] M. Domaratzki, D. Kisman, J. Shallit, On the number of distinct languages accepted by finite automata with n states, J. Automata Languages Combinatorics 7 (4) (2002) 469–486.
[3] G. Gramlich, Probabilistic and nondeterministic unary automata, in: B. Rovan, P. Vojtás (Eds.), MFCS, in: Lecture Notes in Comput. Sci., vol. 2747, Springer-Verlag, 2003, pp. 460–469.
[4] G. Gramlich, G. Schnitger, Minimizing nfa's and regular expressions, in: V. Diekert, B. Durand (Eds.), STACS '05, in: Lecture Notes in Comput. Sci., vol. 3404, Springer-Verlag, 2005.
[5] L. Ilie, G. Navarro, S. Yu, On nfa reductions, in: J. Karhumäki, H.A. Maurer, G. Paun, G. Rozenberg (Eds.), Theory Is Forever, in: Lecture Notes in Comput. Sci., vol. 3113, Springer-Verlag, 2004, pp. 112–124.
[6] L. Ilie, S. Yu, Follow automata, Inform. and Comput. 186 (1) (2003) 140–162.
[7] T. Jiang, E. McDowell, B. Ravikumar, The structure and complexity of minimal nfa's over a unary alphabet, Int. J. Found. Comput. Sci. 2 (2) (1991) 163–182.
[8] T. Jiang, B. Ravikumar, Minimal nfa problems are hard, SIAM J. Comput. 22 (6) (1993) 1117–1141.
[9] M.J. Kearns, L.G. Valiant, Cryptographic limitations on learning boolean formulae and finite automata, J. ACM 41 (1) (1994) 67–95.
[10] O. Matz, A. Potthoff, Computing small nondeterministic finite automata, in: Proc. of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Dpt. of CS., Univ. of Aarhus, 1995, pp. 74–88.
[11] A.R. Meyer, L.J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, in: Proc. 13th Ann. IEEE Symp. on Switching and Automata Theory, 1972, pp. 125–129.
[12] M. Naor, O. Reingold, Number-theoretic constructions of efficient pseudo-random functions, J. ACM 51 (2) (2004) 231–262.
[13] L. Pitt, M.K. Warmuth, Prediction-preserving reducibility, J. Comput. System Sci. 41 (3) (1990) 430–467.
[14] L. Pitt, M.K. Warmuth, The minimum consistent dfa problem cannot be approximated within any polynomial, J. ACM 40 (1) (1993) 95–142.
[15] A.A. Razborov, S. Rudich, Natural proofs, J. Comput. System Sci. 55 (1) (1997) 24–35.
[16] L.J. Stockmeyer, A.R. Meyer, Word problems requiring exponential time: Preliminary report, in: Proc. of the 5th Annual ACM Symposium on Theory of Computing, 1973, pp. 1–9.