On Minimizing Regular Expressions Without Kleene Star

Hermann Gruber[1], Markus Holzer[2], and Simon Wolfsteiner[3]

[1] Knowledgepark GmbH, Leonrodstr. 68, 80636 München, Germany
hermann.gruber@kpark.de
[2] Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
holzer@informatik.uni-giessen.de
[3] Institut für Diskrete Mathematik und Geometrie, TU Wien,
Wiedner Hauptstr. 8–10, 1040 Wien, Austria
simon.wolfsteiner@gmail.com

Abstract. Finite languages lie at the heart of literally every regular expression. Therefore, we investigate the approximation complexity of minimizing regular expressions without Kleene star, or, equivalently, regular expressions describing finite languages. On the side of approximation hardness, given such an expression of size s, we prove that it is impossible to approximate the minimum size required by an equivalent regular expression within a factor of $O\left(\frac{s}{(\log s)^{\delta}}\right)$ if the running time is bounded by a quasipolynomial function depending on δ, for every $\delta > 1$, unless the exponential time hypothesis (ETH) fails. For approximation ratio $O(s^{1-\delta})$, we prove an exponential-time lower bound depending on δ, assuming ETH. The lower bounds apply to alphabets of constant size. On the algorithmic side, we show that the problem can be approximated in polynomial time within $O(\frac{s \log \log s}{\log s})$, with s being the size of the given regular expression. For constant alphabet size, the bound improves to $O(\frac{s}{\log s})$. Finally, we devise a family of superpolynomial approximation algorithms with approximation ratios matching the lower bounds, while the running times are just above the lower bounds excluded by the exponential time hypothesis.

1 Introduction

Regular expressions are used in many applications and it is well known that for each regular expression, there is a finite automaton that defines the same language and *vice versa*. Automata are very well suited for programming tasks and immediately translate to efficient data structures. On the other hand, regular expressions are well suited for human users and therefore are often used as interfaces to specify certain patterns or languages.

Regarding performance optimization, putting effort into the internal representation inside the regex engine is of course a natural choice. On the other hand, most of the time, developers use existing APIs but are not willing, or able, to change the source code of these. Thus, sometimes practitioners, as well as theory researchers, see a need for optimizing the input regular expressions, as witnessed by questions in pertinent Q&A forums.[4] More often than not, the regular expressions under consideration are in fact without Kleene star, that is, they describe only finite languages. Moreover, recently the descriptional complexity of finite languages attracted new attention because of its close connection to well-known measures for the complexity of formal proofs in first-order predicate logic [7].

The problem of minimizing regular expressions accepting infinite languages is PSPACE-complete, and even attaining a sublinear approximation ratio is already equally hard [11]. When restricting to finite languages, there is of course the classical reduction from 3-SAT to the equivalence problem for regular expressions without star [19, Thm. 2.3]. It is sometimes overlooked that, unlike the case of infinite languages, the classical reduction *does not imply* hardness of the corresponding minimization problem. In fact, no lower bounds for minimizing regular expressions without Kleene star were known prior to the present work at all.[5] Also, there are hardness results for minimizing acyclic nondeterministic finite automata [3, 12], and also for minimizing acyclic context-free grammars [16]—but nothing thus far for regular expressions without star. In this work, we fill this gap by proving tight lower and upper approximability bounds.

As a byproduct of our proofs, we also substantially improve the inapproximability bound for minimizing nondeterministic finite automata in the case of finite languages, and give the first nontrivial approximation guarantee. The results are summarized in Table 1.

Recent years have seen a renewed interest in the analysis of computational problems, among others, on formal languages, since more fine-grained hardness results can be achieved based on the exponential time hypothesis (ETH) than with more traditional proofs based on the assumption $\mathsf{P} \neq \mathsf{NP}$ [1, 2, 6, 9, 24, 25]. Namely, ETH posits that there is no algorithm that decides 3-SAT formulae with n variables in time $2^{o(n)}$, and is just one among other strong hypotheses that were used during the last decade to perform fine-grained complexity studies; for a short survey on results obtained by some of these hypotheses, we refer to [26].

We contribute a fine-grained analysis of approximability and inapproximability for minimizing regular expressions without Kleene star. On the side of approximation hardness, given such an expression of size s, we prove that it is

[4] See for example the following questions drawn from various sites: (i) P. Krauss: Minimal regular expression that matches a given set of words, URL: https://cs.stackexchange.com/q/72344, Accessed: 2021-01-02, (ii) J. Mason: A released perl with trie-based regexps! URL: http://taint.org/2006/07/07/184022a.html, Accessed: 2020-07-21, (iii) pdanese (StackOverflow username): Speed up millions of regex replacements in Python 3, URL: https://stackoverflow.com/q/42742810, Accessed: 2021-01-02, (iv) P. Scheibe: RegEx performance: Alternation vs Trie, URL: https://stackoverflow.com/q/56177330, Accessed: 2021-01-02, and (v) Ch. Xu: Minimizing size of regular expression for finite sets, URL: https://cstheory.stackexchange.com/q/16860, Accessed: 2021-01-02.

[5] See, e. g., item (v) of the previous footnote.

	general	unary languages	finite languages
DFA	exactly solvable in P [18]		
NFA	PSPACE-complete [23], *not* approximable within $o(n)$ [11], trivially approximable within $O(n)$	coNP-hard [23], *not* approximable within $o(n)$ [11, 12], trivially approximable within $O(n)$	DP-hard [12], *not* approximable within $\frac{\sqrt{n}}{2^{(\log n)^{7/8-\epsilon}}}$ [4, 13], trivially approximable within $O(n)$.
			Not approximable within $n^{1-\varepsilon}$ (Cor. 16), approximable within $\frac{n}{\log n}$ for fixed alphabet (Thm. 18)
RE			coNP-hard (Cor. 7), *not* approximable within $n^{1-\varepsilon}$ (Cor. 7), approximable within $\frac{n \log \log n}{\log n}$ (Thm. 11)

Table 1. Coarse-grained overview of known and new results for minimization problems. For better comparability, approximability is understood to be in polynomial time, and hardness results are under classical assumptions such as $\mathsf{P} \neq \mathsf{NP}$.

impossible to approximate the minimum size required by an equivalent regular expression within a factor of $O\left(\frac{s}{(\log s)^{\delta}}\right)$ if the running time is bounded by a quasipolynomial function depending on δ, for every $\delta > 1$, unless the ETH fails. For approximation ratio $O(s^{1-\delta})$, we prove an exponential-time lower bound depending on δ, assuming ETH. These lower bounds apply to alphabets of constant size. On the algorithmic side, we show that the problem can be approximated in polynomial time within $O(\frac{s \log \log s}{\log s})$, where s is the size of the given regular expression. For constant alphabet size, the bound improves to $O(\frac{s}{\log s})$. Finally, we devise a family of superpolynomial approximation algorithms that attain the performance ratios of the lower bounds, while their running times are just above those excluded by the ETH. For instance, we attain an approximation ratio of $O\left(\frac{s}{(\log s)^{\delta}}\right)$ in time $2^{O\left((\log s)^{\delta}\right)}$ for $\delta > 1$, and a ratio of $s^{1-\delta}$ in time $2^{O(s^{\delta})}$ for $\delta > 0$. These running times nicely fit with the excluded running times of $2^{o\left((\log s)^{\delta}\right)}$ and of $2^{o(s^{\delta})}$, respectively, for these approximation ratios.

This paper is organized as follows: in the next section, we define the basic notions relevant to this paper. Section 3 covers approximation hardness results for various runtime regimes based on the ETH. Then in Section 4, these negative results are complemented with approximation algorithms that neatly attain these lower bounds. In Section 5, we transfer some of these results to the minimization problem for nondeterministic finite automata. To conclude this work, we indicate possible directions for further research in the last section. Due to space constraints, some of the proofs are omitted.

2 Preliminaries

We assume that the reader is familiar with the basic notions of formal language theory as contained in [18]. In particular, let Σ be an *alphabet* and Σ^* the *set* *of all words over the alphabet Σ* including the *empty word ε*. The *length of a word w* is denoted by $|w|$, where $|\varepsilon| = 0$, and the total number of occurrences of the alphabet symbol a in w is denoted by $|w|_a$. In this paper, we mainly deal with finite languages. The *order* of a finite language L is the length of a longest word belonging to L. A finite language $L \subseteq \Sigma^*$ is called *homogeneous* if all words in the language have the same length. We say that a homogeneous language $L \subseteq \Sigma^n$ is *full* if L is equal to Σ^n. For languages $L_1, L_2 \subseteq \Sigma^*$, the *left quotient of L_1 and L_2* is defined as $L_1^{-1}L_2 = \{\, v \in \Sigma^* \mid \text{there is some } w \in L_1 \text{ such that } wv \in L_2 \,\}$. If L_1 is a singleton, i.e., $L_1 = \{w\}$, for some word $w \in \Sigma^*$, we omit braces, that is, we write $w^{-1}L_2$ instead of $\{w\}^{-1}L_2$. The set $w^{-1}L_2$ is also called the *derivative of L_2 w.r.t. the word w*. In order to fix the notation, we briefly recall the definition of regular expressions and the languages described by them.

The *regular expressions* over an alphabet Σ are defined inductively in the usual way:[6] $\emptyset$, ε, and every letter $a \in \Sigma$ is a regular expression; and when E and F are regular expressions, then $(E + F)$, $(E \cdot F)$, and $(E)^*$ are also regular expressions. The language defined by a regular expression E, denoted by $L(E)$, is defined as follows: $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\varepsilon\}$, $L(a) = \{a\}$, $L(E+F) = L(E) \cup L(F)$, $L(E \cdot F) = L(E) \cdot L(F)$, and $L(E^*) = L(E)^*$. The *alphabetic width* or *size* of a regular expression E over an alphabet Σ, denoted by $\mathsf{awidth}(E)$, is defined as the total number of occurrences of letters of Σ in E. For a regular language L, we define its alphabetic width, $\mathsf{awidth}(L)$, as the minimum alphabetic width among all regular expressions describing L.

We are interested in regular expression minimization w.r.t. its alphabetic width (or, equivalently, its size). An algorithm that returns near-optimal solutions is called an *approximation algorithm.* Assume that we are working on a minimization problem in which each potential solution has a positive cost and that we wish to find a near-minimal solution. We say that an approximation algorithm for the problem has a *performance guarantee of $\rho(n)$* if for any input of size n, the cost C of the solution produced by the approximation algorithm is *within a factor of $\rho(n)$* of the cost C^* of a minimal solution: $\frac{C}{C^*} \leq \rho(n)$. If the approximation algorithm is running in polynomial time, we speak of a *polynomial-time approximation algorithm.* For most of our hardness results, we assume the exponential time hypothesis (ETH) introduced in [20].

Exponential time hypothesis. There is a positive constant c such that the satisfiability of a formula in 3-CNF with n variables and m clauses cannot be decided in time $2^{cn}(m + n)^{O(1)}$.

In particular, using the Sparsification Lemma [20], the ETH implies that there is no algorithm running in time $2^{o(m)}$ that decides satisfiability of a 3-SAT formula with m clauses. This is of course a much stronger assumption than $\mathsf{P} \neq \mathsf{NP}$. For more background on the topic, see, e.g., the survey [21].

[6] For convenience, parentheses in regular expressions are sometimes omitted and concatenation is sometimes simply written as juxtaposition. The priority of operators is specified in the usual fashion: concatenation is performed before union, and star before both concatenation and union.

3 Inapproximability

In this section, we will show that, for a given regular expression without Kleene star, the minimum size required by an equivalent regular expression cannot be approximated within a certain factor if the running time is within certain bounds, assuming the ETH. We start off with an estimate of the required regular expression size for a language which we shall use as gadget.

Lemma 1. *Let $P_r = \{ xy \in \{0,1\}^* \mid |x| = |y| = r \text{ and } x = y^R \}$ denote the language of all binary palindromes of length $2r$. Then $2^r \leq \mathsf{awidth}(P_r) \leq 2^{r+2} - 4$.*

The upper bound is in fact tight, yet proving this takes a lot more effort [15]. Notwithstanding, the simple lower bound above suffices for the purpose of the present work.

It was shown in [14] that taking the quotient of a regular language can cause at most a quadratic blow-up in required regular expression size. *Vice versa*, the alphabetic width of a language can be lower-bounded by the order of the square root of the alphabetic width of any of its quotients. For our reduction, we need a tighter relationship. This is possible if we resort to special cases. Let us consider homogeneous languages and expressions in more detail. First, we need a simple observation that turns out to be very useful in the forthcoming considerations.

Lemma 2. *Let $L \subseteq \Sigma^n$ be a homogeneous language. If E is a regular expression describing L, then any subexpression of E describes a homogeneous language, too.*

Now we are ready to consider the descriptional complexity of quotients of homogeneous languages in detail.

Lemma 3. *Let $L \subseteq \Sigma^n$ be a homogeneous language. Then $\mathsf{awidth}(w^{-1}L) \leq \mathsf{awidth}(L)$, for any word $w \in \Sigma^*$.*

We build upon the classical coNP-completeness proof of the inequality problem for regular expressions without star given in [19, Thm. 2.3]. We recall the reduction to make this paper more self-contained.

Theorem 4. *Let φ be a formula in 3-DNF with n variables and m clauses. Then a regular expression ζ can be computed in time $O(m \cdot n)$ such that the language $Z = L(\zeta)$ is homogeneous and Z is full if and only if φ is a tautology.*

Proof. Let $\varphi = \bigvee_{i=1}^{m} c_i$ be a formula in 3-DNF. We can assume without loss of generality that no clause c_i contains both x_j and $\overline{x}_j$ as a literal. For each clause c_i, let $\zeta_i = \zeta_{i1}\zeta_{i2}\cdots\zeta_{in}$, where

$$\zeta_{ij} = \begin{cases} (0+1) & \text{if both } x_j \text{ and } \overline{x}_j \text{ are not literals in } c_i, \\ 0 & \text{if } \overline{x}_j \text{ is a literal in } c_i, \\ 1 & \text{if } x_j \text{ is a literal in } c_i. \end{cases}$$

Let $\zeta = \zeta_1 + \zeta_2 + \cdots + \zeta_m$. Clearly, $Z = L(\zeta) \subseteq \{0,1\}^n$. Let w in $\{0,1\}^n$. Then w is in Z if and only if w satisfies some clause c_i. Thus $Z = \{0,1\}^n$ if and only if φ is a tautology. This completes the reduction. $\square$

Now if we wanted to apply the reduction from Theorem 4 to the minimization problem for regular expressions, the trouble is that we cannot predict the minimum required regular expression size for $Z = L(\zeta)$ in case it is not full. To make this happen, we use a similar trick as recently used in [16] for the analogous case of context-free grammars. In the following lemma, we embed the language P_r of all binary palindromes of length $2r$ together with the language $Z = L(\zeta)$ (as defined in Theorem 4) into a more complex language Y. Depending on whether or not Z is full, the alphabetic width of Y is at most linear or at least quadratic, respectively, in m. Recall that m refers to the number of clauses in the given 3-DNF formula φ.

Lemma 5. *Let φ be a formula in 3-DNF with n variables and m clauses and let ζ be the regular expression constructed in Theorem 4. Furthermore, let*

$$Y = Z \cdot \{0,1\}^{2r} \cup \{0,1\}^n \cdot P_r,$$

where $Z = L(\zeta)$ and P_r, for $r \leq m$, is defined as in Lemma 1. Then $\mathsf{awidth}(Y) = O(m)$ *if Z is full, and* $\mathsf{awidth}(Y) = \Omega(2^r)$ *if Z is not full.*

The above lemma can serve as a gap introducing reduction. For example, if we take $r = 2\log m$, then $\Omega(2^r)$ is in $\Omega(m^2)$. Now we are in the position to state our first inapproximability result.

Theorem 6. *Let E be a regular expression without Kleene star of size s, and let δ be a constant such that $0 < \delta \leq \frac{1}{2}$. Then no deterministic $2^{o\left(s^\delta\right)}$-time algorithm can approximate* $\mathsf{awidth}(L(E))$ *within a factor of $o(s^{1-\delta})$, unless ETH fails.*

Proof. We give a reduction from the 3-DNF tautology problem as in Lemma 5. That is, given a formula φ in 3-DNF with n variables and m clauses, we construct a regular expression that generates the language $Y = Z \cdot \{0,1\}^{2r} \cup \{0,1\}^n \cdot P_r$. The sets P_r and Z are defined as in Lemma 1 and Theorem 4, respectively. Here, the set Y features some carefully chosen parameter r, which will be fixed later on. For now, we only assume $2\log m \leq r \leq m$.

Next, we need to show that the reduction is correct in the sense that if Z is full, then $\mathsf{awidth}(Y)$ is asymptotically strictly smaller than in the case where it is not full. By Lemma 5, it follows that $\mathsf{awidth}(Y) = O(m)$ if Z is full and $\mathsf{awidth}(Y) = \Omega(2^r)$, otherwise. Thus, the reduction is correct, since we have assumed that $r \geq 2\log m$, and consequently $2^r = \omega(m)$.

It is easy to see that the running time of the reduction is linear in the size of the constructed regular expression describing Y. Now we estimate the size of that regular expression. Recall from Theorem 4 that the regular expression ζ has size $O(m \cdot n)$. Because formula φ is a 3-DNF, we have $m \geq n/3$, and so the size of ζ is in $O(m^2)$. The set $\{0,1\}^{n+2r}$ admits a regular expression of size $O(m+n) = O(m)$; and $\mathsf{awidth}(P_r) = \Theta(2^r)$ by Lemma 1. Since we have assumed that $r \geq 2\log m$, the order of magnitude of the constructed regular expression is $s = \Theta(2^r)$.

Now we need to fix the parameter r in our reduction; let us pick $r = \frac{1}{\delta} \cdot \log m$. Recall that the statement of the theorem requires $\frac{1}{\delta} \geq 2$, thus we have $r \geq 2 \log m$. So this is a valid choice for the parameter r—in the sense that the reduction remains correct.

Towards a contradiction with the ETH, assume that there is an algorithm A_δ approximating the alphabetic width within $o\left(s^{1-\delta}\right)$ running in time $2^{o\left(s^\delta\right)}$. Then A_δ could be used to decide whether Z is full as follows: the putative approximation algorithm A_δ returns a cost C that is at most $o(s^{1-\delta})$ times the optimal cost C^*, that is, $C = o(s^{1-\delta}) \cdot C^* = o(s^{1-\delta}) \cdot \mathsf{awidth}(Y)$.

On the one hand, if Z is full, then $\mathsf{awidth}(Y) = O(m)$ by Lemma 5. In this case, the hypothetical approximation algorithm A_δ returns a cost C with $C = o(m \cdot s^{1-\delta}) = o\left(m \cdot \Theta\left(m^{\frac{1}{\delta}}\right)^{1-\delta}\right) = o\left(m^{\frac{1}{\delta}(1-\delta)+1}\right) = o\left(m^{\frac{1}{\delta}}\right) = o\left(2^r\right)$. In the second step of the above calculation, we used the fact that $s = \Theta(2^r) = \Theta(m^{\frac{1}{\delta}})$ and in the last step, we used the fact that we chose r as $r = \frac{1}{\delta} \cdot \log m$, which is equivalent to $2^r = m^{\frac{1}{\delta}}$.

On the other hand, in case Z is not full, then Lemma 5 states that $\mathsf{awidth}(Y) = \Omega\left(2^r\right)$. Using the constants implied by the O-notation, the size returned by algorithm A_δ could thus be used to decide, for large enough m, whether Z is full, and thus by Theorem 4 whether the 3-DNF formula φ is a tautology.

It remains to show that the running time of A_δ in terms of m is in $2^{o(m)}$, which contradicts the ETH. Recall again that $s = \Theta(m^{\frac{1}{\delta}})$; we thus can express the running time of the algorithm A_δ in terms of m, namely, $2^{o\left(s^\delta\right)} = 2^{o\left(\Theta(m^{\frac{1}{\delta}})^\delta\right)} = 2^{o\left((c \cdot m^{\frac{1}{\delta}})^\delta\right)} = 2^{o(m)}$, for some constant c, which yields the desired contradiction. □

Assuming ETH, the above proof also implies that the problem cannot be solved exactly in time $2^{o(\sqrt{s})}$. The inapproximability result can be stated more simply when using the classical hardness assumption $\mathsf{P} \neq \mathsf{NP}$:

Corollary 7. *Let E be a regular expression without Kleene star of size s, and let δ be a constant with $0 < \delta < 1$. Then no deterministic polynomial-time algorithm can approximate $\mathsf{awidth}(L(E))$ within a factor of $s^{1-\delta}$, unless $\mathsf{P} = \mathsf{NP}$.*

Proof. The reduction in Theorem 6 is from a coNP-complete problem and runs in polynomial time for every choice of $\delta \geq \frac{1}{2}$. Observe that it suffices to show approximation hardness for $\delta \leq \frac{1}{2}$, since the weaker hardness result for $\delta > \frac{1}{2}$ is then implied. □

Again, assuming ETH, we can change the parameter r in the reduction in Theorem 6 to trade a sharper inapproximability ratio against a weaker lower bound on the running time.

Theorem 8. *Let E be a regular expression without Kleene star of size s, and let δ be a constant with $\delta > 1$. Then no deterministic $2^{o(\log s)^\delta}$-time algorithm can approximate $\mathsf{awidth}(L(E))$ within a factor of $o\left(s/(\log s)^\delta\right)$, unless ETH fails.*

4 Approximability

From the previous section, we know that there are severe limits on what we can expect from efficient approximation algorithms. In this section, we present different approximation algorithms for minimizing regular expressions describing finite languages. Each of them introduces a new algorithmic hook, some of which might be useful in implementations. We start off with an algorithm that requires the input to be specified non-succinctly as a list of words. In case the alphabet size is sufficiently large, listing simply all words is enough; otherwise we construct a deterministic finite automaton and further distinguish on the number of states. This leads to the following result.

Theorem 9. *Let L be a finite language given as a list of words, with s being the sum of the word lengths. Then $\mathsf{awidth}(L)$ can be approximated in deterministic polynomial time within a factor of $O(\frac{s}{\sqrt{\log s}})$.*

Recall that the minimal deterministic finite automaton can be exponentially larger than regular expressions in the worst case, also for finite languages [22]. Also, the conversion from deterministic finite automata to regular expressions is only quasipolynomial in the worst case. These facts of course affect the performance guarantee. Nevertheless, we believe that the scheme from the proof of Theorem 9 is worth a look, since the minimal deterministic finite automaton may eliminate a lot of redundancy in practice. Furthermore, the algorithm works equally if we are able to construct a nondeterministic finite automaton which is smaller than the minimal deterministic finite automaton. To this end, some recently proposed effective heuristics for size reduction of nondeterministic automata could be used [5].

Admittedly, regular expressions are exponentially more succinct than a list of words and our inapproximability results crucially rely on that. So, we now turn to the second approximation algorithm. It makes use of the fact that if a given regular expression E describes very short words only, then it is not too difficult to produce a regular expression that is noticeably more succinct than E. In that case, the algorithm builds a trie, which then can be converted into an equivalent regular expression of size linear in the trie.

For the purpose of this paper, a *trie* (also known as *prefix tree*) is simply a tree-shaped deterministic finite automaton with the following properties:

1. The edges are directed away from the root, i.e., towards the leaves.
2. The root is the start state.
3. All leaves are accepting states.
4. Each edge is labelled with a single alphabet symbol.

The last condition is needed if we want to bound the size of an equivalent regular expression in terms of the nodes in the trie. The following lemma seems to be folklore; the observation is used, e.g., in [17].

Lemma 10. *Let T be a trie with n nodes accepting L. Then an equivalent regular expression of alphabetic width at most $n-1$ can be constructed in deterministic polynomial time from T.*

Now we have collected all tools for an approximation algorithm that works with regular expressions as input, which even comes with an improved approximation ratio.

Theorem 11. *Let E be a regular expression without Kleene star of alphabetic width s. Then* $\mathsf{awidth}(L(E))$ *can be approximated in deterministic polynomial time within a factor of* $O\left(\frac{s \log\log s}{\log s}\right)$.

Proof. We again start with a case distinction by alphabet size.

1. The size of the alphabet used in L is at most $\log s$. We further distinguish the cases in which the order of $L(E)$, i.e., the length of a longest word in $L(E)$, is less than $\frac{\log s}{\log\log s}$ or not. The order of $L(E)$ can be easily computed recursively, in polynomial time, by traversing the syntax tree of E. We consider two subcases:
   (a) The order of $L(E)$ is less than $\frac{\log s}{\log\log s}$. We enumerate the words in $L(E)$, e.g., by performing a membership test for each word of length less than $\frac{\log s}{\log\log s}$. Then we use a standard algorithm to construct a trie for $L(E)$. The worst case for the size of T is when L contains all words of length less than $\frac{\log s}{\log\log s}$. Then T is a full $(\log s)$-ary trie of height $\frac{\log s}{\log\log s}$. All nodes are accepting, giving a one-to-one correspondence between the number of nodes in T and the number of words in $L(T)$. That is, the number of nodes in T is equal to $\sum_{i=0}^{\frac{\log s}{\log\log s}-1}(\log s)^i = O\left(\frac{(\log s)^{\frac{\log s}{\log\log s}}}{\log s}\right)$. Using the fact that $(\log s)^{\frac{\log s}{\log\log s}} = s$, this is in $O\left(\frac{s}{\log s}\right)$ and we can construct an equivalent regular expression of that size in deterministic polynomial time by virtue of Lemma 10.
   (b) The order of $L(E)$ is at least $\frac{\log s}{\log\log s}$. We make use of the observation that the order of $L(E)$, i.e., the length of a longest word in $L(E)$, is a lower bound on the required regular expression size, as observed, e.g., in [8, Proposition 6]. That is, the optimal solution is at least of size $\frac{\log s}{\log\log s}$ and thus the regular expression E given as input is already a feasible solution that is at most $\frac{s \log\log s}{\log s}$ times larger than the optimal solution.
2. The size of the alphabet used in L is greater than $\log s$. The size of the alphabet used in L is likewise a lower bound on the required regular expression size and, similarly to the previous case, the input is a feasible solution that is at most $\frac{s}{\log s}$ times greater than the optimal solution size.

This proves the stated claim. □

For alphabets of constant size, the performance ratio can be slightly improved—by a factor of $\log\log s$.

Theorem 12. *Let E be a regular expression without Kleene star of alphabetic width s over a fixed k-ary alphabet. Then* $\mathsf{awidth}(L(E))$ *can be approximated in deterministic polynomial time within a factor of* $O\left(\frac{s}{\log s}\right)$.

A better performance ratio can be achieved if we allow a superpolynomial running time.

Theorem 13. *Let E be a regular expression without Kleene star of alphabetic width s, and let $f(s)$ be a time constructible*[7] *function with $f(s) = \Omega(\log s)$. Then $\mathsf{awidth}(L(E))$ can be approximated in deterministic time $2^{O(f(s))}$ within a factor of $O\left(\frac{s \log f(s)}{f(s)}\right)$.*

Proof. First, as in Theorem 11, we make a case distinction by alphabet size, and then distinguish by the order of the language. Recall that the order of the language can be computed in polynomial time in a recursive manner on the syntax tree of E. The main new ingredient of this proof is a brute-force search for an optimal solution, powered by a context-free grammar that efficiently generates the search space of candidate regular expressions. So, we again start with distinguishing by alphabet size:

1. The size k of the alphabet used in $L(E)$ is at most $f(s)$. Again, we consider two subcases:
   (a) The order of $L(E)$ is less than $f(s)/\log f(s)$. We make use of the fact that there is a context-free grammar generating all regular expressions describing finite languages over the alphabet used in E. Such a grammar can be used to enumerate all regular expressions of size less than $f(s)/\log f(s)$ with polynomial delay [10]. For finite languages, there is an efficient grammar generating at most $O(f(s))^{f(s)/\log f(s)}$ of these candidates[8] in total [17, Prop. 8.3]. Observe that $O(f(s))^{f(s)/\log f(s)} = 2^{O(f(s))}$. For each enumerated candidate regular expression C and each word w of length less than $f(s)/\log f(s)$, we test whether $w \in L(E)$, and if so, we verify that $w \in L(C)$. If C passes all these tests, we can safely conclude that $L(E) \subseteq L(C)$. To verify whether $L(C) \subseteq L(E)$, we enumerate the words in $L(E)$ and build a trie T that accepts the language. Notice that the trie has at most $f(s)^{O(f(s)/\log f(s))} = 2^{O(f(s))}$ nodes. Since T is a deterministic finite automaton, it can be easily complemented, and we can apply the usual product construction—with the position automaton of C—to check whether $L(C) \cap \Sigma^* \setminus L(T) = \emptyset$. In this way, each candidate regular expression can be tested with a running time bounded by a polynomial in $2^{O(f(s))}$. Recall that the total number of candidates is in $2^{O(f(s))}$, and that the candidates can be enumerated

[7] We say that a function $f(n)$ is *time constructible* if there exists an $f(n)$ time-bounded multitape Turing machine M such that for each n there exists some input on which M actually makes $f(n)$ moves [18].

[8] The grammar in [17, Prop. 8.3] does not generate all valid regular expressions, but incorporates some performance tweaks. These tweaks perfectly fit our purpose: while the grammar does not generate all feasible solutions, it still generates at least one optimal solution. More precisely, given a finite language L with $\mathsf{awidth}(L) = k$, the context-free grammar is guaranteed to enumerate a regular expression of alphabetic width k for it.

with polynomial delay. We conclude that in this case, if $L(E)$ admits an equivalent regular expression of size at most $f(s)/\log f(s)$, an *optimal* solution can be found by exhaustive search with a running time bounded by $2^{(O(f(s)))^{O(1)}} \cdot 2^{(O(f(s)))^{O(1)}} = 2^{O(f(s))}$. The performance ratio is $\frac{s \log f(s)}{f(s)}$.

(b) The order of $L(E)$ is at least $f(s)/\log f(s)$. Again, the order of $L(E)$ is a lower bound on required regular expression size. Thus the regular expression E given as input is already a feasible solution with performance ratio $\frac{s \log f(s)}{f(s)}$.

2. The size of the alphabet used in $L(E)$ is greater than $f(s)$. Similarly, the size of the used alphabet is a lower bound on the required regular expression size. Thus, the regular expression E given as input is a feasible solution, whose performance ratio is $\frac{s}{f(s)}$ in this case.

This completes the proof. □

Observe that although the statement of Theorem 13 specializes to Theorem 11 if we set $f(s) = \log s$, the two proofs nevertheless use different algorithms. Both approaches will have their own merits and their own tradeoffs between running time and performance guarantees when put to practice. Again, in case the alphabet size is bounded, we can slightly improve the performance guarantee of the previous theorem—compare with Theorem 12.

Theorem 14. *Let E be a regular expression without Kleene star of alphabetic width s over a fixed k-ary alphabet, and let $f(s)$ be a time constructible function with $f(s) = \Omega(\log s)$. Then $\mathsf{awidth}(L(E))$ can be approximated in deterministic time $2^{O(f(s))}$ within a factor of $O\left(\frac{s}{f(s)}\right)$.*

To compare this with our inapproximability results, we pick $f(s) = s^{\delta}$, for some $\delta \leq \frac{1}{2}$, to obtain an approximation ratio of $s^{1-\delta}$ in time $2^{O(s^{\delta})}$. Here, Theorem 6 rules out an approximation ratio of $o(s^{1-\delta})$ within a running time of $2^{o\left(s^{\delta}\right)}$. Another pick is $f(s) = (\log s)^{\delta}$, for some $\delta > 1$, yielding an approximation ratio $O\left(\frac{s}{(\log s)^{\delta}}\right)$ in time $2^{O(\log s)^{\delta}}$. In contrast, Theorem 8 rules out an approximation ratio of $o\left(\frac{s}{(\log s)^{\delta}}\right)$ in time $2^{o(\log s)^{\delta}}$. In both cases, the upper bound asymptotically matches the obtained lower bounds, and thus there remains little room for improvements, unless the ETH fails.

5 Minimizing Nondeterministic Finite Automata

In this section, we show that several of our results apply *mutatis mutandis* to the problem of minimizing acyclic nondeterministic finite automata, i.e., those accepting finite languages.

Theorem 15. *Let A be an s-state acyclic nondeterministic finite automaton, and let δ be a constant such that $0 < \delta \leq \frac{1}{2}$. Then no deterministic $2^{o\left(s^{\delta}\right)}$-time algorithm can approximate the nondeterministic state complexity of $L(A)$ within a factor of $O(s^{1-\delta})$, unless ETH fails.*

Corollary 16. *Let A be an s-state acyclic nondeterministic finite automaton, and let δ be a constant with $0 < \delta < 1$. Then no deterministic polynomial-time algorithm can approximate the nondeterministic state complexity of $L(A)$ within a factor of $O(s^{1-\delta})$, unless* P= NP. $\square$

The quasipolynomial-time inapproximability result carries over as well:

Theorem 17. *Let A be an s-state acyclic nondeterministic finite automaton, and let δ be a constant with $\delta > 1$. Then no deterministic $2^{o(\log s)^{\delta}}$-time algorithm can approximate the nondeterministic state complexity of $L(A)$ within a factor of $o(s/(\log s)^{\delta})$, unless ETH fails.* $\square$

Regarding positive approximability results, we cannot use the entire toolkit that we have developed for regular expressions. For instance, the size of the used alphabet does not bound the number of states needed. Also, even for binary alphabets, the number of nondeterministic s-state finite automata is in $2^{\Omega(s^2)}$, which renders the enumeration of automata with few states less feasible. At least, the polynomial-time approximation for bounded alphabet size carries over:

Theorem 18. *Let A be an s-state acyclic nondeterministic finite automaton over a fixed k-ary alphabet. Then the nondeterministic state complexity of $L(A)$ can be approximated in deterministic polynomial time within a factor of $O\left(\frac{s}{\log s}\right)$.*

6 Conclusion

We conclude by indicating some possible directions for further research. First, we would like to continue with investigating inapproximability bounds within polynomial time based on the strong exponential time hypothesis (SETH). Further topics are exact exponential-time algorithms and parameterized complexity. In addition to the natural parameter of desired solution size, the order of the finite language and the alphabet size seem to be natural choices.

Given the practical relevance of the problem we investigated, we think that implementing some of the ideas from the above approximation algorithms is worth a try. Also, POSIX regular expressions restricted to finite languages are a more complex model than the one we investigated, but a more practical one as well. Although we would rather not expect better approximability bounds in that model, we suspect that character classes and other mechanisms can offer practical hooks for reducing the size of regular expressions.

Acknowledgments. We would like to thank Michael Wehar for some discussion, and the anonymous reviewers for their valuable comments.

References

1. A. Abboud, A. Backurs, and V. V. Williams. If the Current Clique Algorithms Are Optimal, so Is Valiant's Parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2015.
2. K. Bringmann, A. Grønlund, and K. G. Larsen. A Dichotomy for Regular Expression Membership Testing. In *Proceedings of the* 58*th Annual IEEE Symposium on Foundations of Computer Science*, pages 307–318, Berkeley, California, USA, October 2017. IEEE.
3. P. Chalermsook, S. Heydrich, E. Holm, and A. Karrenbauer. Nearly tight approximability results for minimum biclique cover and partition. In A. S. Schulz and D. Wagner, editors, *Proceedings of the* 22*th Annual European Symposium on Algorithms*, number 8737 in LNCS, pages 235–246, Wroclaw, Poland, September 2014. Springer.
4. P. Chalermsook, S. Heydrich, E. Holm, and A. Karrenbauer. Nearly tight approximability results for minimum biclique cover and partition. In A. S. Schulz and D. Wagner, editors, *Proceedings of the* 22*th Annual European Symposium on Algorithms*, number 8737 in LNCS, pages 235–246. Springer, 2014.
5. L. Clemente and R. Mayr. Efficient reduction of nondeterministic automata with application to language inclusion testing. *Logical Methods in Computer Science*, 15(1), 2019.
6. M. de Oliveira Oliveira and M. Wehar. On the fine grained complexity of finite automata non-emptiness of intersection. In *Proceedings of the* 24*nd International Conference on Developments in Language Theory*, number 12086 in LNCS, pages 69–82, Tampa, Florida, USA, March 2020. Springer.
7. S. Eberhard and St. Hetzl. On the compressibility of finite languages and formal proofs. *Information and Computation*, 259:191–213, 2018.
8. Keith Ellul, Bryan Krawetz, Jeffrey Shallit, and Ming-wei Wang. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics*, 10(4):407–437, 2005.
9. H. Fernau and A. Krebs. Problems on Finite Automata and the Exponential Time Hypothesis. *Algorithms*, 10(1), 2017.
10. Ch. C. Florêncio, J. Daenen, J. Ramon, J. Van den Bussche, and D. Van Dyck. Naive infinite enumeration of context-free languages in incremental polynomial time. *Journal of Universal Computer Science*, 21(7):891–911, 2015.
11. G. Gramlich and G. Schnitger. Minimizing NFA's and Regular Expressions. *Journal of Computer and System Sciences*, 73(6):908–923, September 2007.
12. H. Gruber and M. Holzer. Computational complexity of NFA minimization for finite and unary languages. In *Preproceedings of the* 1*st International Conference on Language and Automata Theory and Applications*, Technical Report 35/07, pages 261–272, Tarragona, Spain, March 2007. Research Group on Mathematical Linguistics, Universitat Rovira i Virgili.
13. H. Gruber and M. Holzer. Inapproximability of nondeterministic state and transition complexity assuming P $\neq$ NP. In T. Harju, J. Karhumäki, and A. Lepistö, editors, *Proceedings of the* 11*th International Conference Developments in Language Theory*, number 4588 in LNCS, pages 205–216, Turku, Finland, July 2007. Springer.
14. H. Gruber and M. Holzer. Language operations with regular expressions of polynomial size. *Theoretical Computer Science*, 410(35):3281–3289, August 2009.

15. H. Gruber and M. Holzer. Optimal regular expressions for palindromes of given length. In F. Bonchi and S. J. Puglisi, editors, *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science*, Leibniz International Proceedings in Informatics. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021. Accepted for publication.
16. H. Gruber, M. Holzer, and S. Wolfsteiner. On minimal grammar problems for finite languages. In M. Hoshi and S. Seki, editors, *Proceedings of the 22nd International Conference on Developments in Language Theory*, number 11088 in LNCS, pages 342–353, Kyoto, Japan, September 2018. Springer.
17. H. Gruber, J. Lee, and J. Shallit. Enumerating regular expressions and their languages. arXiv:1204.4982 [cs.FL], April 2012.
18. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
19. H. B. Hunt, III. On the time and tape complexity of languages I. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, pages 10–19, Austin, Texas, USA, April–May 1973. ACM.
20. R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
21. D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the European Association for Theoretical Computer Science*, 105:41–72, 2011.
22. R. Mandl. Precise bounds associated with the subset construction on various classes of nondeterministic finite automata. In *Proceedings of the 7th Princeton Conference on Information and System Sciences*, pages 263–267, March 1973.
23. A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential time. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE Society Press, October 1972.
24. F. Mráz, D. Průša, and M. Wehar. Two-dimensional pattern matching against basic picture languages. In M. Hospodár and G. Jirásková, editors, *Proceedings of the 24th International Conference on Implementation and Application of Automata*, number 11601 in LNCS, pages 209–221, Košice, Slovakia, July 2019. Springer.
25. M. Wehar. Hardness results for intersection non-emptiness. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming, Part II*, number 8573 in LNCS, pages 354–362, Copenhagen, Denmark, July 2014. Springer.
26. V. Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In B. Sirakov, P. Ney de Souza, and M. Viana, editors, *Proceedings of the International Congress of Mathematicians*, pages 3447–3487, Rio de Janeiro, Brazil, April 2018. World Scientific.