Deciding the Density Type of a Given Regular Language*

Stavros Konstantinidis and Joshua Young

Department of Mathematics and Computing Science,
Saint Mary's University, Halifax, Nova Scotia, B3H 3C3 Canada
s.konstantinidis@smu.ca, jyo04@hotmail.com

Abstract. In this paper, the density of a language is the function that returns, for each n, the number of words in the language of length n. We consider the question of deciding whether the density of a given regular language L is exponential or polynomial. This question can be answered in linear time when L is given via a DFA. When L is given via an NFA, we show that L has exponential density if and only if the NFA has a strongly connected component (SCC) in which two equal length walks from the same state have different labels. This characterization leads to a simple quadratic time algorithm. However, a more elegant approach produces a linear time algorithm whose proof of correctness involves the theorem of Fine and Wilf and the greatest common divisor (gcd) of the lengths of all cycles in the SCC. We have implemented both the quadratic and linear time algorithms using the FAdo library for automata, and present results of a few test cases.

Keywords: algorithm, automaton, complexity, density, strongly connected components, regular language

1 Introduction

Following [11], we define the density of a language L to be the function that returns, for every nonnegative integer n, the number of words in L of length n. This concept is of central importance in language theory. In particular, [11] and [9] characterize regular languages of exponential density, where the characterization of [9] leads to a linear time algorithm for deciding whether a regular language is of exponential density when L is given via a *deterministic* finite automaton (DFA). This characterization is very simple: the DFA has a state that belongs to two different cycles—we note that the same idea was used in [3] in the context of encoding data into DNA languages that are described via certain DFAs.

Here, we consider the question of characterizing regular languages of exponential density when they are given via *nondeterministic* finite automata (NFAs). Our characterization is that the NFA has a strongly connected component (SCC) containing two walks of the same length, starting at the same state, and having different labels. This characterization leads to two algorithms: (i) a 'direct' quadratic time algorithm and (ii) an 'elegant' linear time algorithm that uses breadth first search (BFS) and the greatest common divisor (gcd) of the lengths of all cycles in the SCC. The proof of correctness involves a few technical facts about walk lengths in (directed) graphs, and a simple generalization of the theorem of Fine and Wilf [2].

The paper is organized as follows. Section 2 contains the basic notation and terminology about regular languages, automata and graphs. In Section 3, we consider the question of whether a given regular language L is of exponential density and present

* Research supported by NSERC.

Stavros Konstantinidis, Joshua Young: Deciding the Density Type of a Given Regular Language, pp. 21–34.
Proceedings of PSC 2013, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-05330-0 © Czech Technical University in Prague, Czech Republic

our characterization and the quadratic time algorithm. In Section 4, we present the linear time algorithm, and in Section 5 the correctness of this algorithm. In Section 6, we discuss our implementation using the FAdo library for automata [6], and results of a few test cases.

2 Basic Notation and Background

We begin this section with notation and concepts on words and languages, and then on finite automata. We use [8] as a general reference.

2.1 Sets, Words, Languages

We write $\mathbb{N}$ and $\mathbb{N}_0$ for the sets of positive integers and nonnegative integers, respectively. For a set S, we denote by $|S|$ the cardinality of S. We consider an arbitrary alphabet Σ containing at least two symbols. As usual, the set of all words over Σ is denoted by Σ^*. We write λ for the empty word and Σ^+ for the set of all nonempty words. The length of a word w is the number of alphabet symbols occurring in w and is denoted by $|w|$. For an integer $n \geq 0$, the expression $(w)^n$ is the word consisting of n copies of w. A *prefix* (resp. *suffix*) of a word w is any word u such that $w = ux$ (resp. $w = xu$) for some word x. We write $\mathrm{Prefix}(w)$ for the set of all prefixes of w.

A language is any set of words. A word w is called an L-word if $w \in L$. As usual, for any integer $n \geq 0$, if L is a language then L^n is the language whose words consist of any n concatenated words from L. In particular Σ^n is the set of all words of length n. Also, L^* is the union of L^n, for all $n \geq 0$, and $L^+ = L^* - \{\lambda\}$. A language C is called a *block code* if all C-words are of the same length. When there is no risk of confusion, a singleton language $\{w\}$ is written as w.

A nonempty word w is called *periodic* with a period of length $g \in \mathbb{N}$, if there is a word v of length g such that $w \in \mathrm{Prefix}(v^*)$. For example, *abbabba* is periodic with a period of length 3, as it belongs to $\mathrm{Prefix}((abb)^*)$. A (right) *infinite word* is a sequence $\mathbf{a} : \mathbb{N} \to \Sigma$. It is called periodic with a period of length $g \in \mathbb{N}$, if there is $g \in \mathbb{N}$ such that $\mathbf{a}(i + g) = \mathbf{a}(i)$, for all $i \in \mathbb{N}$. In this case, following [2], we write

$$\mathbf{a} = (v)^\omega,$$

where v is the word $\mathbf{a}(1) \cdots \mathbf{a}(g)$.

The *density* of a language L is the function d_L that maps every nonnegative integer n to $\mathrm{d}_L(n) =$ the number of L-words of length n. We say that a regular language L has *exponential density* if the density of L is not polynomially upper-bounded—see below for the definition of regular language. This definition is justified by a result of [11] stating that the density of any regular language is either polynomially upper-bounded or has a subsequence of order $\Omega(2^n)$.

2.2 Automata, graphs, cycles

A complete deterministic finite automaton (complete DFA, for short) is a quintuple

$$M = (\Sigma, K, \delta, s, F)$$

such that K is the nonempty set of states, s is the start state, F is the set of final states and $\delta : K \times \Sigma \to K$ is the transition function, which can be extended as

$\delta : K \times \Sigma^* \to K$ in the usual way. If the function δ is partial, then M is not complete—we simply call it a *DFA*. A nondeterministic finite automaton (*NFA*) is a quintuple

$$M = (\Sigma, K, T, s, F)$$

such that K, s, F are as in the case of a deterministic automaton, and T is the finite set of *transitions*, which are triples of the form (p, σ, q) with $\sigma \in \Sigma$ and $p, q \in K$. In this case, we say that the *transition is going out of the state* p. A DFA is a special type of an NFA where $(p, \sigma, q) \in T$ exactly when $\delta(p, \sigma) = q$.

The NFA M can be viewed as a directed labeled graph having K as the set of vertices and any triple (p, σ, q) as a labeled arc exactly when (p, σ, q) is a transition in T. A *walk* in M is a sequence

$$(p_0, \sigma_1, p_1, \ldots, \sigma_n, p_n) \tag{1}$$

such that (p_{i-1}, σ_i, p_i) is a transition of M, for each $i = 1, \ldots, n$. In this case, the word $\sigma_1 \cdots \sigma_n$ is called the *label* of the walk. As usual, the language $L(M)$ *accepted* by M is the set of all labels that appear in walks as above such that p_0 is the start state and p_n is a final state. These languages constitute the class of *regular* languages—see [12] for more information on regular languages.

The NFA M is called *trim* if every state of M occurs in some path from the start state to a final state. The *size* of M is $|K| + |T|$, that is the number of states plus the number of transitions in M. We note that if M is trim then $|K| \leq |T| + 1$ and, therefore, the size of M is dominated by the number of transitions in M. A state in an automaton is called a *fork state* if there are at least two transitions going out of that state.

A *path* in the NFA M is a walk in which no state appears twice. A *closed walk* in M is any walk as in (1) where $p_n = p_0$. A *cycle* in the NFA is a closed walk in which only the first and last states are equal—hence, in (1) the p_i's would be unique, for $i = 1, \ldots, n$. The special cycle (p), where p is any state, is called trivial. A *strongly connected component (SCC)*, with respect to some NFA M, is a set $\mathcal{C}$ of states that is maximal with the property that there is a walk in M between any pair of states in $\mathcal{C}$. The component $\mathcal{C}$ is called *nontrivial* if there is at least one transition between two states in $\mathcal{C}$. For the sake of simplicity, we shall say that a component $\mathcal{C}$ '*contains*' a transition (or a walk) to mean that the NFA in which $\mathcal{C}$ exists contains that transition (or walk) with all states involved belonging to $\mathcal{C}$.

3 Characterizing Exponential Density

In this section we consider the following problem.

(P0) Given a regular language L, decide whether L is of exponential density.

In [9] (see also [10]) the author gives a very simple criterion for testing this property for a regular language L: it has exponential density if and only if any trim deterministic automaton accepting L has a state that belongs to two different cycles. Here we consider the case where the language is given via a nondeterministic automaton. We show that L has an exponential density if and only if any trim nondeterministic automaton accepting L has a SCC containing two walks of the same length, starting at the same state, and whose labels are different. For example, in Fig. 1, if $\sigma = a$, then the SCC has two walks from p to 3 of length 6 with different labels:

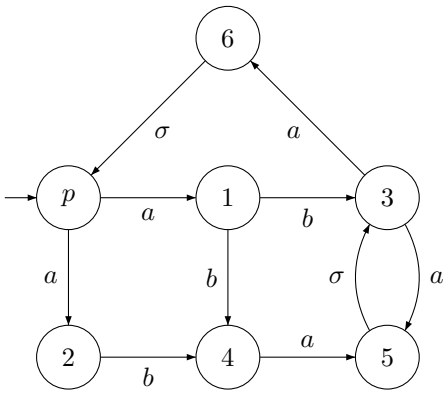

Figure 1. A strongly connected component $\mathcal{C}$ with a chosen state p, which is used as a start state of walks in $\mathcal{C}$.

$$(p, a, 1, b, 3, a, 6, \sigma, p, a, 1, b, 3) \text{ and } (p, a, 2, b, 4, a, 5, \sigma, 3, a, 5, \sigma, 3).$$

Using a 'direct' algorithm, our test can be performed in quadratic time. Note that, when the automaton is deterministic, our test is equivalent to whether a SCC contains a fork state (this is of course equivalent to the test of [9]) and can be performed in linear time. The rest of this section deals with the formalities of the above statements.

Theorem 1. *Let L be a regular language. The following statements are equivalent.*

EXP: L has exponential density.
BL2: There are two words x, y and a two-element code C whose two words have equal lengths such that $xC^*y \subseteq L$.
SCC: For every trim NFA A accepting L, there exists a strongly connected component in A containing two walks of the same length, starting at the same state, and whose labels are different.

Proof. We prove the following sequence of statements:
BL2 $\to$ EXP, SCC $\to$ BL2, EXP $\to$ SCC.

Part BL2 $\to$ EXP: Let $C = \{z_1, z_2\}$ and consider, for every $n \geq 0$, all words in L of length $|x| + |y| + \ell n$, where ℓ is the length of z_1 and z_2. As $xC^n y \subseteq L$, there are at least 2^n such words and, therefore, L must have exponential density.

Part SCC $\to$ BL2: Assume there is a state p in some SCC $\mathcal{C}$, and two walks in $\mathcal{C}$ starting at p, ending at some states q_1 and q_2, and having two *different* labels u_1, u_2 of the same length. Then, there must be two walks in $\mathcal{C}$, one from q_1 to p and the second from q_2 to p with some labels v_1, v_2, respectively. Then there are two closed walks in $\mathcal{C}$ with labels u_1v_1 and u_2v_2. Moreover, it follows that there are two closed walks in $\mathcal{C}$ with labels $z_1 = u_1v_1u_2v_2$ and $z_2 = u_2v_2u_1v_1$, which are different and of the same length, say ℓ. As the NFA A is trim, there are two paths, one from the start state to p with some label x, and the other from p to a final state with some label y. Let $C = \{z_1, z_2\}$. Then it follows that

$$xC^*y \subseteq L.$$

Part EXP $\to$ SCC: We use contraposition by assuming the negation of SCC and showing that the density of $L(A)$ is polynomially upper-bounded. So assume that

in every SCC of A, any two walks starting at the same state and having the same length must have equal labels. This implies that there is no state having two outgoing transitions with different labels. First we have the following claim.

Claim 1: The assumption implies that, in every SCC $\mathcal{C}$, for every $n \geq 0$, there are at most $|\mathcal{C}|$ distinct walk labels of length n.

To see this, we first note that the claim is obvious if $\mathcal{C}$ is trivial. If $\mathcal{C}$ is nontrivial, then for every state q in $\mathcal{C}$ and any integer $n \geq 0$, there is at least one walk in $\mathcal{C}$ of length n starting at state q. At the same time, the assumption implies that, for every state q in $\mathcal{C}$ and integer $n \geq 0$, there is at most one walk label of length n. Thus, for every state q in $\mathcal{C}$ and any $n \geq 0$, there is exactly one walk label of length n starting at q, and the claim follows easily from this observation.

Next we show that the density of $L(A)$ is polynomially upper-bounded using induction on k, where k is the number of SCCs in A. For $k = 1$ this follows immediately from Claim 1. Assume the statement holds when A has at most t SCCs, for some $t \geq 1$, and consider the case where A has $k = t + 1$ SCCs. As $k \geq 2$, there must be a strongly connected component $\mathcal{D}$ with no transitions going out of $\mathcal{D}$—also, as A is trim, $\mathcal{D}$ cannot contain the start state. Consider the set L_n of all words of length n accepted by A. We shall show that $|L_n|$ is of order $O(n^\alpha)$, where α is a constant integer independent of n. First note that

$$L_n = M_n \cup K_n,$$

where M_n is the set of words of length n accepted using walks containing no state in $\mathcal{D}$, and K_n is the set of words of length n accepted using walks ending in $\mathcal{D}$. Let $\bar{\mathcal{D}}$ be the set of states not in $\mathcal{D}$, and let $q_1, \ldots, q_r$ be all states in $\bar{\mathcal{D}}$ having transitions going into $\mathcal{D}$. Let $N_1, \ldots, N_r$ be the languages accepted by the part of A that involves no states from $\mathcal{D}$ and has as final states $\{q_1\}, \ldots, \{q_r\}$, respectively. Let $N'_1, \ldots, N'_r$ be the languages accepted starting, respectively, from the states $q_1, \ldots, q_r$ and then using only states in $\mathcal{D}$, where the final states of A that are in $\mathcal{D}$ are used as final states. Then, it follows that

$$K_n = (N_1 N'_1 \cup \cdots \cup N_r N'_r) \cap \Sigma^n$$

and then

$$L_n = M_n \cup (N_1 N'_1 \cap \Sigma^n) \cup \cdots \cup (N_r N'_r \cap \Sigma^n).$$

By the induction hypothesis, $|M_n| = O(n^c)$, for some constant c. Now for each term $N_i N'_i \cap \Sigma^n$ we have

$$N_i N'_i \cap \Sigma^n = \bigcup_{j=0}^{n} (N_i \cap \Sigma^j)(N'_i \cap \Sigma^{n-j}).$$

Again, as N_i is accepted by an NFA having at most t SCCs we have that $|N_i \cap \Sigma^j| = O(j^{c_i})$, for some constant c_i. Also, by Claim 1, $|N'_i \cap \Sigma^{n-j}| = O(1)$. With these observations, it follows that

$$|N_i N'_i \cap \Sigma^n| = O((n+1) \times n^{c_i}) = O(n^{1+c_i}) \text{ and } |L_n| = O(n^\alpha),$$

where $\alpha = \max(c, 1 + c_1, \ldots, 1 + c_r)$ and, therefore, $L(A)$ is polynomially upper-bounded.

Now we use the previous theorem to devise a 'direct' quadratic time algorithm for deciding the density type of a given NFA language. We shall use a folklore *product construction* for labeled graphs. In particular, for any directed labeled graph $G = (V, E)$, the graph G^2 has vertices all pairs in $V \times V$ and arcs all triples of the form $((p_1, p_2), (a_1, a_2), (q_1, q_2))$ such that (p_1, a_1, q_1) and (p_2, a_2, q_2) are arcs in E. It is evident that for any walk in G^2 there are two corresponding walks in G of the same length and, conversely, for any two walks in G of the same length there is a corresponding walk in G^2. For a walk P in G^2, the *first (resp. second) corresponding walk* is made simply by the sequence of arcs formed by the first (resp. second) SCCs in the sequence of arcs in P. Thus, if

$$P = ((s_0, t_0), (a_1, b_1), (s_1, t_1), \ldots, (a_n, b_n), (s_n, t_n)),$$

then the first corresponding walk is $(s_0, a_1, s_1, \ldots, a_n, s_n)$.

Corollary 2. *There is a quadratic time algorithm for deciding the density type of a given regular language.*

Proof. The required decision algorithm is as follows.

algorithm ExpDensityQT(p)
01. Make the NFA A trim
02. Compute the SCCs of A
03. FOUND = false
04. for each SCC G and while not FOUND
    05. Compute G^2
    06. Compute the set Q_1 of vertices (p_1, p_2) in G^2 such that
        there is an arc $((p_1, p_2), (a_1, a_2), (q_1, q_2))$ with $a_1 \neq a_2$
    07. Compute the set Q_2 of vertices in G^2 of the form (t, t)
    08. if (there is a walk from Q_2 to Q_1) then FOUND = true
09. if (FOUND) **return** TRUE, else **return** FALSE

For the correctness of the algorithm we note that, at the last step, FOUND is true if and only if condition SCC of Lemma 1 is true. Indeed, if there is a walk in G^2 from some (t, t) to some $(p_1, p_2) \in Q_1$ then there is also a walk from (t, t) to some (q_1, q_2) where the last arc in the walk is $((p_1, p_2), (a_1, a_2), (q_1, q_2))$ with $a_1 \neq a_2$; hence, there must be two equal length walks in G starting at t and having different labels. Conversely, if there are two walks in G of the same length, starting at some state t and having different labels, then there are also two such walks differing on their last symbols, which implies that the algorithm will set FOUND to true when it processes the SCC G.

For the time complexity of the algorithm, first we note that Step 1 can be performed in linear time and then Step 2 also in linear time [4]. Now let n be the size of A, let k be the number of strongly connected components in the trimmed A, and let n_i be the size of the SCC i. Then $n_1 + \cdots + n_k = O(n)$. The i-th iteration of the loop requires time $O(n_i^2)$ to construct the product of the i-th SCC, which is of size $O(n_i^2)$, and then the next two steps are linear with respect to n_i^2. Also linear is the last step in the loop via a breadth-first search algorithm. So in the worst case the algorithm requires time $O(n_1^2 + \cdots + n_k^2)$.

4 Deciding the density type in linear time: the algorithm

In this section we consider a fixed SCC $\mathcal{C}$ (of some NFA) containing an arbitrarily chosen state p, which we consider *fixed*, and we show that the question of exponential density for $\mathcal{C}$ can be decided in linear time. We use the following terminology—see Fig. 1 and the example 3 further below.

- $\gcd(\mathcal{C})$ denotes the *greatest common divisor* of the lengths of all cycles in $\mathcal{C}$.
- We say that a state q in $\mathcal{C}$ *occurs at level* i, for some $i \in \mathbb{N}_0$ (when starting at state p), if there is a walk of length i from p to q.
- For each $i \in \mathbb{N}$, $A_p(i)$ denotes the *set of symbols at level* i, that is, all symbols σ such that there is a transition (q, σ, r) in $\mathcal{C}$ and state q occurs at level $i-1$.

algorithm BFS(p)
01. for each state q, set $\mathrm{LEV}_p(q) =?$
02. for each $i \in \{1, \ldots, N\}$, set $\mathbf{b}_p(i) =?$
(Note: N = the number of states in the SCC)
03. Initialize a queue Q to consist of p
04. set $\mathrm{LEV}_p(p) = 0$
05. while (Q is not empty)
06. remove q, the first state in Q
07. for each transition (q, σ, r)
08. set $j = \mathrm{LEV}_p(q)$
09. if $\mathbf{b}_p(j+1) \neq?$ and $\mathbf{b}_p(j+1) \neq \sigma$, **return** λ
10. set $\mathbf{b}_p(j+1) = \sigma$
11. if ($\mathrm{LEV}_p(r) =?$)
set $\mathrm{LEV}_p(r) = j+1$
append r to Q
12. Let k be the last index such that $\mathbf{b}_p(k) \neq?$
13. **return** the word $\mathbf{b}_p(1) \cdots \mathbf{b}_p(k)$

Figure 2. This algorithm is a method of an object $\mathcal{C}$ of type "strongly connected component" containing a state p. The algorithm adds each state to the queue exactly once, and processes all transitions going out of that state. For each state q, $\mathrm{LEV}_p(q)$ is the first level at which q is encountered—this is given initially the special value '?' indicating that q has not yet been encountered. Each $\mathbf{b}_p(i)$ is the symbol found at level i (starting from state p at level 0). The algorithm returns the empty word λ if it finds a level i at which two different symbols occur, or it returns the word made by concatenating the unique symbols found at all the levels visited.

By Theorem 1, $\mathcal{C}$ has exponential density if and only if there is a level i such that $A_p(i)$ contains more than one symbol. To test this condition, we first use the breadth first search algorithm BFS(p) shown in Fig. 2. The expressions $\mathrm{LEV}_p(q)$ and $\mathbf{b}_p(i)$ are explained in that figure. In particular, $\mathrm{LEV}_p(q)$ is the length of the shortest path from p to q, and $\mathbf{b}_p(i)$ is the symbol at level i, as found by BFS(p). Then,

$$\mathbf{b}_p(i) \in A_p(i).$$

We shall show (see Theorem 5) that there is a level i_0 such that $A_p(i_0)$ contains more than one symbol, if and only if, either that level is found by BFS(p), or the word $\mathbf{b}_p$ is *not* periodic with a period of length $\gcd(\mathcal{C})$. This is the main idea for the algorithm deciding exponential density in linear time. This algorithm is shown in Fig 3.

algorithm ExpDensity(p)
1. Let $\mathbf{b}_p = \mathrm{BFS}(p)$
2. if ($\mathbf{b}_p = \lambda$) **return** TRUE
3. Let $g =$ the gcd of the cycles in the SCC
4. Let $v = \mathbf{b}_p(1) \cdots \mathbf{b}_p(g)$
5. if ($\mathbf{b}_p \notin \mathrm{Prefix}(v^*)$) **return** TRUE
6. else **return** FALSE

Figure 3. This linear-time algorithm is a method of an object $\mathcal{C}$ of type "strongly connected component" containing a state p.

Example 3. For the SCC $\mathcal{C}$ in Fig. 1, we have that $\gcd(\mathcal{C}) = 2$. The state 4 occurs at levels $2, 8, 10, 12, \ldots$. The set $A_p(4)$ consists of σ. The algorithm BFS(p) will find that

$$\mathbf{b}_p(1) = \mathbf{b}_p(3) = a, \quad \mathbf{b}_p(2) = b, \quad \mathbf{b}_p(4) = \sigma.$$

If $\sigma = a$, then $A_p(6) = \{a, b\}$ and the density is exponential. In this case, the algorithm in Fig. 3 computes $\mathbf{b}_p = abaa$, $\gcd(\mathcal{C}) = 2$, $\mathbf{b}_p(1)\mathbf{b}_p(2) = ab$, but $abaa \notin \mathrm{Prefix}((ab)^*)$. On the other hand, if $\sigma = b$, then the density is not exponential.

Time complexity. The algorithm in Fig. 3 runs in linear time. Indeed, BFS(p) is a linear time algorithm. The gcd g in Step 3 can be computed in linear time [1,5]. Finally, testing whether $\mathbf{b}_p \in \mathrm{Prefix}(v^*)$ in Step 5, can also be done in linear time, as the length of $\mathbf{b}_p$ is always less than the number of states in the SCC.

5 Correctness

In this section we establish the correctness of the linear-time algorithm—see Theorem 5.

Notation. For any SCC $\mathcal{C}$ (of some NFA) containing a state p, we define the following predicates and infinite word.

- ($\mathbf{U}_\omega$): For all $i \in \mathbb{N}$: $|A_p(i)| = 1$.
- ($\mathbf{U}_{\mathrm{bfs}}$): BFS($p$) returns $\mathbf{b}_p \neq \lambda$.
- If ($\mathbf{U}_\omega$) holds, then we define $\mathbf{a}_p$ to be the infinite word made by the symbols in $A_p(1), A_p(2), \ldots$.

■

Example 4. In Fig. 1, ($\mathbf{U}_\omega$) holds if $\sigma = b$. In this case, $\mathbf{a}_p = (ab)^\omega$.

Theorem 5. *The linear time algorithm in Fig. 3 decides correctly the density type of a given SCC, that is,*

$$\neg(\mathbf{U}_\omega) \quad \leftrightarrow \quad \neg(\mathbf{U}_{\mathrm{bfs}}) \ \vee \ \mathbf{b}_p \notin \mathrm{Prefix}((\mathbf{b}_p(1) \cdots \mathbf{b}_p(g))^*),$$

or equivalently,

$$(\mathbf{U}_\omega) \quad \leftrightarrow \quad (\mathbf{U}_{\mathrm{bfs}}) \ \wedge \ \mathbf{b}_p \in \mathrm{Prefix}((\mathbf{b}_p(1) \cdots \mathbf{b}_p(g))^*),$$

where $g = \gcd(\mathcal{C})$.

How the correctness proof is presented. The 'if' part is shown in Lemma 8. This requires the first two statements of Lemma 7 about lengths of walks in $\mathcal{C}$. The 'only if' part is shown in Lemma 10. This requires the last statement of Lemma 7 and Lemma 9, which is a simple generalization of the theorem of Fine and Wilf [2]. Next we give a notation on walks, paths and cycles in a SCC, which helps make the presentation more rigorous.

Notation. For any states p, q in $\mathcal{C}$, we use the following notation.

- $[\mathcal{C}]_*^{p\to q}$: the set of all *walks* in $\mathcal{C}$ from p to q.
- $[\mathcal{C}]_*^{p\to p}$: the set of all *closed walks* in $\mathcal{C}$ starting at state p.
- $[\mathcal{C}]_0^{p\to q}$: the set of all *paths* in $\mathcal{C}$ from p to q.
- $[\mathcal{C}]_{\min}^{p\to q}$: the set of all *shortest paths* in $\mathcal{C}$ from p to q.
- $[\mathcal{C}]_0^{q\to q}$: the set of all *cycles* in $\mathcal{C}$ from q to q.
- $[\mathcal{C}]_1^{p\to p}$: the set of all *single closed walks* in $\mathcal{C}$ starting at state p. These walks start at p and end at p and they contain exactly one cycle starting at some state $q \neq p$—see the example below.

■

Example 6. Consider the SCC in Fig. 1. The cycle $(5, \sigma, 3, a, 5)$ belongs to $[\mathcal{C}]_*^{5\to 5}$ and $(p, a, 1, b, 3, a, 6, \sigma, p)$ belongs to $[\mathcal{C}]_*^{p\to p}$. The closed walk

$$\varphi = (p, a, 1, b, 3, a, 5, \sigma, 3, a, 6, \sigma, p)$$

is a single closed walk that belongs to $[\mathcal{C}]_1^{p\to p}$. Using the notation in the proof of Lemma 7.3, φ contains exactly one cycle $\varphi'' = (3, a, 5, \sigma, 3)$. If φ'' is removed from φ, then the cycle $\varphi' = (p, a, 1, b, 3, a, 6, \sigma, p)$ is obtained.

Lemma 7. *Let $\mathcal{C}$ be any SCC (of some NFA) containing the states p and q.*

1. *The lengths of all closed walks in $[\mathcal{C}]_*^{p\to p}$ are zero modulo $\gcd(\mathcal{C})$, that is, for every closed walk $\varphi \in [\mathcal{C}]_*^{p\to p}$, $|\varphi| \equiv 0 \pmod{\gcd(\mathcal{C})}$.*
2. *The lengths of all walks in $[\mathcal{C}]_*^{p\to q}$ are equivalent modulo $\gcd(\mathcal{C})$, that is, for all walks $\varphi, \psi \in [\mathcal{C}]_*^{p\to q}$, $|\varphi| \equiv |\psi| \pmod{\gcd(\mathcal{C})}$.*
3. *The greatest common divisor of the lengths of all cycles and single closed walks starting at p is equal to $\gcd(\mathcal{C})$, that is,*

$$\gcd\{|\varphi| : \varphi \in [\mathcal{C}]_0^{p\to p} \cup [\mathcal{C}]_1^{p\to p}\} \;=\; \gcd(\mathcal{C}).$$

Proof. For the first statement, let ψ be any cycle starting at p. By definition of $\gcd(\mathcal{C})$, $|\psi| \equiv 0 \pmod{\gcd(\mathcal{C})}$. Now consider any closed walk $\varphi \in [\mathcal{C}]_*^{p\to p}$. If φ is a cycle, then the claim holds. Else, φ contains at least one cycle. If we remove each cycle occurring in φ, then we shall obtain a cycle ψ such that $|\psi| \equiv |\varphi| \pmod{\gcd(\mathcal{C})}$, as each cycle has a length that is a multiple of $\gcd(\mathcal{C})$. Hence, $|\varphi| \equiv 0 \pmod{\gcd(\mathcal{C})}$, as required.

For the second statement, it is sufficient to show that, for every $\varphi \in [\mathcal{C}]_*^{p\to q}$, we have $|\varphi| \equiv |\psi| \pmod{\gcd(\mathcal{C})}$, where ψ is any shortest path from p to q. Let θ be any shortest path from q to p, and let φ' and ψ' be the closed walks obtained by concatenating the paths φ, θ and ψ, θ (respectively). Then, $|\varphi| - |\psi| = |\varphi'| - |\psi'|$, which is 0 modulo $\gcd(\mathcal{C})$, by the first statement.

For the third statement, we first define, for each $\varphi \in [\mathcal{C}]_1^{p\to p}$, two cycles $\varphi' \in [\mathcal{C}]_0^{p\to p}$ and $\varphi'' \in [\mathcal{C}]_0^{q\to q}$, where q is the only state, other than p, that appears twice in φ. The cycle φ'' is simply the cycle occurring inside φ and the cycle φ' is produced when φ''

is removed from φ—see Example 6. Thus, $|\varphi| = |\varphi'| + |\varphi''|$. Now let $\{\varphi_i\}_{i=1}^m$ be an enumeration of all single closed walks in $[\mathcal{C}]_1^{p\to p}$. Each φ_i contains a cycle φ_i''. In fact, by definition of $[\mathcal{C}]_1^{p\to p}$, the set $\{\varphi_i''\}_{i=1}^m$ consists of all cycles in $\mathcal{C}$ starting at a state other than p. Thus,

$$\gcd(\mathcal{C}) = \gcd(\{|\varphi_i''|\}_{i=1}^m \cup X),$$

where $X = \{|\psi| : \psi \in [\mathcal{C}]_0^{p\to p}\}$. Now let $Y = \{|\varphi| : \varphi \in [\mathcal{C}]_1^{p\to p}\}$. Using basic properties of the gcd function [7], we have that

$$\gcd(X \cup Y) = \gcd(\gcd(X), \gcd(Y)) = \gcd(\gcd(X), \gcd(Y - \{|\varphi_1|\}), |\varphi_1| - |\varphi_1'|)\},$$

as $|\varphi_1'| \in X$. Thus,

$$\gcd(X \cup Y) = \gcd(\gcd(X), \gcd(Y - \{|\varphi_1|\}), |\varphi_1''|).$$

This process can be repeated another $m-1$ times to obtain that

$$\gcd(X \cup Y) = \gcd(\gcd(X), \emptyset, \gcd\{|\varphi_i''|\}_{i=1}^m) = \gcd(X \cup \{|\varphi_i''|\}_{i=1}^m),$$

as required.

Lemma 8. *Assume that, in the BFS(p) algorithm,* $(\mathbf{U}_{\text{bfs}})$ *holds and we have* $\mathbf{b}_p \in \text{Prefix}((\mathbf{b}_p(1)\cdots\mathbf{b}_p(g))^*)$*, where* $g = \gcd(\mathcal{C})$*. Then, the following statements hold true.*

1. *For any states* q *and* q' *and shortest paths* $\psi \in [\mathcal{C}]_{\min}^{p\to q}$ *and* $\psi' \in [\mathcal{C}]_{\min}^{p\to q'}$*, if* ψ *and* ψ' *are of different lengths and there are transitions* (q,σ,r) *and* (q',σ',r') *with* $\sigma \neq \sigma'$*, then* $|\psi| \not\equiv |\psi'| \pmod g$
2. *The predicate* $(\mathbf{U}_\omega)$ *holds.*

Proof. For the first statement, as ψ and ψ' are shortest paths from p, BFS(p) assigns the levels $|\psi|, |\psi'|$ to $\text{LEV}_p(q), \text{LEV}_p(q')$, respectively. Also, when q and q' are removed from the queue, the symbols σ and σ' are assigned to $\mathbf{b}_p(|\psi|+1)$ and $\mathbf{b}_p(|\psi'|+1)$, respectively. Finally, as $\mathbf{b}_p \in \text{Prefix}((\mathbf{b}_p(1)\cdots\mathbf{b}_p(g))^*)$ and $\sigma \neq \sigma'$, the lengths $|\psi|$ and $|\psi'|$ cannot be equivalent $\pmod g$.

For the second statement, assume for the sake of contradiction that there is a level ℓ and two different symbols σ, σ' in $A_p(\ell)$. Then, there are two transitions of the form (q,σ,r) and (q',σ',r') such that q, q' are at level $\ell-1$. Moreover, there are two walks φ and φ' of length $\ell-1$ in $[\mathcal{C}]_*^{p\to q}$ and $[\mathcal{C}]_*^{p\to q'}$, respectively. Let ψ, ψ' be the shortest paths of BFS(p) to q, q' (respectively). By Lemma 7.2, we have

$$|\varphi| \equiv |\psi| \pmod g \text{ and } |\varphi'| \equiv |\psi'| \pmod g$$

On the other hand, the first statement implies $|\psi| \not\equiv |\psi'| \pmod g$. Therefore, $|\varphi| \not\equiv |\varphi'| \pmod g$, which contradicts $|\varphi| = |\varphi'| = \ell - 1$. Hence $(\mathbf{U}_\omega)$ holds.

Lemma 9. *For any positive integer* m *and for any words* $u_1, \ldots, u_m$*,*

$$\text{if } u_1^\omega = \cdots = u_m^\omega, \quad \text{then } u_1, \ldots, u_m \in u^*,$$

for some word u *of length* $\gcd\{|u_1|, \ldots, |u_m|\}$.

Proof. We use induction on m. The base case of $m = 1$ is trivial. Assume the claim holds for some $m \geq 1$, and consider

$$u_1^\omega = \cdots = u_m^\omega = u_{m+1}^\omega.$$

By induction hypothesis, there is a word u such that $u_i \in u^*$, for $i \in \{1, \ldots, m\}$, and $|u| = \gcd\{|u_1|, \ldots, |u_m|\}$. As $u_m^\omega = u^\omega = u_{m+1}^\omega$, the theorem of Fine and Wilf [2] implies that $u, u_{m+1} \in v^*$, for some word v of length $\gcd\{|u|, |u_{m+1}|\}$, which is equal to $\gcd\{|u_1|, \ldots, |u_m|, |u_{m+1}|\}$, by the properties of the function gcd [7].

Lemma 10. *Let $g = \gcd(\mathcal{C})$. The following statements hold.*

1. $(\mathbf{U}_\omega) \rightarrow (\mathbf{U}_{\text{bfs}}) \wedge (\mathbf{b}_p \in \text{Prefix}(\mathbf{a}_p))$
2. $(\mathbf{U}_\omega) \rightarrow \mathbf{a}_p = (\mathbf{b}_p(1) \cdots \mathbf{b}_p(g))^\omega$
3. $(\mathbf{U}_\omega) \rightarrow \mathbf{b}_p \in \text{Prefix}((\mathbf{b}_p(1) \cdots \mathbf{b}_p(g))^*)$

Proof. For the first statement, if $(\mathbf{U}_\omega)$ holds, then BFS(p) cannot find at Step 09 a level with two symbols; hence, $\mathbf{b}_p \neq \lambda$. Also, $(\mathbf{U}_\omega)$ implies that $\mathbf{a}_p$ is well defined and, as $\mathbf{b}_p(i) \in A_p(i)$, for all symbols in $\mathbf{b}_p$, we have that $\mathbf{b}_p$ is a prefix of $\mathbf{a}_p$.

For the second statement, $(\mathbf{U}_\omega)$ implies that there is exactly one symbol at level i, in any path of length i from p. Then, it follows that $\mathbf{a}_p$ is well defined and $\mathbf{a}_p = (\bar{\mathrm{w}}_\varphi)^\omega$, for every closed walk φ starting at p, where $\bar{\mathrm{w}}_\varphi$ denotes the label of the path φ. In particular, $\mathbf{a}_p = (\bar{\mathrm{w}}_\varphi)^\omega$, for every $\varphi \in [\mathcal{C}]_0^{p \to p} \cup [\mathcal{C}]_1^{p \to p}$. Then, Lemma 9 and Lemma 7.3 imply that $\bar{\mathrm{w}}_\varphi \in u^*$ for some word u of length g. Thus, $\mathbf{a}_p = u^\omega$ and, as $\mathbf{b}_p$ is a prefix of $\mathbf{a}_p$, we have that $u = \mathbf{b}_p(1) \cdots \mathbf{b}_p(g)$, as required.

The third statement follows from the previous one and the fact that $\mathbf{b}_p$ is a prefix of $\mathbf{a}_p$.

6 Implementation and testing

We have implemented both the quadratic and linear time algorithms using the FAdo library for automata [6], which is well maintained and provides several useful tools for manipulating automata. In doing so, we have also implemented a Python method

`stronglyConnectedComponents(A)`

receiving a parameter `A`, which is an `NFA` object with respect to FAdo, and returns a list of the SCCs of `A`, where each SCC is a list of states in `A`.

For computing the quantity $\gcd(\mathcal{C})$, one can use the depth first search (DFS) based algorithm in [1], or the breadth first search (BFS) based algorithm in [5]—in fact [5] discusses both the BFS and DFS based algorithms. In our implementation, we have adjusted the BFS algorithm in Fig 2 to compute the required $\gcd(\mathcal{C})$, in addition to the word $\mathbf{b}_p(1) \cdots \mathbf{b}_p(k)$.

Our implementation confirms the theoretical result that indeed the linear time algorithm is much faster. We have used as test cases four sequences of SCCs, which are described in Fig. 4. Each of these SCCs is implemented as an object of type `NFA` and is constructed using `NFA` methods such as

`addState()` and `addTransition()`.

When the answer is FALSE, the linear time algorithm will perform a complete BFS and then all tests in Fig. 3 to find out that $\mathbf{b}_p$ is nonempty and periodic with a period of length g. When the answer is TRUE, it is possible that the linear time

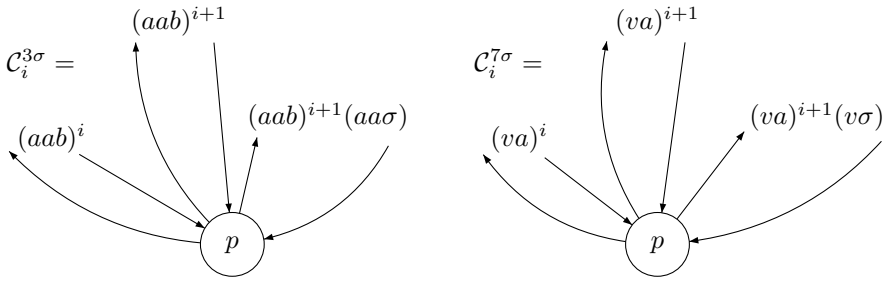

Figure 4. Four sequences of SCCs. On the left, for $\sigma = a, b$, we have the SCCs $\mathcal{C}_i^{3a}$ and $\mathcal{C}_i^{3b}$. On the right, for $\sigma = a, b$, we have the SCCs $\mathcal{C}_i^{7a}$ and $\mathcal{C}_i^{7b}$ with $v = aabbab$. Each SCC has three cycles starting at p with labels as shown in the figure. For example, for each $i \in \mathbb{N}$, the SCC $\mathcal{C}_i^{3a}$ has three cycles with labels $(aab)^i$, $(aab)^{i+1}$, $(aab)^{i+1}(aaa)$. If $\sigma = a$ the density is exponential.

algorithm will finish quickly when BFS(p) finds in Step 09 two different symbols occurring at the same level. However, we have chosen the particular test SCCs such that when the answer is TRUE, the linear time algorithm will still perform a complete BFS and then find out that $\mathbf{b}_p$ is non-periodic only when it scans the last symbol σ of the longest cycle in the SCC.

Each of the four figures in the Appendix concerns one of the two algorithms and a certain sequence $\mathcal{C}_i^{x\sigma}$ of SCCs, and shows a graph with the execution time of the algorithm $T(i)$ vs the value of the parameter i.

References

1. E. Arkin, C. Papadimitriou, and M. Yannakakis: *Modularity of cycles and paths in graphs.* Journal of the ACM, 38 1991, pp. 255–274.
2. C. Choffrut and J. Karhumäki: *Combinatorics on words*, in Rozenberg and Salomaa [8], pp. 329–438.
3. B. Cui and S. Konstantinidis: *DNA coding using the subword closure operation*, in Proceedings of 13th Inter. Meeting on DNA Computing 2007, vol. 4848 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2008, pp. 284–289.
4. S. Dasgupta, C. Papadimitriou, and U. Vazirani: *Algorithms*, McGraw-Hill, 2006.
5. J. Jarvis and D. Shier: *Graph-theoretic analysis of finite Markov chains*, in Applied Mathematical Modeling: a multidisciplinary approach, Chapman & Hall / CRC press, 2000, pp. 271–289.
6. R. Reis and N. Moreira: *FAdo: Tools for language models manipulation*, `http://www.dcc.fc.up.pt/~rvr/FAdoDoc/index.html`. Accessed in March 2013.
7. K. Rosen: *Greatest common divisors*, in Handbook of Discrete and Combinatorial Mathematics, CRC Press, Berlin, 2000.
8. G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages, Vol. I*, Springer-Verlag, 1997.
9. A. Shur: *Combinatorial complexity of rational languages.* Discr. Anal. and Oper. Research, Ser. 1, 12(2) 2005, pp. 78–99.
10. A. Shur: *Factorial languages of low combinatorial complexity*, in In Proceedings of 19th Inter. Conf. on Developments in Language Theory. Lecture Notes in Computer Science, Vol. 4036, Springer-Verlag, Berlin, 2006, pp. 397–407.
11. A. Szilard, S. Yu, K. Zhang, and J. Shallit: *Characterizing regular languages with polynomial densities*, in Proceedings of 7th Inter. Symposium on Mathematical Foundation of Computer Science, vol. 629 of Lecture Notes in Computer Science, Springer-Verlag, London, UK, 1992, pp. 494–503.
12. S. Yu: *Regular languages*, in Rozenberg and Salomaa [8], pp. 41–110.

Appendix

This appendix consists of four graphs showing execution times of the quadratic and linear times algorithms as explained in section 6.

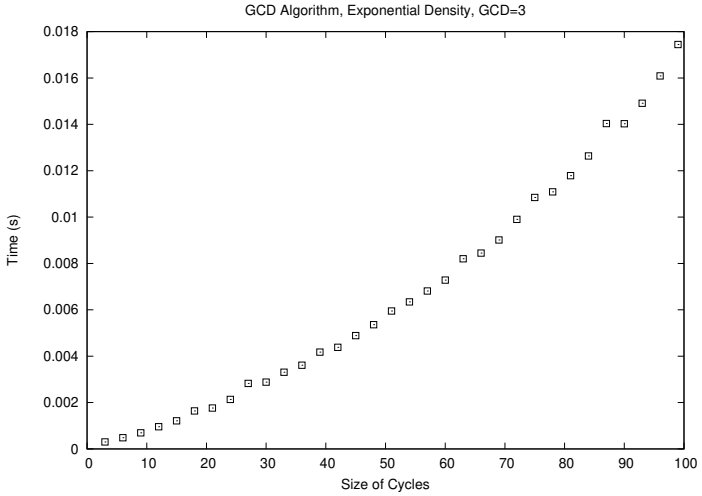

Figure 5. Execution time $T_{\mathrm{lt}}(i)$ of the *linear* time algorithm on $\mathcal{C}_i^{3a}$, for various values of i. The density type is exponential.

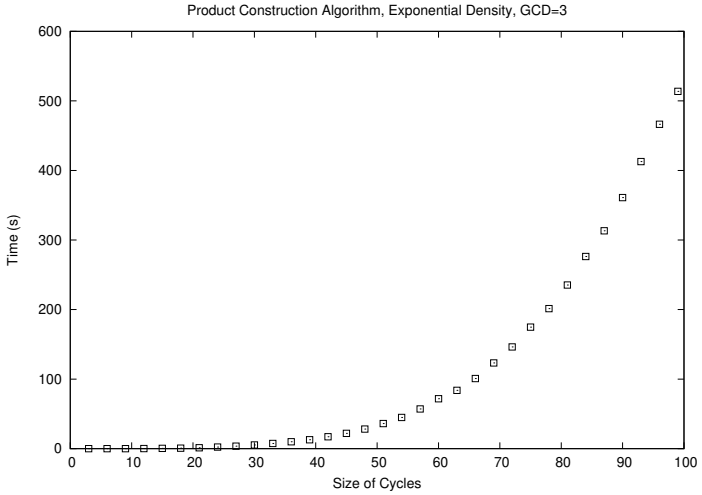

Figure 6. Execution time $T_{\mathrm{qt}}(i)$ of the *quadratic* time algorithm on $\mathcal{C}_i^{3a}$, for various values of i. The density type is exponential.

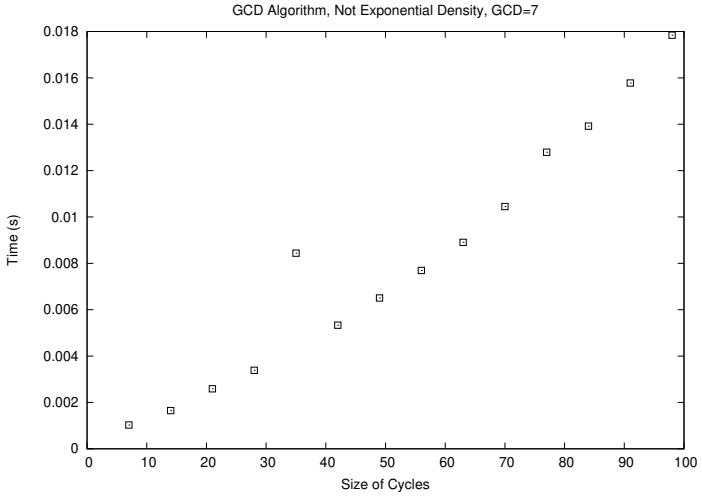

Figure 7. Execution time $T_{\mathrm{lt}}(i)$ of the *linear* time algorithm on $\mathcal{C}_i^{7a}$, for various values of i. The density type is *not* exponential.

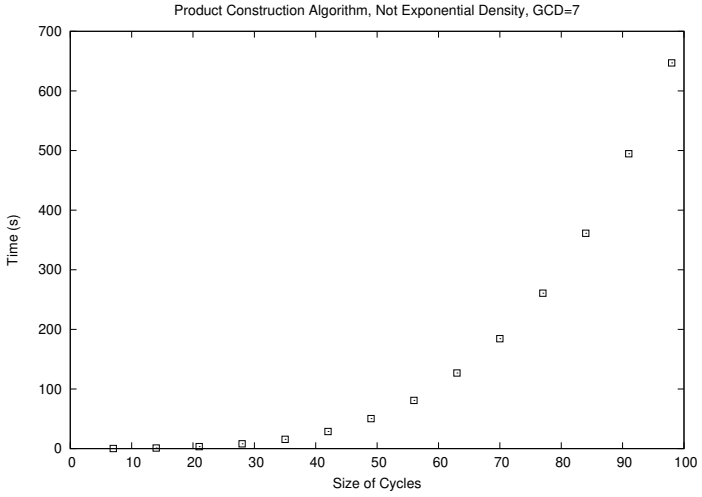

Figure 8. Execution time $T_{\mathrm{qt}}(i)$ of the *quadratic* time algorithm on $\mathcal{C}_i^{7a}$, for various values of i. The density type is *not* exponential.