Branching Measures and Nearly Acyclic NFAs

Chris Keeler and Kai Salomaa(✉)

School of Computing, Queen's University, Kingston, ON K7L 2N8, Canada
{keeler,ksalomaa}@cs.queensu.ca

Abstract. To get a more comprehensive understanding of the branching complexity of nondeterministic finite automata (NFA), we introduce and study the string path width and depth path width measures. The string path width on a string w counts the number of all complete computations on w, and the depth path width on an integer ℓ counts the number of complete computations on all strings of length ℓ. We give an algorithm to decide the finiteness of the depth path width of an NFA. Deciding finiteness of string path width can be reduced to the corresponding question on ambiguity.

An NFA is nearly acyclic if any computation can pass through at most one cycle. The class of nearly acyclic NFAs consists of exactly all NFAs with finite depth path width. Using this characterization we show that the finite depth path width of an m-state NFA over a k-letter alphabet is at most $(k+1)^{m-1}$ and that this bound is tight. The nearly acyclic NFAs recognize exactly the class of constant density regular languages.

1 Introduction

Finite automata are a fundamental model of computation that has been extensively studied since the 1950s. The last decades have seen much work on the descriptional complexity, or state complexity, of regular languages [8,9,25].

The *degree of ambiguity* of a nondeterministic finite automaton (NFA) A on a string w is the number of accepting computations of A on w. Ravikumar and Ibarra [19] have first studied systematically the size-trade-offs between NFAs of different degrees of ambiguity. Leung [15] has shown that general NFAs can be exponentially more succinct than polynomially ambiguous NFAs, and Hromkovič and Schnitger [11] have established a descriptional complexity separation between polynomially ambiguous and finitely ambiguous NFAs.

The degree of ambiguity is defined in terms of the number of accepting computations, and does not directly limit the total amount of nondeterminism in a computation. The computation of an unambiguous NFA may include an unbounded number of nondeterministic steps, as long as at each nondeterministic step, only one choice can lead to acceptance. The *tree width* [1] (a.k.a. leaf size) measure counts the number of leaves of the computation tree [10,17,18]. Other measures of nondeterminism for finite automata have also been considered [6–8,10,18].

[1] Note that this is not the same as the graph theory notion of tree width.

G. Pighizzini and C. Câmpeanu (Eds.): DCFS 2017, LNCS 10316, pp. 202–213, 2017.
DOI: 10.1007/978-3-319-60252-3_16

We study a measure called *string path width* that counts the number of complete accepting and non-accepting computations of an NFA on a given string. The string path width can be viewed as a blending between the tree width measure and the degree of ambiguity. For certain NFAs, the string path width is the same as tree width, and for others the same as ambiguity. In fact, Goldstine et al. [6] have defined 'ambiguity' as the number of complete computations, which coincides with our notion of string path width. The *degree automata* [13] extend these notions by considering the ratio of the number accepting computations and the number of all computations on a given string.

To get a more comprehensive understanding of the degree of branching[2] of an NFA, we introduce the *depth path width* measure, which counts the total number of complete computations on all inputs of a given length. We establish necessary and sufficient conditions for an NFA to have infinite depth path width. These conditions are based on the existence of cycles satisfying certain requirements. This characterization yields a polynomial time algorithm to decide whether or not the depth path width of an NFA is bounded. Finiteness of string path width can be decided with existing algorithms from the literature [24].

It is well known that acyclic finite automata characterize exactly the finite languages. We characterize regular languages having bounded depth path width by an extension of acyclic NFAs, called *nearly acyclic* NFAs. An NFA A is said to be nearly acyclic if A, roughly speaking, it does not contain two distinct cycles where a state of one cycle is reachable from the other cycle.

We show that there exists an m-state nearly acyclic NFA over a k-letter alphabet having depth path width $(k+1)^{m-1}$, and that this is an upper bound for all m-state NFAs over a k-letter alphabet having finite depth path width. Finally, we show that nearly acyclic NFAs recognize exactly the regular languages of bounded density [21]. For nearly acyclic DFAs we have a stronger correspondence: any DFA recognizing a bounded density language must be nearly acyclic.

2 Preliminaries

Here we recall and introduce some notation and definitions. More information on finite automata can be found e.g. in [22,25]. The set of strings over a finite alphabet Σ is Σ^*, and ε is the empty string. The cardinality of a finite set F is denoted $|F|$ and $\mathbb{N}$ is the set of non-negative integers.

A *nondeterministic finite automaton* (NFA) is a tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is the finite set of states, Σ is the input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final states. The transition function δ is in the usual way extended as a function $Q \times \Sigma^* \to 2^Q$, and the language recognized by A is $L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$. If $|\delta(q, b)| \leq 1$ for all $q \in Q$ and $b \in \Sigma$, the automaton A is a *deterministic finite automaton* (DFA). Note that we allow NFAs and DFAs to have undefined

[2] Here and in the title of the paper by "branching" we mean an informal notion of path expansion in computations. A specific technical notion called branching is considered by Goldstine et al. [7].

transitions. Our definition does not allow multiple start states or $\varepsilon-$transitions. Unless otherwise mentioned, we always assume that an NFA does not have any unreachable states.

A *(state) path* of the NFA A with underlying string $w = b_1 b_2 \cdots b_k$, $b_i \in \Sigma$, $i = 1, \ldots, k$, $k \geq 0$, is a sequence of states $(p_0, p_1, \ldots, p_\ell)$, where $p_j \in \delta(p_{j-1}, b_j)$, $j = 1, \ldots \ell$, and either $\ell = k$, or, $\ell < k$ and $\delta(p_\ell, b_{\ell+1}) = \emptyset$. That is, the path must read the entire underlying string unless it encounters an undefined transition. Two paths are equal if and only if they have the same sequence of states and underlying string.

A path beginning in the start state q_0, is a *computation* of A on the underlying string w. A computation $(q_0, p_1, \ldots, p_\ell)$ is a *complete computation* on a string $b_1 b_2 \cdots b_k$ if $\ell = k$. An *accepting computation* is a complete computation that ends in an accepting state of F. The set of all (not necessarily complete) computations of A on the string w is denoted $\text{comp}_A(w)$.

Intuitively, a computation of A on a string w is a sequence of states that A reaches when started with the initial state and the symbols of w are read one by one. A complete computation ends with a state reached after consuming all symbols of w. An incomplete computation ends with a state where the transition on the next symbol of w is undefined.

The *length* of a path $C_1 = (p_0, p_1, \ldots, p_\ell)$ is $|C_1| = \ell$ (the number of transitions). The catenation of C_1 and a path $C_2 = (p_\ell, p'_1, \ldots p'_m)$ is $C_1 \cdot C_2 = (p_0, \ldots, p_\ell, p'_1, \ldots p'_m)$. That is, paths C_1 and C_2 can be catenated if C_1 ends with the first state of C_2.

A path $(p_0, p_1, \ldots, p_k)$, $k \geq 1$, with underlying string $b_1 b_2 \cdots b_k$ is a *cycle* if $p_0 = p_k$. A cycle with one transition from a state to itself is called a *self-loop*. (A path of length zero with no transitions is not a cycle.) An NFA with no cycles is called an *acyclic* NFA (aNFA).

Cycles that are obtained from each other by a cyclical shift are said to be *equivalent:* For $0 < i < k$, the above cycle (with $p_0 = p_k$) is equivalent to the cycle $(p_i, \ldots, p_k, p_1, \ldots p_{i-1}, p_i)$ having underlying string $b_{i+1} \cdots b_k b_1 \cdots b_i$.

We define *path trees* that represent all computations of an NFA on all strings of a given length. Note that this is different than the notion of computation trees [10,17], which represent all computations of an NFA on a given string w. For $\ell \in \mathbb{N}$, the *path tree* of an NFA $A = (Q, \Sigma, \delta, q_0, F)$ of depth ℓ, $T_{A,\ell}$, is a finite tree where the nodes are labelled by elements of Q and the edges are labelled by elements of Σ, defined inductively as follows:

- $T_{A,0}$ consists of a single node labelled by q_0.
- Consider $\ell \geq 1$ and let $\text{leaf}(\ell - 1)$ be the set of leaf nodes of $T_{A,\ell-1}$ having distance $\ell - 1$ from the root. If an $x \in \text{leaf}(\ell - 1)$ is labelled by $q \in Q$, then for each $c \in \Sigma$ and $q' \in \delta(q, c)$, in the tree $T_{A,\ell}$ we add to node x a child y labelled by q', and the edge between x and y is labelled with c.

The *pruned path tree* of depth ℓ, $T^p_{A,\ell}$, is obtained from $T_{A,\ell}$ by recursively removing all leaf nodes which have distance smaller than ℓ from the root node.

The *degree of ambiguity* of an NFA A on a string w, $\text{da}(A, w)$ [8,19], is the number of accepting computations of A on w, and the *tree width* of A on w,

$\mathrm{tw}(A, w)$ [10,17], is the number of (not necessarily complete) computations of A on w. Note that Hromkovič et al. [10] call this "leaf size". Tree width is usually defined as the number of leaves of the computation tree of A on w. This quantity is identical to the cardinality of the set $\mathrm{comp}_A(w)$.

For $\ell \geq 0$, the degree of ambiguity (respectively, tree width) of A on strings of length ℓ is defined as $\mathrm{da}(A, \ell) = \max\{\mathrm{da}(A, w) \mid w \in \Sigma^\ell\}$ (respectively, $\mathrm{tw}(A, \ell) = \max\{\mathrm{tw}(A, w) \mid w \in \Sigma^\ell\}$). Strictly speaking, using common practice, we use $\mathrm{da}(A, \cdot)$ (and $\mathrm{tw}(A, \cdot)$) to denote two different functions where one takes a string and the other an integer as argument.

The ambiguity (respectively, the tree width) of the NFA A is said to be finite if the above values are bounded for all $\ell \in \mathbb{N}$, and in this case, the degree of ambiguity (respectively, the tree width) of A is denoted $\mathrm{da}^{\sup}(A)$ (respectively, $\mathrm{tw}^{\sup}(A)$).

3 String Path Width and Depth Path Width

We consider measures that count the number of complete computations on a given string and on all strings of given length, respectively.

In the following, $A = (Q, \Sigma, \delta, q_0, F)$ is always an NFA. The *string path width* of A on a string $w \in \Sigma^*$, $\mathrm{SPW}(A, w)$, is defined as the number of complete computations of A on w. For $\ell \in \mathbb{N}$, the string path width of A on strings of length ℓ is $\mathrm{SPW}(A, \ell) = \max\{\mathrm{SPW}(A, w) \mid w \in \Sigma^\ell\}$, and when this value is bounded, the *string path width of* A is denoted $\mathrm{SPW}^{\sup}(A)$.

Example 1. For the NFA A_1 given in Fig. 1:

- $\mathrm{SPW}(A_1, ab) = 2$, complete computations $\{(0, 1, 0), (0, 1, 2)\}$
- $\mathrm{SPW}(A_1, aaaa) = 1$, complete computations $\{(0, 1, 0, 1, 0)\}$
- Generally, $\mathrm{SPW}(A_1, (ab)^x) = x + 1$, $x \in \mathbb{N}$ □

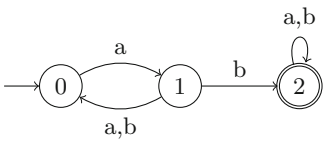

Fig. 1. NFA A_1

In fact, Goldstine et al. [6] have defined 'ambiguity' as the number of complete computations, which coincides with our notion of string path width. The string path width can be viewed as a blend between ambiguity and tree width in the sense of the following lemma. Since string path width counts only complete computations while tree width counts all computations, the string path width of an NFA A on a string w will always be at most the tree width of A on w.

Lemma 1. *Consider an NFA* $A = (Q, \Sigma, \delta, q_0, F)$ *and let* $w \in \Sigma^*$.

(i) $\mathrm{da}(A, w) \leq \mathrm{SPW}(A, w) \leq \mathrm{tw}(A, w)$.
(ii) If A *has no undefined transitions, that is,* $\delta(q, b) \neq \emptyset$ *for all* $q \in Q$, $b \in \Sigma$, *then* $\mathrm{SPW}(A, w) = \mathrm{tw}(A, w)$.
(iii) If all states of A *are final, then* $\mathrm{SPW}(A, w) = \mathrm{da}(A, w)$.

Since string path width is, in the sense of Lemma 1 (iii), a special case of degree of ambiguity, from algorithms and bounds for ambiguity we get corresponding results for string path width. This is established using the transformation of the following lemma. In general, the transformed automaton is not equivalent to the original. Note that Lemma 1 (ii) gives a correspondence between string path width and tree width, but this cannot be used in a similar way because the corresponding transformation changes the string path width of the NFA.

Lemma 2. *Given an NFA* $A = (Q, \Sigma, \delta, q_0, F)$, *we can construct in linear time an NFA* A' *such that* $\mathrm{da}(A', w) = \mathrm{SPW}(A, w)$ *for all strings* $w \in \Sigma^*$.

Using Lemma 2, and the results by Weber and Seidl [24], we get:

Corollary 1 [24]**.** *Let* $A = (Q, \Sigma, \delta, q_0, F)$ *be an NFA.*

(i) In time $O(|Q|^6 \cdot |\Sigma|)$, *a random-access-machine can decide whether or not* $\mathrm{SPW}^{\sup}(A)$ *is finite, and in the positive case,* $\mathrm{SPW}^{\sup}(A) \leq 5^{\frac{|Q|}{2}} \cdot |Q|^{|Q|}$.
(ii) The growth rate of $\mathrm{SPW}(A, \ell)$ *is either bounded by a constant, polynomial in* ℓ, *or exponential in* ℓ. *If the growth rate is polynomial, the degree of the polynomial can be decided in* $O(|Q|^6 \cdot |\Sigma|)$ *time.*
(iii) It can be decided in $O(|Q|^4 \cdot |\Sigma|)$ *time whether or not the growth rate of* $\mathrm{SPW}(A, \ell)$ *is exponential.*

Also, it is known that for a fixed k and a given NFA A it can be decided in polynomial time whether $\mathrm{da}^{\sup}(A)$ (and consequently whether $\mathrm{SPW}^{\sup}(A)$) is at least k, but the question for degree of ambiguity becomes PSPACE-complete if k is part of the input [3].

Next we introduce the depth path width of an NFA as the number of all complete computations of a given length. This metric can be viewed as a broader version of the string path width; while the string path width counts the number of computations on a specific string, the depth path width considers all strings of the same length.

Consider an NFA $A = (Q, \Sigma, \delta, q_0, F)$ and let $\ell \in \mathbb{N}$. The *depth path width* of A on strings of length ℓ is

$$\mathrm{DPW}(A, \ell) = \sum_{w \in \Sigma^\ell} \mathrm{SPW}(A, w).$$

The depth path width of the NFA A is defined as $\mathrm{DPW}^{\sup}(A) = \sup_{\ell \in \mathbb{N}}(\mathrm{DPW}(A, \ell))$.

Example 2. For the DFA $A_2 = (Q, \Sigma, \delta, q_0, F)$ given in Fig. 2:

- $\mathrm{DPW}(A_2, 1) = 2$, complete computations $(0, 0)$ on a, $(0, 1)$ on b.
- Generally, $\mathrm{DPW}(A_2, \ell) = \ell + 1$, $\ell \in \mathbb{N}$. □

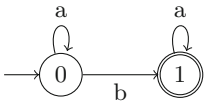

Fig. 2. DFA A_2

Directly from the definition it follows that for NFAs over a unary alphabet, the notion of depth path width coincides with string path width.

We give the necessary and sufficient conditions for an NFA to have unbounded depth path width. For this we use the correspondence between depth path width and the number of leaves in path trees (defined in Sect. 2).

Lemma 3. *Consider an NFA A and $\ell \in \mathbb{N}$. The value* $\mathrm{DPW}(A, \ell)$ *is equal to the number of leaves of the pruned path tree $T^p_{A,\ell}$.*

Intuitively, the conditions of Theorem 1 mean that q_1 and q_2 belong to a cycle and the state q_1 has another transition to a state q_3 such that the computations originating from q_3 are defined on infinitely many strings. Here q_3 may or may not belong to the same cycle as q_1 and q_2. If $q_2 = q_3$, then the alphabet symbols a and b must be distinct.

Theorem 1. *Consider an NFA $A = (Q, \Sigma, \delta, q_0, F)$. The depth path width of A is unbounded if and only if the following holds:*

There exist $q_1, q_2, q_3 \in Q$ and $a, b \in \Sigma$, where $q_2 \neq q_3$ or $a \neq b$, such that

(i) $q_2 \in \delta(q_1, a)$ and state q_1 is reachable from q_2, and,
(ii) $q_3 \in \delta(q_1, b)$ and the language of the NFA $A' = (Q, \Sigma, \delta, q_3, Q)$ is infinite.

Proof. First assume that conditions (i) and (ii) hold. Let C_1 be a computation from q_0 to q_1 (recall that we assume that NFAs have no unreachable states). Let C_2 be a cycle from q_1 back to q_1 that begins with the transition on a to q_2.

To show that $\mathrm{DPW}^{\mathrm{sup}}(A)$ is infinite, it is sufficient to show that for all $M \in \mathbb{N}$ there exists ℓ such that $\mathrm{DPW}(A, \ell) \geq M$. By condition (ii) there exists a path C_M having length $M \cdot |C_2|$ that begins in q_1 with the transition on b to q_3. Now A has M different computations of length $|C_1| + M \cdot |C_2|$:

$$C_1 \cdot C_2^i \cdot D_i, \quad i = 0, 1, \ldots, M - 1,$$

where D_i is an initial part of the path C_M having length $(M - i) \cdot |C_2|$. Note that the above are all distinct computations because the transitions from q_1 to q_2 on a and from q_1 to q_3 on b are distinct.

We sketch the proof in the "only if" direction: If $\mathrm{DPW}^{\mathrm{sup}}(A)$ is infinite, using Lemma 3 we see that the number of leaves of the pruned path tree $T^{p}_{A,\ell}$ can be chosen arbitrarily large for sufficiently large ℓ. When some state of A repeats on a path from the root to a leaf, we get a cycle and states satisfying conditions (i) and (ii). □

The conditions of Theorem 1 yield a polynomial time algorithm to test whether the depth path width of an NFA is infinite.

Theorem 2. *If A is an NFA with m states over an alphabet Σ, we can decide in time $O(|\Sigma| \cdot m^5)$ whether or not the depth path width of A is infinite.*

Proof. Algorithm 1 checks the conditions of Theorem 1. Creating the copy of

Algorithm 1. Deciding if an NFA has infinite depth path width

```
Let A = (Q, Σ, δ, q_0, F) be an NFA where |Q| = m.
Create a copy of A and call it A', where all states of A' are final.
Create a distance matrix M, where M[q, q'] is the minimum distance from state
  q ∈ Q to state q' ∈ Q.
infinityCondition = False
for all q_1 ∈ Q do
   for all q_2 ∈ δ(q_1, a) and q_3 ∈ δ(q_1, b) such that (q_2 ≠ q_3 or a ≠ b) do
      if M[q_2, q_1] ≠ ∞ then
         Set initial state of A' to be q_3
         if L(A') is infinite then
            infinityCondition = True
         end if
      end if
   end for
end for
return infinityCondition
```

the NFA A takes $\Theta(m + |\delta|)$ time. Creating the adjacency matrix takes $\Theta(m^3)$ time and $\Theta(m^2)$ space using the Floyd-Warshall algorithm [5]. The two for all statements multiply the inner complexity by $\Theta(m^3)$, as there are m^3 triples of the form (q_1, q_2, q_3). Checking whether $L(A')$ is infinite takes $O(m + |\delta|)$ time using Tarjan's Strongly Connected Components algorithm [23]. So the worst-case runtime is $O(m + |\delta| + m^3 + m^3 \cdot (m + |\delta|))$ which simplifies to $O(|\Sigma| \cdot m^5)$. □

4 Depth Path Width of Nearly Acyclic NFAs

We want to derive an upper bound for the finite depth path width of an m-state NFA. First we develop bounds for the depth path width measure of acyclic NFAs where the depth path width is naturally guaranteed to be finite.

Proposition 1. *Let A be an m-state unary aNFA. Then* $\mathrm{DPW}^{\sup}(A) \leq \binom{m-1}{\lfloor \frac{m-1}{2} \rfloor}$.

Note that the result of Proposition 1 indicates that the largest possible depth path width of an m-state aNFA is obtained by strings of length, roughly, m divided by two.

We now extend the result for arbitrary alphabet sizes.

Theorem 3. *Let A be an m-state aNFA. Then*

$$\mathrm{DPW}^{\sup}(A) \leq \sup_{\lfloor \frac{m-1}{2} \rfloor \leq \ell \leq m-1} k^{\ell} \cdot \binom{m-1}{\ell}.$$

The upper bound can be improved for acyclic DFAs (aDFA).

Corollary 2. *For an aDFA D with m states and k alphabet characters, the depth path width of D is at most k^{m-1}.*

It is easy to verify that an NFA A does *not* satisfy the conditions of Theorem 1 if and only if A does not have two non-equivalent cycles where one is reachable from the other. (Two cycles are equivalent if they are obtained from each other by a cyclical shift, see Sect. 2.) This condition forms the basis for the following definition.

Definition 1. *An NFA A is* nearly acyclic *(naNFA) if it does not have two non-equivalent cycles, C_1 and C_2, such that a state of C_2 is reachable from a state of C_1. An naNFA with a deterministic transition function is called a* nearly acyclic DFA *(naDFA).*

By Theorem 1, Definition 1 gives the most general class of NFAs that have finite depth path width. The influence of cycles that are reachable from one another is considered in a more general way by Msiska and van Zijl [16].

The limitation on the reachability between cycles implies a limitation on the number of (non-equivalent) cycles in a nearly acyclic NFA.

Lemma 4. *An m-state naNFA has at most $(m-1)$ cycles.*

The naNFAs with a maximal number of acyclic transitions and one self-loop on the initial state turn out to be useful for obtaining bounds for depth path width.

Definition 2. *An m-state* initial self-loop maximal nearly acyclic *NFA, an imax-naNFA, over an alphabet Σ has the set of states $\{0, 1, \ldots, m-1\}$ where 0 is the start state, there exists a transition on each alphabet symbol from i to j for all $0 \leq i < j \leq m-1$, and 0 has a self-loop.*

The transitions of an imax-naNFA are uniquely determined, except for the self-loop on the initial state, which can be on an arbitrary element of Σ. (If needed we could specify the symbol labelling the self-loop.) Also, for purposes of depth path width, the set of final states can be arbitrary. In Fig. 3 illustrating an m-state imax-naNFA, we use $m-1$ as the only final state.

We calculate the depth path width of imax-naNFAs as a function of the number of states and alphabet size.

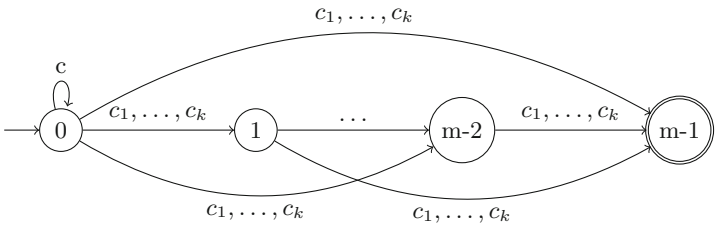

Fig. 3. An m-state imax-naNFA with alphabet $\{c_1, \ldots, c_k\}$.

Lemma 5. *An m-state imax-naNFA over a k-letter alphabet has depth path width $(k+1)^{m-1}$.*

Since acyclic DFAs are a special case of nearly acyclic DFAs, we can use the value acquired in Corollary 2 as a lower limit on the upper bound for the depth path width of an naDFA.

Theorem 4. *For $m \in \mathbb{N}$, there exists an m-state nearly acyclic DFA over a k-letter alphabet having depth path width k^{m-1}.*

Lemma 5 gives the depth path width of imax-naNFAs. From Lemma 4 we recall that an naNFA can have multiple cycles, however, it seems plausible that an m-state imax-naNFA could have maximal depth path width among all m-state naNFAs. This is established in the following lemmas.

Lemma 6. *Let A be an naNFA with (one or more) cycles of length at least two. Then there exists an naNFA A' with the same number of states over the same alphabet where all cycles are self-loops and* $\mathrm{DPW}^{\mathrm{sup}}(A') \geq \mathrm{DPW}^{\mathrm{sup}}(A)$.

Consider an m-state naNFA B where all cycles are self-loops. We can define an injective mapping from the set of computations of B having length ℓ to the length ℓ computations of an m-state imax-naNFA A. This then implies that the depth path width of B is at most that of A, and the observation is the basis for the following lemma.

Lemma 7. *Let A be an m-state imax-naNFA over alphabet Σ and let B be an m-state naNFA over Σ where all cycles are self-loops. Then* $\mathrm{DPW}^{\mathrm{sup}}(B) \leq \mathrm{DPW}^{\mathrm{sup}}(A)$.

Now we get a tight upper bound for the depth path width of an m-state naNFA.

Theorem 5. *If A is an m-state naNFA over a k-letter alphabet, then* $\mathrm{DPW}^{\mathrm{sup}}(A) \leq (k+1)^{m-1}$. *For each $m, k \geq 1$, there exists an m-state naNFA B_{imax} over a k-letter alphabet such that* $\mathrm{DPW}^{\mathrm{sup}}(B_{\mathrm{imax}}) = (k+1)^{m-1}$.

Proof. By Lemma 6, A can be converted to an m-state naNFA A' over the same alphabet without decreasing the depth path width where all cycles in A' are self-loops. Let B_{imax} be an m-state imax-naNFA over the same alphabet. Now

$$\text{DPW}^{\text{sup}}(A) \leq \text{DPW}^{\text{sup}}(A') \leq \text{DPW}^{\text{sup}}(B_{\text{imax}}) = (k+1)^{m-1},$$

where the second inequality follows from Lemma 7 and the equality from Lemma 5. The equality also establishes the second claim of the theorem. □

4.1 Languages Recognized by NaNFAs

Acyclic NFAs recognize the family of finite languages and, similarly, the nearly acyclic NFAs recognize a proper subfamily of the regular languages. The *density* of a language $L \subseteq \Sigma^*$ is defined as the function $d_L(\ell) = |L \cap \Sigma^\ell|$, $\ell \in \mathbb{N}$.

Proposition 2 (Shallit [21])**.** *The density of a regular language L over Σ is bounded, that is $d_L(\ell) \in O(1)$, if and only if L can be represented as a finite union of regular expressions xy^*z, where $x, y, z \in \Sigma^*$.*

The nearly acyclic NFAs recognize exactly the constant density languages.

Theorem 6. *A regular language L has constant density if and only if L is recognized by a nearly acyclic NFA.*

Proof. Suppose that $L \subseteq \Sigma^*$ is recognized by an m-state naNFA A. We show that $d_L(\ell) \leq m^3 \cdot |\Sigma|^m$ for all $\ell \in \mathbb{N}$. For $\ell \leq m-1$ there is nothing to prove.

Consider then strings of length $\ell \geq m$. For each $w \in \Sigma^\ell$ accepted by A, fix one accepting computation C_w. Since A is nearly acyclic and $\ell \geq m$, the computation C_w must pass through exactly one cycle. Thus, we can write $w = w_{\text{pref}} w_{\text{cyc}} w_{\text{suf}}$ where w_{cyc} is the maximal substring of w that in the computation C_w is "processed" by transitions of the cycle, and $|w_{\text{pref}} \cdot w_{\text{suf}}| \leq m-1$. The number of strings of length at most $m-1$ is upper bounded by $|\Sigma|^m$. In a string of length at most $m-1$ the cycle can occur in at most m locations and, according to Lemma 4, A has at most m cycles and, furthermore, each cycle (equivalence class) can be started in at most m positions.[3] Once a particular cycle and its position in the "acyclic part" of the computation (consuming the prefix w_{pref} and suffix w_{suf}) are chosen, the length of the computation in the cycle is determined by the total length ℓ. Thus, the number of accepted strings of length ℓ is upper bounded by the constant $m^3 \cdot |\Sigma|^m$.

Conversely, if L has constant density then, by Proposition 2, L can be represented as a finite union of regular expressions of the form xy^*z, $x, y, z \in \Sigma^*$. An naNFA with one cycle recognizes xy^*z, and the languages recognized by naNFAs are clearly closed under union. □

By considering unary regular languages it is easy to see that a constant density language can be recognized by an NFA that is not nearly acyclic. However, for DFAs, we get the implication also in the converse direction.

[3] This is a conservative upper bound chosen to keep the argument simple. If A were to have m cycles, the length of the cycles naturally could not be m.

Theorem 7. *Any DFA recognizing a constant density language must be nearly acyclic.*

As a corollary, we get that determinizing an naNFA must result in a nearly acyclic DFA. This could of course also be seen using a direct construction but it would require some effort.

Corollary 3. *Let A be an naNFA and let D be the DFA obtained from A using the subset construction. Then D is nearly acyclic.*

5 Conclusion

We have given an algorithm to decide whether the depth path width of an NFA is unbounded, and characterized automata with bounded depth path width as the class of nearly acyclic NFAs. We have given an upper bound for the finite depth path width of an m-state NFA over an alphabet of size k and shown that this bound is tight.

Nearly acyclic NFAs extend the class of acyclic NFAs that characterize the class of finite languages. A tight state complexity bound for determinizing acyclic NFAs is known [20]. From Corollary 3 we know that determinizing a nearly acyclic NFA always results in a nearly acyclic DFA. Establishing the worst-case size blow-up of determinizing a nearly acyclic NFA is a topic for future research. The size blow-up is at least as great as the exponential lower bound for determinizing unary (nearly acyclic) NFAs having cycles of different prime lengths [4].

Minimization of NFAs is PSPACE-complete [9] and remains NP-hard even for restricted subclasses of acyclic NFAs [1]. A linear time minimization algorithm for acyclic DFAs is given by Bubenzer [2] and incremental minimization techniques for acyclic NFAs have been considered e.g. by Lamperti et al. [14]. A topic for future work could be also to extend such methods for nearly acyclic NFAs.

Acknowledgments. Research supported by NSERC grant OGP0147224. Full version of the work can be found in [12].

References

1. Björklund, H., Martens, W.: The tractability frontier for NFA minimization. J. Comput. Syst. Sci. **78**(1), 198–210 (2012)
2. Bubenzer, J.: Cycle-aware minimization of acyclic deterministic finite-state automata. Discrete Appl. Math. **163**, 238–246 (2014)
3. Chan, T.-H., Ibarra, O.: On the finite valuedness problem for sequential machines. Theoret. Comput. Sci. **23**, 95–101 (1983)
4. Chrobak, M.: Finite automata and unary languages. Theoret. Comput. Sci. **47**, 149–158 (1986)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2009)

6. Goldstine, J., Leung, H., Wotschke, D.: On the relation between ambiguity and nondeterminism in finite automata. Inform. Comput. **100**, 261–270 (1992)
7. Goldstine, J., Kintala, C.M.R., Wotschke, D.: On measuring nondeterminism in regular languages. Inform. Comput. **86**(2), 261–270 (1990)
8. Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. Univ. Comput. Sci. **8**(2), 193–234 (2002)
9. Holzer, M., Kutrib, M.: Descriptional and computational complexity of finite automata - a survey. Inform. Comput. **209**(3), 456–470 (2011)
10. Hromkovič, J., Seibert, S., Karhumäki, J., Klauck, H., Schnitger, G.: Communication complexity method for measuring nondeterminism in finite automata. Inform. Comput. **172**(2), 202–217 (2002)
11. Hromkovič, J., Schnitger, G.: Ambiguity and communication. Theory Comput. Syst. **48**(3), 517–534 (2011)
12. Keeler, C.: New metrics for finite automaton complexity and subregular language hierarchies. QSPACE. https://qspace.library.queensu.ca/handle/1974/15329
13. Kintala, C.M.R., Pun, K.Y., Wotschke, D.: Concise representations of regular languages by degree and probabilistic finite automata. Math. Syst. Theory **26**(4), 379–395 (1993)
14. Lamperti, G., Scandale, M., Zanella, M.: Determinization and minimization of finite acyclic automata by incremental techniques. Softw. Pract. Exp. **46**(4), 513–549 (2016)
15. Leung, H.: Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata. SIAM J. Comput. **27**(4), 1073–1082 (1998)
16. Msiska, M., van Zijl, L.: Interpreting the subset construction using finite sublanguages. In: Proceedings of Prague Stringology Conference 2016, pp. 48–62 (2016)
17. Palioudakis, A., Salomaa, K., Akl, S.G.: State complexity of finite tree width NFAs. J. Automata Lang. Comb. **17**(2–4), 245–264 (2012)
18. Palioudakis, A., Salomaa, K., Akl, S.G.: Quantifying nondeterminism in finite automata. Ann. Univ. Bucharest Informatica **62**(2), 89–100 (2015)
19. Ravikumar, B., Ibarra, O.H.: Relating the type of ambiguity of finite automata to the succinctness of their representation. SIAM J. Comput. **18**(6), 1263–1282 (1989)
20. Salomaa, K., Yu, S.: NFA to DFA transformation for finite languages over arbitrary alphabets. J. Automata Lang. Comb. **2**(3), 177–186 (1997)
21. Shallit, J.: Numeration systems, linear recurrences, and regular sets. Inf. Comput. **113**(2), 331–347 (1994)
22. Shallit, J.: Second Course in Formal Languages and Automata Theory. Cambridge University Press, Cambridge (2009)
23. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972)
24. Weber, A., Seidl, H.: On the degree of ambiguity of finite automata. Theoret. Comput. Sci. **88**(2), 325–349 (1991)
25. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, pp. 41–110. Springer, Heidelberg (1997)