S©MR ISSN 1813-3304

СИБИРСКИЕ ЭЛЕКТРОННЫЕ
МАТЕМАТИЧЕСКИЕ ИЗВЕСТИЯ

Siberian Electronic Mathematical Reports
http://semr.math.nsc.ru

Том 15, стр. 1426–1442 (2018) УДК 519.713.1, 510.633
DOI 10.17377/semi.2018.15.117 MSC 68Q45

Special issue: Groups and Graphs, Metrics and Manifolds — G2M2 2017

USING SAT SOLVERS FOR SYNCHRONIZATION ISSUES IN NON-DETERMINISTIC AUTOMATA

H. SHABANA, M.V. VOLKOV

Abstract. We approach the problem of computing a D_3-synchronizing word of minimum length for a given nondeterministic automaton via its encoding as an instance of SAT and invoking a SAT solver. We also present some experimental results.

Keywords: nondeterministic automaton, synchronizing word, SAT, SAT-solver, random automaton.

1. Background and overview

We assume the reader's familiarity with some basic concepts of computational complexity theory that can be found in the early chapters of any general complexity theory text such as, e.g., [1]. As far as automata theory is concerned, we have tried to make the paper, to a reasonable extent, self-contained.

One of the significant concepts for digital systems is *synchronization*. It means that all parts of the system are in agreement regarding the present state of the system. This concept is of immense importance in fields such as coding theory, conformance testing, biocomputing, industrial robotics, and many others, and also leads to intriguing mathematical questions, see, e.g., the survey [2] or the chapter [3] of the forthcoming "Handbook of Automata Theory".

Shabana, H., Volkov, M.V., Using Sat solvers for synchronization issues in non-deterministic automata.

Supported by the Ministry of Education and Science of the Russian Federation, project no. 1.3253.2017, and the Competitiveness Enhancement Program of Ural Federal University.
Received January, 15, 2018, published November, 15, 2018.

From the viewpoint of mathematics, discrete systems are often modeled as finite automata. A *finite automaton* is a triple $\mathscr{A} = (Q, \Sigma, \delta)$, where Q is a finite non-empty set which elements are referred to as *states*, Σ is a finite non-empty set which is called the *input alphabet* and which elements are referred to as *input symbols* or *input letters*, and δ is a map, called the *transition function*, that describes the action of symbols in Σ at states in Q. Finite automata are usually classified into three categories according to the nature of their transition function.

DFA: $\mathscr{A} = (Q, \Sigma, \delta)$ is a *deterministic finite automaton* (DFA) if the transition function δ is a total map $Q \times \Sigma \to Q$, that is, $\delta(q, s)$ is defined for every state $q \in Q$ and for every symbol $s \in \Sigma$. We interpret $\delta(q, s)$ as the next state where the DFA would move to if it was at the state q and read the symbol s.

PFA: $\mathscr{A} = (Q, \Sigma, \delta)$ is a *partial finite automaton* (PFA) if the transition function δ is a partial map $Q \times \Sigma \to Q$, that is, $\delta(q, s)$ is defined for some pairs $(q, s) \in Q \times \Sigma$ but may be undefined for some other pairs. We again interpret $\delta(q, s)$, provided it is defined, as the next state where the PFA would move to if it was at the state q and read the symbol s, and we write $\delta(q, s) = \varnothing$ to indicate that $\delta(q, s)$ is undefined[1].

NFA: $\mathscr{A} = (Q, \Sigma, \delta)$ is a *nondeterministic finite automaton* (NFA) if the transition function δ is a map $Q \times \Sigma \to \mathcal{P}(Q)$, where $\mathcal{P}(Q)$ is the power set of Q, that is, for every state $q \in Q$ and for every symbol $s \in \Sigma$, the expression $\delta(q, s)$ is not a single state, but rather a subset of states. If this subset is non-empty, we interpret it as the set of all possible states where the NFA could move to if it was at the state q and read the symbol s. If $\delta(q, s) = \varnothing$, we say that the action of s is undefined at q.

Clearly, both DFSs and PFAs can be considered as special instances of NFAs. Therefore, in the sequel, we define all concepts for NFAs, commenting on their specializations for NFAs and PFAs, if necessary.

We represent a given automaton $\mathscr{A} = (Q, \Sigma, \delta)$ by the labeled directed graph with the vertex set Q, the label alphabet Σ, and the set of labeled edges

$$\{q \xrightarrow{s} q' \mid q, q' \in Q,\ s \in \Sigma,\ q' \in \delta(q, s)\}.$$

Figure 1 shows examples of a DFA (left) and a NFA (right). We adopt the convention that edges with multiple labels represent bunches of parallel edges. Thus, the edge $1 \xrightarrow{a,c} 0$ in Figure 1 represents the two parallel edges $1 \xrightarrow{a} 0$ and $1 \xrightarrow{c} 0$, etc.

Given an alphabet Σ, a *word* over Σ is a finite sequence of symbols from Σ. We do not exclude the empty sequence from this definition; that is, we allow the *empty word*. The set of all words over Σ including the empty word is denoted by Σ^* and is referred to as the *free monoid over* Σ. If $w = a_1 \cdots a_\ell$ with $a_1, \ldots, a_\ell \in \Sigma$ is a non-empty word over Σ, the number ℓ is said to be the *length* of w and is denoted by $|w|$. The length of the empty word is defined to be 0. The set of all words of a given length ℓ over Σ is denoted by Σ^ℓ.

For every NFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$, the transition function δ can be extended to a function $\mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ (still denoted by δ) by induction on the length of $w \in \Sigma^*$. If $|w| = 0$, that is, w is the empty word, then, for each $X \subseteq Q$, we let $\delta(X, w) = X$. If $|w| > 0$, we represent w as $w = sw'$ with $w' \in \Sigma^*$ and $s \in \Sigma$

[1]It should be noted that in the literature, automata that we call PFAs sometimes are referred to as deterministic finite automata while our DFAs are called *complete* deterministic finite automata.

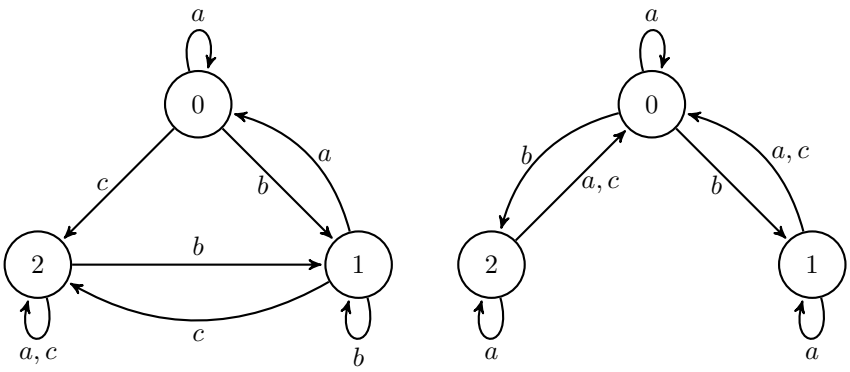

FIGURE 1. A DFA (left) and a NFA (right) with $Q = \{0, 1, 2\}$ and $\Sigma = \{a, b, c\}$

and, for each $X \subseteq Q$, let $\delta(X, w) = \bigcup_{q \in X} \delta(\delta(q, s), w')$. (The right hand side of the latter equality is defined by the induction assumption since $|w'| < |w|$.) To lighten the notation, we write $q.w$ for $\delta(q, w)$ and $X.w$ for $\delta(X, w)$ whenever we deal with a fixed automaton.

Here we are interested in *synchronization* of finite automata. The idea of *synchronization* is as follows: for a given automaton, we are looking for an input word that directs the automaton to a specific state, no matter at which state the automaton was at the beginning. This input is called a *synchronizing word*, and if an automaton possesses such a word, it is called *synchronizing*.

The above informal idea of synchronization is easy to formalize for DFAs but for NFAs it admits several non-equivalent formalizations. First, we recall the three versions that were suggested in [4] and have been widely studied thereafter.

Let $\mathscr{A} = (Q, \Sigma, \delta)$ be an NFA, $i = 1, 2, 3$. A word $w \in \Sigma^*$ is said to be D_i-*synchronizing* for $\mathscr{A}$ if it satisfies the condition (D_i) from the list below:

(D_1): $\forall q \in Q\ (q.w \neq \varnothing\ \wedge\ |q.w| = |Q.w| = 1)$;
(D_2): $\forall q \in Q\ (q.w \neq \varnothing\ \wedge\ q.w = Q.w)$;
(D_3): $\bigcap_{q \in Q} q.w \neq \varnothing$.

A NFA is called D_i-*synchronizing*, $i = 1, 2, 3$, if it has a D_i-synchronizing word[2]. It should be clear that every D_1-synchronizing word is also D_2-synchronizing and every D_2-synchronizing word is also D_3-synchronizing. The converse is not true in general. For an illustration, consider the NFA $\mathscr{A}$ in Figure 1 (right). It is easy to see that for it, the word abc is D_1-synchronizing, the word ab is D_2-synchronizing, but not D_1-synchronizing, and the word a is D_3-synchronizing, but not D_2-synchronizing. Moreover, the NFA obtained from $\mathscr{A}$ by omitting the letter c is D_2-synchronizing, but not D_1-synchronizing, while the NFA obtained from $\mathscr{A}$ by omitting the letters b and c is D_3-synchronizing, but not D_2-synchronizing.

Yet another version of synchronization for NFAs has been studied by Martyugin, see, e.g., [5]. Let $\mathscr{A} = (Q, \Sigma, \delta)$ be an NFA. A word $w = a_1 \cdots a_\ell$ with $a_1, \ldots, a_\ell \in \Sigma$

[2]In some sources, the requirement $q.w \neq \varnothing$ is not included in the definition of D_2-synchronization. If one omits this requirement, every word that is nowhere defined becomes D_2-synchronizing.

is said to be *carefully synchronizing* for $\mathscr{A}$ if it satisfies the condition (C), being the conjunction of $(C1)$–$(C3)$ below:

$(C1)$: $q.a_1$ is defined for all $q \in Q$,
$(C2)$: $p.a_i$ with $1 < i \leq \ell$ is defined for all $p \in Q.a_1 \cdots a_{i-1}$,
$(C3)$: $|Q.w| = 1$.

Thus, when w is applied at any state in Q, no undefined transition occurs during the course of application. Clearly, every carefully synchronizing word is also D_1-synchronizing but the converse is not true. For instance, the word abc is not carefully synchronizing for the NFA $\mathscr{A}$ in Figure 1 (right); moreover, this NFA possesses no carefully synchronizing word. We call a NFA *carefully synchronizing* if it admits a carefully synchronizing word. Thus, if we denote by $\mathbf{D}_i$, $i = 1, 2, 3$, the class of all D_i-synchronizing NFAs and by $\mathbf{C}$ the class of all carefully synchronizing NFAs, we have the following strict inclusions:

$$\mathbf{C} \subset \mathbf{D}_1 \subset \mathbf{D}_2 \subset \mathbf{D}_3.$$

In this paper, we consider D_3-synchronization. As it can been seen from the above discussion, it is the most general version of synchronization for NFAs amongst those considered in the literature so far. Besides that, we think that it reasonably reflects the basic nature of non-determinism. Indeed, if an NFA $\mathscr{A} = (Q, \Sigma, \delta)$ is used as an *acceptor*, we designate some states in Q as initial and final and then say that $\mathscr{A}$ *accepts* a word $w \in \Sigma^*$ whenever there exists a path labeled w that starts at an initial state and terminates at a final state. The definition of a D_3-synchronizing word very much resembles this concept: a word $w \in \Sigma^*$ is D_3-synchronizing whenever for each $q \in Q$, there exists a path labeled w that starts at q and terminates at a certain common state, independent of q. In both cases we do not require that a starting state uniquely determines the path labeled w nor that every path labeled w with a given starting state should arrive at a final/common state.

We also mention in passing that D_3-synchronization gets a very transparent meaning within a standard matrix representation of NFAs. In this representation, an NFA $\mathscr{A} = (Q, \Sigma, \delta)$ becomes a collection of $|\Sigma|$ Boolean $Q \times Q$-matrices where to each input symbol $s \in \Sigma$, a matrix $M(s)$ is assigned such that the (q, q')-entry of $M(s)$ is 1 if $q' \in \delta(q, s)$ and 0 otherwise. Then it is not hard to realize that the automaton $\mathscr{A}$ is D_3-synchronizing if and only if some product of the matrices $M(s)$, $s \in \Sigma$, has a column consisting entirely of 1s.

Some information about D_3-synchronization can be found in Chapter 8 of Ito's monograph [6]; recently, some aspects of D_3-synchronization has been considered in [7–10]. (The papers [7, 8] use the language of matrices rather than that of automata.)

It is easy to see that each of the conditions (C), (D_1), (D_2), (D_3) leads to the same notion when restricted to PFAs. Thus, for PFAs and, in particular, for DFAs, we call a word *synchronizing* if it satisfies any of these conditions. A PFA (in particular, a DFA) is said to be *synchronizing* if it has a synchronizing word.

It is known that the problem of determining whether or not a DFA with n states is synchronizing can be solved in $O(n^2)$ time, see, e.g., [2,3] or [11]. If such a DFA is synchronizing, it always has a synchronizing word of length $(n^3 - n)/6$, see [12], and is conjectured that it must have a synchronizing word of length $(n - 1)^2$ (this is the famous Černý conjecture). In contrast, the problem of determining whether or not a given PFA is synchronizing is known to be PSPACE-complete and there

is no polynomial in n upper bound on the length of synchronizing words for a synchronizing PFA with n states. (These results were found by Rystsov in the early 1980s [13, 14] and later rediscovered (and strengthened) by Martyugin [15].) This readily implies that the problem of determining whether or not a given NFA is D_3-synchronizing as well as the problem of finding a D_3-synchronizing word of minimum length are computationally hard.

Nowadays, a popular approach to computationally hard problems consists in encoding them as instances of the Boolean satisfiability problem (SAT) that are then fed to a SAT-solver, that is, a specialized program designed to solve instances of SAT. We refer to this approach as the *SAT-solver method*. Modern SAT solvers can solve instances with hundreds of thousands of variables and millions of clauses within a few minutes. Thanks to this remarkable progress, the SAT-solver method has proved to be very efficient for an extremely wide range of problems of both theoretical and practical importance. Its applications are far too numerous to be listed here; some examples of such applications can be found in the survey [16], which also gives a smart introduction into the area. Here we mention only three recent papers that deal with two difficult problems related to finite automata. Geldenhuys, van der Merwe, and van Zijl [17] have used the SAT-solver method to attack the minimization problem for NFAs. In the minimization problem, which is known to be PSPACE-complete [18], an NFA $\mathscr{A}$ with designated initial and final states is given, and one looks for an NFA of minimum size that accepts the same set of words as $\mathscr{A}$. Skvortsov and Tipikin [19] have applied the method to find a synchronizing word of minimum length for a given DFA with two input symbols, and Güniçen, Erdem, and Yenigün [20] have extended their approach to DFAs with arbitrary input alphabets. The problem of finding a synchronizing word of minimum length is known to be hard for the complexity class $\mathrm{FP}^{\mathrm{NP[log]}}$, the functional analogue of the class of problems solvable by a deterministic polynomial-time Turing machine that has an access to an oracle for an NP-complete problem, with the number of queries being logarithmic in the size of the input [21].

In the present paper, we use the SAT-solver method to approach the problem of computing a D_3-synchronizing word of minimum length for a given NFA. It should be stressed that neither the encoding of NFAs used in [17] nor the encodings of synchronization used in [19, 20] work for our problem, and therefore, we have had to invent essentially different encodings.

The rest of the paper is divided into three sections. Section 2 describes our basic encoding and Section 3 presents implementation details and some of our experimental results. The final section contains several concluding remarks and a discussion of possible further developments.

2. Encoding

By the encoding of a problem, we mean a polynomial reduction from this problem to SAT. First, let us precisely formulate the problem which we are interested in.

D3W (the existence of a D_3-synchronizing word of a given length):
Input: A NFA $\mathscr{A}$ with two input symbols and a positive integer ℓ.
Output: YES if $\mathscr{A}$ has a D_3-synchronizing word of length ℓ; NO otherwise.

The integer ℓ is assumed to be given in unary. With ℓ given in binary, a polynomial reduction from D3W to SAT is hardly possible. Indeed, it is known that every D_3-synchronizing NFA with n states has a D_3-synchronizing word of length at most 2^n, see [6, Proposition 8.3.10]. Hence, given a NFA $\mathscr{A}$ with n states and two input symbols, the answer to the problem D3W for the instance $(\mathscr{A}, 2^n)$ is YES if and only if $\mathscr{A}$ is D_3-synchronizing. As it was mentioned, the problem of determining whether or not a given NFA is D_3-synchronizing is PSPACE-complete, whence the version of D3W in which the integer parameter is given in binary is PSPACE-hard. On the other hand, SAT is an archetypical problem in NP, and clearly, the existence of a polynomial reduction from a PSPACE-hard problem to a problem in NP would imply that the polynomial hierarchy collapses at level 1. While, as it is usual in complexity theory, the question of whether or not the polynomial hierarchy collapses at any level is open, a common opinion is that it does not.

In contrast, the version of D3W with the integer parameter given in unary is easily seen to belong to NP. Indeed, given an instance $(\mathscr{A}, \ell)$ of D3W in this setting, one has right to guess a word w of length ℓ over the input alphabet of $\mathscr{A}$ as w is obviously of polynomial size in terms of the size of the instance. Then one just checks whether or not w is D_3-synchronizing for $\mathscr{A}$, and time spent for this check is clearly polynomial in the size of $(\mathscr{A}, \ell)$. By Cook's classic theorem (see, e.g., [1, Theorem 8.2]), SAT is NP-complete, and by the very definition of NP-completeness, there exists a polynomial reduction from our version of D3W to SAT.

Recall that an instance of SAT is a pair (V, C), where V is a set of Boolean variables and C is a collection of clauses over V. (A *clause* over V is a disjunction of literals and a *literal* is either a variable in V or the negation of a variable in V.) Any *truth assignment* on V, i.e., any map $\varphi\colon V \to \{0, 1\}$, extends to a map $C \to \{0, 1\}$ (still denoted by φ) via the usual rules of propositional calculus: $\varphi(\neg x) = 1 - \varphi(x)$, $\varphi(x \vee y) = \max\{\varphi(x), \varphi(y)\}$. A truth assignment φ *satisfies* C if $\varphi(c) = 1$ for all $c \in C$. The answer to an instance (V, C) is YES if (V, C) has a *satisfying assignment* (i.e., a truth assignment on V that satisfies C) and NO otherwise.

Thus, a polynomial reduction from D3W to SAT is an algorithm that, given an arbitrary instance $(\mathscr{A}, \ell)$ of D3W, constructs, in polynomial time with respect to the size of $(\mathscr{A}, \ell)$, an instance (V, C) of SAT such that the answer to $(\mathscr{A}, \ell)$ is YES if and only if so is the answer to (V, C). Of course, neither a pure existence statement nor any general construction that can be extracted from one of the proofs of Cook's theorem can be used for our purposes. We need a sort of "practical" reduction: it should be explicit, easy to implement, and economical in the sense that the degrees of the polynomials that bound the number of variables in V and the number of clauses in C in terms of the size of $(\mathscr{A}, \ell)$ should be as small as possible.

In the following presentation of our encoding, precise definitions and statements are interwoven with less formal comments explaining the "physical" meaning of variables and clauses we introduce and with estimations of their numbers.

So, take a NFA $\mathscr{A} = (Q, \Sigma, \delta)$ and an integer $\ell > 0$. Denote the size of Q by n and fix some numbering of the states in Q so that $Q = \{q_1, \dots, q_n\}$. Recall that we consider the problem D3W for NFAs with two input symbols, so let $\Sigma = \{0, 1\}$.

We start with introducing the variables used in the instance (V, C) of SAT that encodes $(\mathscr{A}, \ell)$. The set V consists of three sorts of variables: *letter variables*, *token variables*, and *synchronization variables*.

The letter variables are $x_1, \dots, x_\ell$. They are just placeholders for the input symbols 0 and 1. There is an obvious 1-1 correspondence between the truth assignments on the set $X = \{x_1, \dots, x_\ell\}$ and the words of length ℓ over $\{0, 1\}$: given a truth assignment $\varphi\colon X \to \{0, 1\}$, the corresponding word is $\varphi(x_1) \cdots \varphi(x_\ell)$, and, conversely, given a word $a_1 \cdots a_\ell$ with $a_1, \dots, a_\ell \in \{0, 1\}$, the corresponding truth assignment is $x_t \mapsto a_t$ for each $t = 1, \dots, \ell$.

The token variables are y_{ij}^t where $i, j = 1, \dots, n$ and $t = 0, 1, \dots, \ell$. To explain the role of these variables, we use a solitaire-like game Γ on the labeled directed graph representing the NFA $\mathscr{A}$. In the initial position of Γ, each state $q_i \in Q$ holds exactly one token denoted **i**. In the course of the game, tokens migrate and may multiply or disappear according to certain rules that will be specified a bit later, when we describe the clauses in C. For the moment, it is sufficient to say that the rules are designed to ensure that the variable y_{ij}^t gets value 1 in a satisfying truth assignment for C if and only if after t rounds of the game, one of the tokens held by the state q_j is **i**.

The synchronization variables are $z_1, \dots, z_n$. They play the role of indicators showing which states may occur at the end of the synchronization process. By the definition of D_3-synchronization, the answer to the instance $(\mathscr{A}, \ell)$ is YES if and only if there exists a word $w \in \Sigma^\ell$ such that $\bigcap_{q \in Q} q.w \neq \varnothing$. The clauses of C will be chosen so that the variable z_j gets value 1 in a satisfying assignment for C if and only the state q_j belongs to the set $\bigcap_{q \in Q} q.w$, where w is the word defined by the restriction of the assignment to X.

We see that the total number of variables in V is $\ell + n^2(\ell + 1) + n$.

Now we turn to constructing the set of clauses C. It is the disjoint union of $\ell + 1$ sets: the set C_0 of *initial clauses*, the sets C_t, $t = 1, \dots, \ell$, of *transition clauses*, and the set S of *synchronization clauses*.

The clauses in C_0 describe the initial position of our game Γ. As mentioned, in this position, each state $q_i \in Q$ holds the token **i** and nothing else. It order to reflect this setting, we let C_0 consist of the clauses $y_{11}^0, \dots, y_{nn}^0$ along with all clauses of the form $\neg y_{ij}^0$ with $i \neq j$. Altogether, C_0 contains n^2 one-literal clauses.

They are the clauses in C_t, $t = 1, \dots, \ell$, that encode the rules of Γ. The rules are as follows. At each move an input symbol $a \in \Sigma$ is chosen. Then for each state $q \in Q$ such that $q.a \neq \varnothing$, all tokens that were held by q slide along the edges labeled a to all states in the set $q.a$. (If $|q.a| > 1$, then every token held by q multiplies to $|q.a|$ identical tokens, one for each state in $q.a$.) If $q.a = \varnothing$, then all tokens that were held by q disappear. Thus, after the move, the token **i** occurs at a state $p \in Q$ if and only if $p \in q.a$ for some state q that had held **i** just prior to the move.

For an illustration, Figure 2 demonstrates the initial distribution of tokens on a 5-state NFA with the input alphabet $\{0, 1\}$ (top), along with the outcomes of the first move, depending on whether 0 or 1 has been chosen for the move (bottom left and bottom right, respectively).

The following observation is immediate.

Lemma 1. *Suppose that in the game Γ played on $\mathscr{A} = (Q, \Sigma, \delta)$, the sequence of chosen symbols forms a word $w \in \Sigma^*$. Then for each $i = 1, \dots, n$, the set of states holding the token* **i** *at the end of the game is $q_i.w$.*

Now we express the rules of Γ by formulas of propositional logic. For a state $q \in Q$, let $P_0(q)$ and $P_1(q)$ stand for the sets of all preimages of q under the actions of the input symbols 0 and respectively 1, that is, if a is either of the two symbols,

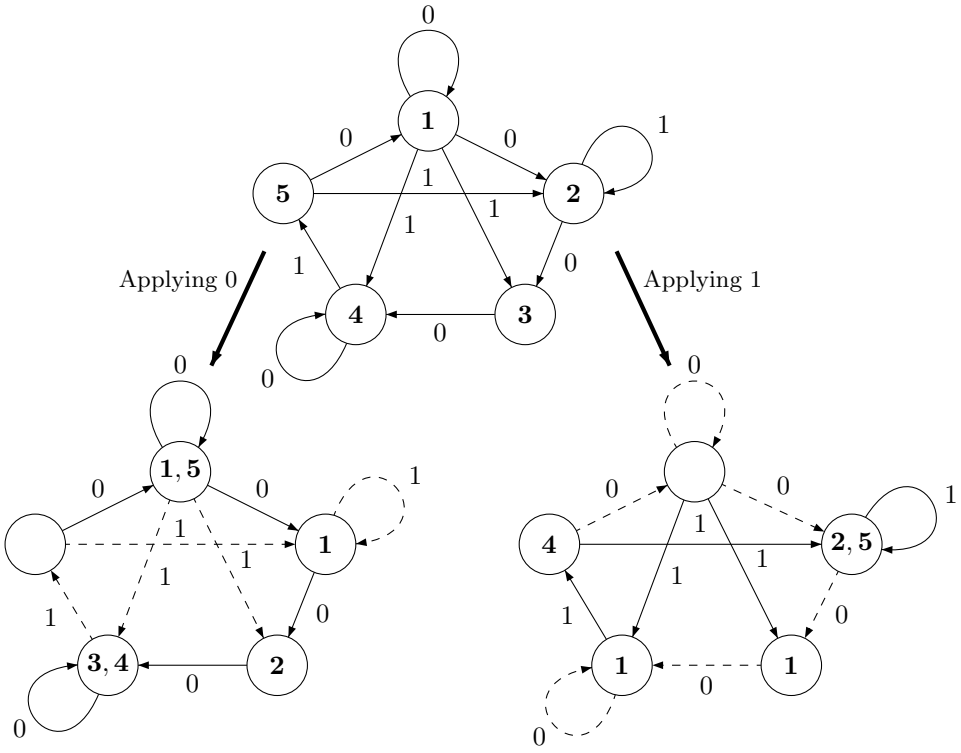

FIGURE 2. Redistribution of tokens after the first move

$P_a(q) = \{p \in Q \mid q \in p.a\}$. Consider for every $t = 1, \ldots, \ell$ and all $i, j = 1, \ldots, n$, the following formulas:

$$\Psi_{ij}^t : \quad y_{ij}^t \equiv \Big(x_t \wedge \bigvee_{q_k \in P_1(q_j)} y_{ik}^{t-1}\Big) \vee \Big(\neg x_t \wedge \bigvee_{q_h \in P_0(q_j)} y_{ih}^{t-1}\Big).$$

Observe that the equivalence Ψ_{ij}^t just translates in the language of propositional logic our propagation rule for the tokens that says that the token **i** occurs at the state q_j after t moves if and only if one of the following alternatives takes place:

- the t-th move was done with the input symbol 1 and one of the preimages of q_j under the actions of 1 was holding **i** after $t-1$ moves, or
- the t-th move was done with the input symbol 0 and one of the preimages of q_j under the actions of 0 was holding **i** after $t-1$ moves.

Lemma 2. *For every $t = 0, 1, \ldots, \ell$, every truth assignment φ on the set X of letter variables has a unique extension $\overline{\varphi}$ to the token variables y_{ij}^s that makes the clauses in C_0 and the formulas Ψ_{ij}^s hold $(i, j = 1, \ldots, n,\ s = 1, \ldots, t)$. The token variable y_{ij}^s gets value 1 under $\overline{\varphi}$ if and only if after the moves $\varphi(x_1), \ldots, \varphi(x_s)$ of the game Γ, one of the tokens held by the state q_j is* **i**.

Proof. We induct on t. The indiction basis $t = 0$ is clear: we have to satisfy the clauses in C_0 and the only way to satisfy a one-literal clause is to assign value 1 to

its only literal. Hence, independently of φ, we have to set for all $i, j = 1, \dots, n$,

$$\overline{\varphi}(y_{ij}^0) = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Observe that then, in the accordance with the initial setting of the game Γ, the variable y_{ij}^0 gets value 1 exactly when the token held by the state q_j is $\mathbf{i}$.

Now suppose that $t > 0$ and there exists a unique way to define $\overline{\varphi}(y_{ij}^s)$ for all $i, j = 1, \dots, n$, $s = 0, \dots, t-1$, such that the clauses in C_0 and the formulas Ψ_{ij}^s with $i, j = 1, \dots, n$ and $s = 1, \dots, t-1$ hold. If the variable x_t is assigned the value $\varphi(x_t)$, the value of the right hand side of each equivalence Ψ_{ij}^t is uniquely defined, and to make this equivalence hold, we must assign the value to the left hand side, that is, the variable y_{ij}^t. This gives a unique way to extend $\overline{\varphi}$ to the variables y_{ij}^t, where $i, j = 1, \dots, n$. As observed prior to the formulation of the lemma, the equivalences Ψ_{ij}^t express the rule of Γ. Therefore the token $\mathbf{i}$ migrates to the state q_j after the move $\varphi(x_t)$ if and only if the variable y_{ij}^t gets value 1 under this extension. □

For each $t = 1, \dots, \ell$, we define the set C_t as the set of all clauses of a suitable CNF (conjunctive normal form) equivalent to $\bigwedge\limits_{1 \le i,j \le n} \Psi_{ij}^t$. In our basic encoding, the set C_t consists of the following clauses:

$$\neg y_{ij}^t \vee x_t \vee \bigvee_{q_h \in P_0(q_j)} y_{ih}^{t-1}, \qquad \neg y_{ij}^t \vee \neg x_t \vee \bigvee_{q_k \in P_1(q_j)} y_{ik}^{t-1}, \tag{1}$$

$$y_{ij}^t \vee \neg x_t \vee \neg y_{ik}^{t-1} \quad \text{for each } q_k \in P_1(q_j), \tag{2}$$

$$y_{ij}^t \vee x_t \vee \neg y_{ih}^{t-1} \quad \text{for each } q_h \in P_0(q_j). \tag{3}$$

The verification of the equivalence between $\bigwedge\limits_{1 \le i,j \le n} \Psi_{ij}^t$ and the conjunction of the clauses in (1)–(3) is routine, and we omit it.

It may be worth explaining how the clauses of the form (1)–(3) are understood in the case when one of the sets $P_0(q_j)$ or $P_1(q_j)$ or both of these sets happen to be empty. In (1) the disjunctions over the empty sets are omitted so that if, say, $P_0(q_j) = \varnothing$, then the first clause in (1) reduces to $\neg y_{ij}^t \vee x_t$. As for (2) or (3), these clauses disappear whenever $P_1(q_j)$ or, respectively $P_0(q_j)$ are empty.

In order to calculate the number of clauses in C_t, denote by m the number of all *transitions in* $\mathscr{A}$, that is, triples $(q, a, q') \in Q \times \Sigma \times Q$ with $q' \in \delta(q, a)$. Clearly, for each fixed i, the number $\sum_{j=1}^n (|P_1(q_j)| + |P_0(q_j)|)$ of clauses of the forms (2) and (3) is equal to m, whence the total number of such "short" clauses is mn. As for "long" clauses in (1), there are at most two such clauses for each fixed pair (i, j), whence their total number does not exceed $2n^2$. Altogether, $|C_t| \le n(m + 2n)$ for each $t = 1, \dots, \ell$.

Lemma 1 readily implies that a word $w = a_1 \cdots a_\ell$ is D_3-synchronizing for $\mathscr{A}$ if and only if after the moves $a_1, \dots, a_\ell$ in the game Γ on $\mathscr{A}$, some state q_j holds all tokens $\mathbf{1}, \dots, \mathbf{n}$. This is equivalent to saying that the formula

$$\bigvee_{j=1}^n \bigwedge_{i=1}^n y_{ij}^\ell \tag{4}$$

holds under the extension, specified in Lemma 2, of the truth assignment on X defined by w. A little difficulty is that a direct conversion of the formula (4) into a CNF produces 2^n clauses. To overcome this difficulty, we use a standard trick for which we need new variables (this is why we introduce synchronization variables). Let S consist of the following n^2+1 clauses:

$$\bigvee_{j=1}^{n} z_j \quad \text{and} \quad \neg z_j \vee y_{ij}^{\ell} \quad \text{for all} \quad i,j=1,\dots,n.$$

It is easy to see that the set S and the formula (4) are equisatisfiable; moreover, if $Y=\{y_{ij}^{\ell} \mid i,j=1,\dots,n\}$ and $Z=\{z_1,\dots,z_n\}$, then every truth assignment on Y that satisfies (4) can be extended to a truth assignment on $Y\cup Z$ that satisfies S, and, conversely, for every truth assignment on $Y\cup Z$ that satisfies S, its restriction to Y satisfies (4).

The whole set $C=S\cup\bigcup_{t=0}^{\ell} C_t$ consists of at most $n(m+2n)\ell+2n^2+1$ clauses. The number of transitions in a NFA with n states two input symbols is upper-bounded by $2n^2$, whence $|C|\le 2\ell n^3+2(\ell+1)n^2+1$. Thus, constructing (V,C) from $\mathscr{A}$ takes time polynomial in n and ℓ. Summarizing the above discussion, we arrive at the main result of the section.

Theorem 3. *An NFA $\mathscr{A}$ has a D_3-synchronizing word of length ℓ if and only if the instance (V,C) of SAT constructed above is satisfiable, and the construction takes time polynomial in the size of $\mathscr{A}$ and the value of ℓ. Moreover, by the construction, there is a 1-1 correspondence between the D_3-synchronizing words of length ℓ for $\mathscr{A}$ and the restrictions of satisfying assignments of (V,C) to the letter variables.*

Remark 4. We do not claim that the above reduction of D3W to SAT is optimal. For instance, it is possible to reduce the number of variables by getting rid of the letter variables. Namely, for each pair of $i,j\in\{1,\dots,n\}$ and each $t\in\{1,\dots,\ell\}$, one could take the clause

$$\neg y_{ij}^{t} \vee \bigvee_{q_h\in P_0(q_j)} y_{ih}^{t-1} \vee \bigvee_{q_k\in P_1(q_j)} y_{ik}^{t-1} \tag{5}$$

instead of the clauses in (1) and the set of clauses of the form

$$y_{ij}^{t} \vee \neg y_{ih}^{t-1} \vee \neg y_{ik}^{t-1} \quad \text{for } h \text{ and } k \text{ such that } q_h\in P_0(q_j) \text{ and } q_k\in P_1(q_j) \tag{6}$$

instead of the ones in (2) and (3). It is easy to see that (1) and (5) are equisatisfiable, and so are the sets of clauses in (2), (3) on the one hand and in (6) on the other.

We have preferred to keep the letter variables because of the fact mentioned in Theorem 3: if a D_3-synchronizing word of length ℓ exists, we can immediately recover it from the the restrictions of a satisfying assignment to the letter variables.

3. Experimental results

Here we overview our experiments and present some of their results. Our basic procedure has been organized as follows.

1. A positive integer n (the number of states) is fixed. In the experiments which results we report here, we have considered $n\le 100$.
2. A random NFA $\mathscr{A}$ with n states and 2 input symbols is generated. We have used two models of random generation that are specified below.

3. We check whether $\mathscr{A}$ has an input symbol whose action is defined at each state. If it is not the case, the NFA $\mathscr{A}$ cannot be D_3-synchronizing, and we return to Step 2 to generate another random NFA.
4. A positive integer ℓ_0 (the hypothetical length of the shortest D_3-synchronizing word for $\mathscr{A}$) is chosen. Initially, we chose ℓ_0 to be close to n but, as our early experiments have revealed, it is much more practical to start with smaller values of ℓ_0. We introduce three integer variables $\ell_{\min}$, ℓ, and $\ell_{\max}$ and initialize them as follows: $\ell_{\min} := 1$, $\ell := \ell_0$, $\ell_{\max} := 2\ell_0$.
5. The pair $(\mathscr{A}, \ell)$ is encoded into a SAT instance as described in Section 2.
6. A SAT solver is invoked to solve the SAT instance obtained in Step 5. We have used MiniSat 2.2.0; see [22] for a description of the underlying ideas of MiniSat and [23] for a discussion and the source code of the solver.
7. The binary search on ℓ is performed. In more detail, if the SAT solver returns YES on the encoding of the pair $(\mathscr{A}, \ell)$, we first check whether or not $\ell = \ell_{\min}$. If $\ell = \ell_{\min}$, then ℓ is the length of the shortest D_3-synchronizing word for $\mathscr{A}$, and we go to Step 2 to generate another random NFA. If $\ell > \ell_{\min}$, we update the variables $\ell_{\max}$ and ℓ by letting
$$\ell_{\max} := \ell, \quad \ell := \lfloor \frac{\ell_{\min} + \ell_{\max}}{2} \rfloor,$$
keep the value of $\ell_{\min}$ and go to Step 5. If the SAT solver returns NO on the encoding of the pair $(\mathscr{A}, \ell)$, we check whether or not $\ell = \ell_{\max}$. If $\ell = \ell_{\max}$, we interpret this as the evidence that the NFA $\mathscr{A}$ fails to be D_3-synchronizing[3] and go to Step 2 to generate another random NFA. If $\ell < \ell_{\max}$, we update the variables $\ell_{\min}$ and ℓ by letting
$$\ell_{\min} := \ell + 1, \quad \ell := \lceil \frac{\ell_{\min} + \ell_{\max}}{2} \rceil,$$
keep the value of $\ell_{\max}$ and go to Step 5.

We implemented the algorithm outlined above in C++ and compiled with GCC 4.9.2. In our experiments we used a personal computer with an Intel(R) Core(TM) i5-2520M processor with 2.5 GHz CPU and 4GB of RAM. For each fixed n, up to 400 NFAs that passed Step 3 were analyzed. The average calculation time (for one NFA) was 400 seconds for $n = 30$ and 4350 seconds for $n = 60$.

Now we describe the methods used for random generation of NFAs. It turns out that the literature on random NFAs is sparse and no "standard" notion of a random NFA seems to have been developed so far. For instance, Tabakov and Vardi, in their widely cited paper [24], defined a random NFA $\mathscr{A} = (Q, \{0, 1\}, \delta)$ as the pair (D_0, D_1) of directed graphs sharing Q as the vertex set, where the edges of D_0 and D_1 represented the transitions caused by the inputs 0 and 1, respectively, and for some integer parameter $k > 0$, each of the directed graphs D_0 and D_1 had k edges chosen uniformly at random from all possible $|Q|^2$ directed edges that could be drawn between the vertices in Q. While this definition was well suited for the purposes of [24], it does not look natural for us since there is no obvious reason why different input symbols should label the same number of transitions.

[3]Of course, the equality $\ell = \ell_{\max}$ only means that $\mathscr{A}$ has no D_3-synchronizing word of length $\leq 2\ell_0$, and it is not excluded, in principle, that the NFA is D_3-synchronizing but its shortest D_3-synchronizing word is very long. However, by suitable preprocessing and choosing an appropriate value of the parameter ℓ_0, we have got rid of the "bad" cases when the SAT solver returns NO and $\ell = \ell_{\max}$ in our experiments.

We have suggested and examined two other models to produce a random NFA $\mathscr{A} = (Q, \Sigma, \delta)$ with n states and 2 input symbols: the *uniform model* based on the uniform distribution and the *Poisson model* based on the Poisson distribution with some parameter λ. For each state $q \in Q$ and each symbol $s \in \Sigma$, we first choose a number $k \in \{0, 1, 2, \ldots, n\}$ that serves as the cardinality of the set $\delta(q, s)$. In the uniform model, each k is chosen with probability $\frac{1}{n+1}$ while in the Poisson model with parameter λ, each $k < n$ is chosen with probability $e^{-\lambda}\frac{\lambda^k}{k!}$ and n is chosen with probability $1 - e^{-\lambda}\sum_{k=0}^{n-1}\frac{\lambda^k}{k!}$. With k being chosen, we proceed the same in both models, by choosing a k-element subset from all $\binom{n}{k}$ subsets of Q with cardinality k uniformly at random and letting $\delta(q, s)$ be the chosen subset.

In each of the two models, it is easy to estimate the fraction of automata that survive Step 3. The corresponding results are stated in the following proposition which proof amounts to straightforward calculations and is therefore omitted.

Proposition 5. *The probability that a random NFA with n states and 2 input symbols has an input symbol whose action is defined at each state is*

$$2(1 - \frac{1}{n+1})^n - (1 - \frac{1}{n+1})^{2n} \tag{7}$$

if the NFA is generated under the uniform model and

$$2(1 - e^{-\lambda})^n - (1 - e^{-\lambda})^{2n} \tag{8}$$

if the NFA is generated under the Poisson model with parameter λ.

Observe that as n grows, the expression in (7) tends to $2e^{-1} - e^{-2} \approx 0.600$ while the expression in (8) tends to 0. In the further discussion, we always assume that the NFA considered have passed Step 3.

For the uniform model, our experiments produced results that may seem surprising at the first glance. Namely, it turns out that for an overwhelming majority of NFAs, the length of the shortest D_3-synchronizing word is equal to 2, and this conclusion does not depend on the state number n, at least within the range of our experiments (recall that we have considered $n \leq 100$). For an illustration, see Figure 3 in which the horizontal axis is the length of the shortest D_3-synchronizing word and the vertical axis is the number of NFAs. The blue and the yellow circles represent NFAs with 20 and 30 states respectively.

Insofar, we have got no rigorous theoretical explanation of the observed phenomenon. However, even a quick analysis of the uniform model reveals that NFAs it produces should tend to have rather short D_3-synchronizing words. Indeed, if an NFA $\mathscr{A} = (Q, \Sigma, \delta)$ with n states and 2 input symbols is generated under the uniform model, then the expected cardinality of the set $q.s$ is $\frac{n}{2}$ for every $q \in Q$ and $s \in \Sigma$. Therefore the expected size of every set of the form $q.w$ with $w \in \Sigma^2$ is close to n. Hence it is quite likely that $\bigcap_{q \in Q} q.w \neq \varnothing$ for some word w of length 2, which is then a D_3-synchronizing word for $\mathscr{A}$.

Since the uniform model produces no "slowly synchronizing" NFAs, there is no real need in using SAT-solvers in the uniform setting. Instead, we could have used a brute-force approach that, given an NFA $\mathscr{A} = (Q, \Sigma, \delta)$, writes all words over Σ up to a given length in the short-lex order and then lets each of these words act on $\mathscr{A}$ until it encounters a D_3-synchronizing word. As the above discussion shows, for a majority of NFAs, this approach would require to check only words up to length 2.

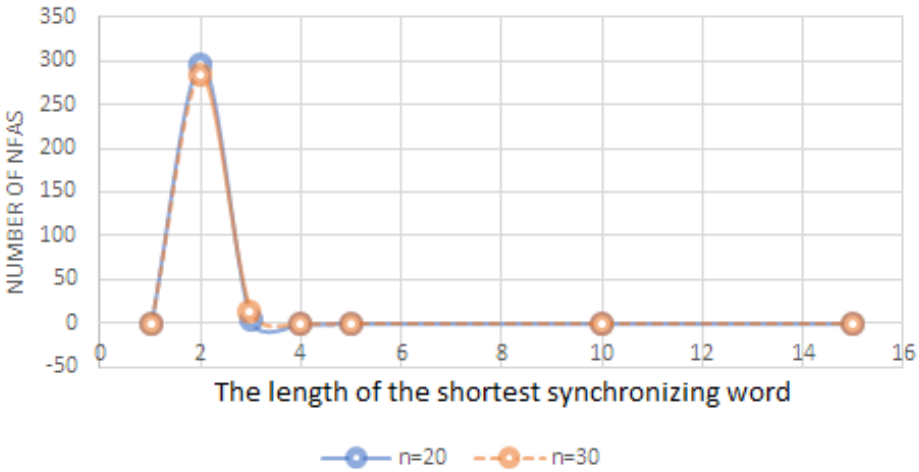

FIGURE 3. Distributions of 20- and 30-state NFAs generated under the uniform model according to the length of their shortest D_3-synchronizing words

Since we have realized that we need a more flexible way to generate random NFAs, we have developed the Poisson model. Some sample experimental results for the Poisson model are presented in Figure 4. The three histograms in Figure 4 correspond to 60-state NFAs generated under the Poisson models with three different values of the parameter λ and demonstrate how these NFAs are distributed according to the length of their shortest D_3-synchronizing words. As in Figure 3, the horizontal axis is the length of the shortest D_3-synchronizing word and the vertical axis is the number of NFAs. Observe that for $\lambda = 1$, we have registered some NFAs whose shortest D_3-synchronizing words have length more than 20. Under rather limited computational resources we used, analyzing such NFAs via the brute-force approach outlined in the above discussion of the uniform model appears to be problematic.

We see that if the number of states is fixed, the expected length of the shortest D_3-synchronizing word decreases as the parameter λ grows. This can be explained by an informal argument of the same flavour as the reasoning used above to explain the outcome of our experiments with NFAs generated under the uniform model. Indeed, if an NFA $\mathscr{A} = (Q, \Sigma, \delta)$ with n states and 2 input symbols is generated under the Poisson model with parameter λ, it follows from a basic property of the Poisson distribution that λ is close to the expected cardinality of sets $\delta(q, s)$ for every $q \in Q$ and $s \in \Sigma$. The larger are these sets, the smaller is the value of ℓ such that the expected size of sets of the form $q.w$ with $w \in \Sigma^\ell$ becomes close to n.

Our experiments also show that if the parameter λ is fixed, the expected length of the shortest D_3-synchronizing word grows with the number of states but the growth rate is rather small. For each $n \leq 100$, we have calculated the average length $E_1(n)$ of the shortest D_3-synchronizing words for n-state NFAs generated under the Poisson model with $\lambda = 1$. Then, using the method of least squares, we have searched for an explicit function of n that approximates $E_1(n)$ and found the

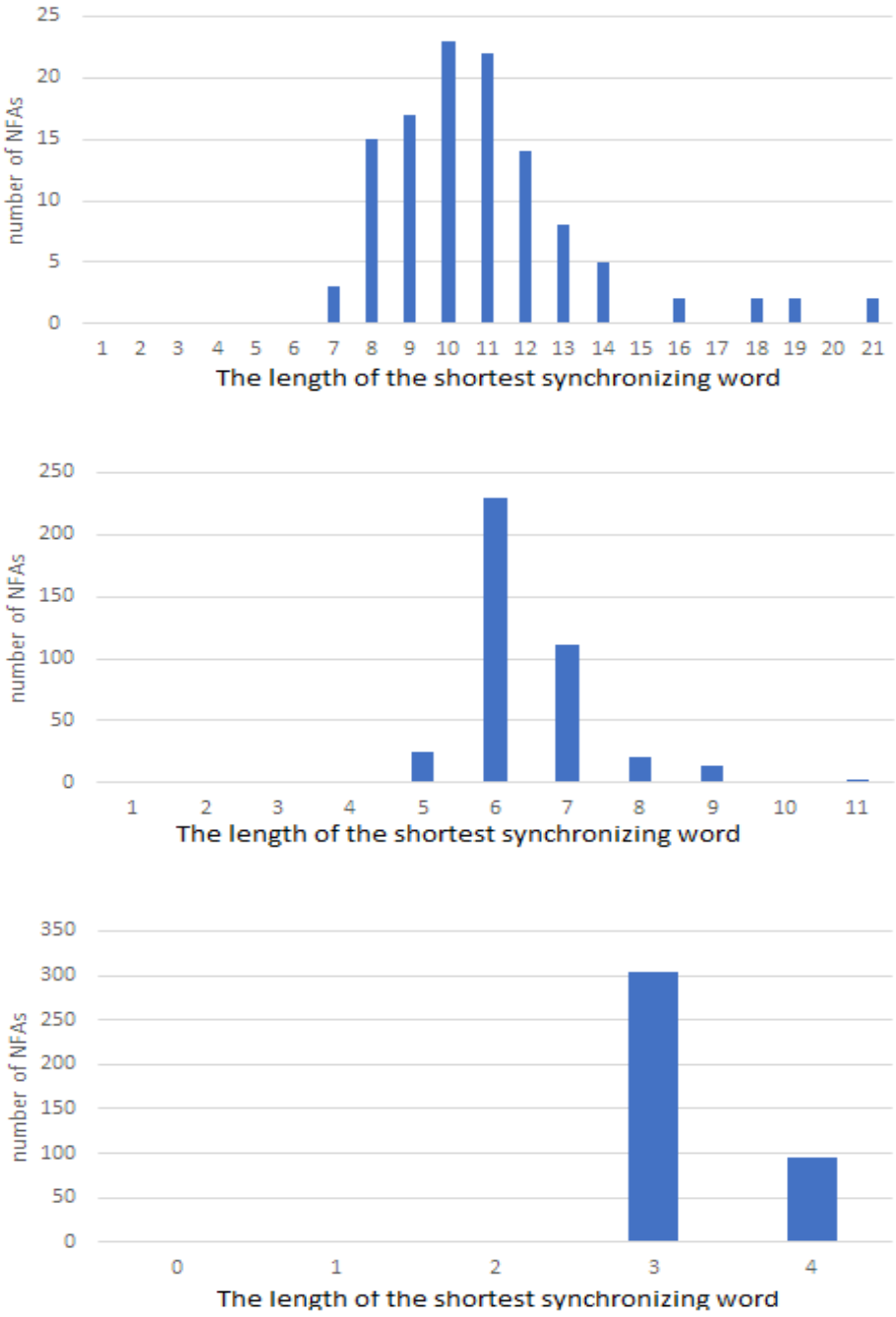

FIGURE 4. Distributions of 60-state NFAs generated under the Poisson models with λ=1 (top), λ=2 (middle), and $\lambda = 5$ (bottom) according to the length of their shortest D_3-synchronizing words

following solution:

$$E_1(n) \approx (0.57 + 0.66 \ln n)^2.$$

For $\lambda = 2$, the same procedure has led to the following approximation of the similarly defined quantity $E_2(n)$ calculated from our experimental data:

$$E_2(n) \approx (0.77 + 0.43 \ln n)^2.$$

Similar approximations have been obtained for other values of the parameter λ.

4. Conclusion and future work

We have presented an attempt to approach the problem of computing a D_3-synchronizing word of minimum length for a given NFA via the SAT-solver method. Even though our results constitute only the very first steps, we think that they do provide some evidence for this approach to be feasible in principle. Of course, more work is needed to improve the performance of our implementation and to enlarge its range.

We see several resources for improvements. First of all, we may try to modify the basic encoding described in Section 2. There are several options for such modifications that all look promising, but it is hard to predict a priori which one will prove to be the most efficient, and we have to go through several rounds of trial-and-error. As an example of a relatively successful trial, we briefly report one of the modifications that have already been implemented by the first author.

As mentioned in the description of our basic algorithm in Section 3, every D_3-synchronizing NFA $\mathscr{A}$ must have an everywhere defined input symbol. If all input symbols of $\mathscr{A}$ are everywhere defined, one can use the transformations described in [6, Lemma 8.3.8] or [9, Section 2] to convert $\mathscr{A}$ into a DFA $\mathscr{A}'$ such that $\mathscr{A}$ is D_3-synchronizing if and only if $\mathscr{A}'$ is synchronizing and the minimum length of D_3-synchronizing words for $\mathscr{A}$ is the same as the minimum length of synchronizing words for $\mathscr{A}'$. Since there are powerful methods to compute shortest synchronizing words for DFAs with up to 350 states (see, e.g., [25]), we can apply one of these methods to $\mathscr{A}'$. Hence, we can restrict ourselves to the case when one of the input symbols of $\mathscr{A}$ is not everywhere defined.

If we consider only NFAs with 2 input symbols, 0 and 1, say, we conclude that we may assume that 0 is everywhere defined while 1 is not. Every D_3-synchronizing word for such an NFA should start with the symbol 0. Therefore one can start our solitaire-like game Γ described in Section 2 from the position that arises after the first application of 0, and the basic encoding can be modified accordingly[4]. For an NFA with n states and m transitions, this preprocessing allows one to save n^2 variables and around $n^2 + 2m$ clauses in the resulting instance of SAT. Our experiments show that this modification indeed reduces the execution time of solving D3W-instances for NFAs with ≥ 20 states, and the average time decrease reaches 50% for NFAs with ≥ 50 states. Also, the modification has allowed us to solve D3W for NFAs with more than 100 states which size was out of reach with the basic encoding.

Of course, the efficiency of our approach depends not only on the way we encode the problem but also on software and hardware used in the implementation. Besides

[4]If we re-use the illustrative example in Figure 2, the new initial position for this example will be the one shown in bottom left.

optimizing our own code, we have plan to experiment with more advanced SAT-solvers, namely, with CryptoMiniSat [26] and lingeling [27]. Using more powerful computers constitutes other obvious direction for improvements. In particular, our approach is clearly amenable to parallelization since calculations needed for different automata are completely independent so that in principle, we can work in parallel with as many automata as many processors are available.

Our future work should include theoretical explanations for phenomena observed in our experiments (which may be a challenging task) as well as extending our study to automata with arbitrarily many input symbols and to other versions of NFA synchronization such as D_1- and D_2-synchronization mentioned in Section 1. We also plan to test our algorithm on certain non-random benchmarks, including some"slowly synchronizing" automata such as PFAs constructed in [28, 29].

Acknowledgements. The authors are grateful to the referee for his/her remarks and suggestions, in particular, for the proposal to compare the methods of the paper with the brute force approach.

References

[1] C. H. Papadimitriou, Computational Complexity, Addison-Wesley, 1994. MR1251285

[2] M. V. Volkov, *Synchronizing automata and the Černý conjecture*, in: Language and Automata Theory and Applications, Lect. Notes Comput. Sci., **5196** (2008), 11–27. MR2540309

[3] J. Kari, M. V. Volkov, *Černý's conjecture and the Road Coloring Problem*, Chapter 15 in: J.-É. Pin (ed.), Handbook of Automata Theory, Volume I, EMS Publishing House, in print.

[4] B. Imreh, M. Steinby, *Directable nondeterministic automata*, Acta Cybernetica, **14**:1 (1999), 105–115. MR1682017

[5] P. Martyugin, *A lower bound for the length of the shortest carefully synchronizing words*, Russ. Math., **54**:1 (2010), 46–54. MR2664485

[6] M. Ito, Algebraic Theory of Automata and Languages, World Scientific, 2004. MR2078698

[7] V. D. Blondel, R. M. Jungers, A. Olshevsky, *On primitivity of sets of matrices*, Automatica J. IFAC, **61** (2015), 80–88. MR3401692

[8] B. Gerencsér, V. V. Gusev, R. M. Jungers, *Primitive sets of nonnegative matrices and synchronizing automata*, SIAM J. Matrix Analysis and Applications, **39**:1 (2018), 83–98. MR3746507

[9] H. Don, H. Zantema, *Synchronizing non-deterministic finite automata*, J. Automata, Languages and Combinatorics, **23**:4 (2018), 307–328. DOI 10.25596/jalc-2018-307

[10] M. Steinby, *Directable fuzzy and nondeterministic automata*, Preprint, 2017. Available at `https://arxiv.org/abs/1709.07719`.

[11] S. Sandberg, *Homing and synchronizing sequences*, in: Model-Based Testing of Reactive Systems, Lect. Notes Comput. Sci., **3472** (2005), 5–33. DOI 10.1007/11498490_2

[12] J.-E. Pin, *On two combinatorial problems arising from automata theory*, Ann. Discrete Math., **17** (1983), 535–548. MR0841339

[13] I. K. Rystsov, *Asymptotic estimate of the length of a diagnostic word for a finite automaton*, Cybernetics, **16** (1980), 194–198; translation from Kibernetika, 1980:2 (1980), 31–35. MR0699331

[14] I. K. Rystsov, *Polynomial complete problems in automata theory*, Inf. Process. Lett., **16**:3 (1983), 147–151. MR0701758

[15] P. Martyugin, *Synchronization of automata with one undefined or ambiguous transition*, in: Implementation and Application of Automata, Lect. Notes Comput. Sci., **7381** (2012), 278–288. MR2993192

[16] C. P. Gomes, H. Kautz, A. Sabharwal, B. Selman, *Satisfiability Solvers*, Chapter 2 in: Handbook of Knowledge Representation, Elsevier, 2008, 89–134. Zbl 1183.68611

[17] J. Geldenhuys, B. van der Merwe, L. van Zijl, *Reducing nondeterministic finite automata with SAT solvers*, in: Finite-State Methods and Natural Language Processing, Lect. Notes Comput. Sci., **6062** (2009), 81–92. DOI 10.1007/978-3-642-14684-8_9

[18] T. Jiang, B. Ravikumar, *Minimal NFA problems are hard*, in: Automata, Languages and Programming, Lect. Notes Comput. Sci., **510** (1991), 629–640. MR1129941

[19] E. Skvortsov, E. Tipikin, *Experimental study of the shortest reset word of random automata*, in: Implementation and Application of Automata, Lect. Notes Comput. Sci., **6807** (2011), 290–298. MR2862922

[20] C. Güniçen, E. Erdem, H. Yenigün, *Generating shortest synchronizing sequences using Answer Set Programming*, in: Proceedings of Answer Set Programming and Other Computing Paradigms (ASPOCP 2013), 6th International Workshop (2013), 117–127. Available at `https://arxiv.org/abs/1312.6146`.

[21] J. Olschewski, M. Ummels, *The complexity of finding reset words in finite automata*, in: Mathematical Foundations of Computer Science, Lect. Notes Comput. Sci., **6281** (2010), 568–579. MR2727259

[22] N. Eén, N. Sörensson, *An extensible SAT-solver*, in: Theory and Applications of Satisfiability Testing (SAT 2003), Lect. Notes Comput. Sci., **2919** (2004), 502–518. Zbl 1204.68191

[23] N. Eén, N. Sörensson, The MiniSat Page. Available at `http://minisat.se`.

[24] D. Tabakov, M. Y. Vardi, *Experimental evaluation of classical automata constructions*, in: Logic for Programming, Artificial Intelligence, and Reasoning, Lect. Notes Comput. Sci., **3835** (2005), 396–411. Zbl 1143.68443

[25] A. Kisielewicz, J. Kowalski, M. Szykuła, *Computing the shortest reset words of synchronizing automata*, J. Comb. Optim. **29**:1 (2015), 88–124. MR3296258

[26] M. Soos, CryptoMiniSat 2. Available at `http://www.msoos.org/cryptominisat2/`.

[27] A. Biere, *Yet another Local Search Solver and Lingeling and Friends entering the SAT Competition 2014*, in: Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions, University of Helsinki, 2014, 39–40.

[28] M. de Bondt, H. Don, H. Zantema, *DFAs and PFAs with long shortest synchronizing word length*, in: Developments in Language Theory, Lect. Notes Comput. Sci., **10396** (2017), 122–133. MR3691074

[29] M. de Bondt, H. Don, H. Zantema, *Lower bounds for synchronizing word lengths in partial automata*, Preprint, 2018. Available at `https://arxiv.org/abs/1801.10436`.

Hanan Shabana
Institute of Natural Sciences and Mathematics,
Ural Federal University,
Lenina 51, 620000 Yekaterinburg, Russia
Faculty of Electronic Engineering, Menoufia University, Egypt
E-mail address: `hananshabana22@gmail.com`

Mikhail V. Volkov
Institute of Natural Sciences and Mathematics,
Ural Federal University
Lenina 51,
620000 Yekaterinburg, Russia
E-mail address: `Mikhail.Volkov@usu.ru`