URAL MATHEMATICAL JOURNAL, Vol. 4, No. 2, 2018, pp. 99–110

DOI: 10.15826/umj.2018.2.011

D_2-SYNCHRONIZATION IN NONDETERMINISTIC AUTOMATA[1]

Hanan Shabana

Institute of Natural Sciences and Mathematics,

Ural Federal University, 51 Lenin aven., Ekaterinburg, Russia, 620000

Faculty of Electronic Engineering, Menoufia University, Egypt

hananshabana22@gmail.com

Abstract: We approach the problem of computing a D_2-synchronizing word of minimum length for a given nondeterministic automaton via its encoding as an instance of SAT and invoking a SAT solver. In addition, we report some of the experimental results obtained when we had tested our method on randomly generated automata and certain benchmarks.

Keywords: Nondeterministic automata, Synchronizing word, SAT solver

Introduction

A *nondeterministic finite automaton* (NFA) is a triple $\mathscr{A} = (Q, \Sigma, \delta)$, where Q is a finite non-empty set of *states*, Σ is a finite non-empty set of input symbols, and δ is a map $Q \times \Sigma \to \mathcal{P}(Q)$, where $\mathcal{P}(Q)$ is the power set of Q. The map δ is called the *transition function* of $\mathscr{A}$; it describes the action of symbols in Σ at states in Q. As usual, we represent the NFA $\mathscr{A}$ by the labeled digraph with the vertex set Q, the label alphabet Σ, and the set of labeled edges

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

A *word* over Σ is a finite (maybe, empty) sequence of symbols from Σ. The set of all words over Σ including the empty word is denoted by 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 $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$, we extend the function δ 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 $s \in \Sigma$ 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.

The present note is a follow-up of the paper [12] by Volkov and the present author. We briefly recall the problem approached in [12] and, in parallel, introduce the problem that we tackle here. We are interested in *synchronization* of finite automata. The basic idea of *synchronization* is as follows: for a given automaton, we look for a sequence of input signals that allows us to predict the behaviour of the automaton after consuming these signals, 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*.

[1]Supported by the Competitiveness Enhancement Program of Ural Federal University.

The above informal idea of synchronization is fairly easy to formalize for deterministic automata but for NFAs it admits several non-equivalent formalizations. We are not going to survey all formalizations that appear in the literature and restrict ourselves to the two following versions of synchronization, both originating in [7].

Let $\mathscr{A} = (Q, \Sigma, \delta)$ be an NFA. A word $w \in \Sigma^*$ is said to be *D_3-synchronizing* if $\bigcap_{q \in Q} q.w \neq \varnothing$. In terms of the labeled digraph representing $\mathscr{A}$, this condition amounts to saying that for each $q \in Q$, there exists a directed path, whose consecutive labels form the word w, that starts at q and terminates at a certain common state, independent of q. Observe that this definition implies that the action of any D_3-synchronizing word must be defined at every state of $\mathscr{A}$. A NFA is called *D_3-synchronizing* if it admits a D_3-synchronizing word.

A word $w \in \Sigma^*$ is said to be *D_2-synchronizing* for $\mathscr{A} = (Q, \Sigma, \delta)$ if $q.w = q'.w$ for all $q, q' \in Q$. To understand the 'physical meaning' of this concept, imagine a big quantity of identical NFAs which get the same input sequence and work on it in parallel. If the sequence constitutes a D_2-synchronizing word, then after consuming the input, the NFAs will demonstrate identical (that is, synchronous) behaviour, even though originally they all might have been in different states that were unknown to us.

In contrast to the condition $\bigcap_{q \in Q} q.w \neq \varnothing$, the equality $q.w = q'.w$ does not imply that the action of w must be everywhere defined. However, the equality ensures that if a D_2-synchronizing word is undefined at some state, the word must be nowhere defined. Thus, a D_2-synchronizing word is either nowhere or everywhere defined; in the latter case, it is easy to see that the word is also D_3-synchronizing. (The converse is not true: a D_3-synchronizing word can fail to be D_2-synchronizing.) A NFA is called *D_2-synchronizing* if it has a D_2-synchronizing word.

We mention that both D_3- and D_2-synchronization get very transparent meanings 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 each input symbol $s \in \Sigma$ is encoded by a matrix $M(s)$ whose (q, q')-entry is 1 if $q' \in \delta(q, s)$ and 0 otherwise. It is not hard to check 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, and $\mathscr{A}$ is D_2-synchronizing if and only if in some product of the matrices $M(s)$, $s \in \Sigma$, every column consists either entirely of 0s or entirely of 1s.

The problems of determining whether or not a given NFA is either D_3- or D_2-synchronizing are known to be PSPACE-complete and there is no polynomial in n upper bound on the length of D_3- or D_2-synchronizing words for NFAs with n states. (These facts follow from results found by Rystsov in the early 1980s [10, 11] and later rediscovered (and strengthened) by Martyugin [9].) Thus, given an NFA, finding a D_3- or D_2-synchronizing word of minimum length for it is computationally hard. In [12] the author and Volkov have approached the problem of finding a D_3-synchronizing word of minimum length for a given NFA via the *SAT-solver method*. The method of treating 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. Modern SAT solvers are extremely powerful: they can solve instances with hundreds of thousands of variables and millions of clauses within a few minutes. Therefore the SAT-solver method has a very wide range of applications, see [5] for a survey. In particular, the method has been successfully invoked for studying synchronization in deterministic automata, see [6, 13]. Our results in [12] have demonstrated that the SAT-solver method can also be applied in the realm of NFAs. Here we extend the approach to the case of D_2-synchronization.

The paper is organized as follows. Sect. 1 describes the encoding reducing our problem to SAT. Sect. 2 presents the main algorithm, outlines the settings of our experiments and gives samples of our experimental results. Sect. 3 collects a few final remarks, including a discussion of the future work in the direction of the present paper.

1. Encoding to SAT

We start with a precise formulation of the problem which we are going to study here.

D2W (the existence of a D_2-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_2-synchronizing word of length ℓ; NO otherwise.

In [12] the present author and Volkov have considered the problem D3W that has exactly the same instances as D2W but asks whether or not $\mathscr{A}$ has a D_3-synchronizing word of length ℓ. For both D2W and D3W, the integer ℓ is assumed to be given in unary; as explained in [12, Sect. 2], with ℓ given in binary, it is not feasible to expect the existence of a polynomial reduction from D3W to SAT, and the very same argument applies to D2W.

It is fair to say that our encoding of D2W has been obtained as a modification of the encoding of D3W suggested in [12]. However, restricting here to the "new" part of the encoding only would make the present paper difficult to follow without looking at [12] at every single step of the way. Therefore, we have preferred to describe our encoding in a self-contained manner, even though this causes a few overlaps with [12].

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.) 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. We aim to construct a polynomial reduction of D2W to SAT. For this, we have to find two polynomials $v(x, y)$ and $c(x, y)$ (preferably of low degrees in x and y) with the following property: given an arbitrary instance $(\mathscr{A}, \ell)$ of D2W, where $\mathscr{A} = (Q, \Sigma, \delta)$ is an NFA with two input symbols, we are able to produce 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), while $|V| \le v(|Q|, \ell)$ and $|C| \le c(|Q|, \ell)$.

Throughout our encoding, we let $\Sigma := \{0, 1\}$ and $Q := \{q_0, \dots, q_{n-1}\}$. For a state $q \in Q$, we use the expressions $P_0(q)$ and $P_1(q)$ to denote the sets of all preimages of q under the actions of the input symbols 0 and 1 respectively; that is, if a is either of the two symbols, then

$$P_a(q) := \{p \in Q \mid q \in p.a\}.$$

First we define the set V of variables. We need two sorts of variables: *letter variables* and *token variables*.

The letter variables are $x_1, \dots, x_\ell$. The variable x_t, $1 \le t \le \ell$, plays the role of an indicator for the t-th symbols a_t in the input word $w := a_1 \cdots a_\ell \in \Sigma^\ell$: the value of x_t is 1 if and only if $a_t = 1$.

The token variables are y_{ij}^t where $i, j = 0, \dots, n-1$ and $t = 0, 1, 2, \dots, \ell$. To explain the role of these variables, we use a solitaire-like game Γ on the labeled digraph representing the NFA $\mathscr{A}$. In the initial position of Γ, each state $q_i \in Q$ holds exactly one token denoted $\mathbf{i}$. In the course of the game, tokens migrate and may multiply or disappear according to the previous position of the game and the action of the player. Namely, at each move an input symbol $a \in \Sigma$ is chosen. Then for each state $q \in Q$ such that $q.a \ne \varnothing$, 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 gives rise 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 $\mathbf{i}$ occurs at a state $p \in Q$ if and only if $p \in q.a$ for some state q that had held $\mathbf{i}$ just prior to the move. For an illustration, Fig. 1 (borrowed from [12]) demonstrates the initial case of 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).

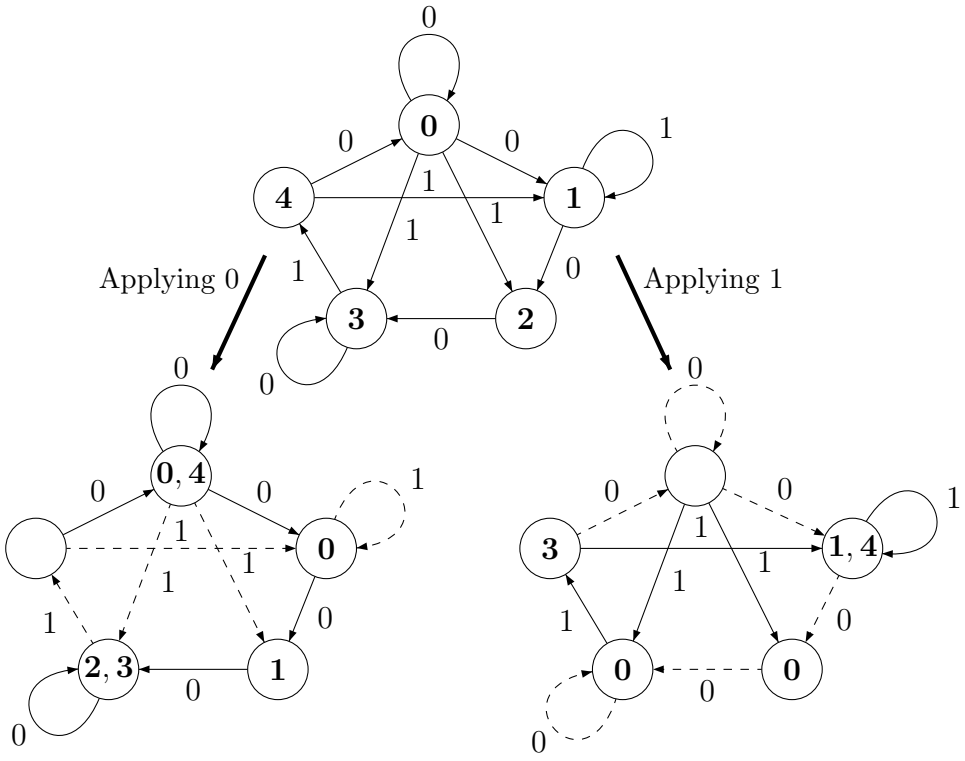

Figure 1. Redistribution of tokens after the first move

The intended meaning of the variables y_{ij}^t (which will be enforced by the condition we impose on them later) is as follows: $y_{ij}^t = 1$ should mean that after t rounds of the game Γ, one of the tokens held by the state q_j is $\mathbf{i}$.

Perhaps, it makes sense to add a matrix interpretation of the game Γ as the token variables get quite a clear meaning under this interpretation. The initial position of Γ can be thought of as the identity Boolean $Q \times Q$-matrix. At each move, an input symbol $a \in \Sigma$ is chosen and the matrix of the current position is right multiplied by the matrix $M(a)$. Then for each fixed t, the values of the variables y_{ij}^t are exactly the entries of the matrix corresponding to the position of Γ after t moves. For instance, the matrices that correspond to two possible positions of the game played on the 5-state NFA in Fig. 1 are

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Altogether, V consists of $n^2(\ell+1)+\ell$ variables so that we can take the polynomial $x^2(y+1)+y$ to play the role of $v(x,y)$ from the above definition of polynomial reduction. For the reduction from D3W to SAT in [12], an extra set of n variables (the so-called *synchronization variables*) was used. Here we have managed to slightly decrease the number of variables.

Now we describe the set C of clauses over V corresponding to the instance $(\mathscr{A}, \ell)$. As in [12], C is the disjoint union of set C_0 of *initial clauses*, the sets C_t, $t = 1, \ldots, \ell$, of *transition clauses*, and the set S of *synchronization clauses*. The clauses in $C_0, C_1, \ldots, C_\ell$ are constructed exactly as in [12] (but we will recall the construction for the reader's convenience) while the clauses in S are

essentially different as these are the clauses that reflect the essence of D_2-synchronization.

The clauses in C_0 describe the initial position of the game Γ. In this position, each state $q_i \in Q$ holds the token **i** and nothing else. Therefore C_0 consists of n^2 one-literal clauses, namely, the clauses $y^0_{00}, \dots, y^0_{n-1\,n-1}$ along with all clauses of the form $\neg y^0_{ij}$ with $i \neq j$.

In order to define C_t for $t = 1, \dots, \ell$, consider for all $i, j = 0, \dots, n-1$, the following formulas:

$$\Psi^t_{ij}: \quad y^t_{ij} \iff \Big(x_t \wedge \bigvee_{q_k \in P_1(q_j)} y^{t-1}_{ik}\Big) \vee \Big(\neg x_t \wedge \bigvee_{q_h \in P_0(q_j)} y^{t-1}_{ih}\Big).$$

The equivalence Ψ^t_{ij} is nothing but a direct translation of the above propagation rule for the tokens in the language of propositional logic. Indeed, it 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.

The following fact is a special instance of [12, Lemma 2]:

Lemma 1. *Every truth assignment $\varphi\colon \{x_1, \dots, x_\ell\} \to \{0,1\}$ on the letter variables has a unique extension $\overline{\varphi}$ to the token variables y^t_{ij} that makes all the clauses in C_0 and all the formulas Ψ^t_{ij} hold true $(i, j = 0, \dots, n-1,\ t = 1, \dots, \ell)$. The token variable y^t_{ij} gets value 1 under $\overline{\varphi}$ if and only if after the moves $\varphi(x_1), \dots, \varphi(x_t)$ of the game Γ, one of the tokens held by the state q_j is* **i**.

Now, for each $t = 1, \dots, \ell$, we let C_t be the set of all clauses of a suitable CNF (conjunctive normal form) equivalent to $\bigwedge_{1 \le i,j \le n} \Psi^t_{ij}$. Of course, there are many ways to convert the latter formula to an equivalent CNF, but in order to reuse a part of code written for [12], we retain for C_t the following set of clauses:

$$\neg y^t_{ij} \vee x_t \vee \bigvee_{q_h \in P_0(q_j)} y^{t-1}_{ih}, \qquad \neg y^t_{ij} \vee \neg x_t \vee \bigvee_{q_k \in P_1(q_j)} y^{t-1}_{ik}, \tag{1.1}$$

$$y^t_{ij} \vee \neg x_t \vee \neg y^{t-1}_{ik} \quad \text{for each } q_k \in P_1(q_j), \tag{1.2}$$

$$y^t_{ij} \vee x_t \vee \neg y^{t-1}_{ih} \quad \text{for each } q_h \in P_0(q_j). \tag{1.3}$$

Clauses of the form (1.1)–(1.3) simplify if one of the sets $P_0(q_j)$ or $P_1(q_j)$ is empty. In (1.1) the disjunctions over the empty sets are omitted so that if, say, $P_0(q_j) = \varnothing$, then the first clause in (1.1) reduces to $\neg y^t_{ij} \vee x_t$. As for (1.2) or (1.3), these clauses disappear whenever $P_1(q_j)$ or, respectively $P_0(q_j)$ are empty. Thus, if the state q_j is such that $P_0(q_j) = P_1(q_j) = \varnothing$, then both (1.2) and (1.3) vanish and the two clauses in (1.1) reduce to $\neg y^t_{ij} \vee x_t$ and $\neg y^t_{ij} \vee \neg x_t$. The latter pair of clauses is clearly equivalent to just $\neg y^t_{ij}$ whence all clauses (1.1)–(1.3) reduce to $\neg y^t_{ij}$ for this particular j and for all $i = 0, \dots, n-1$ and $t = 1, \dots, \ell$. This fact amounts to expressing the following simple idea: if the state q_j has no incoming edges, then no token can arrive at q_j after any move of the game Γ.

Let m stand for 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, $m \le 2n^2$. For each fixed i, the number $\sum_{j=1}^{n} (|P_1(q_j)| + |P_0(q_j)|)$ of clauses of the forms (1.2) and (1.3) is equal to m, whence the total number of such "short" clauses is mn. As for "long" clauses in (1.1), there are at most two such clauses for each fixed pair (i, j), whence their total number does not exceed $2n^2$. Altogether, $|C_t| \le n(m + 2n) \le 2n^2(n+1)$ for each $t = 1, \dots, \ell$.

While clauses in $\bigcup_{t=0}^{\ell} C_t$ coincide with those used in [12], the sets of synchronization clauses in [12] and here are different. The present set S contains n^2 disjunctions of the following form:

$$\neg y_{ij}^{\ell} \vee y_{i+1 \,(\mathrm{mod}\ n)\, j}^{\ell}, \quad i,j = 0,\dots,n-1. \tag{1.4}$$

Clearly, for each fixed j, the clauses (1.4) are equivalent to the cycle of implications

$$y_{0j}^{\ell} \to y_{1j}^{\ell},\ y_{1j}^{\ell} \to y_{2j}^{\ell}, \dots, y_{n-1\,j}^{\ell} \to y_{0j}^{\ell}$$

that expresses the idea of D_2-synchronization as follows: if the state q_j holds some token after ℓ moves, then q_j must hold all n tokens $\mathbf{0}, \mathbf{1}, \dots, \mathbf{n-1}$. Observe that the clauses (1.4) are satisfied if all variables y_{ij}^{ℓ} get value 0; by Lemma 1 this happens exactly when all tokens disappear after ℓ moves which means that the word $w \in \Sigma^{\ell}$ corresponding to the chosen sequence of moves is nowhere defined. In this paper we are interested in finding only those D_2-synchronizing words that are somewhere defined; we refer to them as *proper* D_2-synchronizing words. Therefore, we add to the set S the following clause:

$$\bigvee_{0 \le j \le n-1} y_{0j}^{\ell}. \tag{1.5}$$

The clause (1.5) is satisfied if and only if some state holds the token $\mathbf{0}$ after ℓ moves; in the presence of (1.4), the latter fact is equivalent to the claim that some state holds *some* token after ℓ moves, which in turn means that the word w is somewhere defined.

The whole set $C = S \cup \bigcup_{t=0}^{\ell} C_t$ consists of at most $2n^2((n+1)\ell + 1) + 1$ clauses. Thus, the polynomial $2x^2((x+1)y+1)+1$ can be taken as $c(x,y)$ from the definition of polynomial reduction. Summarizing the above discussion, we arrive at the following result parallel to [12, Theorem 3].

Theorem 1. *An NFA $\mathscr{A}$ has a proper D_2-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, a word $w = a_1 \cdots a_\ell$ with $a_1, \dots, a_\ell \in \{0,1\}$ is proper D_2-synchronizing for $\mathscr{A}$ if and only if the map $x_t \mapsto a_t$, $t = 1,\dots,\ell$, extends to a satisfying assignment for (V,C).*

2. Experimental results

The general scheme of our experiments follows [12] mutatis mutandis. We outline our basic procedure, commenting on similarities with and differences from the procedure implemented in [12].

1. A positive integer n (the number of states) is fixed. As in [12], we have considered $n \le 100$.

2. A random NFA $\mathscr{A}$ with n states and 2 input symbols is generated. We have used the same two models of random generation that were used in [12] but we provide details below for the reader's convenience. As in [12], we disregard NFAs that have no everywhere defined input symbol because such NFAs possess neither D_3-synchronizing nor proper D_2-synchronizing words.

3. A positive integer ℓ_0 (the hypothetical length of the shortest D_2-synchronizing word for $\mathscr{A}$) is chosen. Taking into account the fact that proper D_2-synchronization is more restrictive than D_3-synchronization, we have used slightly larger values of ℓ_0 than in [12]. We introduce three integer variables $\ell_{\min}$, ℓ, and $\ell_{\max}$ and initialize them as follows: $\ell_{\min} := 1$, $\ell := \ell_0$, $\ell_{\max} := 2\ell_0$.

4. The pair $(\mathscr{A}, 1)$ is encoded into a SAT instance (V', C') as described in Sect. 1.

5. The instance (V', C') is scaled to the instance (V, C) that encodes the pair $(\mathscr{A}, \ell)$, see Remark 1 below.

6. The SAT solver MiniSat 2.2.0. is invoked to solve the SAT instance (V, C). We refer to [3] for a description of the underlying ideas of MiniSat and to [4] for a discussion and the source code of the solver.

7. The binary search on ℓ is performed. If MiniSat returns YES on the instance (V, C), we check whether or not $\ell = \ell_{\min}$. If $\ell = \ell_{\min}$, then ℓ is the minimum length of proper D_2-synchronizing words for $\mathscr{A}$, and we pass to Step 2 to generate another NFA. If $\ell > \ell_{\min}$, we keep the value of $\ell_{\min}$, update $\ell_{\max}$ and ℓ by letting

$$\ell_{\max} := \ell, \quad \ell := \left\lfloor \frac{\ell_{\min} + \ell_{\max}}{2} \right\rfloor,$$

and pass to Step 5.

If the MiniSat returns NO on the instance (V, C), 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 properly D_2-synchronizing[2] and go to Step 2 to generate another NFA. If $\ell < \ell_{\max}$, we keep the value of $\ell_{\max}$, update $\ell_{\min}$ and ℓ by letting

$$\ell_{\min} := \ell + 1, \quad \ell := \left\lceil \frac{\ell_{\min} + \ell_{\max}}{2} \right\rceil,$$

and pass to Step 5.

Remark 1. In the course of the binary search outlined above, we have to consider instances of D2W with the same NFA $\mathscr{A}$ but different values of ℓ. An important feature of the encoding presented in Sect. 1 is that as soon as we have constructed the "primary" SAT instance (V', C') that encodes the D2W instance $(\mathscr{A}, 1)$, we are in a position to scale (V', C') to the SAT instance encoding the D2W instance $(\mathscr{A}, \ell)$ for any value of ℓ. In order to explain this feature, recall that MiniSAT accepts its input in the following text format (so-called simplified DIMACS CNF format). Every line beginning `c` is a comment. The first non-comment line is of the form:

`p cnf NUMBER_OF_VARIABLES NUMBER_OF_CLAUSES`

Variables are represented by integers from 1 to `NUMBER_OF_VARIABLES`. The first non-comment line is followed by `NUMBER_OF_CLAUSES` non-comment lines each of which defines a clause. Every such line starts with a space-separated list of different non-zero integers corresponding to the literals of the clause: a positive integer corresponds to a literal which is a variable, and a negative integer corresponds to a literal which is the negation of a variable; the line ends in a space and the number 0.

Given an NFA $\mathscr{A}$ with n states, we write the SAT instance (V', C'), which corresponds to $(\mathscr{A}, 1)$, in DIMACS CNF format, representing the variables y_{ij}^0, y_{ij}^1, and x_1 by the numbers, respectively, $in + j + 1$, $n^2 + in + j + 2$, and $n^2 + 1$. Consider, for a simple illustration, the NFA $\mathscr{E}_2$ shown in Fig. 2.

Table 1 in the next page presents our encoding of the D2W instance $(\mathscr{E}_2, 1)$ as a SAT instance. In the left column the SAT instance is shown as a list of clauses while the right column shows it in DIMACS CNF format.

[2]Of course, the equality $\ell = \ell_{\max}$ only means that $\mathscr{A}$ has no proper D_2-synchronizing word of length $\leq 2\ell_0$, and it is not excluded, in principle, that the NFA is properly D_2-synchronizing but its shortest proper D_2-synchronizing word is very long. However, by choosing appropriate values of the parameter ℓ_0, we have drastically minimized the number of the "bad" cases when the SAT solver returns NO and $\ell = \ell_{\max}$ so that we have been able to analyze each of them individually.

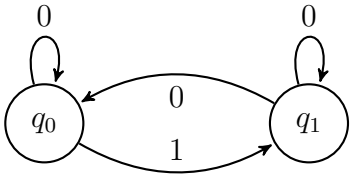

Figure 2. The NFA $\mathscr{E}_2$

Clauses		**DIMACS CNF lines**
		`p cnf 9 25`
C'_0	y^0_{00}	`1 0`
	$\neg y^0_{01}$	`-2 0`
	$\neg y^0_{10}$	`-3 0`
	y^0_{11}	`4 0`
C'_1	$\neg y^1_{00} \vee x_1 \vee y^0_{00} \vee y^0_{01}$	`-6 5 1 2 0`
	$\neg y^1_{00} \vee \neg x_1$	`-6 -5 0`
	$y^1_{00} \vee x_1 \vee \neg y^0_{00}$	`6 5 -1 0`
	$y^1_{00} \vee x_1 \vee \neg y^0_{01}$	`6 5 -2 0`
	$\neg y^1_{01} \vee x_1 \vee y^0_{01}$	`-7 5 2 0`
	$\neg y^1_{01} \vee \neg x_1 \vee y^0_{00}$	`-7 -5 1 0`
	$y^1_{01} \vee \neg x_1 \neg y^0_{00}$	`7 -5 -1 0`
	$y^1_{01} \vee x_1 \vee \neg y^0_{01}$	`7 5 -2 0`
	$\neg y^1_{10} \vee x_1 \vee y^0_{10} \vee y^0_{11}$	`-8 5 3 4 0`
	$\neg y^1_{10} \vee \neg x_1$	`-8 -5 0`
	$y^1_{10} \vee x_1 \vee \neg y^0_{10}$	`8 5 -3 0`
	$y^1_{10} \vee x_1 \vee \neg y^0_{11}$	`8 5 -4 0`
	$\neg y^1_{11} \vee x_1 \vee y^0_{11}$	`-9 5 4 0`
	$\neg y^1_{11} \vee \neg x_1 \vee y^0_{10}$	`-9 -5 3 0`
	$y^1_{11} \vee \neg x_1 \neg y^0_{10}$	`9 -5 -3 0`
	$y^1_{11} \vee x_1 \vee \neg y^0_{11}$	`9 5 -4 0`
S'	$\neg y^1_{00} \vee y^1_{01}$	`-6 7 0`
	$\neg y^1_{01} \vee y^1_{00}$	`-7 6 0`
	$\neg y^1_{10} \vee y^1_{11}$	`-8 9 0`
	$\neg y^1_{11} \vee y^1_{10}$	`-9 8 0`
	$y^1_{00} \vee y^1_{01}$	`6 7 0`

Table 1. The SAT encoding of the D2W instance $(\mathscr{E}_2, 1)$

Now, in order to scale (V', C') to the SAT instance (V, C) that encodes the pair $(\mathscr{A}, \ell)$ for some given $\ell > 1$, we perform the following transformations on the DIMACS CNF representation of $C' = C'_0 \cup C'_1 \cup S'$:

1. In the first non-comment line, replace `NUMBER_OF_VARIABLES` and `NUMBER_OF_CLAUSES` by

$n^2(\ell+1)+\ell$ and respectively $\ell N + 2n^2 + 1$, where N is the number of clauses in C'_1.

2. Keep the lines corresponding to the clauses in C'_0 and C'_1.

3. For each $t = 2, \ldots, \ell$, add all the lines obtained from the ones corresponding to the clauses C'_1 by keeping the sign of every non-zero integer and adding $(t-1)n^2 + t - 1$ to its absolute value.

4. In each line corresponding to a clause in S', substitute every nonzero integer $\pm k$ by the integer $\pm(k + (\ell-1)n^2 + \ell - 1)$.

Our experiments have been performed on a personal computer equipped with an Intel(R) Core(TM) i5-2520M processor with 2.5 GHz CPU and 4GB of RAM. We have implemented the described algorithm in C++ and compiled with GCC 4.9.2. For various fixed $n \leq 100$, up to 1000 NFAs with n states have been generated and analyzed. We have generated 1000 automata for each $n \in \{5, 10, \ldots, 30\}$, 700 automata for each $n \in \{35, 40, \ldots, 60\}$, 500 automata for each $n \in \{65, 70, \ldots, 80\}$, and 200 automata for each $n \in \{90, 100\}$. The calculations have taken ≈ 400 seconds for $n = 10$ and $\approx 1.2 \cdot 10^5$ seconds for $n = 80$.

As in [12], the two models used for random generation of an NFA $\mathscr{A} = (Q, \Sigma, \delta)$ with n states and 2 input symbols were 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 $1/(n+1)$ while in the Poisson model with parameter λ, each $k < n$ is chosen with probability $e^{-\lambda}\lambda^k/k!$ and n is chosen with probability $1 - e^{-\lambda}\sum_{k=0}^{n-1}\lambda^k/k!$. With k having been chosen, we proceeded the same in both models, by choosing $\delta(q, s)$ from all $\binom{n}{k}$ subsets of Q with cardinality k uniformly at random.

For NFAs generated under the uniform model, we have observed that for an overwhelming majority of properly D_2-synchronizing NFAs, the length of the shortest proper D_2-synchronizing word is 3, and this conclusion does not depend on the number of states within the range of our experiments. Recall that the experiments in [12] revealed quite a similar phenomenon for D_3-synchronization: if a NFA generated under the uniform model is D_3-synchronizing (which happens with the probability $\approx 60\%$, see [12, Proposition 5]), then its shortest D_3-synchronizing word has length 2, and this fact does not depend on the number of states. An informal explanation of the latter phenomenon can be found in [12]; similar arguments apply also in the present situation.

Thus, the uniform model fails to produce any "slowly synchronizing" NFA. This indicates that using SAT-solvers in the uniform setting was not really necessary since a brute-force approach would suffice. Indeed, given an NFA $\mathscr{A} = (Q, \Sigma, \delta)$, one can write all words over Σ up to a given length in the short-lex order and apply each of these words to $\mathscr{A}$ until one finds a D_2-synchronizing word. As our experiments reveal, for a majority of NFAs generated under the uniform model, the brute-force approach requires to check only words up to length 3.

For random NFAs generated under the Poisson model, our experiments show that if the parameter λ is fixed, the length of the shortest proper D_2-synchronizing word grows with the number of states but the growth rate is rather small. Some sample experimental results are presented in Fig. 3. The three graphs in Fig. 3 correspond to NFAs with 30, 45, and 60 states generated under the Poisson models with $\lambda = 2$ and demonstrate how these NFAs are distributed according to the length of their shortest proper D_2-synchronizing words. The horizontal axis is the minimum length of proper D_2-synchronizing words and the vertical axis is the number of NFAs. We have applied the method of least squares to our experimental data, searching for an explicit function of n that approximates the mean value $E_\lambda(n)$ of the minimum lengths of proper D_2-synchronizing words for n-state NFAs generated under the Poisson model with a given parameter λ. The best

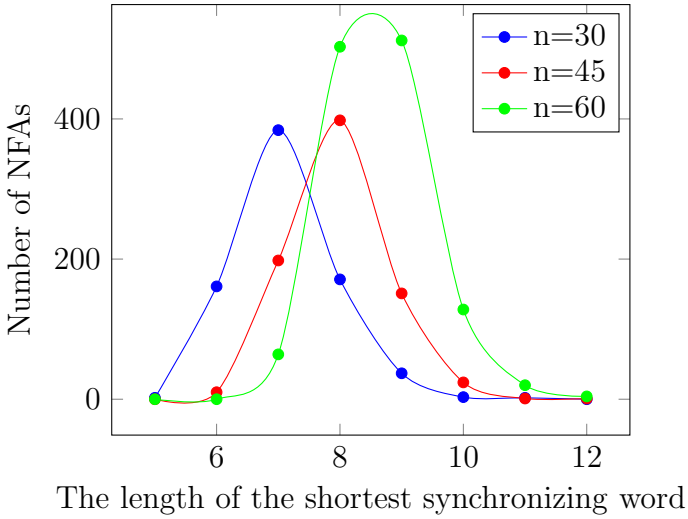

Figure 3. Distributions of random NFAs with 30, 45, and 60 states generated under the Poisson model with $\lambda = 2$ according to the minimum lengths of their proper D_2-synchronizing words

approximations have been provided by logarithmic functions; for instance, for $\lambda = 2$, we have found the following solution:

$$E_2(n) \approx -0.39 + 2.2 \ln(n).$$

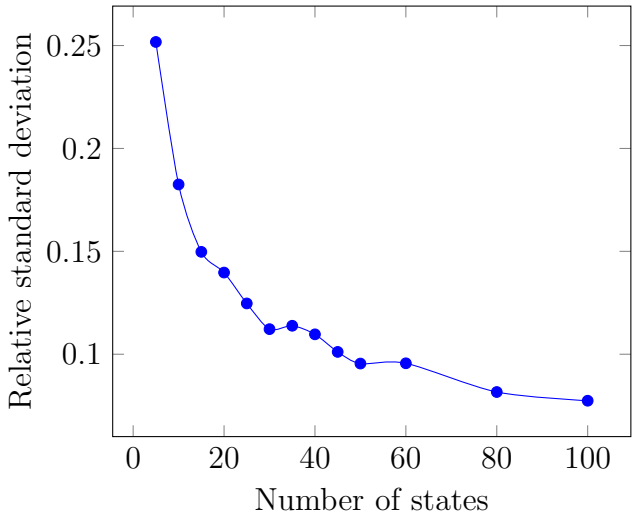

Figure 4. The relative standard deviation of the minimum lengths of proper D_2-synchronizing words for n-state NFAs as a function of n

Fig. 4 shows the relation between the relative standard deviation of our datasets and the number of states (for $\lambda = 2$).

Besides experimenting with randomly generated NFAs, we have tested our approach on certain provably "slowly synchronizing" NFAs considered in the literature. Here we report a set of results in which we used as benchmarks several automata from the series $\mathscr{P}_n$ suggested by de Bondt, Don, and Zantema [2]. The state set of $\mathscr{P}_n$ is $\{1, 2, \ldots, n\}$, $n \geq 3$, and the input alphabet consists of

two letters a and b whose actions are defined as follows:

$$q.a := \begin{cases} q+1 & \text{if } q = 1, 2, \\ q & \text{if } q = 3, \ldots, n; \end{cases} \qquad q.b := \begin{cases} \text{undefined} & \text{if } q = 1, \\ q+1 & \text{if } q = 2, \ldots, n-1, \\ 1 & \text{if } q = n. \end{cases}$$

Thus, the automata $\mathscr{P}_n$ are partial deterministic, and it is easy to see that for partial deterministic automata, proper D_2-synchronizing words coincide with D_3-synchronizing words and coincide with so-called carefully synchronizing words considered in [2]. Hence we can compare the information about the length of shortest synchronizing words for $\mathscr{P}_n$ obtained in [2, Theorem 3] and the results produced by an application of our procedure. In our experiments, we have examined all automata $\mathscr{P}_n$ with $n = 4, 5, \ldots, 11$, and for each of them, our result has matched the theoretical value predicted by [2, Theorem 3]. The time consumed ranges from 0.301 sec for $n = 4$ to 4303 sec for $n = 11$. Observe that in the latter case the shortest synchronizing word has length 116; clearly, this value is out of reach for any brute-force method.

3. Conclusion and future work

We have presented a modification of the approach originated in [12] that has allowed us to find shortest proper D_2-synchronizing words for nondeterministic automata with two input letters and up to 100 states. The size of automata that we are able to analyze may seem modest in comparison with the results of [8] whose authors describe sophisticated methods to compute shortest synchronizing words for *deterministic* automata with up to 350 states. However, two important nuances should be taken into account. First, for the time being, the approach of [12] and the present paper appears to be the only one that has proved to work in the realm of nondeterministic automata. Second, it is well known that nondeterministic automata may be exponentially more succinct than equivalent deterministic ones, and, say, an NFA with 100 states may encode the same amount of information as a DFA with 2^{100} states.

We have concentrated on D_2-synchronizing words which are everywhere defined. In fact, shortest nowhere defined words are even easier to be find with a similar method. The point is that in terms of our game Γ from Sect. 1, a nowhere defined word is just a word which application removes all tokens. However, if all tokens are going to be eventually removed, there is no need to distinguish between them! Therefore one can drastically reduce the number of variables and clauses used in the encoding. Instead of the 3-parameter set of variables $\{y_{ij}^t\}$ used in Sect. 1, it suffices to consider the 2-parameter set $\{y_j^t\}$ where $y_j^t = 1$ should mean that after t rounds of the game Γ, the state q_j holds a token; similarly, the role of the 3-parameter set of formulas $\{\Psi_{ij}^t\}$ can be played by the 2-parameter set consisting of the formulas

$$y_j^t \iff \Big(x_t \wedge \bigvee_{q_k \in P_1(q_j)} y_k^{t-1}\Big) \vee \Big(\neg x_t \wedge \bigvee_{q_h \in P_0(q_j)} y_h^{t-1}\Big)$$

for all $j = 0, \ldots, n-1$ and $t = 1, \ldots, \ell$. Similar simplifications apply to the sets of initial and synchronization clauses. Therefore, it made no sense to search for nowhere and everywhere defined D_2-synchronizing words simultaneously, although it was possible (for this, one just had to remove the clause (1.5) from the set of synchronization clauses).

Yet another version of synchronization for nondeterministic automata suggested in [7] is D_1-synchronization. A word $w \in \Sigma^*$ is said to be D_1-*synchronizing* for $\mathscr{A} = (Q, \Sigma, \delta)$ if $q.w = q'.w$ and $|q.w| = 1$ for all $q, q' \in Q$. Clearly, every D_1-synchronizing word is everywhere defined and is D_2-synchronizing but the converse is not true: an everywhere defined D_2-synchronizing word need not be D_1-synchronizing. We can use encodings similar to those in [12] and the present paper in

order to find shortest D_1-synchronizing words for NFAs of reasonable sizes; one only has to adjust the set of synchronization clauses.

We think that the results presented here and in [12] demonstrate that our approach works in principle but, of course, its present implementation is only a toy prototype for a system that could be used for real-world applications. There are several resources, on both software and hardware sides, which can be employed to speed up our calculations and enlarge their range. In particular, one can try more advanced SAT-solvers, such as CryptoMiniSat [14] and lingeling [1], and run a version of our program on a multiprocessor grid.

Acknowledgments. The author thanks the anonymous referees for their constructive comments and recommendations.

REFERENCES

1. Biere A. 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. P. 39–40. URL: http://fmv.jku.at/papers/Biere-SAT-Competition-2014.pdf
2. de Bondt M., Don H., Zantema H. *Lower bounds for synchronizing word lengths in partial automata.* Preprint, 2018. URL: https://arxiv.org/abs/1801.10436
3. Eén N., Sörensson N. An extensible SAT-solver. *Lect. Notes Comput. Sci.*, Vol. 2919: Theory and Applications of Satisfiability Testing (SAT 2003). 2004. P. 502–518. DOI: 10.1007/978-3-540-24605-3_37
4. Eén N., Sörensson N. *The MiniSat Page*. URL: http://minisat.se
5. Gomes C. P., Kautz H., Sabharwal A., Selman B. Satisfiability Solvers. Ch. 2. In: *Handbook of Knowledge Representation*, Elsevier, 2008. P. 89–134. DOI: 10.1016/S1574-6526(07)03002-7
6. Güniçen C., Erdem E., Yenigün H. Generating shortest synchronizing sequences using Answer Set Programming. In: *Proceedings of Answer Set Programming and Other Computing Paradigms (ASPOCP 2013)*. P. 117–127. URL: https://arxiv.org/abs/1312.6146.
7. Imreh B., Steinby M. Directable nondeterministic automata. *Acta Cybernetica*. 1999. Vol. 14, no. 1. P. 105–115.
8. Kisielewicz A., Kowalski J., Szykuła M. Computing the shortest reset words of synchronizing automata. *J. Comb. Optim.* 2015. Vol. 29, no. 1. P. 88–124. DOI: 10.1007/s10878-013-9682-0
9. Martyugin P. Synchronization of automata with one undefined or ambiguous transition. *Lect. Notes Comput. Sci.*, Vol. 7381: Implementation and Application of Automata (CIAA 2012). 2012. P. 278–288. DOI: 10.1007/978-3-642-31606-7_24
10. Rystsov I. K. Polynomial complete problems in automata theory. *Inf. Process. Lett.* 1983. Vol. 16, no. 3. P. 147–151. DOI: 10.1016/0020-0190(83)90067-4
11. Rystsov I. K. Asymptotic estimate of the length of a diagnostic word for a finite automaton. *Cybernetics*. 1980. Vol. 16, no. 1. P. 194–198. DOI: 10.1007/bf01069104
12. Shabana H., Volkov M. V. Using Sat solvers for synchronization issues in nondeterministic automata. *Siberian Electronic Math. Reports*. 2018. Vol. 15. P. 1426–1442. URL: http://semr.math.nsc.ru/v15/p1426-1442.pdf
13. Skvortsov E., Tipikin E. Experimental study of the shortest reset word of random automata. *Lect. Notes Comput. Sci.*, Vol. 6807: Implementation and Application of Automata (CIAA 2011), 2011. P. 290–298. DOI: 10.1007/978-3-642-22256-6_27
14. Soos M. CryptoMiniSat 2. URL: http://www.msoos.org/cryptominisat2/