Using Sat solvers for synchronization issues in partial deterministic automata*

Hanan Shabana and Mikhail V. Volkov

Institute of Natural Sciences and Mathematics
Ural Federal University, Lenina 51, 620000 Ekaterinburg, Russia
hananshabana22@gmail.com, m.v.volkov@urfu.ru

Abstract. We approach the task of computing a carefully synchronizing word of minimum length for a given partial deterministic automaton, encoding the problem as an instance of SAT and invoking a SAT solver. Our experimental results demonstrate that this approach gives satisfactory results for automata with up to 100 states even if very modest computational resources are used.

Keywords: Nondeterministic automaton, deterministic automaton, partial deterministic automaton, careful synchronization, exact synchronization, carefully synchronizing word, SAT, SAT-solver

1 Introduction

A *nondeterministic finite automaton* (NFA) is a triple $\langle Q, \Sigma, \delta\rangle$, where Q and Σ are finite non-empty sets called the *state set* and the *input alphabet* respectively, and δ is a subset of $Q \times \Sigma \times Q$. The elements of Q and Σ are called *states* and *letters*, respectively, and δ is referred to as the *transition relation*[1]. For each pair $(q, a) \in Q \times \Sigma$, we denote by $\delta(q, a)$ the subset $\{q' \mid (q, a, q') \in \delta\}$ of Q; this way δ can be viewed as a function $Q \times \Sigma \to \mathcal{P}(Q)$, where $\mathcal{P}(Q)$ is the power set of Q. When we treat δ as a function, we refer to it as the *transition function*.

Let Σ^* stand for the collection of all finite words over the alphabet Σ, including the empty word ε. The transition function extends to a function $\mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$, still denoted δ, in the following inductive way: for every subset $S \subseteq Q$ and every word $w \in \Sigma^*$, we set

$$\delta(S, w) := \begin{cases} S & \text{if } w = \varepsilon, \\ \bigcup_{q \in \delta(S,v)} \delta(q, a) & \text{if } w = va \text{ with } v \in \Sigma^* \text{ and } a \in \Sigma. \end{cases}$$

* Supported by the Ministry of Science and Higher Education of the Russian Federation, projects no. 1.580.2016 and 1.3253.2017, and the Competitiveness Enhancement Program of Ural Federal University.

[1] The conventional concept of an NFA includes distinguishing two non-empty subsets of Q consisting of *initial* and *final* states. As these play no role in our considerations, the above simplified definition well suffices for the purpose of this paper.

(Here the set $\delta(S,v)$ is defined by the induction assumption since v is shorter than w.) We say that a word $w \in \Sigma^*$ is *undefined at a state* $q \in Q$ if the set $\delta(q,w)$ is empty; otherwise w is said to be *defined at* q.

When we deal with a fixed NFA, we suppress the sign of the transition relation, introducing the NFA as the pair $\langle Q, \Sigma\rangle$ rather than the triple $\langle Q, \Sigma, \delta\rangle$ and writing $q.w$ for $\delta(q,w)$ and $S.w$ for $\delta(S,w)$.

A *partial* (respectively, *complete*) *deterministic* automaton is an NFA $\langle Q, \Sigma\rangle$ such that $|q.a| \leq 1$ (respectively, $|q.a| = 1$) for all $(q,a) \in Q \times \Sigma$. We use the acronyms PFA and CFA for the expressions 'partial deterministic automaton' and 'complete deterministic automaton', respectively.

A CFA $\mathscr{A} = \langle Q, \Sigma\rangle$ is called *synchronizing* if there exists a word $w \in \Sigma^*$ whose action leaves the automaton in one particular state no matter at which state in Q it is applied: $q.w = q'.w$ for all $q, q' \in Q$. Any w with this property is said to be a *synchronizing* word for the automaton.

Synchronizing automata serve as simple yet adequate models of error-resistant systems in many applied areas (system and protocol testing, information coding, robotics). At the same time, synchronizing automata surprisingly arise in some parts of pure mathematics and theoretical computer science (symbolic dynamics, theory of substitution systems, formal language theory). We refer to the survey [37] and the chapter [18] of the forthcoming 'Handbook of Automata Theory' for a discussion of synchronizing automata as well as their diverse connections and applications. From both applied and theoretical viewpoints, the key question is to find the optimal, i.e., shortest reset word for a given synchronizing automaton. Under standard assumptions of complexity theory, this optimization question is known to be computationally hard; see [18, Section 2] for a summary of various hardness results in the area. As it is quite common for hard problems of applied importance, there have been many attempts to develop practical approaches to the question. These approaches have been based on certain heuristics [15,1,16] and/or popular techniques, including (but not limiting to) binary decision diagrams [27], genetic and evolutionary algorithms [30,17], satisfiability solvers [36], answer set programming [10], hierarchical classifiers [28], and machine learning [29].

The present authors [35,34] have initiated an extension to the realm of NFAs of the approach of [36]. Here we consider a more restricted class, namely, that of PFAs, where studying synchronization issues appears to be much better motivated. While we follow the general strategy of and reuse some technical tricks from [35,34], our present constructions heavily depend on the specifics of partial automata and have not been obtained via specializing the constructions of those earlier papers.

The rest of the paper is structured as follows. In Sect. 2 we describe and motivate the version of PFA synchronization that we have studied. In Sect. 3 we first outline the approach based on satisfiability solvers and then explain in detail how we encode PFAs and their synchronization problems as instances of the Boolean satisfiability problem. In Sect. 4 we provide samples of our experimental results and conclude in Sect. 5 with a brief discussion of the future work.

We have tried to make the paper, to a reasonable extent, self-contained, except for a few discussions that involve some basic concepts of computational complexity theory. These concepts can be found, e.g., in the early chapters of the textbook [26].

2 Synchonization of NFAs and PFAs

The concept of synchronization of CFAs as defined in Sect. 1 was extended to NFAs in several non-equivalent ways. The following three nowadays popular versions were suggested and analyzed in [11] in 1999 (although, in an implicit form, some of them appeared in the literature much earlier, see, e.g., [4,9]). For $i \in \{1, 2, 3\}$, an NFA $\mathscr{A} = \langle Q, \Sigma \rangle$ is called *D_i-synchronizing* if there exists a word $w \in \Sigma^*$ that satisfies the condition (D_i) from the list below:

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

Any word satisfying (D_i) is called *D_i-synchronizing* for $\mathscr{A}$. The definition readily yields the following properties of D_i-synchronizing words:

Lemma 1 *a) A D_1- or D_3-synchronizing word is defined at each state.*

b) A D_2-synchronizing word is either defined at each state or undefined at each state.

c) Every D_1-synchronizing word is both D_2- and D_3-synchronizing; every D_2-synchronizing word defined at each state is D_3-synchronizing.

In [35] we adapted the approach based on satisfiability solvers to finding D_3-synchronizing words of minimum length for NFAs. The first-named author used a similar method in the cases of D_1- and D_2-synchronization; results related to D_2-synchronization were reported in [34].

Yet another version of synchronization for NFAs was introduced in [13] and systematically studied in [21,22,23,24,25], which terminology we adopt. An NFA $\mathscr{A} = \langle Q, \Sigma \rangle$ is called *carefully synchronizing* if there is a word

$w = a_1 \cdots a_\ell$, with $a_1, \ldots, a_\ell \in \Sigma$, that satisfies the condition (C), being the conjunction of $(C1)$–$(C3)$ below:

$(C1)$: the letter a_1 is defined at every state in Q;
$(C2)$: the letter a_t with $1 < t \leq \ell$ is defined at every state in $Q.a_1 \cdots a_{t-1}$,
$(C3)$: $|Q.w| = 1$.

Any w satisfying (C) is called a *carefully synchronizing word* (c.s.w., for short) for $\mathscr{A}$. Thus, when a c.s.w. is applied at any state in Q, no undefined transition occurs during the course of application. Every carefully synchronizing word is clearly D_1-synchronizing but the converse is not true in general; moreover, a D_1-synchronizing NFA may admit no c.s.w.

In this paper we focus on carefully synchronizing words for PFAs. There are several theoretical and practical reasons for this.

On the theoretical side, it is easy to see that each of the conditions (C), (D_1), (D_3) leads to the same notion when restricted to PFAs. As for D_2-synchronization, if a word w is D_2-synchronizing for a PFA $\mathscr{A}$, then w carefully synchronizes $\mathscr{A}$ whenever w is defined at each state. Otherwise w is nowhere defined by Lemma 1b, and such 'annihilating' words are nothing but usual synchronizing words for the CFA obtained from $\mathscr{A}$ by adding a new sink state and making all transitions undefined in $\mathscr{A}$ lead to this sink state. Synchronization of CFAs with a sink state is relatively well understood (see [33]), and therefore, we may conclude that D_2-synchronization also reduces to careful synchronization in the realm of PFAs. On the other hand, there exists a simple transformation that converts every NFA $\mathscr{A} = \langle Q, \Sigma \rangle$ into a PFA $\mathscr{B} = \langle Q, \Sigma' \rangle$ such that $\mathscr{A}$ is D_3-synchronizing if and only if so is $\mathscr{B}$ and the minimum lengths of D_3-synchronizing words for $\mathscr{A}$ and $\mathscr{B}$ are equal; see [12, Lemma 8.3.8] and [14, Lemma 3]. These observations demonstrate that from the viewpoint of optimal synchronization, studying carefully synchronizing words for PFAs may provide both lower and upper bounds applicable to arbitrary NFAs and all aforementioned kinds of synchronization.

Probably even more important is the fact that careful synchronization of PFAs is relevant in numerous applications. Due to the page limit, we mention only two examples here.

In industrial robotics, synchronizing automata are widely used to design feeders, sorters, and orienters that work with flows of certain objects carried by a conveyer. The goal is achieved by making the flow encounter passive obstacles placed appropriately along the conveyer belt; see [19,20] for the origin of this automata approach and [2] for an illustrative example. Now imagine that the objects to be oriented or sorted have a fragile

side that could be damaged if hitting an obstacle. In order to prevent any damage, we have to forbid 'dangerous' transitions in the automaton modelling the orienter/sorter so that the automaton becomes partial and the obstacle sequences must correspond to carefully synchronizing words. (Actually, the term 'careful synchronization' has been selected with this application in mind.)

Our second example relates to so-called synchronized codes[2]. Recall that a *prefix code* over a finite alphabet Σ is a set X of words in Σ^* such that no word of X is a prefix of another word of X. Decoding of a finite prefix code X over Σ can be implemented by a finite deterministic automaton $\mathscr{A}_X$ whose state Q is the set of all proper prefixes of the words in X (including the empty word ε) and whose transitions are defined as follows: for $q \in Q$ and $a \in \Sigma$,

$$q.a = \begin{cases} qa & \text{if } qa \text{ is a proper prefix of a word of } X, \\ \varepsilon & \text{if } qa \in X, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In general, $\mathscr{A}_X$ is a PFA (it is complete if and only if the code X is not contained in another prefix code over Σ). It can be shown that if $\mathscr{A}_X$ is carefully synchronizing, the code X has a very useful property: whenever a loss of synchronization between the decoder and the coder occurs (because of a channel error), it suffices to transmit a c.s.w. w of $\mathscr{A}_X$ such that w sends all states in Q to the state ε to ensure that the following symbols will be decoded correctly.

We may conclude that the problems of determining whether or not a given PFA is carefully synchronizing and of finding its shortest carefully synchronizing words are both natural and important. The bad news is that these problems turn out to be quite difficult: it is known that the first problem is PSPACE-complete and that the minimum length of carefully synchronizing words for carefully synchronizing PFAs can be exponential as a function of the number of states. (These results were found in [31,32] and later rediscovered and strengthened in [23].) Thus, one has to use some tools that have proved to be efficient for dealing with computationally hard problems. As mentioned in Section 1, in this paper we make an attempt to employ a satisfiability solver as such a tool.

[2] We refer the reader to [3, Chapters 3 and 10] for a detailed account of profound connections between codes and automata.

3 Encoding

For completeness, recall the formulation of the Boolean satisfiability problem (SAT). 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.

By Cook's classic theorem (see, e.g., [26, Theorem 8.2]), SAT is NP-complete, and by the very definition of NP-completeness, every problem in NP reduces to SAT. On the other hand, over the last score or so, many efficient *SAT-solvers*, i.e., specialized programs designed to solve instances of SAT have been developed. Modern SAT solvers can solve instances with hundreds of thousands of variables and millions of clauses within a few minutes. Due to this progress, the following approach to computationally hard problems has become quite popular nowadays: one encodes instances of such problems into instances of SAT that are then fed to a SAT-solver[3]. It is exactly the strategy that we want to apply.

We start with the following problem:

CSW (the existence of a c.s.w. of a given length):
INPUT: a PFA $\mathscr{A}$ and a positive integer ℓ (given in unary);
OUTPUT: YES if $\mathscr{A}$ has a c.s.w. of length ℓ;
NO otherwise.

We have to assume that the integer ℓ is given in unary because with ℓ given in binary, a polynomial time reduction from CSW to SAT is hardly possible. (Indeed, it easily follows from [23] that the version of CSW in which the integer parameter is given in binary is PSPACE-hard, and the existence of a polynomial reduction from a PSPACE-hard problem to SAT would imply that the polynomial hierarchy collapses at level 1.) In contrast, the version of CSW with the unary integer parameter is easily seen to belong to NP: given an instance $(\mathscr{A} = \langle Q, \Sigma\rangle, \ell)$ of CSW in this setting, guessing a word $w \in \Sigma^*$ of length ℓ is legitimate. Then one just checks whether or not w is carefully synchronizing for $\mathscr{A}$, and time spent for this check is clearly polynomial in the size of $(\mathscr{A}, \ell)$.

[3] We refer the reader to the survey [8] a detailed discussion of the approach and examples of its successful applications in various areas.

Now, given an arbitrary instance $(\mathscr{A}, \ell)$ of CSW, we construct 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). 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.

So, take a PFA $\mathscr{A} = \langle Q, \Sigma \rangle$ and an integer $\ell > 0$. Denote the sizes of Q and Σ by n and m respectively, and fix some numbering of these sets so that $Q = \{q_1, \ldots, q_n\}$ and $\Sigma = \{a_1, \ldots, a_m\}$.

We start with introducing the variables used in the instance (V, C) of SAT that encodes $(\mathscr{A}, \ell)$. The set V consists of two sorts of variables: $m\ell$ *letter variables* $x_{i,t}$ with $1 \le i \le m$, $1 \le t \le \ell$, and $n(\ell+1)$ *state variables* $y_{j,t}$ with $1 \le j \le n$, $0 \le t \le \ell$. We use the letter variables to encode the letters of a hypothetical c.s.w. w of length ℓ: namely, we want the value of the variable $x_{i,t}$ to be 1 if and only if the t-th letter of w is a_i. The intended meaning of the state variables is as follows: we want the value of the variable $y_{j,t}$ to be 1 whenever the state q_j belongs to the image of Q under the action of the prefix of w of length t, in which situation we say that q_j *is active after* t *steps*. We see that the total number of variables in V is $m\ell + n(\ell + 1) = (m + n)\ell + n$.

Now we turn to constructing the set of clauses C. It consists of four groups. The group I of *initial clauses* contains n one-literal clauses $y_{j,0}$, $1 \le j \le n$, and expresses the fact that all states are active after 0 steps.

For each $t = 1, \ldots, \ell$, the group L of *letter clauses* includes the clauses

$$x_{1,t} \vee \cdots \vee x_{m,t}, \quad \neg x_{r,t} \vee \neg x_{s,t}, \quad \text{where } \ 1 \le r < s \le m. \tag{1}$$

Clearly, the clauses (1) express the fact that the t-th position of our hypothetical c.s.w. w is occupied by exactly one letter in Σ. Altogether, L contains $\ell \left(\frac{m(m-1)}{2} + 1 \right)$ clauses.

For each $t = 1, \ldots, \ell$ and each triple (q_j, a_i, q_k) in the transition relation of $\mathscr{A}$, the group T of *transition clauses* includes the clause

$$\neg y_{j,t-1} \vee \neg x_{i,t} \vee y_{k,t}. \tag{2}$$

Invoking the basic laws of propositional logic, one sees that the clause (2) is equivalent to the implication $y_{j,t-1} \,\&\, x_{i,t} \to y_{k,t}$, that is, (2) expresses the fact that if the state q_j has been active after $t-1$ steps and a_i is the t-th letter of w, then the state $q_k = q_j.a_i$ becomes active after t steps. Further, for each $t = 1, \ldots, \ell$ and each pair (q_j, a_i) such that a_i is undefined at q_j in $\mathscr{A}$, we add to T the clause

$$\neg y_{j,t-1} \vee \neg x_{i,t}. \tag{3}$$

The clause is equivalent to the implication $y_{j,t-1} \to \neg x_{i,t}$, and thus, it expresses the requirement that the letter a_i should not be occur in the t-th position of w if q_j has been active after $t-1$ steps. Obviously, this corresponds to the conditions $(C1)$ (for $t=0$) and $(C2)$ (for $t>0$) in the definition of careful synchronization. For each $t=1,\ldots,\ell$ and each pair $(q_j,a_i)\in Q\times\Sigma$, exactly one of the clauses (2) or (3) occurs in T, whence T consists of ℓmn clauses.

The final group S of of *synchronization clauses* includes the clauses

$$\neg y_{r,\ell} \vee \neg y_{s,\ell}, \quad \text{where} \quad 1 \leq r < s \leq n. \tag{4}$$

The clauses (4) express the requirement that at most one state remains active when the action of the word w is completed, which corresponds to the condition $(C3)$ from the definition of careful synchronization. The group S contains $\frac{n(n-1)}{2}$ clauses.

Summing up, the number of clauses in $C := I \cup L \cup T \cup S$ is

$$n+\ell\left(\tfrac{m(m-1)}{2}+1\right)+\ell mn+\tfrac{n(n-1)}{2} = \ell\left(\tfrac{m(m-1)}{2}+mn+1\right)+\tfrac{n(n+1)}{2}. \tag{5}$$

In comparison with encodings used in our earlier papers [35,34], the encoding suggested here produces much smaller SAT instances. Since in the applications the size of the input alphabet is a (usually small) constant, the leading term in (5) is $\Theta(\ell n)$ while the restriction to PFAs of the encodings from [35,34] has $\Theta(\ell n^2)$ clauses.

Theorem 2 *A PFA $\mathscr{A}$ has a c.s.w. of length ℓ if and only if the instance (V,C) of SAT constructed above is satisfiable. Moreover, the carefully synchronizing words of length ℓ for $\mathscr{A}$ are in a 1-1 correspondence with the restrictions of satisfying assignments of (V,C) to the letter variables.*

Proof. Suppose that $\mathscr{A}$ has a c.s.w. of length ℓ. We fix such a word w and denote by w_t its prefix of length $t=1,\ldots,\ell$. Define a truth assignment $\varphi\colon V\to\{0,1\}$ as follows: for $1\leq i\leq m$, $0\leq j\leq n$, $1\leq t\leq\ell$, let

$$\varphi(x_{i,t}) := \begin{cases} 1 & \text{if the } t\text{-th letter of } w \text{ is } a_i,\\ 0 & \text{otherwise;}\end{cases} \tag{6}$$

$$\varphi(y_{j,0}) := 1; \tag{7}$$

$$\varphi(y_{j,t}) := \begin{cases} 1 & \text{if the state } q_j \text{ lies in } Q.w_t,\\ 0 & \text{otherwise.}\end{cases} \tag{8}$$

In view of (6) and (7), φ satisfies all clauses in L and respectively I. As $w_\ell = w$ and $|Q.w| = 1$, we see that (8) ensures that φ satisfies all clauses in S. It remains to analyze the clauses in T. For each fixed $t = 1, \dots, \ell$, these clauses are in a 1-1 correspondence with the pairs in $Q \times \Sigma$. We fix such a pair (q_j, a_i), denote the clause corresponding to (q_j, a_i) by c and consider three cases.

***Case 1**: the letter a_i is not the t-th letter of w.* In this case $\varphi(x_{i,t}) = 0$ by (6), and hence, $\varphi(c) = 1$ since the literal $\neg x_{i,t}$ occurs in c, independently of c having the form (2) or (3).

***Case 2**: the letter a_i is the t-th letter of w but it is undefined at q_j.* In this case the clause c must be of the form (3). Observe that $t > 1$ in this case since the first letter of the c.s.w. w must be defined at each state in Q. Moreover, the state q_j cannot belong to the set $Q.w_{t-1}$ because a_i must be defined at each state in this state. Hence $\varphi(y_{j,t-1}) = 0$ by (8), and $\varphi(c) = 1$ since the literal $\neg y_{j,t-1}$ occurs in c.

***Case 3**: the letter a_i is the t-th letter of w and it is defined at q_j.* In this case the clause c must be of the form (2), in which the literal $y_{k,t}$ corresponds to the state $q_k = q_j.a_i$. If the state q_j does not belong to the set $Q.w_{t-1}$, then as in the previous case, we have $\varphi(y_{j,t-1}) = 0$ and $\varphi(c) = 1$. If q_j belongs to $Q.w_{t-1}$, then the state q_k belongs to the set $(Q.w_{t-1}).a_i = Q.w_t$, whence $\varphi(y_{k,t}) = 1$ by (8). We conclude that $\varphi(c) = 1$ since the literal $y_{k,t}$ occurs in c.

Conversely, suppose that $\varphi\colon V \to \{0,1\}$ is a satisfying assignment for (V, C). Since φ satisfies the clauses in L, for each $t = 1, \dots, \ell$, there exists a unique $i \in \{1, \dots, m\}$ such that $\varphi(x_{i,t}) = 1$. This defines a map $\chi\colon \{1, \dots, \ell\} \to \{1, \dots, m\}$. Let $w := a_{\chi(1)} \cdots a_{\chi(\ell)}$. We aim to show that w is a c.s.w. for $\mathscr{A}$, i.e., to verify that w fulfils the conditions $(C1)$–$(C3)$ from the definition of a c.s.w. For this, we first prove two auxiliary claims. Recall that a state is said to be active after t steps if it lies in $Q.w_t$, where, as above, w_t is the length t prefix of the word w. (By the length 0 prefix we understand the empty word ε.)

***Claim 1**. For each $t = 0, 1, \dots, \ell$, there are states active after t steps.*

***Claim 2**. If a state q_k is active after t steps, then $\varphi(y_{k,t}) = 1$.*

We prove both claims simultaneously by induction on t. The induction basis $t = 0$ is guaranteed by the fact that all states are active after 0 steps and φ satisfies the clauses in I. Now suppose that $t > 0$ and there are states active after $t - 1$ steps. Let q_r be such a state. Then $\varphi(y_{r,t-1}) = 1$ by the induction assumption. Let $i := \chi(t)$, that is, a_i is the t-th letter of the word w. Then $\varphi(x_{i,t}) = 1$, whence φ cannot satisfy the clause of the form (3) with $j = r$. Hence this clause cannot appear in T as φ satisfies

the clauses in T. This means that the letter a_i is defined at q_r in $\mathscr{A}$, and the state $q_s := q_r.a_i$ is active after t steps. Claim 1 is proved.

Now let q_k be an arbitrary state that is active after $t > 0$ steps. Since a_i is the t-th letter of w, we have $Q.w_t = (Q.w_{t-1}).a_i$, whence $q_k = q_j.a_i$ for same $q_j \in Q.w_{t-1}$. Therefore the clause (2) occurs in T, and thus, it is satisfied by φ. Since q_j is active after $t-1$ steps, $\varphi(y_{j,t-1}) = 1$ by the induction assumption; besides that, $\varphi(x_{i,t}) = 1$. We conclude that in order to satisfy (2), the assignment φ must fulfil $\varphi(y_{k,t}) = 1$. This completes the proof of Claim 2.

We turn to prove that the word w fulfils $(C1)$ and $(C2)$. This amounts to verifying that for each $t = 1, \ldots, \ell$, the t-th letter of the word w is defined at every state q_j that is active after $t-1$ steps. Let, as above, a_i stand for the t-th letter of w. If a_j were undefined at q_j, then by the definition of the set T of transition clauses, this set would include the corresponding clause (3). However, $\varphi(x_{i,t}) = 1$ by the construction of w and $\varphi(y_{j,t-1}) = 1$ by Claim 2. Hence φ does not satisfy this clause while the clauses from T are satisfied by φ, a contradiction.

Finally, consider $(C3)$. By Claim 1, some state is active after ℓ steps. On the other hand, the assignment φ satisfies the clauses in S, which means that $\varphi(y_{j,\ell}) = 1$ for at most one index $j \in \{1, \ldots, n\}$. By Claim 2 this implies that at most one state is active after ℓ steps. We conclude that exactly one state is active after ℓ steps, that is, $|Q.w| = 1$. □

4 Experimental results

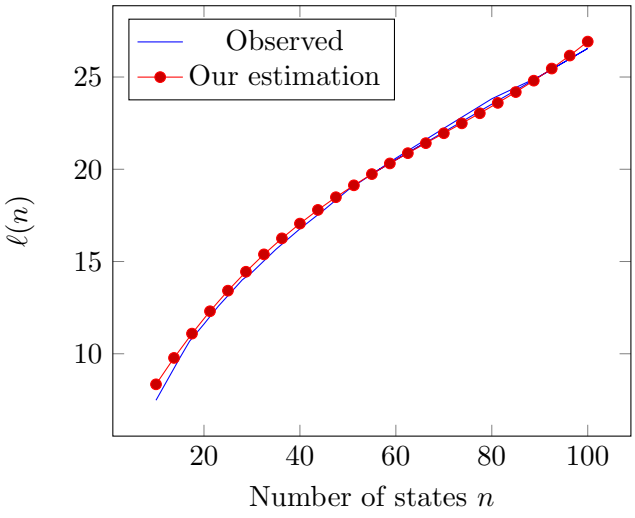
We have successfully applied the encoding constructed in Sect. 3 to solve CSW instances with the help of a SAT-solver. As in [36,10,35,34], we have used MiniSat 2.2.0 [6,7]. In order to find a c.s.w. of minimum length for a given PFA $\mathscr{A}$, we have considered CSW instances $(\mathscr{A}, \ell)$ with fixed $\mathscr{A}$ and performed binary search on ℓ. Even though our encoding is different form those we used in [35,34], it shares with them the following useful feature: when presented in DIMACS CNF format, the 'primary' SAT instance that encodes the CSW instance $(\mathscr{A}, 1)$ can be easily scaled to the SAT instances that encode the CSW instances $(\mathscr{A}, \ell)$ with any value of ℓ. Due to this feature, one radically reduces time needed to prepare the input data for the SAT-solver. We refer the reader to [34, Sect. 3] for a detailed explanation of the trick and an illustrative example.

We implemented the algorithm outlined above in C++ and compiled with GCC 4.9.2. In our experiments we used a personal computer with

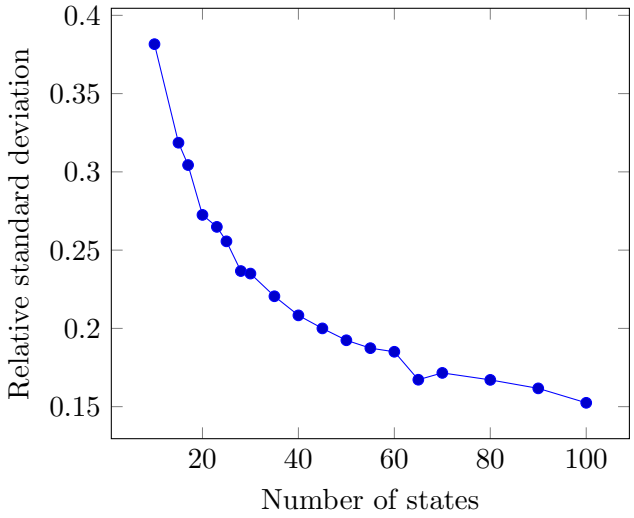
an Intel(R) Core(TM) i5-2520M processor with 2.5 GHz CPU and 4GB of RAM. The code can be found at `https://github.com/hananshabana/SynchronizationChecker`.

As a sample of our experimental findings, we present here our results on synchronization of PFAs with a unique undefined transition. Observe that the problem of deciding whether or not a given PFA is carefully synchronizing remains PSPACE-complete even if restricted to this rather special case [23]. We considered random PFAs with $n \leq 100$ states and two input letters. The condition $(C1)$ in the definition of a carefully synchronizing PFA implies that such a PFA must have an everywhere defined letter. We denoted this letter by a and the other letter, called b, was chosen to be undefined at a unique state. Further, it is easy to see that for a PFA $\langle Q, \{a, b\}\rangle$ with a, b so chosen to be carefully synchronizing, it is necessary that $|Q.a| < |Q|$. Therefore, we fixed a state $q_a \in Q$ and then selected a uniformly at random from all n^{n-1} maps $Q \to Q \setminus \{q_a\}$. Similarly, to ensure there is a unique undefined transition with b, we fixed a state $q_b \in Q$ (not necessarily different from q_a) and then selected b uniformly at random from all $(n-1)^n$ maps $Q \setminus \{q_b\} \to Q$. For each fixed n, we generated up to 1000 random PFAs this way and calculated the average length $\ell(n)$ of their shortest carefully synchronizing words. We used the least squares method to find a function that best reflects how $\ell(n)$ depends on n, and it turned out that our results are reasonably well approximated by the following expression:

$$\ell(n) \approx 3.92 + 0.49n - 0.005n^2 + 0.000024n^3.$$



The next graph shows the relation between the relative standard deviation of our datasets and the number of states. We see that the relative standard deviation gradually decreases as the number of states grows.



We performed similar experiments with random PFAs that have two or three undefined transition. We also tested our algorithm on PFAs from the series $\mathscr{P}_n$ suggested in [5]. The state set of $\mathscr{P}_n$ is $\{1, 2, \ldots, n\}$, $n \geq 3$, on which the input letters a and b act as follows:

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

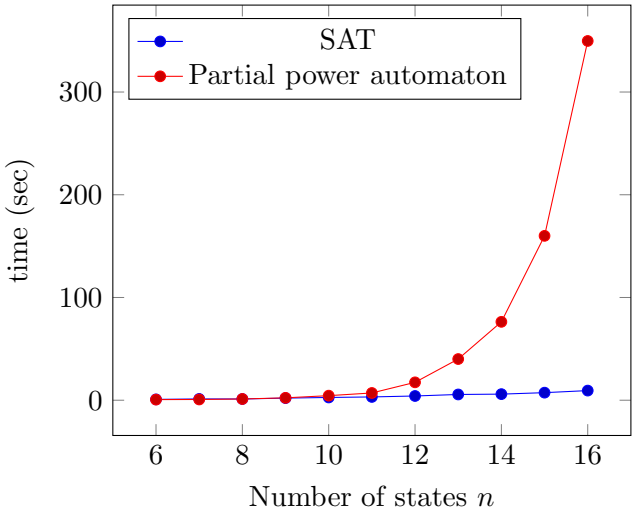
We examined all automata $\mathscr{P}_n$ with $n = 4, 5, \ldots, 11$, and for each of them, our result matched the theoretical value predicted by [5, Theorem 3]. The time consumed ranges from 0.301 sec for $n = 4$ to 4303 sec for $n = 11$. Observe that in the latter case the shortest c.s.w. has length 116 so that honest binary search started with $(\mathscr{P}_{11}, 1)$ required 11 iterations.

We made also a comparison with the only approach to computing carefully synchronizing words of minimum length that we had found in the literature, namely, the approach based on partial power automata. Given a PFA $\mathscr{A} = \langle Q, \Sigma \rangle$, its *partial power automaton* $\mathcal{P}(\mathscr{A})$ has the subsets of Q as the states, the same input alphabet Σ, and the transition function defined as follows: for each $a \in \Sigma$ and each $P \subseteq Q$,

$$P.a := \begin{cases} \{q.a \mid q \in P\} & \text{provided } q.a \text{ is defined for all } q \in P, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

It is easy to see that $w \in \Sigma^*$ is a c.s.w. of minimum length for $\mathscr{A}$ if and only if w labels a minimum length path in $\mathcal{P}(\mathscr{A})$ starting at Q and ending at a singleton. Such a path can be found by breadth-first search in the underlying digraph of $\mathcal{P}(\mathscr{A})$.

The result of the comparison are presented in the picture below. In this experiment we had to restrict to PFAs with at most 16 states since beyond this number of states, our implementation of the method based on partial power automata could not complete the computation due to memory restrictions (recall that we used rather modest computational resources). However, we think that the exhibited data suffice to demonstrate that the approach based on SAT-solvers shows a by far better performance.



5 Conclusion and future work

We have presented an attempt to approach the problem of computing a c.s.w. of minimum length for a given PFA via the SAT-solver method. For this, we have developed a new encoding, which, in comparison with encodings used in our earlier papers [35,34], requires a more sophisticated proof but leads to more economic SAT instances.

We plan to continue our experiments. In particular, it is interesting to compare the minimum lengths of a synchronizing word for a synchronizing DFA and of carefully synchronizing words for PFAs that can be obtained from the DFA by removing one or more of its transitions.

We also plan to extend the SAT-solver approach to so-called *exact synchronization* of PFAs which is of interest for certain applications.

References

1. Altun, Ö.F., Atam, K.T., Karahoda, S., Kaya, K.: Synchronizing heuristics: Speeding up the slowest. In: N. Yevtushenko, A. R. Cavalli, H. Yenigün (eds.), Testing Software and Systems, 29th Int. Conf., ICTSS 2017. LNCS, vol. 10533, pp. 243–256. Springer (2017)
2. Ananichev, D.S., Volkov, M.V. Some results on Černý type problems for transformation semigroups. In: I. M. Araújo, M. J. J. Branco, V. H. Fernandes, G. M. S. Gomes (eds.), Semigroups and Languages, World Scientific, pp. 23–42 (2004)
3. Berstel, J., Perrin, D., Reutenauer, C.: Codes and Automata. Cambridge University Press (2009)
4. Burkhard, H.-D.: Zum Längenproblem homogener Experimente an determinierten und nicht-deterministischen Automaten. Elektronische Informationsverarbeitung und Kybernetik 12, 301–306 (1976)
5. de Bondt M., Don H., Zantema H.: Lower bounds for synchronizing word lengths in partial automata. Available at `https://arxiv.org/abs/1801.10436`
6. Eén, N., Sörensson, N.: An extensible SAT-solver. In: E. Giunchiglia, A. Tacchella (eds.), Theory and Applications of Satisfiability Testing, 6th Int. Conf., SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer (2004)
7. Eén, N., Sörensson, N.: The MiniSat Page. Available at `http://minisat.se`.
8. Gomes, C.P., Kautz, H., Sabharwal, A., Selman, B.: Satisfiability solvers. Chapter 2 in: F. van Harmelen, V. Lifschitz, B. Porter (eds.), Handbook of Knowledge Representation, Vol. I, Elsevier, 89–134 (2008)
9. Goralčk, P., Hedrlín, Z., Koubek, V.; Ryšlinková, J.: A game of composing binary relations. RAIRO Inform. Théor. 16(4), 365-369 (1982)
10. Güniçen, C., Erdem, E., Yenigün, H.: Generating shortest synchronizing sequences using Answer Set Programming. In: M. Fink, Yu. Lierler (eds.), Answer Set Programming and Other Computing Paradigms, 6th Int. Workshop, ASPOCP 2013, pp. 117–127 (2013) Available at `https://arxiv.org/abs/1312.6146`
11. Imreh, B., Steinby, M.: Directable nondeterministic automata. Acta Cybernetica 14, 105–115 (1999)
12. Ito, M.: Algebraic Theory of Automata and Languages. World Scientific (2004)
13. Ito, M., Shikishima-Tsuji, K.: Some results on directable automata. In: J. Karhumäki, H. Mauer, Gh. Păun, G. Rozenberg (eds.), Theory Is Forever. Essays dedicated to Arto Salomaa on the occasion of his 70th birthday. LNCS, vol. 3113, pp. 125–133. Springer (2004)
14. Ito, M., Shikishima-Tsuji, K.: Shortest directing words of nondeterministic directable automata. Discrete Math. 308(21), 4900–4905 (2008)
15. Karahoda, S., Erenay, O.T., Kaya, K., Türker, U.C.,Yenigün, H.: Parallelizing heuristics for generating synchronizing sequences. In: F. Wotawa, M. Nica, N. Kushik (eds.), Testing Software and Systems, 28th Int. Conf., ICTSS 2016. LNCS, vol. 9976, pp. 106–122. Springer (2016)
16. Karahoda, S., Kaya, K., Yenigün, H.: Synchronizing heuristics: Speeding up the fastest. Expert Syst. Appl. 94:265–275 (2018)
17. Kowalski, J., Roman, A.: A new evolutionary algorithm for synchronization. In: G. Squillero, K. Sim (eds), Applications of Evolutionary Computation, 20th European Conf., EvoApplications 2017, Part I. LNCS, vol. 10199, pp. 620–635. Springer (2017)
18. Kari, J., Volkov, M.V.: Černý's conjecture and the Road Coloring Problem. Chapter 15 in: J.-É. Pin (ed.), Handbook of Automata Theory, Vol. I, EMS Publishing House (in print)

19. Natarajan, B.K.: An algorithmic approach to the automated design of parts orienters. In: Proc. 27th Annual Symp. Foundations Comput. Sci., pp. 132–142. IEEE Press (1986)
20. Natarajan, B.K.: Some paradigms for the automated design of parts feeders. Int. J. Robotics Research 8(6), 89–109 (1989)
21. Martyugin, P.V.: Lower bounds for the length of the shortest carefully synchronizing words for two- and three-letter partial automata. Diskretn. Anal. Issled. Oper. 15(4), 44–56 (2008)
22. Martyugin, P.V.: A lower bound for the length of the shortest carefully synchronizing words. Russian Math. (Iz. VUZ) 54(1), 46-54 (2010)
23. Martyugin, P.V.: Synchronization of automata with one undefined or ambiguous transition. In: N. Moreira, R. Reis (eds.), Implementation and Application of Automata, 17th Int. Conf., CIAA 2012. LNCS, vol. 7381, pp. 278–288. Springer (2012)
24. Martyugin, P.V.: Careful synchronization of partial automata with restricted alphabets. In: A. A. Bulatov, A. M. Shur (eds.): Computer Science – Theory and Applications, 8th Int. Comp. Sci. Symp. in Russia, CSR 2013. LNCS, vol. 7913, pp. 76–87. Springer (2013)
25. Martyugin, P.V.: Complexity of problems concerning carefully synchronizing words for PFA and directing words for NFA. Theory Comput. Syst. 54(2), 293–304 (2014)
26. Papadimitriou, C.H.: Computational Complexity, Addison-Wesley (1994)
27. Pixley, C., Jeong, S.-W., Hachtel, G.D: Exact calculation of synchronization sequences based on binary decision diagrams. In: Proc. 29th Design Automation Conf., pp. 620–623. IEEE Press (1992)
28. Podolak, I.T., Roman, A., Jędrzjczyk, D.: Application of hierarchical classifier to minimal synchronizing word problem. In L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, J. M. Zurada (eds.), Artificial Intelligence and Soft Computing, 11th Int. Conf., ICAISC 2012, Part I. LNCS, vol. 7267, pp. 421–429. Springer (2012)
29. Podolak, I.T., Roman, A., Szykuła, M., Zieliński, B.: A machine learning approach to synchronization of automata. Expert Syst. Appl. 97:357–371 (2018)
30. Roman, A.: Genetic algorithm for synchronization. In: A. Dediu, A. Ionescu, C. Martín-Vide (eds.), Language and Automata Theory and Applications, 3rd Int. Conf., LATA 2009. LNCS, vol. 5457, pp. 684–695. Springer (2009)
31. Rystsov, I.K.: Asymptotic estimate of the length of a diagnostic word for a finite automaton. Cybernetics 16(1), 194–198 (1980)
32. Rystsov, I.K.: Polynomial complete problems in automata theory, Inf. Process. Lett. 16(3), 147–151 (1983)
33. Rystsov, I.K.: Reset words for commutative and solvable automata. Theoret. Comput. Sci. 172(1), 273–279 (1997)
34. Shabana, H.: D_2-synchronization in nondeterministic automata, Ural Math. J. 4(2), 99–110 (2018)
35. Shabana, H., Volkov, M.V.: Using Sat solvers for synchronization issues in nondeterministic automata, Siberian Electronic Math. Reports 15, 1426–1442 (2018).
36. Skvortsov, E., Tipikin, E.: Experimental study of the shortest reset word of random automata. In: B. Bouchou-Markhoff, P. Caron, J.-M. Champarnaud, D. Maurel (eds.), Implementation and Application of Automata, 16th Int. Conf, CIAA 2011. LNCS, vol. 6807, pp. 290–298. Springer (2011)
37. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: C. Martín-Vide, F. Otto, H. Fernau (eds.), Languages and Automata Theory and Applications, 2nd Int. Conf., LATA 2008. LNCS, vol. 5196, pp. 11–27. Springer (2008)