Acta Cybernetica **19** (2009) 517–536.

Complexity of Problems Concerning Reset Words for Some Partial Cases of Automata

Pavel Martyugin*

Abstract

A word w is called a reset word for a deterministic finite automaton $\mathscr{A}$ if it maps all states of $\mathscr{A}$ to one state. A word w is called a compressing to M states for a deterministic finite automaton $\mathscr{A}$ if it maps all states of $\mathscr{A}$ to at most M states. We consider several subclasses of automata: aperiodic, $\mathscr{D}$-trivial, monotonic, partially monotonic automata and automata with a zero state. For these subclasses we study the computational complexity of the following problems. Does there exist a reset word for a given automaton? Does there exist a reset word of given length for a given automaton? What is the length of the shortest reset word for a given automaton? Moreover, we consider complexity of the same problems for compressing words.

Keywords: synchronization, automata, reset words, computational complexity

1 Synchronization

A *deterministic finite automaton* (DFA) $\mathscr{A}$ is a triple $\langle Q, \Sigma, \delta\rangle$, where Q is a finite set of *states*, Σ is a finite *alphabet*, and $\delta : Q \times \Sigma \to Q$ is a totally defined *transition function*. The function δ extends in a unique way to an action $Q \times \Sigma^* \to Q$ of the free monoid Σ^* over Σ; this extension is also denoted by δ. We denote $\delta(q, w)$ by $q.w$. We also define for $S \subseteq Q$, $w \in \Sigma^*$, $\delta(S, w) = S.w = \{q.w | q \in S\}$.

A DFA $\mathscr{A}$ is called *synchronizing* if there exists a word $w \in \Sigma^*$ whose action resets $\mathscr{A}$, that is, leaves the automaton in one particular state no matter at which state in Q it started: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$. Any word w with this property is said to be a *reset* or *synchronizing word* for the automaton.

In [1], Černý produced for each n a synchronizing automaton with n states and 2 input letters whose shortest reset word has length $(n-1)^2$ and conjectured that these automata represent the worst possible case, that is, every synchronizing automaton with n states can be reset by a word of length $(n-1)^2$. The conjecture is arguably the most longstanding open problem in the combinatorial theory of finite automata.

*Ural State University, E-mail: `martugin@mail.ru`

Upper bounds within the confines of the Černý conjecture have been obtained for the maximum length of the shortest reset words for synchronizing automata in some special classes, see, e.g., [2, 6, 7, 9, 10, 13, 14]. Some of these classes are considered in this paper.

One of these classes is the class of commutative DFA. An automaton $\mathscr{A} = (Q, \Sigma, \delta)$ is said to be *commutative* if its transformation monoid is commutative, that is, for every state $q \in Q$ and for all letters $a, b \in \Sigma$, $\delta(q, ab) = \delta(q, ba)$. Rystsov in [10] proved that every commutative synchronizing automaton with n states has a reset word of length $n - 1$. This means that the Černý conjecture is true for commutative automata.

Another class of DFA considered by Rystsov in [11] is a class of automata with simple idempotents. Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a DFA. Let $a \in \Sigma$. If $\delta(Q, a) = Q$, then the letter a is called a *permutation.* If $\delta(Q, a^2) = \delta(Q, a)$, then the letter a is called an *idempotent.* If the letter a is an idempotent and $|\delta(Q, a)| = |Q| - 1$ then a is called a *simple idempotent.* If for DFA $\mathscr{A}$ all letters are permutations or simple idempotents, then such an automaton is called an *automaton with simple idempotents.* Every n-state synchronizing automaton with simple idempotents admits a reset word of length $2(n-1)^2$ (see [11]).

Another natural class of DFA is a class of automata with a zero state. A state z of a DFA $\mathscr{A} = \langle Q, \Sigma, \delta\rangle$ is said to be a *zero* state if $\delta(z, a) = z$ for all $a \in \Sigma$. It is clear that a synchronizing automaton may have at most one zero state and each word that resets a synchronizing automaton possessing a zero state must bring all states to the zero state. A rather straightforward argument shows that any n-state synchronizing automaton can be reset by a word of length $\frac{n(n-1)}{2}$, see, e.g., [10]. This upper bound is in fact tight because, for each n, there exists a synchronizing automaton with n states and $n-1$ input letters which cannot be reset by any word of length less than $\frac{n(n-1)}{2}$. In [5] it was proved that for each integer $n \geq 8$, there exists a synchronizing automaton with n states and 2 input letters such that the length of the shortest reset word for this automaton is $\left\lceil \frac{n^2+6n-16}{4} \right\rceil$.

Another two classes of DFA can be defined via Greens Relations $\mathscr{H}$ and $\mathscr{D}$. Let M be a transition monoid of some DFA $\mathscr{A}$. Let $U, V \subseteq M$. Denote $UV = \{uv | u \in U, v \in V\}$. The relations $\mathscr{H}$ and $\mathscr{D}$ can be defined on any monoid. Let M be a finite monoid and $u, v \in M$ then $u\mathscr{H}v \Leftrightarrow uM = vM$ and $Mu = Mv$; $u\mathscr{D}v \Leftrightarrow MuM = MvM$. An automaton is called *aperiodic* or *$\mathscr{H}$-trivial* if its transition semigroup has no nontrivial $\mathscr{H}$-classes. An automaton is called *$\mathscr{D}$-trivial* if its transition semigroup has no nontrivial $\mathscr{D}$-classes. Every synchronizing strongly connected aperiodic automaton has a reset word of length $\lfloor \frac{n(n+1)}{6} \rfloor$ (see [9]). Moreover, any synchronizing aperiodic automaton has a reset word of length $\frac{n(n-1)}{2}$.

We also consider another three classes of automata. A DFA $\mathscr{A} = (Q, \Sigma, \delta)$ is called *monotonic* if its state set admits a linear order $\leq$ such that for each letter $a \in \Sigma$ the transformation $\delta(\_, a)$ of Q preserves $\leq$ in the sense that $\delta(q, a) \leq \delta(q', a)$ whenever $q \leq q'$. Every synchronizing monotonic automaton has a reset word of length at most $n - 1$ (see [6]). A DFA is called *cyclically monotonic* if its state set admits a cyclic order preserving by an action of any letter. Every synchronizing cyclically monotonic automaton has a reset word of length at most $(n-1)^2$ (see [2]).

A deterministic incomplete automaton is an automaton with a partial transition function (in an incomplete automaton the value $\delta(q, a)$ can be undefined on some pairs (q, a)). A deterministic incomplete automaton is called *partially monotonic* if its state set admits a linear order $\leq$ such that for each $a \in \Sigma$ the partial transformation $\delta(\_, a)$ preserves the restriction of order to the domain of the transformation. Every incomplete automaton $\mathscr{A} = (Q, \Sigma, \delta)$ can be transformed to a complete automaton $\mathscr{A}' = (Q \cup \{end\}, \Sigma, \delta')$ by an adding of one state *end* such that if for some $q \in Q, a \in \Sigma$ the value $\delta(q, a)$ is undefined then $\delta'(q, a) = end$. If the automaton $\mathscr{A}$ is partially monotonic then we call the DFA $\mathscr{A}'$ *partially monotonic* too. Every partially monotonic DFA is an aperiodic automaton with a zero state. Every synchronizing partially monotonic automaton has a reset word of length at most $n-1$ $(n-1) + \lfloor \frac{n-2}{2} \rfloor$. On the other hand, for any $n \geq 6$ there exists a 2-letter partially monotonic DFA such that its shortest reset word has length $(n-1) + \lfloor \frac{n-2}{2} \rfloor$ (see [8]).

2 Complexity

It is natural to consider computational complexity of various problems arising from the study of automata synchronization. Most natural questions are: is the given automaton synchronizing or not, and what is the length of the shortest reset word for a given automaton?

In [2], Eppstein presented an algorithm which checks whether the given DFA $\mathscr{A} = (Q, \Sigma, \delta)$ is synchronizing. This algorithm works within $O(|\Sigma| \cdot |Q|^2) + |Q|^3$ times bound. Moreover, for a synchronizing automaton this algorithm finds some reset word. This word can be not a shortest reset word for $\mathscr{A}$.

In [3], Salomaa proved that the following problem is NP-hard. Let a DFA $\mathscr{A}$ and an integer number L be given. The question is the following: is there a word of length $\leq L$ resetting the automaton $\mathscr{A}$. This problem remains NP-complete even if the input automaton has a 2-letter alphabet.

In [4], Samotij considered another problem. Let a DFA $\mathscr{A}$ and an integer number L be given. The question is the following: is the length of the shortest reset word for automaton $\mathscr{A}$ equal to L. It turns out that this problem is NP-hard and co-NP-hard. To prove these statements the construction from [3] can be applied. This method gives also that the considered problem remains NP-hard and co-NP-hard for 2-letter automata.

There is a further natural problem for DFA. Given a DFA $\mathscr{A} = (Q, \Sigma, \delta)$, we define a rank of the word $w \in \Sigma^*$ as the cardinality of the image of the transformation $\delta(\_, w)$ of the set Q. (Thus, in this terminology reset words are exactly the words of rank 1). In 1978 Pin conjectured that for every M, if an n-state automaton admits a word of rank at most M, then it also has a word with rank at most M and of length $(n-M)^2$. But Kari [12] has found a counterexample in the case $n - M = 4$. Let some word have a rank M in the automaton $\mathscr{A}$. Then we say that this word *compresses* the automaton $\mathscr{A}$ to M states. We consider the complexity of the problem of finding the length of the shortest word compressing a given automaton to M states.

In the present paper we consider these problems for partial cases of the DFA. We give a denotation to any considered class of DFA. Let

- DFA be the denotation of the class of all DFA,
- COM be the denotation of the class of commutative automata,
- SIMPID be the denotation of the class of automata with simple idempotents,
- APER be the denotation of the class of aperiodic automata,
- $\mathscr{D}$-TRIV be the denotation of the class of $\mathscr{D}$-trivial automata,
- MON be the denotation of the class of monotonic automata,
- PMON be the denotation of the class of partially monotonic automata,
- ZERO be the denotation of the class of automata with a zero state.

Let C be some class of DFA. Let us give formal definitions of the following problems.

Problem: $SYN(C)$
Input: A DFA $\mathscr{A} = (Q, \Sigma, \delta)$ from the class C.
Question: Is there a reset word $w \in \Sigma^*$ for the automaton $\mathscr{A}$?

Problem: $SYN(C, \leq L)$
Input: A DFA $\mathscr{A} = (Q, \Sigma, \delta)$ from the class C and an integer $L > 0$.
Question: Is there a reset word $w \in \Sigma^*$ of length $\leq L$ for the automaton $\mathscr{A}$?

Problem: $SYN(C, = L)$
Input: A DFA $\mathscr{A} = (Q, \Sigma, \delta)$ from the class C and an integer $L > 0$.
Question: Does the shortest reset word $w \in \Sigma^*$ for the automaton $\mathscr{A}$ have length L?

Problem: $COMP(C, M, \leq L)$
Input: A DFA $\mathscr{A} = (Q, \Sigma, \delta)$ from the class C and integers $M, L > 0$.
Question: Is there a word $w \in \Sigma^*$ of length $\leq L$ such that $|\delta(Q, w)| \leq M$?

Problem: $COMP(C, M, = L)$
Input: A DFA $\mathscr{A} = (Q, \Sigma, \delta)$ from the class C and integers $M, L > 0$.
Question: Does the shortest word $w \in \Sigma^*$ such that $|\delta(Q, w)| \leq M$ have length L?

In applications automata usually have not an arbitrary alphabet, but an alphabet of fixed size. If the input of some PROBLEM contains only automata having an alphabet of size $\leq k$ for some fixed k, then we call such a problem k-PROBLEM (for example, k-$SYN(ZERO)$).

Let the DFA $\mathscr{A} = (Q, \Sigma, \delta)$, $|Q| = n, |\Sigma| = k$ and the integer $L > 0$ be input data. In this paper we obtain the following results.

- The problems $SYN(ZERO)$, $SYN(\mathscr{D}\text{-}TRIV)$, $SYN(MON)$, $SYN(PMON)$ can be solved in time $O(nk)$.
- The problems $SYN(ZERO, \leq L)$, $SYN(APER, \leq L)$, $SYN(\mathscr{D}\text{-}TRIV, \leq L)$, $SYN(PMON, \leq L)$ are NP-complete together with the corresponding k-problems for $k \geq 2$.
- The problems $SYN(ZERO, = L)$, $SYN(APER, = L)$, $SYN(\mathscr{D}\text{-}TRIV, = L)$, $SYN(PMON, = L)$ are NP-hard and co-NP-hard together with the corresponding k-problems for $k \geq 2$.
- The problem $SYN(COM)$ can be solved in time $O(kn \ln n)$.
- The problem $SYN(COM, \leq L)$ is NP-complete.
- The problem k-$SYN(COM, \leq L)$ for some fixed $k \geq 1$ can be solved in time $O(n^k \ln n)$.
- The problem $SYN(COM, = L)$ is NP-hard and co-NP-hard.
- The problem k-$SYN(COM, = L)$ for some fixed $k \geq 1$ can be solved in time $O(n^k \ln n)$.
- The problem $SYN(SIMPID, \leq L)$ is NP-complete.
- The problem 2-$SYN(SIMPID, \leq L)$ can be solved in time $O(n)$.
- The problem $SYN(SIMPID, = L)$ is NP-hard and co-NP-hard.
- The problem 2-$SYN(SIMPID, = L)$ can be solved in time $O(n)$.
- The problems $COMP(MON, M, \leq L)$ and k-$COMP(MON, M, \leq L)$ for $k \geq 2$ are NP-complete.
- The problems $COMP(MON, M, = L)$ and k-$COMP(MON, M, = L)$ for $k \geq 2$ are NP-hard and co-NP-hard.

The problems $SYN(APER)$ and $SYN(SIMPID)$ can be solved in time $O(kn^2)$ because these problems can be solved in such time for arbitrary input DFA (not necessarily aperiodic or automata with simple idempotents), see [2]. It follows from [2] that the problems $SYN(MON, \leq L)$ and $SYN(MON, = L)$

can be solved in time $O(kn^2)$ (the algorithm from [2] works with cyclically monotonic automata; any monotonic automaton is cyclically monotonic). The only open question is what is the complexity of the problems k-$SYN(SIMPID, \leq L)$ and k-$SYN(SIMPID, = L)$ for fixed $k > 2$?

For the sequel, we need some notation. We denote by $|Q|$ the cardinality of a set Q. We denote the set of all subsets of a set Q by 2^Q. For a word $w \in \{a, b\}^*$, we denote by $|w|$ the length of w and by $w[i]$, where $1 \leq i \leq |w|$, the i^{th} letter in w from the left. If $1 \leq i \leq j \leq |w|$, we denote by $w[i, j]$ the word $w[i] \cdots w[j]$.

3 Checking the synchronizability

Proposition 3.1. *Let C be a subclass of DFA, then*

1. The problems SYN(C) and k-SYN(C) for $k \geq 1$ can be solved in time $O(n^2k)$, where n is a number of states, k is an alphabet size.

2. The problems $SYN(C, \leq L)$ and k-$SYN(C, \leq L)$ for $k \geq 1$ belong to NP.

Proof. From [2] we have that the problem $SYN(DFA)$ can be solved in time $O(n^2k)$. From [3] we have that the problem $SYN(DFA, \leq L)$ belongs to NP. The problems $SYN(C)$ and k-$SYN(C)$ are partial cases of the problem $SYN(DFA)$. Therefore they can be solved in time $O(n^2k)$ too. The problems $SYN(C, \leq L)$ and k-$SYN(C, \leq L)$ are partial cases of the problem $SYN(DFA, \leq L)$. Therefore these problems belong to NP. □

Proposition 3.2. *The problem $SYN(ZERO)$ can be solved in time $O(nk)$, where n is a number of states, k is an alphabet size.*

Proof. We construct an algorithm checking whether the DFA $\mathscr{A}$ is synchronizing or not. Let q_0 be a zero state in automaton $\mathscr{A}$. Our algorithm is a breadth first search from the state q_0 in the automaton $\mathscr{A}$. We move along the arrows in back direction. If some state $q \in Q$ was not visited during the search, then the automaton is not synchronizing, because there is no word $w \in \Sigma$ such that $q.w = q_0$. Otherwise the automaton is synchronizing. Every arrow can be used no more than once during the search. Therefore, the complexity of the algorithm is $O(nk)$, because the automaton $\mathscr{A}$ contains exactly nk arrows. □

Proposition 3.3. *The problems $SYN(\mathscr{D}$-$TRIV)$ and $SYN(PMON)$ can be solved in time $O(nk)$, where n is a number of states, k is an alphabet size.*

Proof. Any $\mathscr{D}$-trivial automaton is $\mathscr{R}$-trivial. For every $\mathscr{R}$-trivial automaton $\mathscr{A} = (Q, \Sigma, \delta)$ the linear order $\leq$ can be defined on the set Q such that for any $q \in Q, a \in \Sigma$, $q.a \geq q$. Let $Q = \{1, \ldots, n\}$ and $1 < 2 < \ldots < n$, then for any word $w \in \Sigma^*$, $n.w = n$. Therefore, the state n is a zero state in the automaton $\mathscr{A}$. From Proposition 3.2, we have that the problem $SYN(\mathscr{D}$-$TRIV)$ can be solved in time $O(nk)$.

There is a state *end* in any partially monotonic DFA. The state *end* is always a zero state. Therefore, any partially monotonic automaton has a zero state. Hence, the problem $SYN(PMON)$ can be solved in time $O(nk)$. □

Proposition 3.4. *The problem $SYN(MON)$ can be solved in time $O(nk)$, where n is a number of states, k is an alphabet size.*

Proof. The automaton $\mathscr{A}$ is monotonic, therefore there is a linear order $\leq$ on the set Q such that for any $q_1, q_2 \in Q$ and $a \in \Sigma$ if $q_1 \leq q_2$ then $q_1.a \leq q_2.a$. Let $Q = \{1, \ldots, n\}$ and $1 < 2 < \ldots < n$, then for any word $w \in \Sigma^*$, $1.w \leq 2.w \leq \ldots \leq n.w$. Therefore, a word w is synchronizing if and only if $1.w = n.w$.

Let $p = \max\{q \in Q | \exists w \in \Sigma^*, 1.w = q\}$. Let $v \in \Sigma^*$ such that $1.v = p$. If $n.w > p$ for every word $w \in \Sigma^*$ then $n.w > p \geq 1.w$ by the choice of p, therefore the automaton $\mathscr{A}$ is not synchronizing. If there is a word $u \in \Sigma^*$ such that $n.u \leq p$, then the word uv resets the automaton $\mathscr{A}$ into the state p (because for any $q \in Q$, $q.uv \leq n.uv \leq p.v \leq p$, and $q.uv \geq 1.uv \geq 1.v = p$). Our algorithm finds words u and v. The letter $u[1]$ can be found in time $O(k)$, then the letter $u[2]$ can be found in time $O(k)$ and so on. Therefore, the word u can be found in time $O(nk)$. The word v can be found in the same way in time $O(nk)$. Hence, the problem $SYN(MON)$ can be solved in time $O(nk)$. □

4 Finding the length of the shortest reset words

We will use the classical NP-complete problem SAT to prove the NP-hardness of different problems.

Problem: *SAT*

Input: A set of Boolean variables $x_1, \ldots, x_n$ (the value of any variable can be 0 or 1) and a set of p Boolean expressions (which are called *clauses*) of kind $c_i(x_1, \ldots, x_n) = y_1^i \vee \ldots \vee y_{s_i}^i$, $i \in \{1, \ldots, p\}$ where $y_j^i \in \{x_\ell | \ell \in \{1, \ldots, n\}\} \cup \{\neg x_\ell | \ell \in \{1, \ldots, n\}\}$.

Question: Is there values for variables $x_1, \ldots, x_n \in \{0, 1\}$ such that for any $i \in \{1, \ldots, p\}$, $c_i(x_1, \ldots, x_n) = 1$?

In [3] and [4] the complexity of the problems $SYN(DFA, \leq L)$ and $SYN(DFA, = L)$ were considered. In the next proposition we formulate these results and recall a construction from the proof of these results (the construction is taken from [3]).

Proposition 4.1.

1. The problems $SYN(DFA, \leq L)$ and k-$SYN(DFA, \leq L)$ for $k \geq 2$ are NP-hard.

2. The problems $SYN(DFA, = L)$ and k-$SYN(DFA, = L)$ for $k \geq 2$ are NP-hard and co-NP-hard.

Proof. 1. The problems $SYN(DFA, \leq L)$ and k-$SYN(DFA, \leq L)$ for $k \geq 2$ belong to NP (see Proposition 3.1).

We reduce the problem SAT to the problem 2-$SYN(DFA, \leq L)$. Let the set of clauses $c_1(x_1, \ldots, x_n), \ldots, c_p(x_1, \ldots, x_n)$ over the variables $x_1, \ldots, x_n$ be an input of the problem SAT. We are going to construct a 2-letter automaton $\mathscr{A}_{dfa} = (Q, \{a,b\}, \delta)$ and a number L. Let $\Sigma = \{a,b\}$, $Q = \{q(m,i) | i \in \{1,\ldots,p\}, m \in \{1, n+1\} \cup \{end\}$. Let $i \in \{1,\ldots,p\}, m \in \{1,\ldots,n\}$, then

$$q(m,i).a = \begin{cases} end, & \text{if } x_m \text{ is contained in } c_i \text{ without } \neg \\ q(m+1,i), & \text{otherwise} \end{cases},$$

$$q(m,i).b = \begin{cases} end, & \text{if } \neg x_m \text{ is contained in } c_i \\ q(m+1,i), & \text{otherwise} \end{cases}.$$

For $i \in \{1,\ldots,p\}$, let $q(n+1,i).a = q(n+1,i).b = end$, $end.a = end.b = end$. We put $L = n$.

An example of the automaton $\mathscr{A}_{dfa}$ is represented by the Figure 1. The action of the letter a is denoted by solid lines. The action of the letter b is denoted by dotted lines. The figure contains three columns of states. In the i-th column there are states of kind $q(m,i)$ for fixed i. In any horizontal row there are states of kind $q(m,i)$ for some fixed m.

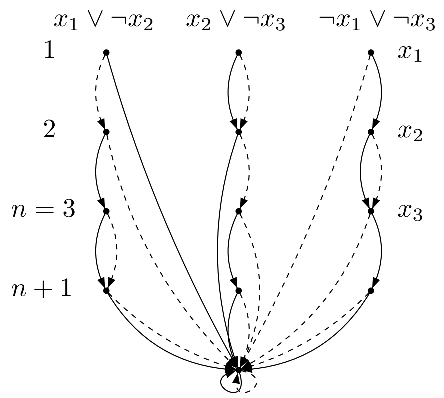

Figure 1: Automaton $\mathscr{A}_{dfa}$ for clauses $x_1 \vee \neg x_2$, $x_2 \vee \neg x_3$, $\neg x_1 \vee \neg x_3$

It is easy to see that the automaton $\mathscr{A}_{dfa}$ has polynomial size. It follows from [3] that there exists a reset word of length L for the automaton $\mathscr{A}_{dfa}$ if and only if there exist values of the variables $x_1,\ldots,x_p$ such that $c_1(x_1,\ldots,x_n) = \ldots = c_p(x_1,\ldots,x_n) = 1$. Therefore, the problem 2-$SYN(DFA, \leq L)$ is NP-complete.

Let $k > 2$. The problem 2-$SYN(DFA, \leq L)$ reduces to the problem k-$SYN(DFA, \leq L)$ with adding $k-2$ new letters. Any new letter acts identically on the set of states. Moreover, the problem 2-$SYN(DFA, \leq L)$ is a partial case of the problem $SYN(DFA, \leq L)$. Therefore, the problems $SYN(DFA, \leq L)$ and k-$SYN(DFA, \leq L)$ for $k \geq 2$ are NP-complete.

2. The proof of the NP-hardness of the problems $SYN(DFA, = L)$ and k-$SYN(DFA, = L)$ for $k \geq 2$ is the same. To prove the co-NP-hardness we should construct the automaton $\mathscr{A}_{dfa}$ again using the clauses $c_1,\ldots,c_p$. Now we put $L = n+1$. Then we ask: does the shortest reset word for the automaton $\mathscr{A}_{dfa}$

has length L. The length of the shortest reset word for the automaton $\mathscr{A}_{dfa}$ is less or equal to $L = n + 1$, because there are only $n + 1$ rows of states and the state *end* in the automaton $\mathscr{A}_{dfa}$ and every letter maps every state from the some row to a state from the next row or to the state *end*. The shortest reset word for the automaton $\mathscr{A}_{dfa}$ has length L if and only if there are no values for the variables $x_1, \dots, x_n$ such that $c_1(x_1, \dots, x_n) = \dots = c_p(x_1, \dots, x_n) = 1$. Therefore the problems $SYN(DFA, = L)$ and k-$SYN(DFA, = L)$ for $k \geq 2$ are co-NP-hard. □

Now we consider the same problems for some subclasses of automata. It is very easy to prove the NP-hardness for these subclasses because the automaton $\mathscr{A}_{dfa}$ belongs to each of these classes.

Proposition 4.2. *1. The problems $SYN(ZERO, \leq L)$, $SYN(APER, \leq L)$, $SYN(\mathscr{D}\text{-}TRIV, \leq L)$, $SYN(PMON, \leq L)$ are NP-complete together with the corresponding k-problems for $k \geq 2$.*

2. The problems $SYN(ZERO, = L)$, $SYN(APER, = L)$, $SYN(\mathscr{D}\text{-}TRIV, = L)$, $SYN(PMON, = L)$ are NP-hard and co-NP-hard together with the corresponding k-problems for $k \geq 2$.

Proof. Let us prove that the DFA $\mathscr{A}_{dfa} = (Q, \{a, b\}, \delta)$ is a $\mathscr{D}$-trivial, aperiodic, partially monotonic automaton with a zero state. Is can be easily proved that the state *end* is a zero state in the automaton $\mathscr{A}_{dfa}$. Therefore $\mathscr{A}_{dfa}$ is an automaton with a zero state.

Now we prove that the automaton $\mathscr{A}_{dfa}$ is partially monotonic. Let us define the linear order $\leq$ on the set $Q \backslash \{end\}$. We put $q(m_1, i_1) \leq q(m_2, i_2)$, if $i_1 < i_2$, or $i_1 = i_2$ and $m_1 \leq m_2$. It is easily proved that if $q, q' \in Q \backslash \{end\}$, $q \leq q'$ and $\alpha \in \{a, b\}$ such that $q.\alpha \neq end$ and $q'.\alpha \neq end$ then $q.\alpha \leq q'.\alpha$. Therefore $\mathscr{A}_{dfa}$ is partially monotonic.

Now we prove that the automaton $\mathscr{A}_{dfa}$ is $\mathscr{D}$-trivial. Let M be a transition monoid of the DFA $\mathscr{A}_{dfa}$. Let words $u, v \in \Sigma^*$ act on the state Q as different transitions. This means that there is a state $q \in Q$ such that $q.u \neq q.v$. It is easy to see that $q \neq end$, therefore $q = q(m, i)$ for some $m \in \{1, \dots, n\}, i \in \{1, \dots, p\}$. We put $q(n+2, i) = end$ for any $i \in \{1, \dots, p\}$. From the definition of the $\mathscr{A}_{dfa}$, we have that $q.u = q(m_1, i)$ and $q.v = q(m_2, i)$ for some $m_1 \neq m_2$. Let $m_1 < m_2$. Let us take $f, g \in M$. The state $q.fvg$ is equal to $q(m_3, i)$ for some $m_3 \geq m_2 > m_1$. Hence, $fvg \neq \lambda u \lambda$, for any $f, g \in M$, where λ is an empty word. Therefore, $MuM \neq MvM$ and $(u, v) \notin \mathscr{D}$. Thus, the automaton $\mathscr{A}_{dfa}$ is $\mathscr{D}$-trivial. Every $\mathscr{D}$-trivial automaton is aperiodic, therefore $\mathscr{A}_{dfa}$ is aperiodic. □

5 Commutative automata

Proposition 5.1. *The problem $SYN(COM)$ can be solved in time $O(kn \ln n)$, where $n = |Q|$, $k = |\Sigma|$.*

Proof. Let $\Sigma = \{a_1, \dots, a_k\}$. Every synchronizing commutative DFA $\mathscr{A}$ with n states can be synchronized by a word of length $n - 1$ (see [10]). Whence, there exists a reset word for the automaton $\mathscr{A}$, containing at most $n-1$ occurrences of the letter a_1, at most $n-1$ occurrences of the letter a_2, and so on. If we add one extra letter to a reset word then we obtain a reset word again. Moreover, the letters contained in a reset word can be permuted and the obtained word will be a reset word again. Therefore, the word $w = a_1^{n-1}a_2^{n-1}\dots a_k^{n-1}$ synchronizes every n-state automaton with alphabet $\{a_1,\dots,a_k\}$. It means that $|Q.w| = 1$ if and only if the automaton $\mathscr{A}$ is synchronizing. The value $Q.w$ can be calculated in time $O(kn\ln n)$, using the famous idea of the fast power calculation.

If the transformation defined by some word u is known, then for every set $S \subseteq Q$, the set $S.u$ can be calculated in time $O(n)$. Let $a \in \Sigma$. The transformation defined by the word a^{n-1} can be calculated in time $O(n\ln n)$ using the fast power calculation. Therefore, if we already know the set $Q.a_1^{n-1}a_2^{n-1}\dots a_{j-1}^{n-1}$, then the calculation of the set $Q.a_1^{n-1}a_2^{n-1}\dots a_{j-1}^{n-1}a_j^{n-1}$ takes time $O(n\ln n)$. Whence, the set $Q.w$ can be calculated in time $O(kn\ln n)$. When all the calculations are finished, we should look at the cardinality of the set $Q.w$. If $|Q.w| = 1$, then the automaton $\mathscr{A}$ is synchronizing, if $|Q.w| \neq 1$, then $\mathscr{A}$ is not synchronizing. The proposition is proved. □

Proposition 5.2. *Let $k \geq 1$, then the problems k-$SYN(COM, \leq L)$ and k-$SYN(COM, = L)$ can be solved in time $O(n^k \ln n)$, where $n = |Q|$.*

Proof. Every synchronizing commutative automaton $\mathscr{A} = (Q, \Sigma, \delta)$ with n states has a reset word of length at most $n-1$. Let $\Sigma = \{a_1,\dots,a_k\}$. If we take a shortest reset word and place its letters in the alphabetic order, we obtain a shortest reset word of kind $a_1^{s_1}a_2^{s_2}\dots a_k^{s_k}$. Therefore we can search a shortest reset word among the words of this kind and length at most $n-1$.

If the numbers $s_1,\dots,s_{k-1}$ are fixed, then the set $Q.a_1^{s_1}a_2^{s_2}\dots a_{k-1}^{s_{k-1}}$ can be found. For fixed numbers $s_1,\dots,s_{k-1}$, the number s_k can be found by the binary search as a minimal s such that $|Q.a_1^{s_1}a_2^{s_2}\dots a_{k-1}^{s_{k-1}}a_k^{s}| = 1$. Every set of the form $Q.a_1^{s_1}a_2^{s_2}\dots a_{k-1}^{s_{k-1}}$ and the sets $Q.a_1^{s_1}a_2^{s_2}\dots a_k^{s}$ (which appear during the binary search) can be calculated in time $O(kn\ln n)$ using the fast power calculation. The number k is fixed, hence $O(kn\ln n) = O(n\ln n)$. Therefore, the shortest reset word of language $a_1^{s_1}a_2^{s_2}\dots a_k^{*}$ can be found in time $O(n\ln n)$ for every vector $(s_1,\dots,s_{k-1})$ with $s_1+\ldots+s_{k-1} < n$. The number of these vectors is $\binom{n+k-2}{k-1} = O(n^{k-1})$. To find the answer, the length of the shortest reset word should be compared with L. The complete working time of the algorithm is $O(n^k\ln n)$. The proposition is proved. □

Proposition 5.3. *The problem $SYN(COM, \leq L)$ is NP-complete. The problem $SYN(COM, = L)$ is NP-hard and co-NP-hard.*

Proof. Let us consider the proof of NP-completeness of the problem $SYN(DFA, \leq L)$ from [4] (it is called there SYNCH WORD). We prove that the automaton from this proof is commutative. We reduce the problem SAT to the problem $SYN(COM, \leq L)$. Let the set of clauses $c_1(x_1,\dots,x_n),\dots,c_p(x_1,\dots,x_n)$ over the boolean variables $x_1,\dots,x_n$ be an input of the problem SAT. We are going to construct an automaton $\mathscr{A}_{com} = (Q, \Sigma, \delta)$ and a number L such that there exists a reset word of length L for the automaton $\mathscr{A}_{com}$ if and only if there exist values of the variables $x_1, \ldots, x_n$ such that $c_1(x_1, \ldots, x_n) = 1, \ldots, c_p(x_1, \ldots, x_n) = 1$.

Let $\mathscr{A}_{com} = (Q, \Sigma, \delta)$, where $Q = \{v_1, \ldots, v_n, q_1, \ldots, q_p, end\}$, $\Sigma = \{a_1, b_1, \ldots, a_n, b_n\}$, and the function δ is the following:

$$\text{For } m \in \{1, \ldots, n\},\ v_m.a_m = v_m.b_m = end,$$

$$\text{for } j \in \{1, \ldots, n\},\ j \neq m,\ v_m.a_j = v_m.b_j = v_m,$$

$$\text{For } i \in \{1, \ldots, p\},\ m \in \{1, \ldots, n\},$$
$$q_i.a_m = \begin{cases} end, & \text{if } x_m \text{ is contained in } c_i \text{ without } \neg \\ q_i, & \text{otherwise} \end{cases},$$

$$q_i.b_m = \begin{cases} end, & \text{if } \neg x_m \text{ is contained in } c_i \\ q_i, & \text{otherwise} \end{cases},$$

$$\text{For } m \in \{1, \ldots, n\},\ end.a_m = end.b_m = end.$$

An example of the automaton $\mathscr{A}_{com}$ for clauses $x_1 \vee \neg x_2$, $x_2 \vee \neg x_3$ and $\neg x_1 \vee \neg x_3$ is represented by Figure 2. We put $L = n$. It is evident that the size of the automaton $\mathscr{A}_{com}$ is polynomial with respect to the input size.

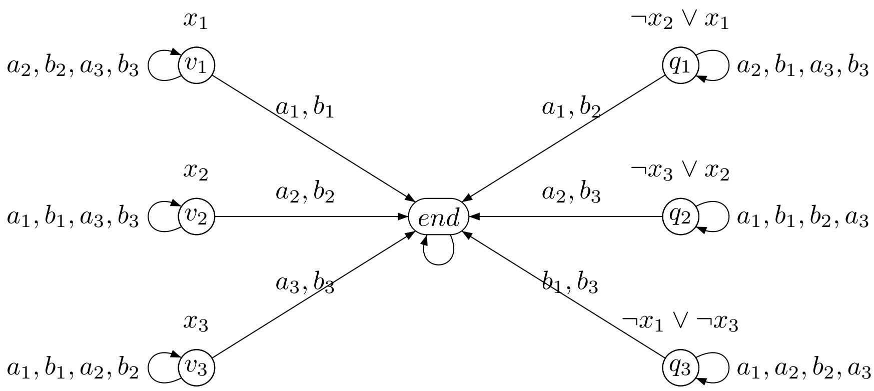

Figure 2: Automaton $\mathscr{A}_{com}$ for clauses $x_1 \vee \neg x_2$, $x_2 \vee \neg x_3$, $\neg x_1 \vee \neg x_3$

There is no letter which maps the state *end* to another state. Whence, the automaton $\mathscr{A}_{com}$ can be synchronized only to the state *end* and the states $v_1, \ldots, v_n$ should be mapped to the state *end*. Therefore, for every $m \in \{1, \ldots, n\}$ one of the letters a_m and b_m should be used. This means that there is no reset word of length less than n.

Let there exist values of the variables $x_1, \dots, x_n$ such that $c_1(x_1, \dots, x_n) = 1, \dots, c_p(x_1, \dots, x_n) = 1$. Consider the word w of length n where

$$w[m] = \begin{cases} a_m, & \text{if } x_m = 1 \\ b_m, & \text{if } x_m = 0 \end{cases} \quad \text{for } m \in \{1, \dots, n\}.$$

For every $i \in \{1, \dots, p\}$, $c_i(x_1, \dots, x_n) = 1$. Therefore there exists a number m such that x_m without $\neg$ is contained in c_i and $x_m = 1$, in this case $w[m] = a_m$; or $\neg x_m$ is contained in c_i and $x_m = 0$, in this case $w[m] = b_m$. In both cases $q_i.w[m] = end$. Therefore $Q.w = \{end\}$.

Assume that there exists a reset word w of length n for the automaton $\mathscr{A}_{com}$. For each $m \in \{1, \dots, n\}$ there exists one and only one of letters a_m and b_m in the word w. We put

$$x_m = \begin{cases} 1, & \text{if } a_m \text{ is contained in } w \\ 0, & \text{if } b_m \text{ is contained in } w \end{cases}.$$

If the letter a_m maps the state q_i to the state *end*, then the value $x_m = 1$ provides $c_i = 1$. If the letter b_m maps the state q_i to the state *end*, then the value $x_m = 0$ provides $c_i = 1$. Thus $c_1(x_1, \dots, x_n) = 1, \dots, c_p(x_1, \dots, x_n) = 1$.

Now we prove that the automaton $\mathscr{A}_{com}$ is commutative. Let $\alpha, \beta \in \Sigma$.

- Let $m \in \{1, \dots, n\}$. If $\alpha \notin \{a_m, b_m\}$ and $\beta \notin \{a_m, b_m\}$, then $v_m.\alpha\beta = v_m.\beta\alpha = v_m$, else $v_m.\alpha\beta = v_m.\beta\alpha = end$.
- Let $i \in \{1, \dots, p\}$. If $q_i.\alpha \neq end$ and $q_i..\beta \neq end$, then $q_i.\alpha\beta = q_i.\beta\alpha = q_i$, else $q_i.\alpha\beta = q_i.\beta\alpha = end$.
- $end.\alpha\beta = end.\beta\alpha = end$.

Whence the automaton $\mathscr{A}_{com}$ is commutative.

For any synchronizing commutative automaton $\mathscr{A} = (Q, \Sigma, \delta)$ the length of the shortest reset word does not exceed $|Q| - 1$ (see [10]). This means that if $L \geq |Q| - 1$, then for solving problem $SYN(COM, \leq L)$ it is enough to check whether the automaton is synchronizing. It can be done in polynomial time. Thus, the problem $SYN(COM, \leq L)$ is contained in the class NP.

The proof of the NP-hardness of the problem $SYN(COM, = L)$ is the same. Our proof of the co-NP-hardness and the proof from [4] are different (the proof from [4] is more complicated). To prove the co-NP-hardness we should construct the automaton $\mathscr{A}_{com}$ again using the clauses $c_1, \dots, c_p$. Now we put $L = n+1$. Then we ask the question: is it correct that a shortest reset word for the automaton $\mathscr{A}_{com}$ has length L. The shortest reset word for the automaton $\mathscr{A}_{com}$ has length L if and only if there are no values for the variables $x_1, \dots, x_n$ such that $c_1(x_1, \dots, x_n) = \dots = c_p(x_1, \dots, x_n) = 1$. Therefore the problem $SYN(COM, = L)$ is co-NP-hard. The proposition is proved. □

Thus, the problems k-$SYN(COM, \leq L)$ and k-$SYN(COM, = L)$ for a fixed number $k \geq 1$ can be solved in polynomial time, but at the same time the problems $SYN(COM, \leq L)$ and $SYN(COM, = L)$ are hard.

6 Automata with simple idempotents

A construction similar to the construction of the automaton $\mathscr{A}_{com}$ can be used to estimate the computational complexity of problems stated for the class of DFA with simple idempotents. As for commutative automata, the complexity of the problems $SYN(SIMPID, \leq L)$ and $SYN(SIMPID, = L)$ differ from the corresponding 2-problems.

Proposition 6.1. *The problem $SYN(SIMPID, \leq L)$ is NP-complete. The problem $SYN(SIMPID, = L)$ is NP-hard and co-NP-hard.*

Proof. We reduce the problem SAT to the problem $SYN(SIMPID, \leq L)$.

Let the input of the problem SAT is a set of clauses $c_1(x_1, \dots, x_n), \dots, c_p(x_1, \dots, x_n)$ over the variables $x_1, \dots, x_n$. We are going to construct an automaton $\mathscr{A}_{sid} = (Q, \Sigma, \delta)$ and a number L such that there exists a reset word of length L for the automaton $\mathscr{A}_{sid}$ if and only if there exist values of the variables $x_1, \dots, x_p$ such that $c_1(x_1, \dots, x_n) = 1, \dots, c_p(x_1, \dots, x_n) = 1$.

Let $\mathscr{A}_{sid} = (Q, \Sigma, \delta)$, where $Q = \{v_1, \dots, v_n, q_1, \dots, q_p, u, r_1, \dots, r_p, end\}$, $\Sigma = \{a_1, b_1, \dots, a_n, b_n, z, y_1, \dots, y_p\}$, and the function δ is the following:

For $m \in \{1, \dots, n\}$, $v_m.a_m = v_m.b_m = u$, $u.a_m = u.b_m = v_m$

For $j \in \{1, \dots, n\}$, $j \neq m$, $v_m.a_j = v_m.b_j = v_m$

For $i \in \{1, \dots, p\}$, $m \in \{1, \dots, n\}$,

if x_m is contained in c_i without $\neg$, then $q_i.a_m = r_i$, $r_i.a_m = q_i$

else $q_i.a_m = q_i$, $r_i.a_m = r_i$

if $\neg x_m$ is contained in c_i, then $q_i.b_m = r_i$, $r_i.b_m = q_i$

else $q_i.b_m = q_i$, $r_i.b_m = r_i$

For $q \in Q$, $q.z = \begin{cases} end, & \text{if } q = u \\ q, & \text{otherwise} \end{cases}$

For $i \in \{1, \dots, p\}$, and $q \in Q$, $q.y_i = \begin{cases} end, & \text{if } q = r_i \\ q, & \text{otherwise} \end{cases}$

For $m \in \{1, \dots, n\}$, $i \in \{1, \dots, p\}$, $end.a_m = end.b_m = end.y_i = end.z = end$.

An example of the automaton $\mathscr{A}_{sid}$ for clauses $x_1 \vee \neg x_2$, $x_2 \vee \neg x_3$ and $\neg x_1 \vee \neg x_3$ is represented by Figure 3. We put $L = 2n + 2p + 1$. It is obvious that the size of the automaton $\mathscr{A}_{sid}$ is a function with respect to the input size. The letters $a_1, \dots, a_n, b_1, \dots, b_n$ are permutations of the set Q, the letters $z, y_1, \dots, y_p$ are simple idempotents. Whence $\mathscr{A}_{sid}$ is an automaton with simple idempotents.

The state *end* can be mapped only to the state *end*. Therefore the automaton $\mathscr{A}_{sid}$ can be synchronized only to the state *end*. The automaton $\mathscr{A}_{sid}$ contains $n + 2p + 2$ states. At most one state (except *end*) can be mapped to the state *end* under an action of one letter. The only letters that map some states to the state *end* are $z, y_1, \dots, y_p$. This means that every reset word should contain at least $n + 2p + 1$ letters from the set $\{z, y_1, \dots, y_p\}$. Furthermore, a word maps the states

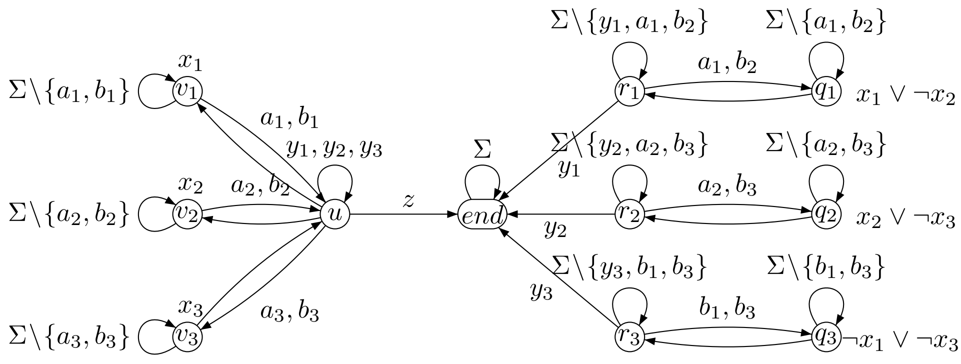

Figure 3: Automaton $\mathscr{A}_{sid}$ for clauses $x_1 \vee \neg x_2$, $x_2 \vee \neg x_3$, $\neg x_1 \vee \neg x_3$

$v_1, \dots, v_n$ to the state *end* only if it contains a letter from every pair $\{a_m, b_m\}$ for $m \in \{1, \dots, n\}$. Therefore there is no reset word of length less than $2n + 2p + 1$.

Let there exist values of the variables $x_1, \dots, x_n$ such that $c_1(x_1, \dots, x_n) = \dots = c_p(x_1, \dots, x_n) = 1$. We construct a word $w \in \Sigma^*$ of length $2n + 2p + 1$. We put $w[1, p+1] = zy_1 \dots y_p$. The states $u, r_1, \dots, r_p$ map to the state *end* under the action of the word $w[1, p+1]$. We put $w[p+2] = \begin{cases} a_1, & \text{if } x_1 = 1 \\ b_1, & \text{if } x_1 = 0 \end{cases}$ and $w[p+3] = z$. In this case the state v_1 also maps to the state *end*. Let after the variable x_1 get value, t_1 of clauses $c_{i_1(1)}, \dots, c_{i_1(t_1)}$ become true (=1 independently of the values of $x_2, \dots, x_n$). Then we put $w[p+4, p+t_1+3] = y_{i_1(1)} \cdots y_{i_1(t_1)}$, and the states $q_{i_1(1)}, \dots, q_{i_1(t_1)}$ map to the state *end*. In the same way, we put $w[p+t_1+4] = \begin{cases} a_2, & \text{if } x_2 = 1 \\ b_2, & \text{if } x_2 = 0 \end{cases}$ and $w[p+t_1+5] = z$. Let then the variable x_2 get value, t_2 of the clauses $c_{i_2(1)}, \dots, c_{i_2(t_2)}$ become true (we consider only the clauses been false before the variable x_2 get a value). We put $w[p+t_1+6, p+t_1+t_2+5] = y_{i_2(1)} \cdots y_{i_2(t_2)}$. We repeat this process for all variables. For every variable x_i a number t_i is defined. All the clauses becomes true after all the variables get their values. Therefore $t_1 + \dots + t_n = p$, and the length of the word w is equal to $2n + 2p + 1$. Furthermore, $Q.w = end$.

Assume that there exists a reset word w of length $2n + 2p + 1$ for the automaton $\mathscr{A}_{sid}$. If w is a reset word, then it contains at least $n + 2p + 1$ letters from the set $\{z, y_1, \dots, y_p\}$ and just one letter from every pair $\{a_m, b_m\}$. The states $q_1, \dots, q_p$ map to the states $r_1, \dots, r_p$ under the action of word w, because for each $m \in \{1, \dots, n\}$ the letter a_m or the letter b_m is contained in w. We put $x_m = \begin{cases} 1, & \text{if } a_m \text{ is contained in } w \\ 0, & \text{if } b_m \text{ is contained in } w \end{cases}$, $m \in \{1, \dots, n\}$. If the letter a_m maps the state q_i to r_i, then the equality $x_m = 1$ provides $c_i(x_1, \dots, x_n) = 1$; if the letter b_m maps the state q_i to r_i, then the equality $x_m = 0$ provides $c_i(x_1, \dots, x_n) = 1$. Thus all the clauses are true.

A length of every reset word for the given synchronizing automaton $\mathscr{A} = (Q, \Sigma, \delta)$ with simple idempotents does no exceed $2(|Q|-1)^2$ (see [11]). It means that if $L \geq 2(|Q|-1)^2$, then it is enough to check whether the automaton is synchronizing. It can be done in polynomial time. We can check in polynomial time whether the word $w \in \Sigma^*$ of length L is a reset word for the automaton $\mathscr{A}$. Thus, the problem $SYN(SIMPID, \leq L)$ is in the class NP.

The proof of the NP-hardness of the problem $SYN(SIMPID, = L)$ is the same. To prove the co-NP-hardness we should construct the automaton $\mathscr{A}_{sid}$ again using the clauses $c_1, \dots, c_p$ and put $L = 2n + 2p + 2$. The shortest reset word for the automaton $\mathscr{A}_{sid}$ has length L if and only if there are no values for the variables $x_1, \dots, x_n$ such that $c_1(x_1, \dots, x_n) = \dots = c_p(x_1, \dots, x_n) = 1$. Therefore, $SYN(SIMPID, = L)$ is co-NP-hard. The proposition is proved. □

Proposition 6.2. *The problems 2-$SYN(SIMPID)$, 2-$SYN(SIMPID, \leq L)$ and 2-$SYN(SIMPID, = L)$ can be solved in time $O(n)$, where $n = |Q|$.*

Proof. Let us consider the automaton $\mathscr{A} = (Q, \{a, b\}, \delta)$ and let $|Q| = n$. If a and b are permutations, then for $n > 1$ the automaton $\mathscr{A}$ is not synchronizing. If a and b are simple idempotents, then for $n > 3$ the automaton $\mathscr{A}$ is not synchronizing too. All variants of the automaton for $n \leq 3$ can be easily considered in a constant time. Therefore, we can assume that a is a simple idempotent and b is a permutation.

Let $q_1.a = q_2 \neq q_1$ for $q_1, q_2 \in Q$. The permutation b can be represented as a product of simple cycles. If the permutation b consists of more than two cycles, then the automaton $\mathscr{A}$ is not synchronizing, because the letter a can merge states from at most two cycles. Let b consists of two cycles. In this case the states q_1 and q_2 should contain in different cycles C_1 and C_2. Moreover, if the cycle C_2 consists of more then one state, then the automaton $\mathscr{A}$ is not synchronizing, because different states from the cycle C_2 cannot be merged. In this case the automaton $\mathscr{A}$ looks like the automaton represented by Figure 4.

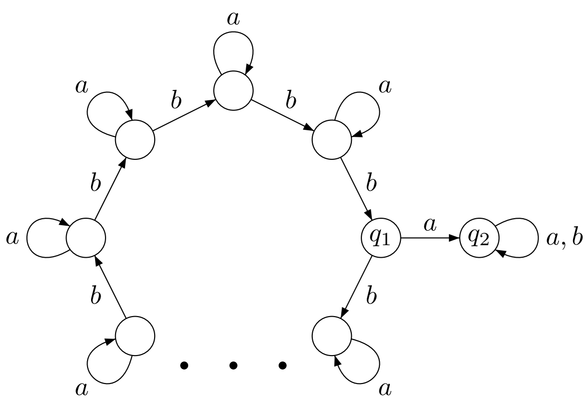

Figure 4: Automaton with two cycles

It is not difficult to check that the word $a(ba)^{n-2}$ is a shortest reset word for $\mathscr{A}$ in this case.

Let the letter b act on the set Q as a single cycle. Let $Q = \{1, \dots, n\}$, $q_1 = 1$, $q_2 = p$ for some $p \in \{1, \dots, n\}$, and $n.b = n-1, \dots, 2.b = 1, 1.b = n$. Such an automaton is represented by Figure 5.

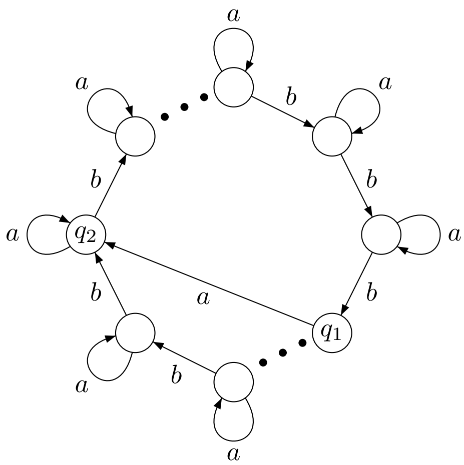

Figure 5: Automaton with one cycle

For $q_2 = n$, the automaton $\mathscr{A}$ is a Černý automaton with n states described in [1]. Is it not difficult to obtain that if $\gcd(n,p) > 1$ then the automaton $\mathscr{A}$ is not synchronizing. If $\gcd(n,p) = 1$, then the word $a(b^{p-1}a)^{n-2}$ is a shortest reset word for the automaton $\mathscr{A}$. The proof of this fact is very similar to the proof from [1] (in [1] it was proved that the word $a(b^{n-1}a)^{n-2}$ is a shortest reset word for the Černý automaton). We skip this proof here.

Let an automaton $\mathscr{A}$ be given. There exists a very simple algorithm taking time $O(n)$ and checking whether the automaton $\mathscr{A}$ can be represented by Figure 4 or by Figure 5. This algorithm finds the states q_1 and q_2, calculates the length of cycles of permutation b. If the states q_1 and q_2 are contained in one cycle, then the algorithm also finds the distance between q_1 and q_2 along the cycle. Finally, the algorithm compares one of the values $a(ba)^{n-2}$ (for the case of two cycles) and $a(b^{p-1}a)^{n-2}$ (for the case of one cycle) with the number L. The proposition is proved.

$\square$

Thus, the problems 2-$SYN(SIMPID, \leq L)$ and 2-$SYN(SIMPID, = L)$ can be solved in polynomial time, but at the same time the problems $SYN(SIMPID, \leq L)$ and $SYN(SIMPID, = L)$ are hard. The question about the computational complexity of the problems k-$SYN(SIMPID, \leq L)$ and k-$SYN(SIMPID, = L)$ for $k > 2$ is open.

7 Finding the length of shortest compressing words

It was proved in [2] that the shortest synchronizing word for a given k-letter cyclically monotonic automaton with n states can be found in time $O(n^2k)$. Every monotonic automaton is cyclically monotonic too. Hence the problems $SYN(MON, \leq L)$, k-$SYN(MON, \leq L)$, $SYN(MON, = L)$ and k-$SYN(MON, = L)$ for $k \geq 1$ can be solved in time $O(n^2k)$, i.e. in polynomial time. But there are problems concerning synchronization of the monotonic DFA which cannot be solved in polynomial time (if $P \neq NP$).

In the proof of the next proposition we use a token model of synchronization. Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a DFA and $w \in \Sigma^*$. Suppose that at the beginning there is a token on any state from Q. We apply letters of the word w step by step. The action of the letter $a \in \Sigma$ moves the token from the state $q \in Q$ to the state $\delta(q, a)$. If two tokens arrive at one state, then one of them must be removed. If after the action of the word w there is only one token on the set Q, then the word w is a reset word. If after the action of the word w there are M tokens on the states of the set Q, then the word w compresses the automaton $\mathscr{A}$ to M states.

Proposition 7.1. *1. The problems $COMP(MON, M, \leq L)$ and k-$COMP(MON, M, \leq L)$ for $k \geq 2$ are NP-complete.*

2. The problems $COMP(MON, M, = L)$ and k-$COMP(MON, M, = L)$ for $k \geq 2$ are NP-hard and co-NP-hard.

Proof. It can be checked in polynomial time, whenever a given word compresses a given DFA to M states. Hence, the problem $COMP(MON, M, \leq L)$ belongs to NP. If we prove the NP-hardness of the problem 2-$COMP(MON, M, \leq L)$ then the NP-completeness of the problems $COMP(MON, M, \leq L)$ and k-$COMP(MON, M, \leq L)$ for $k \geq 2$ will be proved as well.

We reduce the problem SAT to the problem 2-$COMP(MON, M, \leq L)$. Let the input of the problem SAT is a set of clauses $c_1(x_1, \dots, x_n), \dots, c_p(x_1, \dots, x_n)$ over the variables $x_1, \dots, x_n$. We are going to construct a 2-letter automaton $\mathscr{A}_{mon} = (Q, \{a, b\}, \delta)$ and the numbers M and L such that there exists a word of length L compressing the automaton $\mathscr{A}_{mon}$ to the M states if and only if there exist values of variables $x_1, \dots, x_n$ such that $c_1(x_1, \dots, x_n) = \dots = c_p(x_1, \dots, x_n) = 1$.

Let $\Sigma = \{a, b\}$, $Q = \{q(m', i) | i \in \{1, \dots, p\}, m' \in \{1, 2n + 2\}\}$. Let $i \in \{1, \dots, p\}, m \in \{1, \dots, n\}$, then

$$q(2m-1, i).a = \begin{cases} q(2m+2, i), & \text{if } x_m \text{ is contained in } c_i \text{ without } \neg \\ q(2m+1, i), & \text{otherwise} \end{cases}$$

$$q(2m-1, i).b = \begin{cases} q(2m+2, i), & \text{if } \neg x_m \text{ is contained in } c_i \\ q(2m+1, i), & \text{otherwise} \end{cases}$$

$$q(2m, i).a = q(2m+2, i), q(2m, i).b = q(2m+2, i).$$

$$q(2n+1, i).a = q(2n+1, i).b = q(2n+2, i).a = q(2n+2, i).b = q(2n+2, i).$$

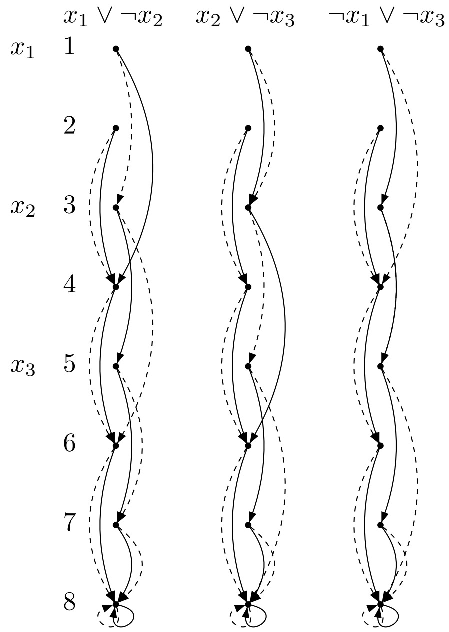

Figure 6: Automaton $\mathscr{A}_{mon}$ for clauses $x_1 \vee \neg x_2$, $x_2 \vee \neg x_3$, $\neg x_1 \vee \neg x_3$

We also put $M = p$, $L = n$. An example of the automaton $\mathscr{A}_{mon}$ is represented by Figure 6. The action of the letter a is denoted with solid lines. The action of the letter b is denoted with dotted lines. The figure contains three columns of states. In the i-th column there are states of kind $q(m, i)$ for fixed i. In any horizontal row there are states $q(m, i)$ for some fixed m.

We define a linear order $\leq$ on the set Q. We put

$$q(m_1, i_1) \leq q(m_2, i_2), \text{ if } i_1 < i_2, \text{ or } i_1 = i_2 \text{ and } m_1 \leq m_2.$$

It is not difficult to verify that for each letter $a \in \Sigma$ the transformation $\delta(\_, a)$ of the set Q preserves $\leq$. Thus, the automaton $\mathscr{A}_{mon}$ is monotonic. The size of the automaton $\mathscr{A}_{mon}$ is a polynomial in common number of clauses and variables.

The set Q can be represented as a table with p columns and $2n + 2$ rows. In the i-th column K_i there are states of kind $q(*, i)$, in the m-th row R_m there are states of kind $q(m, *)$. Suppose that there is a token on every state of the set Q at the start of the synchronization. If some word compresses the automaton $\mathscr{A}_{mon}$ to p states, then it moves all tokens to the states $q(2n+2, 1), \dots, q(2n+2, p)$, i.e. to the $2n+2$-th row. Let some token be in the row R_m, $m \in \{1, \cdots, 2n\}$. This token can be moved to the row R_{m+2} or to the row R_{m+3} under the action of some letter. Thus, if $q \in R_2 \cup \ldots \cup R_{2n+2}$ and $w \in \Sigma^*$, $|w| = n$, then $q.w \in R_{2n+2}$. Therefore, a word w of length n compresses the automaton $\mathscr{A}_{mon}$ if and only if $R_1.w = R_{2n+2}$.

Let there exist values of the variables $x_1, \dots, x_n$ such that $c_1(x_1, \dots, x_n) = \ldots = c_p(x_1, \dots, x_n) = 1$. Consider a word w of length n such that

$w[m] = \begin{cases} a, & \text{if } x_m = 1 \\ b, & \text{if } x_m = 0 \end{cases}$ for $m \in \{1, \ldots, n\}$. Let $i \in \{1, \ldots, p\}$. Let m be a minimal number from the set $\{1, \ldots, n\}$ such that $x_m = 1$ and x_m is contained in c_i without $\neg$, or $x_m = 0$ and $\neg x_m$ is contained in c_i. Then

$$q(1,i).w[1] = q(3,i), q(3,i).w[2] = q(5,i) \ldots q(2m-3,i).w[m-1] = q(2m-1,i)$$

$$q(2m-1,i).w[m] = q(2m+2,i)$$

$$q(2m+2,i).w[m+1] = q(2m+4,i), \ldots q(i,2n).w[n] = q(i,2n+2)$$

Therefore, $R_1.w = R_{2n+2}$ and $|Q.w| = p = M$.

Let there exist a word $w \in \Sigma^*$ of length n such that $|Q.w| = p$. In this case $Q.w = \{q(2n+2,1), \ldots, q(2n+2,p)\}$. We put $x_m = \begin{cases} 1, & \text{if } w[m] = a \\ 0, & \text{if } w[m] = b \end{cases}$. Let $i \in \{1, \ldots, p\}$, then $q(1,i).w = q(2n+2,i)$. Let us consider a token from the state $q(1,i)$. If each letter of the word w moves this token from row with number j to row with number $j+2$, then after applying the word w the token cannot be on the state $q(2n+2,i)$. Therefore, there is an $m \in \{1, \ldots, n\}$ such that $q(2m-1,i).w[m] = q(2m+2,i)$. This holds only if the variable x_m is contained in c_i without $\neg$ and $x_m = 1$; or $\neg x_m$ is contained in c_i and $x_m = 0$. In this case $c_i(x_1, \ldots, x_n) = 1$.

2. The statement can be proved using the idea of the proof of the NP and co-NP-hardness of the problem 2-$SYN(DFA)$. But in this case idea should be applied to the automaton $\mathscr{A}_{mon}$. □

Acknowledgement

The author is grateful to his supervisor Dr. D.S. Ananichev for suggesting the research problem and for his valuable help. The author acknowledges support from the Federal Education Agency of Russia, grant 2.1.1/3537, and from the Russian Foundation for Basic Research, grant 09-01-12142.

References

[1] Černý, J. Poznámka k homogénnym eksperimentom s konecnými avtomatami. *Mat.-Fyz. Cas. Slovensk. Akad. Vied.*, 14:208–216, 1964 [in Slovak].

[2] Eppstein, D. Reset sequences for monotonic automata. *SIAM J. Comput.*, 19:500–510, 1990.

[3] Salomaa, A. Composition sequences for functions over a finite domain. *Theor. Comput. Sci.*, 292:263–281, 2003.

[4] Samotij, W. A note on the complexity of the problem of finding shortest synchronizing words. Palermo, AUTOMATA 2007.

[5] Martugin, P. A series of slowly synchronizable automata with a zero state over a small alphabet. *Inf. and Comput.*, 206:1197–1203, 2008.

[6] Ananichev, D. S. and Volkov, M. V. Synchronizing monotonic automata. *Theoret. Comput. Sci.*, 327:225–239, 2004.

[7] Ananichev, D. S. and Volkov, M. V. Synchronizing generalized monotonic automata. *Theoret. Comput. Sci.*, 330:3–13, 2005.

[8] Ananichev, D. S. The mortality threshold for partially monotonic automata. In de Felice, C. and Restivo, A., editors, *Developments in Language Theory, 9th International Conference, DLT 2005, Palermo, Italy, July 4-8, 2005, Proceedings.*, Lecture Notes in Computer Science 3572, pages 112–121. Springer, 2005.

[9] Trahtman, A. N. The Černý conjecture for aperiodic automata. *Discr. Math. and Theoret. Comput. Sci.*, 9(2):3–10, 2007.

[10] Rystsov, I. C. Reset words for commutative and solvable automata. *Theoret. Comput. Sci.*, 172:273–279, 1997.

[11] Rystsov, I. C. Reset words for automata with simple idempotents. *Kibernetika i Sistemnyj Analiz*, 3:32–39, 2000, [in Russian; English translation: *Cybernetics and System Analysis*, 36:339–344, 2000]

[12] Kari, J. A counter example to a conjecture concerning synchronizing words in finite automata. *EATCS Bull.*, 73:146, 2001.

[13] Kari, J. Synchronizing finite automata on Eulerian digraphs. *Theoret. Comput. Sci.*, 295:223–232, 2003.

[14] Dubuc, L. Sur les automates circulaires et la conjecture de Černý. *RAIRO Inform. Theor. Appl.*, 32:21–34, 1998 [in French].

Received 30th September 2008