World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:7, 2007

New Algorithms for Finding Short Reset Sequences in Synchronizing Automata

Adam Roman

***Abstract*—**Finding synchronizing sequences for the finite automata is a very important problem in many practical applications (part orienters in industry, reset problem in biocomputing theory, network issues etc). Problem of finding the shortest synchronizing sequence is NP-hard, so polynomial algorithms probably can work only as heuristic ones. In this paper we propose two versions of polynomial algorithms which work better than well-known Eppstein's Greedy and Cycle algorithms.

***Keywords*—**Synchronizing words, reset sequences, Černý Conjecture.

I. Introduction

LET us first define the finite automaton without initial or final states as a triple **A** = (Q, A, δ), where Q is the finite set of states, A is the finite alphabet, and δ is the transition function from Q×A into Q. The free monoid A^* is the set of all words over A. It contains the empty word ε. We use the notation $|w|$ for the length of w, i.e. the number of letters in w. The length of an empty word is 0. If u and v are two words, then $u.v$ is a word uv which is the concatenation of them. If $w=w_1w_2...w_k$, we say that w_i is a subword of w. We extend the transition function on the whole free monoid A^* in a natural way:

$$\delta(q,aw)=\delta(\delta(q,a),w),\ \ q\in Q, a\in A, w\in A^* \qquad (1)$$

Let **A** be the finite automaton. We say that the word w synchronizes **A** iff

$$\exists w\in A^* : \forall p,q\in Q\ \ \delta(p,w)=\delta(q,w). \qquad (2)$$

If such a word exists for **A,** we say that **A** is synchronizing. If w is the synchronizing word for **A** and there is no shorter one, we say that w is the minimal synchronizing word (MSW) for **A**. It is easy to find a synchronizing word for a given automaton, but the problem of finding *minimal* synchronizing word is NP-complete [1].

In 1964 Černý stated the following conjecture:

Conjecture (Černý). If the n-state automaton is synchronizing, then the length of its MSW is not greater than $(n-1)^2$.

The Conjecture turned out to be true for some special cases ([1], [4], [7], [8]) but in general the problem is still open.

The problem of finding the minimal synchronizing word for a given automaton seems to be just a nice combinatoric puzzle, but in fact there is a deep connection between the problem and applications (see for example the pioneer works of Natarajan [2], [3]). Synchronizing sequences are used in part orienters (see the very good example in [4]), biocomputing ([5], [6]), network theory etc. Kari [7] gives other examples for possible applications: simple error recovery in finite automata, leader identification in processor networks, road map problem.

The paper is organized as follows: in Section II we introduce the notion of a pair automaton. This construction will be used in our algorithms. In Section III we present the well-known Eppstein's Greedy and Cycle algorithms, and then we introduce two new algorithms: SynchroP and SynchroPL. Section IV includes the numerical results for all four algorithms. Then, in Section V we present the example of SynchroPL action for particular automaton. Finally, in Section VI we discuss obtained results.

II. The Pair Automaton

Let **A**=(Q, A, δ) be the finite automaton. For a given automaton **A** we define the pair automaton $\mathbf{A^2}$ as a triple (Q', A', δ') where:

- Q' is a set of states. Each element of Q' is either the 2-element subset of Q: $\{p,q\}$, where p and q belong to Q ($p\neq q$) or a special state q_0.
- A'=A
- δ' is a transition function defined in the following way:

$$\delta'(\{p,q\},a)=\begin{cases} q_0 & if\ \ \delta(p,a)=\delta(q,a), \\ \{\delta(p,a),\delta(q,a)\} & otherwise, \end{cases} \qquad (3)$$

where $\{p, q\} \in$ Q', $a \in$ A'. We also define $\delta(q_0, a)=q_0$ for all $a \in$ A'. The following Lemma establishes the relation between the synchronization of **A** and some property of its pair automaton.

Lemma 1. Let **A** be the finite automaton and $\mathbf{A^2}$ its pair automaton. Then **A** is synchronizing iff the following

Manuscript received July 07, 2005. This work was supported in part by the State Committee for Scientific Research under Grant 3 T11C 010 27.

Adam Roman is with the Institute of Computer Science, Jagiellonian University, 30-072 Cracow, Poland (email: roman@ii.uj.edu.pl).

condition holds:

$$\forall s \in Q' \ \exists w \in A'^{*} : \delta'(s,w) = q_0. \quad (4)$$

The proof comes directly from the definition of synchronizing word.

We also define $d(q) = \min_{w \in A^*}\{|w| : \delta(q,w) = q_0\}$ as the minimal distance in the pair automaton from q to q_0. This distance is defined in the terms of the proper word's length. By W*(q)* we denote the word which realizes this minimum.

If for a given word w we have $\delta(Q,w)=P$, then the states which at this moment belong to P are called the active states. States from Q\P are called inactive states. Of course, at the beginning of the synchronization process each state is active. If w is the synchronizing word for A and $\delta(Q,w)=q_0$, only q_0 is an active state at the end of the synchronization process.

III. Algorithms

Now we will describe two well-known heuristic algorithms which find possibly the shortest synchronizing word for a given automaton **A**. They were introduced by Eppstein in [1].

The Greedy Algorithm finds a pair of states such that the word synchronizing them is the shortest one and transforms all active states with this word. The Cycle Algorithm does the same, but one state in the pair (in which the synchronization takes place) is fixed.

```
procedure Greedy
INPUT: automaton A
OUTPUT: synchronizing word w for A
1. A2=(Q',A,δ') ← A²(A);
2. w←ε;
3. while (Q' != {q0}) do {
4.    find p: d(p)=min{d(q), q in Q'};
5.    w←w.W(p);
6.    Q'← δ'(Q',W(p)); }
7. return w;
```

In the case of Cycle Algorithm we need to modify the definition of Q' in pair automaton because now we have to distinguish the "single" states of $\mathbf{A}^2$. In the pair automaton there is only one such state – q_0. Let us define the new set of states Q'' in the extended pair automaton as Q'\{q_0} ∪ Q and redefine the transition function in a following way:

$$\delta'(\{p,q\},a) = \begin{cases} \delta(p,a) \ \ if \ \ \delta(p,a) = \delta(q,a), \\ \{\delta(p,a), \delta(q,a)\} \ \ otherwise. \end{cases} \quad (5)$$

We also redefine the *n(q)*. |P| denotes the cardinality of the set P. We put $d(q) = \min_{w \in A^*}\{|w| : |\delta(q,w)| = 1\}$

```
procedure Cycle
INPUT: automaton A
OUTPUT: synchronizing word w for A
1. A2=(Q'',A,δ') ← A²(A);
2. w←ε;
3. find r: d({r,_})=1;
4. while |Q''| != 1 do {
5.    find p={r,_}: d(p)=min{d(q), q in
   Q''};
6.    w←w.W(p); Q''← δ'(Q'',W(p));
7.    r←p.W(p); }
8.  return w;
```

We will now introduce two new algorithms based on *d* function for the pair automata. In each step the algorithms will find the sequence which synchronizes at least two states of the pair automaton, but now we will also look one step forward, that is – we will be checking how the choice of a particular state q (and, automatically, the synchronizing subword W(q)) in the pair automaton will affect the positions of active states. The estimation of how "good" is a given distribution of active states among all states of the pair automaton requires introducing a measure for it. We will use the *d* function and the following heuristics:

Suppose that at some stage of our procedure p is an active state. We choose a word w=W(q) and we look how the value of *d* for p changes before and after the word w is applied. The considered difference is defined as follows:

$$\Delta_q(p,w) = \begin{cases} d(\delta(p,w)) - d(p) \ \ if \ \ p \neq q \\ 0 \ \ if \ \ p = q. \end{cases} \quad (6)$$

For a given word w=W(q) we can compute Δ_q for all active states and summarize this values:

$$\Phi_1(w,q) = \sum_{p \in Act} \Delta_q(p,w), \quad (7)$$

where *Act* is the set of all active states and q is the currently considered state. We compute Φ_1 for all words W*(r)*, where r is an active state. We also assume that if for two words u and v $\Phi_1(u) < \Phi_1(v)$, then it is better to apply u than v at the stage because after applying u to all active states, all transformed active states are closer to the synchronizing state than if v is applied. The Φ_1 function is our measure described above and it is the base measure in the first algorithm.

In the second algorithm we add the reward function which equals the length of the chosen word. The Φ_2 function with "reward" factor is defined as follows:

$$\Phi_2(w,q) = (\sum_{p \in Act} \Delta_q(p,w)) + |w|. \quad (8)$$

Now, let us define the two new algorithms. We name them SynchroP and SynchroPL. "P" denotes use of Φ_1 (Phi) function and "L" means that we add the reward factor related to the length of the word.

```
procedure SynchroP
INPUT: automaton A
OUTPUT: synchronizing word w for A
1. A2=(Q',A,δ') ← A²(A);
2. w←ε;
3. while |Q'| != 1 do {
4.    min_phi←∞;
5.    for each active state p do
6.       if (Φ1(W(p))<min_phi) then
7.          {min_phi←Φ1(W(p)); v←W(p);}
8.    w←w.v; Q'← δ'(Q',W(p)); }
9. return w;
```

The procedure **SynchroPL** is almost the same as **SynchroP** except line 6. in which Φ_2 stands for Φ_1.

One can prove the following facts (we omit the proofs):

Proposition 2. The time complexity of greedy and cycle algorithm is $O(n^3)$.

Proposition 3. The time complexity of SynchroP and SynchroPL is $O(\frac{1}{8}n^5)$.

IV. Numerical Results

It is very difficult to analyse the synchronizing algorithms because still little is known about the property of "being synchronizable". For example, we don't even know any simple property of the synchronization. Although there are many properties (for example, see Lemma 1.), they are all defined in algorithmic rather than in theoretical way and therefore are worthless in theoretical analysis.

That is why the only way to compare algorithms is to do a computer experiment: generate all *n*-state synchronizing automata, find the synchronizing words for them using all four algorithms, and compare the lengths of words being the algorithms output. The best algorithm should find the shortest synchronizing words.

We did the experiment for *n=2,3,4,5*. We generated all synchronizing automata over binary alphabet with transition functions $\delta=(a_1...a_n)(b_1...b_n)$, where $t_i=\delta(i,t)$ and $(a_1...a_n)\leq(b_1...b_n)$ in lexicographic order.

Now, let us use two methods for estimating the quality of our algorithms. The first one is some kind of a "global method" – we take into consideration all results returned by the algorithm. The second one is focused only on cases in which the algorithm works optimal, i.e. the returned word is exactly a MSW.

Method 1. We define *m(n,ALG)* in the following way:

$$m(n, ALG) = \frac{\sum_{A \in Syn(n)} (ALG(A) - MSW(A))}{|Syn(n)|}, \quad (9)$$

where:

- A is an automaton
- n is the (fixed) number of states
- ALG(A) is the length of synchronizing word for A found with algorithm ALG
- Syn(n) is the set of all synchronizing *n*-state automata.

The value *m(n,ALG)* says how much longer is a synchronizing word found by algorithm ALG than MSW length. For example, if ALG is the exponential, optimal algorithm which always finds the shortest synchronizing word, then *m(n,ALG)=0*. If for a given *n* and two algorithms A1 and A2 we have *m(n,A1)<m(n,A2)*, we say that A1 works better in finding synchronizing words for *n*-state automata.

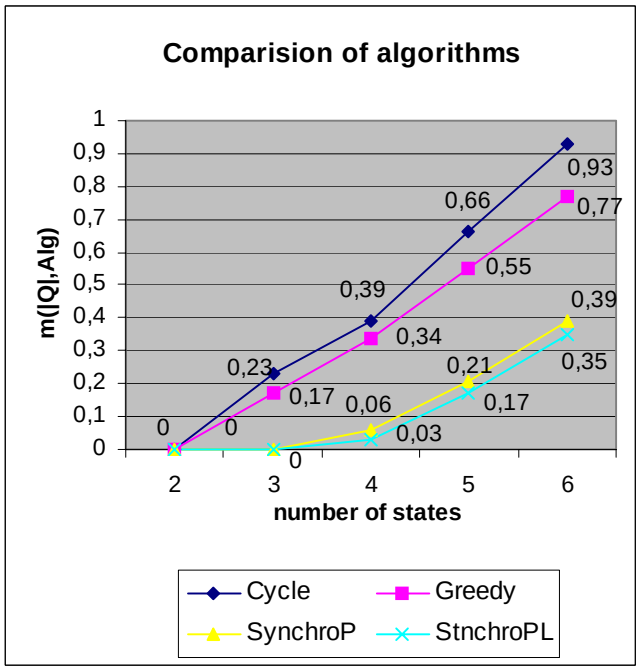

Fig. 1 Comparision of algorithms

The results of our experiment, in terms of *m* value from the Method 1., is presented in Fig. 1. We can see that the Cycle Algorithm is the least efficient one. Our two new algorithms work better than Eppstein's greedy algorithms. The second one (with reward factor) is the most efficient one: for example, for 5-state automata it finds the synchronizing word of average length |MSW|+0.17, whereas the Cycle Algorithm finds the word of length |MSW|+0.66.

Method 2. This is a very simple method; we just compute the ratio of optimal results returned by a given algorithm. Let us define the value *k*:

$$k(n, ALG) = \frac{\sum_{A \in Syn(n)} [ALG(A) = MSW(A)]}{|Syn(n)|}, \quad (10)$$

where [*expression*]=1 iff *expression* is true and 0 otherwise. If for two algorithms A1 and A2 we have *k(n,A1)<k(n,A2)*, we can say that for *n*-state automata algorithm A2 works better because it finds more optimal synchronizing words (MSWs).

TABLE I
OPTIMAL RESULTS RATIO

n	Number of synchr. automata	Cycle (%)	Greedy (%)	SynchroP (%)	SynchroPL (%)
2	5	5 (**100**)	5 (**100**)	5 (**100**)	5 (**100**)
3	270	208 (**77**)	225 (**83**)	270 (**100**)	270 (**100**)
4	25728	17674 (**69**)	18465 (**72**)	24341 (**95**)	24910 (**97**)
5	4031380	2221524 (**55**)	2423148 (**60**)	3428673 (**85**)	3510181 (**87**)
6	935719560	430721082 (**46**)	476010680 (**51**)	705120829 (**75**)	719721209 (**77**)

The results of the experiment are presented in Table I. The cell in *n*-th row and "ALG" column contains the value *k(n,ALG)*. This method gives the same order of algorithms quality: Cycle Algorithm is the least efficient one, SynchroPL is the most efficient one. For example, SynchroPL gives the optimal result cases (i.e. it returns MSW) for 5-state automata in 87% of. Cycle algorithm does it only in 55% and Greedy – in 60% of cases.

V. THE EXAMPLE

Let us give an example which shows how SynchroPL algorithm works. We will use the algorithm to find the possibly shortest synchronizing word for automaton $\mathbf{A_0}$=(Q,A,δ), where Q={0,1,2,3,4}, A={a,b} and the transition function is presented in Table II.

The SynchroPL algorithm works as it is described below. First, the pair automaton $\mathbf{A^2(A)}$ is generated. Next, for each state *s* from $\mathbf{A^2(A)}$ we compute W(*s*) and |*s*|. It can be done in a very simple way: we build the spanning tree for $\mathbf{A^2(A)}$

TABLE II
TRANSITION FUNCTION FOR A_0

δ	0	1	2	3	4
a	0	0	3	2	4
b	3	0	1	4	2

where q_0 is its root. This allows us to compute W(*s*) for any *s*. In order to compute *d*(*s*) we reverse the arrows in the spanning tree and use BFS procedure starting with q_0. We define $d(q_0)$=0 and in each step of BFS procedure, when we process the state *q*, we put *d*(*q*)=*d*(*r*)+1, where *r* is the parent of *q* in the spanning tree with reverse arrows. When all W(*s*) and *d*(*s*) values are computed, the algorithm builds the table with this values, of size n^4. Such table for $\mathbf{A_0}$ is shown in Table III. The cell in the row *s* and column *t* contains Δ_t(s,W(t)).

Now, at every stage of the main loop we compute Φ_2(W(*p*),*q*) for all *p*,*q* from the set of active states. We choose the state *r* (and the word W(*r*)), which minimalizes the value of Φ_2.

The computed values for $\mathbf{A_0}$ are presented in Table IV. The

TABLE III
TABLE OF D FUNCTION USED BY SYNCHROPL

n	01	02	03	04	12	13	14	23	24	34
01	0	-1	8	4	5	-1	8	3	2	-1
02	-1	0	3	3	0	3	3	-1	3	2
03	1	4	0	-5	-2	-5	-1	0	-1	4
04	0	-4	-4	0	-1	-4	0	-4	0	0
12	3	-2	-2	2	0	7	-2	7	4	2
13	-1	2	2	-2	2	0	-2	2	1	2
14	2	-2	-3	-2	-2	-2	0	2	-1	2
23	0	1	1	1	1	1	-3	0	-3	-3
24	1	2	1	6	2	1	-3	-3	0	2
34	-1	0	1	-4	0	1	1	-4	1	0

lowest values at every stage are bolded. At the first stage the lowest value is 5. This is the value for the state {0,1}, for which the corresponding word W({0,1}) is *a*. We put *w*=*w.a* (at the beginning *w* is an empty word). Now, the set of active states is transformed according to the found word: Q←Q.*a*={0,2,3,4}. For active states 0,2,3,4 the minimal value

TABLE IV
PHI VALUES COMPUTED BY SYNCHROPL

step (act. states)	01	02	03	04	12	13	14	23	24	34
1 (01234)	**5**	6	12	12	7	8	8	10	9	10
2 (0234)		9	7	10				**-4**	3	5
3 (03)			**0**							

of Φ_2 is realized by the word *bbababba* related with state {2,3}. We put *w*=*w.bbababba* and again transform the set of active states: Q←Q.*bbababba*={0,3}. For state {0,3} we have only one possibility: the word W({0,3})=*babba*. Again, we concatenate *w* with *babba* and finally we obtain the synchronizing word *w*=*abbababbababba*. This word has the length 14. The greedy algorithm finds the word of length 17, *w*=*abbaabbabbbababba*. The MSW for $\mathbf{A_0}$ has the length 13: *w*=*abbabbbababba*.

VI. CONCLUSION AND FUTURE WORK

We presented two new algorithms for finding possibly the shortest synchronizing words for synchronizing automata. We presented two methods for evaluating the quality of the algorithms and we used them for Greedy, Cycle, SynchroP and SynchroPL algorithms. The numerical experiments

indicated that the SynchroP and SynchroPL work better than well-known greedy algorithms.

We can try to modify the Φ function from equation (7) and check if it improves the algorithm quality. For example, one could add the weights for two components of Φ_2 - Δ and the reward factor. The heuristics here would be as follows: if there are a few active states, then in the current stage of the algorithm we should emphasize minimizing the length of synchronizing subword rather than minimize the Δ value. If there are many active states, it is important to minimize the Δ component because in later stages we will emphasize the length of synchronizing word (there will be fewer states than before). Minimizing the Δ component allows us to increase the possibility that in future configuration we will find short synchronizing subwords. In the greedy algorithm only the length of synchronizing subword is taken into consideration. Experiments show that minimizing the Δ component allows us to improve the quality of an algorithm. It would be interesting to check if a fusion of this two approaches gives better results than SynchroPL.

REFERENCES

[1] D. Eppstein, Reset Sequences for Monotonic Automata, *SIAM J. Comput.* 19(1990), 500-510.
[2] B. K. Natarajan, An algorithmic Approach to the Automated Design of Parts Orienters, *Proc. 27th Annual Symp. Foundations of Computer Science, IEEE* (1986), 132-142.
[3] B. K. Natarajan, Some paradigms for the automated design of parts feeders, *Internat. J. Robotics Research* 8(1989) 6, 89-109.
[4] D. S. Ananichev, M. V. Volkov, Synchronizing Monotonic Automata, *Lecture Notes in Computer Science*, 2710(2003), 111--121.
[5] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, E. Shapiro, Programmable and autonomous computing machine made of biomolecules, *Nature* 414(2001).
[6] Y. Benenson, R. Adar, T. Paz-Elizur, L. Livneh, E. Shapiro, DNA molecule provides a computing machine with both data and fuel, *Proc. National Acad. Sci. USA* 100(2003) 2191-2196.
[7] L. Dubuc, Les automates circulaires et la conjecture de Černý, *Inform. Theor. Appl.* 32(1998), 21-34.
[8] A. N. Trahtman, The existence of synchronizing word and Cerny Conjecture for some finite automata, *Second Haifa Workshop on Graph Theory, Combinatorics and Algorithms*, Haifa (2002).
[9] A. Salomaa, Compositions over a Finite Domain: from Completeness to Synchronizable Automata, *Turku Centre for Computer Science, Technical Report* No 350(2000).