Approximating Minimum Reset Sequences

Michael Gerbush[1] and Brent Heeringa[2]

[1] Department of Computer Science, The University of Texas at Austin, Taylor Hall 2.124, Austin, TX 78712
mgerbush@cs.utexas.edu

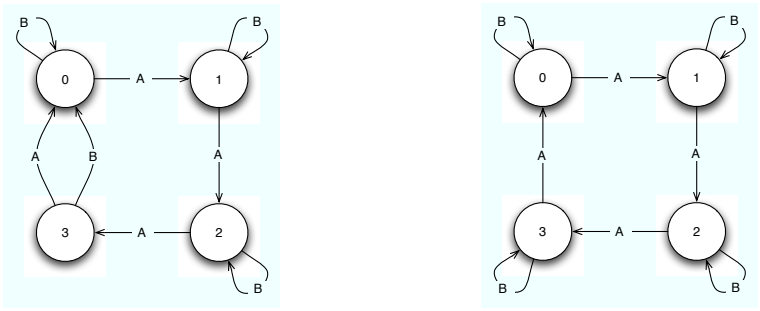
[2] Department of Computer Science, Williams College, 47 Lab Campus Drive, Williamstown, MA 01267
heeringa@cs.williams.edu

Abstract. We consider the problem of finding minimum reset sequences in synchronizing automata. The well-known Černý conjecture states that *every n-state synchronizing automaton has a reset sequence with length at most* $(n-1)^2$. While this conjecture gives an upper bound on the length of every reset sequence, it does not directly address the problem of finding the *shortest* reset sequence. We call this the MINIMUM RESET SEQUENCE (MRS) problem. We give an $O(kmn^k + n^4/k)$-time $\lceil \frac{n-1}{k-1} \rceil$-approximation for the MRS problem for any $k \geq 2$. We also show that our analysis is tight. When $k = 2$ our algorithm reduces to Eppstein's algorithm and yields an $(n-1)$-approximation. When $k = n$ our algorithm is the familiar exponential-time, exact algorithm. We define a nontrivial class of MRS which we call STACK COVER. We show that STACK COVER naturally generalizes two classic optimization problems: MIN SET COVER and SHORTEST COMMON SUPERSEQUENCE. Both these problems are known to be hard to approximate, although at present, SET COVER has a slightly stronger lower bound. In particular, it is $\mathbf{NP}$-hard to approximate SET COVER to within a factor of $c \cdot \log n$ for some $c > 0$. Thus, the MINIMUM RESET SEQUENCE problem is as least as hard to approximate as SET COVER. This improves the previous best lower bound which showed that it was $\mathbf{NP}$-hard to approximate the MRS on binary alphabets to within any constant factor. Our result requires an alphabet of arbitrary size.

1 Introduction

In the *part orienteering problem* [1], a part drops onto a pan which is moving along a conveyor belt. The part is in some unknown orientation. The goal is to move the part into a known orientation through a sequence of pan tilts. The sequence of tilts should be *universal* in the sense that, regardless of its initial position, the tilts always bring the part back to the some known orientation. If Q is a finite set of n possible states that the part can occupy, Σ is a finite alphabet of m symbols representing the types of tilts, and $\delta : Q \times \Sigma \to Q$ is a state transition function mapping states to states based on the action of a tilt from Σ, then $A = (Q, \Sigma, \delta)$ forms a simple deterministic finite automaton (omitting start and accept states).

We extend δ to sequences $\delta : Q \times \Sigma^* \to Q$ in the natural way: $\delta(q, \varepsilon) = q$ and $\delta(q, xw) = \delta(\delta(q, x), w)$ where $q \in Q$, ε is the empty sequence, $x \in \Sigma$ and $w \in \Sigma^*$.



(a) A synchronizing automaton

(b) A non-synchronizing automaton

Fig. 1: An example of a synchronizing and a non-synchronizing automata. Notice that the automaton in (a) admits the reset sequence BAAABAAAB while the automaton in (b) has no reset sequence.

We also let δ operate on *sets of states* instead of a single state. That is, if $Q' \subseteq Q$ and $z \in \Sigma^*$ then $\delta(Q', z) = \{\delta(q, z) \mid q \in Q'\}$.

Given an automaton $A = (Q, \Sigma, \delta)$, a sequence $w \in \Sigma^*$ is a *reset sequence* for A if and only if $|\delta(Q, w)| = 1$. We call w a reset sequence because it *resets* the state of a finite automaton back to some known state. Given the automaton formulation of the part orienteering problem above, it is easy to see that finding an orienting sequence of tilts is equivalent to finding a reset sequence in an automaton.

Some automata have reset sequences and some do not (see Figure 1). If an automaton has a reset sequence, then we say it is *synchronizing*. Checking if an automaton is synchronizing can be done in polynomial time using dynamic programming [2], however finding the shortest reset sequence is **NP**-hard [3]. We call finding the shortest reset sequence of an automaton the MINIMUM RESET SEQUENCE (MRS) problem. Since it is unlikely that efficient algorithms exist for the MRS problem, we consider approximating the shortest reset sequence.

1.1 Prior and Related Work

Arguably the most well-known open problem in automata theory today is a conjecture on the length of the minimum reset sequence. Posed in 1964, Černý conjectured that any n-state, synchronizing automaton has a reset sequence with length at most $(n-1)^2$ [4]. Over 40 years later, this problem remains open and continues to draw considerable attention. The current best upper bound on the MRS is $(n^3 - n)/6$ [5, 6]. For large classes of automata, Černý's conjecture holds [3, 7, 8]. In particular, Eppstein [3] gives a polynomial time algorithm for finding reset sequences with length at most $(n-1)^2$ for automata where the transition function has certain monotonicity properties and Kari [8] shows that Černý's conjecture holds when the graph underlying the automaton is Eulerian.

While Černý's conjecture claims an upper bound on the length of the shortest reset sequence, it does not directly address the problem of *finding* the shortest reset sequence. For example, in the part orienteering problem, when an $O(\log n)$ reset sequence exists, improving from an $O(n^2)$ solution to an $O(n \log n)$ solution may provide an enormous savings in production costs. Berlinkov recently showed that it $\mathbf{NP}$-hard to approximate the MRS problem to within any constant factor [9]. This result holds for binary alphabets. In addition, Olschewski and Ummels showed that finding the minimum reset sequence and determining the length of the minimum reset sequence are in $\mathbf{FP}^{\mathbf{NP}}$ and $\mathbf{FP}^{\mathbf{NP}[log]}$ respectively [10].

1.2 Results

We begin in Section 2 by giving a simple $O(kmn^k + n^4/k)$-time $\lceil \frac{n-1}{k-1} \rceil$-approximation for MINIMUM RESET SEQUENCE for any $k \geq 2$. Here n is the number of states in the automaton and m is the size of the alphabet. When $k = 2$, this algorithm reduces to Eppstein's algorithm [3] so our analysis shows that his algorithm produces an $(n-1)$-approximation. When $k = n$ our algorithm becomes the standard exponential-time, exact algorithm. We also show that our analysis is tight. In Section 3 we define a non-trivial class of MRS which we call STACK COVER. We show that STACK COVER is a natural generalization of two classic optimization problems: SET COVER and SHORTEST COMMON SUPERSEQUENCE. Under the assumption that $\mathbf{P} \neq \mathbf{NP}$, SHORTEST COMMON SUPERSEQUENCE has no α-approximation for any constant $\alpha > 0$ [11]. This matches the lower bound given by Berlinkov, albeit for alphabets of arbitrary, instead of constant size. However, assuming $\mathbf{P} \neq \mathbf{NP}$, SET COVER has no $c \cdot \log n$-approximation for some constant $c > 0$ [12]. This offers a significant improvement over the previous best lower bound. We conclude in Section 4 with some conjectures and open problems. In particular, we offer a roadmap for combining the hardness of approximating SHORTEST COMMON SUPERSEQUENCE and SET COVER to achieve even stronger lower bounds for MINIMUM RESET SEQUENCE.

2 A simple approximation algorithm

Our algorithm relies on the following property.

Property 1. Let $A = (Q, \Sigma, \delta)$ be a finite automaton. For every $k \geq 2$, A is synchronizing if and only if for every $Q' \subseteq Q$ such that $|Q'| \leq k$ there exists $x \in \Sigma^*$ such that $|\delta(Q', x)| = 1$.

Proof. When $k = 2$, the property reduces to a well-known necessary and sufficient condition for synchronization [2]. It is easy to see that since the property holds for $k = 2$ then it holds in general since any synchronizing sequence $x \in \Sigma^*$ for $\{\delta(p, y), \delta(s, y)\} \subseteq Q$ can be appended to a synchronizing sequence $y \in \Sigma^*$ for $\{p, q\} \subseteq Q$ to form a synchronizing sequence yx for $\{p, q, s\}$. □

Given some $k \geq 2$ we create a k-dimensional dynamic programming table D such that, for all subsets $Q' \subseteq Q$ where $2 \leq |Q'| \leq k$ if $x = ay$ is an MRS for Q' where $a \in \Sigma$

and $y \in \Sigma^*$ then $D(Q') = (a, \delta(Q', a))$. That is, D yields the first letter in the MRS for Q' as well as pointer to the next subset of states in the MRS. The base case is when $|Q'| = 1$. In this case we simply return the empty sequence. The following lemma establishes an upper bound on the time required to construct D.

Lemma 1. *Given $k \geq 2$, constructing D takes times $O(mn^k)$.*

Proof. Given an automaton $A = (Q, \Sigma, \delta)$, define an edge-labelled, directed multigraph $G = (V, E)$ such that $V = \{Q' \subseteq Q \mid 1 \leq |Q'| \leq k\}$ and for every $U, W \in V$ we have $(U, W) \in E$ labelled with $a \in \Sigma$ if and only if $\delta(W, a) = U$. That is, if a brings W to U then there is an edge from U to W labelled with a. We perform a breadth-first search on G, starting with all the singleton sets of Q. That is, we begin by considering all sets Q' such that $|Q'| = 1$. Whenever we encounter a node $R \in V$ for the first time we let $D(R) = (a, R')$ where R' is the parent of R in the breadth-first search and (R', R) is labelled with a. We let $D(Q') = \varepsilon$ if $|Q'| = 1$. If the breadth-first search fails to reach every node in V then, by Property 1, A is not synchronizing. Correctness of the construction follows from the fact that we are always considering the shortest sequences that take a singleton node to a non-singleton node (and, by way of reversing the edge orientations, the shortest reset sequences). Since the graph has $O(n^k)$ nodes and $O(mn^k)$ edges and we are performing a simple BFS, constructing the table takes time $O(mn^k)$. □

Algorithm 1 APPROX-MRS(A, k) where $A = (Q, \Sigma, \delta)$

```
1: X ← Q
2: z ← ϵ
3: Let D be the dynamic programming table given by Lemma 1.
4: while |X| > 1 do
5:     α ← min{|X|, k}
6:     Choose an arbitrary subset Q' ⊆ X such that |Q'| = α.
7:     while |Q'| > 1 do
8:         (a, Q') ← D(Q')
9:         z ← z · a
10:        X ← δ(X, a)
11:    end while
12: end while
13: return z
```

Algorithm 1 uses D to successively synchronize k states until only one state remains. The correctness of the algorithm follows from Property 1.

Theorem 1. *Algorithm 1 is an $O(kmn^k + n^4/k)$-time $\lceil \frac{n-1}{k-1} \rceil$-approximation for the* MINIMUM RESET SEQUENCE *problem for any $k \geq 2$.*

Proof. Let $k \geq 2$ be given and let z be the MRS for A with length OPT. For any $Q' \subseteq Q$ such that $|Q'| \leq k$, if y is the sequence given for Q' by D then $|y| \leq OPT$. This

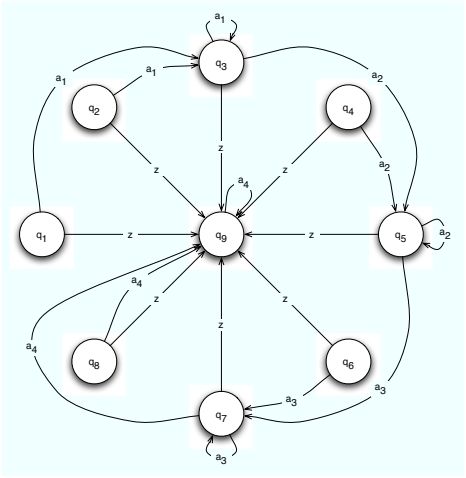


Fig. 2: A tight example for the case $n = 9$ and $k = 3$. All transitions not shown are self-transitions. The optimal solution is z, but Algorithm 1 could return the solution $a_1a_2a_3a_4$.

follows by construction since D gives us a method to find the shortest reset sequence for all $Q' \subseteq Q$ such that $|Q'| \leq k$. If $OPT < |y|$ then z would be a shorter synchronizing sequence for Q', a contradiction.

Notice that in each iteration of line 4 (other than the final iteration) we decrease the size of X by $k-1$ (once $|X| < k$ we perform at most one additional iteration). Thus, after at most $\lceil \frac{n-1}{k-1} \rceil$ iterations X contains a single state. Since each iteration of line 4 appends a sequence of length at most OPT to z, Algorithm 1 yields a sequence with length at most $\lceil \frac{n-1}{k-1} \rceil \cdot OPT$ which yields the desired approximation ratio.

Constructing the dynamic programming table takes time $O(kmn^k)$. The inner loop in line 7 executes at most $O(n^4/k)$ times since $O(n^3)$ is an upper bound on the reset sequence. Since the outer loop in line 4 executes at most n times, Algorithm 1 takes time at most $O(kmn^k + n^4/k)$. □

When $k = 2$, Algorithm 1 reduces to Eppstein's algorithm. Thus, this algorithm yields an $(n-1)$-approximation for the MINIMUM RESET SEQUENCE problem. When $k = n$ the dynamic programming table becomes the power-set automata and a breadth-first search yields the standard exponential-time exact algorithm [13]. It also turns out that the analysis of Algorithm 1 is tight.

Theorem 2. *The analysis of Algorithm 1 is tight.*

Proof. We construct an instance of the MRS problem such that Algorithm 1 gives a solution that is $\lceil \frac{n-1}{k-1} \rceil$ times larger than the optimal solution. Let n and k be given. Choose $j = \lceil \frac{n-1}{k-1} \rceil$ and take $Q = \{q_1, q_2, ..., q_n\}$ as the set of states and $\Sigma = \{z, a_1, a_2, ..., a_j\}$ as the alphabet. We define the transition function δ as

$$\delta(q_i, a_h) = \begin{cases} q_{\lceil \frac{i}{k-1} \rceil (k-1)+1} & \text{if } h = \lceil \frac{i}{k-1} \rceil \\ q_n & \text{if } a_h = z \\ q_i & \text{otherwise.} \end{cases}$$

The optimal solution to an instance of this form is the string z which has length 1. However, Algorithm 1 chooses to merge k arbitrary states. If the sequence of k-states are $(q_1, \ldots, q_k), (q_k, \ldots, q_{2(k-1)+1}), (q_{2(k-1)+1}, \ldots, q_{3(k-1)+1})$..., $(q_{(j-1)(k-1)+1}, \ldots, q_n)$, and if for each tuple the algorithm chooses a_i instead of z to merge the states, then only k states are merged at a time. In this case, the solution that Algorithm 1 gives is $a_1 a_2 \ldots a_j$ which has length $j = \lceil \frac{n-1}{k-1} \rceil$. Thus, our analysis is tight. Figure 2 gives an example for $n = 9$ and $k = 3$.

3 The STACK COVER problem

Here we define a non-trivial class of MINIMUM RESET SEQUENCE problems which we call STACK COVER. Our primary purpose in introducing this class is to show that MINIMUM RESET SEQUENCE is hard to approximate, however, STACK COVER may have independent appeal.

Imagine you have n stacks of cards where each card is painted with multiple colors. You can peek at any card in any stack at any time. Selecting a *color* removes the top card of each stack provided that card is painted with the selected color. The goal is to select the shortest sequence of colors that empties all the stacks. This is the STACK COVER problem. When each stack has a single card, then the colors represent sets and STACK COVER becomes the SET COVER problem. When the stacks have varying heights but each card is painted with a single color then a stack of cards is a string and STACK COVER becomes the SHORTEST COMMON SUPERSEQUENCE problem. Below we review the definitions of SET COVER and SHORTEST COMMON SUPERSEQUENCE and formally show how STACK COVER generalizes both problems.

We now define STACK COVER within the MINIMUM RESET SEQUENCE framework. We treat the symbols in Σ as colors and then impose some structure on the transition function. Figure 3 shows the general form of STACK COVER instances. We partition the set of states Q into n stacks $Q_1 \cup Q_2 \cup \cdots \cup Q_n$ plus a single sink state $\hat{q}$. Furthermore, we linearly order each stack Q_i as $q_{i1}, q_{i2}, \ldots, q_{il_i}, q_{i(l_i+1)}$ where each q_{ij} is a state in Q_i, and for convenience, $q_{i(l_i+1)} = \hat{q}$. That is, we assume the final state in every stack is the sink state. The transition function must obey this order, so for each $1 \leq i \leq n$ and for each $x \in \Sigma$, either $\delta(q_{ij}, x) = q_{ij}$ or $\delta(q_{ij}, x) = q_{i(j+1)}$ for all $1 \leq j \leq l_i$. Finally, we have $\delta(\hat{q}, x) = \hat{q}$ for all $x \in \Sigma$. If $A = (Q, \Sigma, \delta)$ is a STACK COVER instance, then let OPT(A) be the length of the MINIMUM RESET SEQUENCE for A.

SHORTEST COMMON SUPERSEQUENCE as STACK COVER An instance of SHORTEST COMMON SUPERSEQUENCE (SCS) is a set of strings $R = \{t_1, \ldots, t_n\}$ over an alphabet Σ of size m. That is, each t_i is a string in Σ^*. The goal is to find the shortest string $w \in \Sigma^*$ such that each string t_i is a subsequence of w. We can reduce an SCS

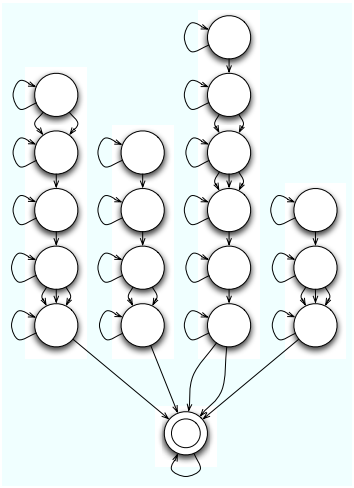


Fig. 3: An instance of STACK COVER.

instance to a STACK COVER instance as follows. Refer to the jth character in the string t_i as t_{ij}. Given a set of strings R, construct $A = (Q, \Sigma, \delta)$, such that

$$Q = \{q_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq |t_i|\} \cup \{\hat{q}\}$$

and for all $q_{ij} \in Q \setminus \{\hat{q}\}$ and $a \in \Sigma$

$$\delta(q_{ij}, a) = \begin{cases} \hat{q} & \text{if } a = t_{ij} \text{ and } j = |t_i| \\ q_{i(j+1)} & \text{if } a = t_{ij} \text{ and } j < |t_i| \\ q_{ij} & \text{otherwise.} \end{cases}$$

and $\delta(\hat{q}, a) = \hat{q}$ for all $a \in \Sigma$.

From the definition, notice that we have created a single state for each character in R and a transition symbol for each character in Σ. Also, we have added one additional state $\hat{q}$, which acts as a single sink node attached to each stack. Every transition from $\hat{q}$ is a self transition. The transition function δ guarantees that each state has only a single transition symbol to the next state in the stack. Notice that $w \in \Sigma^*$ is a reset sequence for A if and only if w is a supersequence for R.

Jiang and Li [11] showed that SCS has no α-approximation for any $\alpha > 0$ unless $\mathbf{P} = \mathbf{NP}$. Here n is the number of strings. This hardness result holds even when the strings have constant length, so given the reduction above, it applies to the MINIMUM RESET SEQUENCE problem where n is the number of states.

SET COVER as STACK COVER An instance of SET COVER is a base set $X = \{x_1, \ldots, x_n\}$ and a collection $S = \{S_1, \ldots, S_m\}$ of subsets of X. The goal is to find the smallest

subset $S' \subseteq S$ such that $\cup_{S_i \in S'} S_i = X$. We can reduce SET COVER to STACK COVER as follows. Given a SET COVER instance (X, S), construct $A = (Q, \Sigma, \delta)$ such that

$$Q = X \cup \{\hat{q}\}$$
$$\Sigma = \{1, \ldots, m\}$$
$$\delta(x, j) = \begin{cases} \hat{q} & \text{if } x \in S_j \\ x & \text{otherwise.} \end{cases}$$

In our automaton, we have a single state for each element of X and a single transition symbol for each subset in S. Again, we add a sink node, $\hat{q}$, that is connected to every stack and has only self transitions. We define the transition function so that a node can be brought to $\hat{q}$ if and only if a subset containing that character is selected. Notice that $w_1 \cdots w_l \in \Sigma^*$ is a reset sequence for A if and only if $\bigcup_{1 \leq i \leq l} S_{w_i}$ is a cover for X.

SET COVER has no $c \log n$-approximation for some constant $c > 0$ unless $\mathbf{P} = \mathbf{NP}$ [12]. Thus, this lower bound also extends to STACK COVER.

4 Open Problems and Conjectures

The lowers bounds in Section 3 require alphabets of finite, yet arbitrary size. It is an open problem to determine if these results extend to the case where the alphabet has constant size.

In addition, the gap between the upper bound on the approximation ratio offered by Algorithm 1 and the lower bound offered by SET COVER is quite large. One line of attack in closing this gap is to combine an instance of SHORTEST COMMON SUPERSEQUENCE with an instance of SET COVER to produce an instance of STACK COVER that is harder to approximate than either problem on its own. For example, given $A = (Q_A, \Sigma_A, \delta_A)$ and $B = (Q_B, \Sigma_B, \delta_B)$, where A represents an SCS problem and B represents a SET COVER problem, we define the natural cross product $A \times B =$

$$Q = (Q_A \setminus \{\hat{q_A}\}) \times (Q_B \setminus \{\hat{q_B}\}) \cup \{\hat{q}\}$$
$$\Sigma = \Sigma_A \times \Sigma_B$$
$$\delta((q_{ij}, q), (a, s)) = \begin{cases} \hat{q} & \text{if } a = t_{ij} \text{ and } j = |t_i| \text{ and } \delta_B(q, s) = \hat{q_B} \\ (q_{i(j+1)}, q) & \text{if } a = t_{ij} \text{ and } j < |t_i| \text{ and } \delta_B(q, s) = \hat{q_B} \\ (q_{ij}, q) & \text{if } a \neq t_{ij} \text{ or } \delta_B(q, s) \neq \hat{q_B} \end{cases}$$

where $\hat{q_A}$ is the sink state of A and $\hat{q_B}$ is the sink state of B.

Each state in the SCS automaton A is now paired with a state from the SET COVER automaton B, creating n stacks for each stack in the original SCS instance, where n is the number of elements in the SET COVER instance. Likewise, each transition in the SCS automaton is now paired with a transition from the SET COVER automaton, creating m transitions for each transition in the original SCS instance. Here m is the number of subsets in the SET COVER instance. The transition function has become more complex, but the general concept is straightforward: we can only move downward in a stack if we

select a symbol that corresponds to both the current node's symbol in the SCS instance and to one of its subsets in the SET COVER instance.

Assuming OPT$(\cdot)$ gives the length of an optimal solution, it's clear that OPT$(A \times B) \leq$ OPT$(A) \cdot$ OPT(B). However, if we can show that OPT$(A) \cdot$ OPT$(B) \leq \tau \cdot$ OPT$(A \times B)$ for some constant τ then the following conjecture holds:

Conjecture 1. For any $\alpha > 0$, the MINIMUM RESET SEQUENCE problem has no polynomial-time algorithm with approximation ratio $\alpha \log n$, where n is the total number of states, unless $\mathbf{P} = \mathbf{NP}$.

This lower bound is stronger than the lower bounds for both SET COVER and SHORTEST COMMON SUPERSEQUENCE. However, showing that OPT$(A) \cdot$ OPT$(B) \leq \tau \cdot$ OPT$(A \times B)$ for some constant τ seems challenging because of the interaction between A and B. More specifically, it is tempting to think that OPT$(A) \cdot$ OPT$(B) =$ OPT$(A \times B)$, but this is not the case. Consider an SCS instance $A = \{ab, ba\}$ and a SET COVER instance $B = (X, \mathcal{C})$ where $X = \{1, 2, 3, 4\}$ and $\mathcal{C} = \{\{2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}\}$. An optimal solution to $A \times B$ uses only fives symbols:

$$(B, \{1, 2, 4\}), (A, \{2, 3\}), (B, \{1, 3, 4\}), (A, \{1, 3, 4\}), (B, \{1, 2, 4\})$$

however OPT$(A) = 3$ (either aba or bab) and OPT$(B) = 2$ (since no subset contains all 4 elements).

Acknowledgements

The authors wish to thank the anonymous reviewers for their helpful comments. This material is based upon work supported by the National Science Foundation under Grant No. 0812514

References

1. Natarajan, B.K.: An algorithmic approach to the automated design of parts orienters. In: FOCS. (1986) 132–142
2. Salomaa, A.: Generation of constants and synchronization of finite automata. J. UCS **8**(2) (2002) 332–347
3. Eppstein, D.: Reset sequences for monotonic automata. SIAM J. Computing **19**(3) (June 1990) 500–510
4. Černý, J.: Poznamka k homogenym eksperimentom s konechnymi automatami. Math.-Fyz. Čas **14** (1964) 208–215
5. Pin, J.E.: On two combinatorial problems arising from automata theory. Annals of Discrete Mathematics **17** (1983) 535–548
6. Klyachko, A.A., Rystsov, I.K., Spivak, M.A.: In extremal combinatorial problem associated with the length of a synchronizing word in an automaton. Cybernetics and Systems Analysis **23**(2) (March 1987) 165–171
7. Ananichev, D., Volkov, M.: Synchronizing generalized monotonic automata. Theoretical Computer Science **330**(1) (2005) 3 – 13
8. Kari, J.: Synchronizing finite automata on eulerian digraphs. Theoretical Computer Science **295**(1-3) (2003) 223 – 232

9. Berlinkov, M.V.: On calculating length of minimal synchronizing words. CoRR **abs/0909.3787** (2009) To appear at CSR 2010.
10. Olschewski, J., Ummels, M.: The Complexity of Finding Reset Words in Finite Automata. CoRR **abs/1004.3246v1** (April 2010)
11. Jiang, T., Li, M.: On the approximation of shortest common supersequences and longest common subsequences. SIAM J. Comput. **24**(5) (October 1995) 1122–1139
12. Alon, N., Moshkovitz, D., Safra, S.: Algorithmic construction of sets for k-restrictions. ACM Trans. Algorithms **2** (April 2006) 153–177
13. Volkov, M.V.: Synchronizing automata and the Černý conjecture. In: Language and Automata Theory and Applications: Second International Conference, Berlin, Heidelberg, Springer-Verlag (2008) 11–27