Synchronizing Finite Automata

II. Algorithmic and Complexity Issues

Mikhail Volkov

Ural State University, Ekaterinburg, Russia

CSClub, St Petersburg, November 13, 2010

1. Recap

Deterministic finite automata: $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.

- Q the state set
- Σ the input alphabet
- $\delta : Q \times \Sigma \to Q$ the transition function

$\mathscr{A}$ 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 which state in Q it started at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$.

$|Q\,.\,w| = 1$. Here $Q\,.\,v = \{\delta(q, v) \mid q \in Q\}$.

Any w with this property is a reset word for $\mathscr{A}$.

1. Recap

Deterministic finite automata: $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.

- Q the state set
- Σ the input alphabet
- $\delta : Q \times \Sigma \to Q$ the transition function

$\mathscr{A}$ 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 which state in Q it started at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$.

$|Q\,.\,w| = 1$. Here $Q\,.\,v = \{\delta(q, v) \mid q \in Q\}$.

Any w with this property is a reset word for $\mathscr{A}$.

Deterministic finite automata: $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.

- Q the state set
- Σ the input alphabet
- $\delta : Q \times \Sigma \to Q$ the transition function

$\mathscr{A}$ 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 which state in Q it started at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$.

$|Q\,.\,w| = 1$. Here $Q\,.\,v = \{\delta(q, v) \mid q \in Q\}$.

Any w with this property is a reset word for $\mathscr{A}$.

1. Recap

Deterministic finite automata: $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.

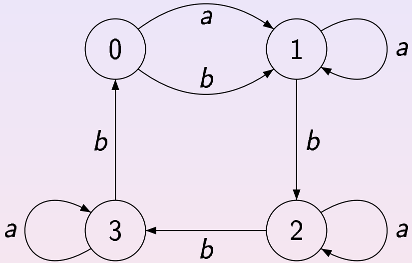
- Q the state set
- Σ the input alphabet
- $\delta : Q \times \Sigma \to Q$ the transition function

$\mathscr{A}$ 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 which state in Q it started at: $\delta(q, w) = \delta(q', w)$ for all $q, q' \in Q$.

$|Q\,.\,w| = 1$. Here $Q\,.\,v = \{\delta(q, v) \mid q \in Q\}$.

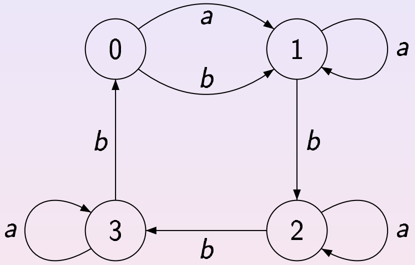
Any w with this property is a reset word for $\mathscr{A}$.

2. Example



A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

2. Example



A reset word is *abbbabbba*. In fact, we will see that this is the shortest reset word for this automaton.

Not every DFA is synchronizing. Therefore, the very first question is the following one: *given an automaton, how to determine whether or not it is synchronizing?* This question is easy, and a straightforward solution comes from the classic power automaton construction.

The *power automaton* $\mathcal{P}(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:
- states are the non-empty subsets of Q,
- $\delta(P, a) = P\,.\,a = \{\delta(p, a) \mid p \in P\}$

A $w \in \Sigma^*$ is a reset word for the DFA $\mathscr{A}$ iff w labels a path in $\mathcal{P}(\mathscr{A})$ starting at Q and ending at a singleton.

3. Power Automaton

Not every DFA is synchronizing. Therefore, the very first question is the following one: *given an automaton, how to determine whether or not it is synchronizing?* This question is easy, and a straightforward solution comes from the classic power automaton construction.

The *power automaton* $\mathcal{P}(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:
- states are the non-empty subsets of Q,
- $\delta(P, a) = P \,.\, a = \{\delta(p, a) \mid p \in P\}$

A $w \in \Sigma^*$ is a reset word for the DFA $\mathscr{A}$ iff w labels a path in $\mathcal{P}(\mathscr{A})$ starting at Q and ending at a singleton.

3. Power Automaton

Not every DFA is synchronizing. Therefore, the very first question is the following one: *given an automaton, how to determine whether or not it is synchronizing?* This question is easy, and a straightforward solution comes from the classic power automaton construction.

The *power automaton* $\mathcal{P}(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:
- states are the non-empty subsets of Q,
- $\delta(P, a) = P \,.\, a = \{\delta(p, a) \mid p \in P\}$

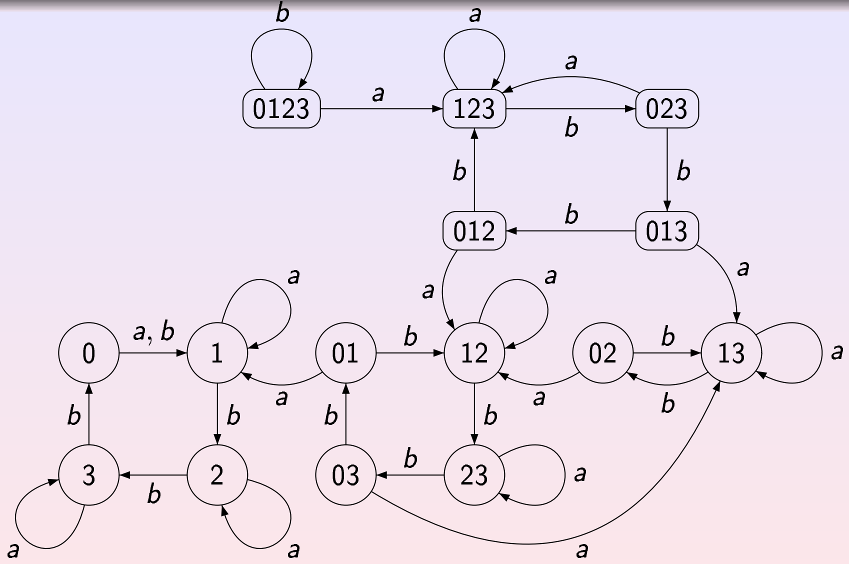
A $w \in \Sigma^*$ is a reset word for the DFA $\mathscr{A}$ iff w labels a path in $\mathcal{P}(\mathscr{A})$ starting at Q and ending at a singleton.

Not every DFA is synchronizing. Therefore, the very first question is the following one: *given an automaton, how to determine whether or not it is synchronizing?* This question is easy, and a straightforward solution comes from the classic power automaton construction.

The *power automaton* $\mathcal{P}(\mathscr{A})$ of a given DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$:
- states are the non-empty subsets of Q,
- $\delta(P, a) = P\,.\,a = \{\delta(p, a) \mid p \in P\}$

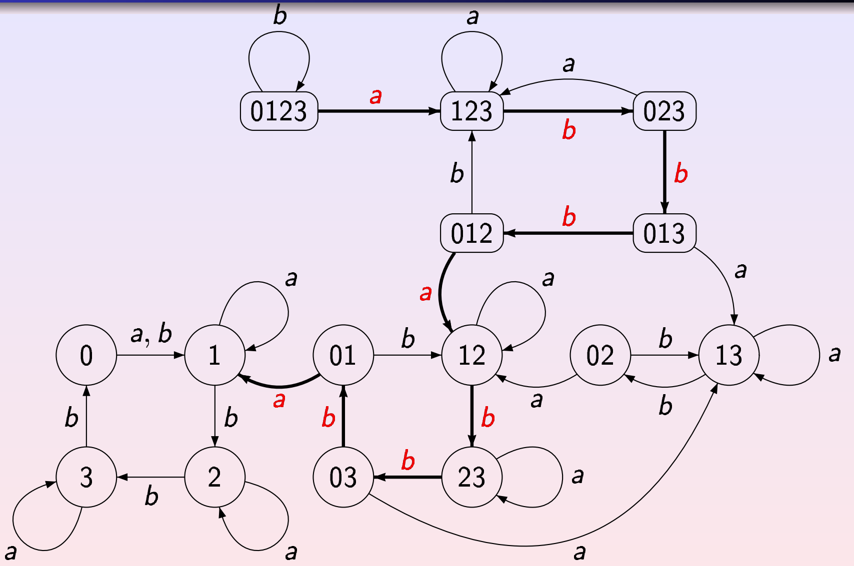
A $w \in \Sigma^*$ is a reset word for the DFA $\mathscr{A}$ iff w labels a path in $\mathcal{P}(\mathscr{A})$ starting at Q and ending at a singleton.

4. Example



CSClub, St Petersburg, November 13, 2010

4. Example



5. Polynomial Algorithm

Thus, the question of whether or not a given DFA $\mathscr{A}$ is synchronizing reduces to the following reachability question in the underlying digraph of the power automaton $\mathcal{P}(\mathscr{A})$: is there a path from Q to a singleton? The latter question can be easily answered by BFS. This algorithm is however exponential w.r.t. the size of $\mathscr{A}$.

The following result by Černý gives a polynomial algorithm:

Proposition. *A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ is synchronizing iff for every $q, q' \in Q$ there exists a word $w \in \Sigma^*$ such that $\delta(q, w) = \delta(q', w)$.*

5. Polynomial Algorithm

Thus, the question of whether or not a given DFA $\mathscr{A}$ is synchronizing reduces to the following reachability question in the underlying digraph of the power automaton $\mathcal{P}(\mathscr{A})$: is there a path from Q to a singleton? The latter question can be easily answered by BFS. This algorithm is however exponential w.r.t. the size of $\mathscr{A}$.

The following result by Černý gives a polynomial algorithm:

Proposition. *A DFA $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ is synchronizing iff for every $q, q' \in Q$ there exists a word $w \in \Sigma^*$ such that $\delta(q, w) = \delta(q', w)$.*

5. Polynomial Algorithm

Thus, the question of whether or not a given DFA $\mathscr{A}$ is synchronizing reduces to the following reachability question in the underlying digraph of the power automaton $\mathcal{P}(\mathscr{A})$: is there a path from Q to a singleton? The latter question can be easily answered by BFS. This algorithm is however exponential w.r.t. the size of $\mathscr{A}$.

The following result by Černý gives a polynomial algorithm:

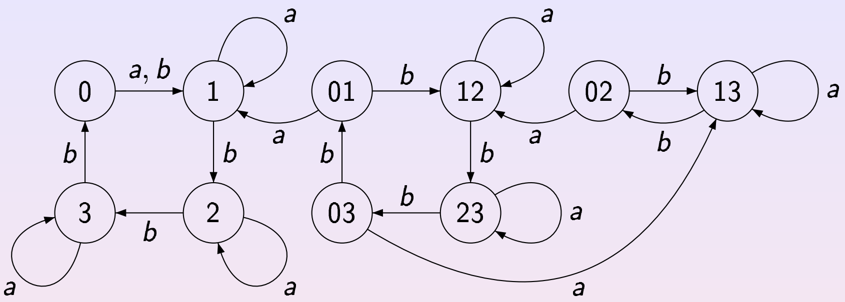
Proposition. *A DFA* $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ *is synchronizing iff for every* $q, q' \in Q$ *there exists a word* $w \in \Sigma^*$ *such that* $\delta(q, w) = \delta(q', w)$.

Thus, the question of whether or not a given DFA $\mathscr{A}$ is synchronizing reduces to the following reachability question in the underlying digraph of the power automaton $\mathcal{P}(\mathscr{A})$: is there a path from Q to a singleton? The latter question can be easily answered by BFS. This algorithm is however exponential w.r.t. the size of $\mathscr{A}$.

The following result by Černý gives a polynomial algorithm:

Proposition. *A DFA* $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ *is synchronizing iff for every* $q, q' \in Q$ *there exists a word* $w \in \Sigma^*$ *such that* $\delta(q, w) = \delta(q', w)$.

6. Example

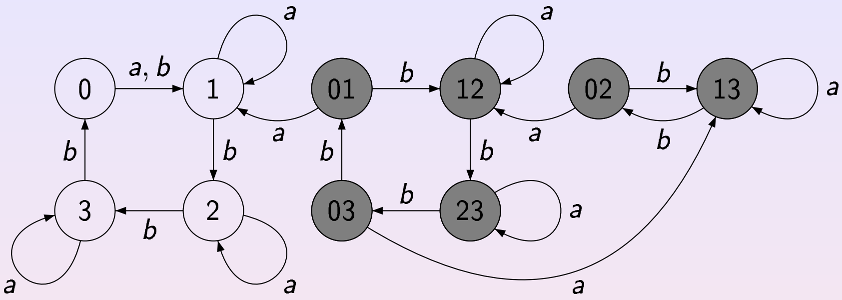


a, $Q\,.\,a = \{1,2,3\}$; $a \cdot bba$, $Q\,.\,abba = \{1,3\}$

$abba \cdot babbba$, $Q\,.\,abbababbba = \{1\}$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

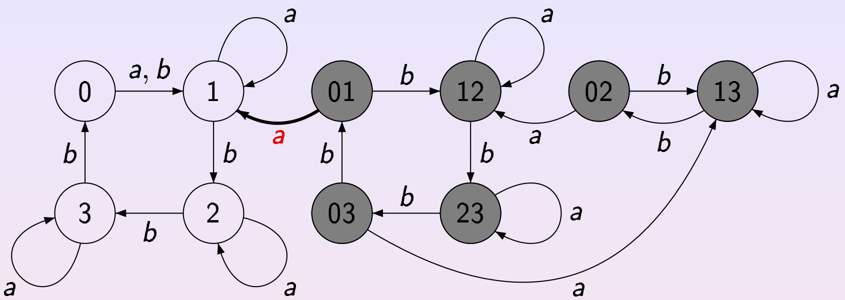
6. Example



$a, \;\; Q \,.\, a = \{1, 2, 3\}; \qquad a \cdot bba, \;\; Q \,.\, abba = \{1, 3\}$

$abba \cdot babbba, \;\; Q \,.\, abbababbba = \{1\}$

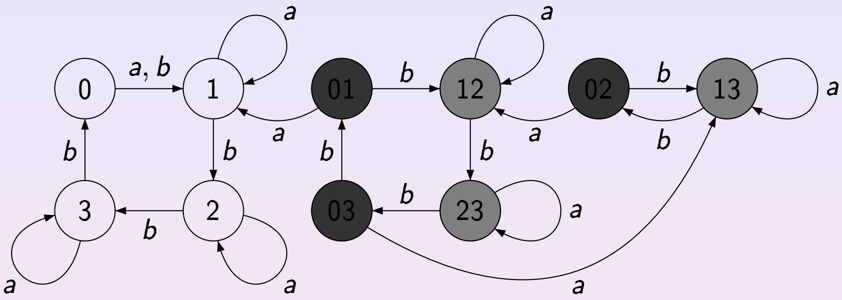
Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.



$a, \;\; Q\,.\,a = \{1, 2, 3\};$ $\quad a \cdot bba, \;\; Q\,.\,abba = \{1, 3\}$

$abba \cdot babbba, \;\; Q\,.\,abbababbba = \{1\}$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

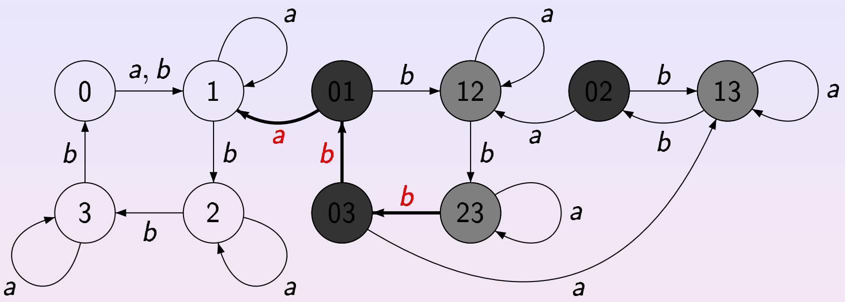


a, $Q\,.\,a = \{1,2,3\}$; $a \cdot bba$, $Q\,.\,abba = \{1,3\}$

$abba \cdot babbba$, $Q\,.\,abbababbba = \{1\}$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

6. Example

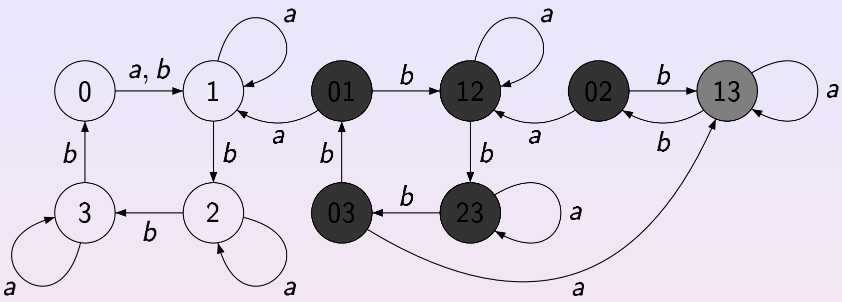


$$a, \;\; Q\,.\,a = \{1,2,3\}; \qquad a \cdot bba, \;\; Q\,.\,abba = \{1,3\}$$

$$abba \cdot babbba, \;\; Q\,.\,abbababbba = \{1\}$$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

6. Example

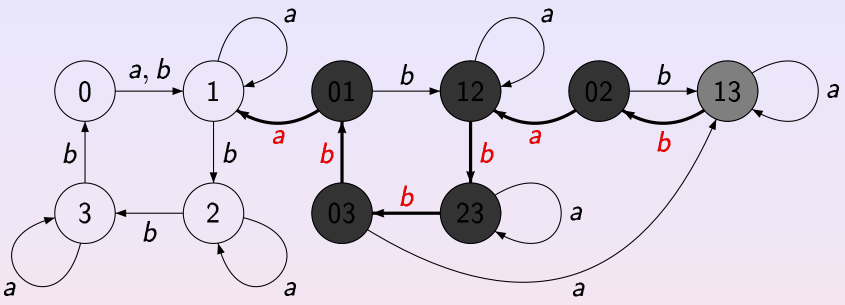


$a, \ Q\,.\,a = \{1, 2, 3\}; \quad a \cdot bba, \ Q\,.\,abba = \{1, 3\}$

$abba \cdot babbba, \ Q\,.\,abbababbba = \{1\}$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

6. Example

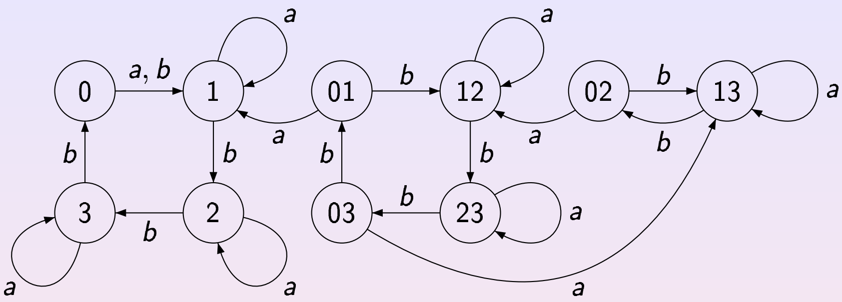


$$a,\ \ Q\,.\,a = \{1,2,3\};\qquad a\cdot bba,\ \ Q\,.\,abba = \{1,3\}$$

$$abba\cdot babbba,\ \ Q\,.\,abbababbba = \{1\}$$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

6. Example



$$a,\ Q\,.\,a = \{1,2,3\};\quad a \cdot bba,\ Q\,.\,abba = \{1,3\}$$

$$abba \cdot babbba,\ Q\,.\,abbababbba = \{1\}$$

Observe that the reset word constructed this way is of length 10 while we know a reset word of length 9.

7. Results

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.

If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n - 1$ steps and the length of the segment added in the step when k states are still to be compressed ($n \geq k \geq 2$) is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3 - n}{3}$. Can we do better? What is the exact bound?

7. Results

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.

If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n - 1$ steps and the length of the segment added in the step when k states are still to be compressed ($n \geq k \geq 2$) is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3 - n}{3}$. Can we do better? What is the exact bound?

7. Results

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.
If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n-1$ steps and the length of the segment added in the step when k states are still to be compressed $(n \geq k \geq 2)$ is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3-n}{3}$. Can we do better? What is the exact bound?

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.
If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n - 1$ steps and the length of the segment added in the step when k states are still to be compressed $(n \geq k \geq 2)$ is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3 - n}{3}$. Can we do better? What is the exact bound?

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.
If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n - 1$ steps and the length of the segment added in the step when k states are still to be compressed $(n \geq k \geq 2)$ is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3 - n}{3}$. Can we do better? What is the exact bound?

7. Results

Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.
If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

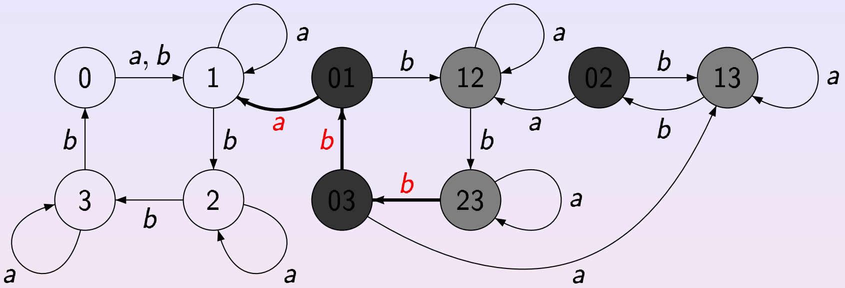
Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n-1$ steps and the length of the segment added in the step when k states are still to be compressed ($n \geq k \geq 2$) is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3-n}{3}$. Can we do better? What is the exact bound?

7. Results

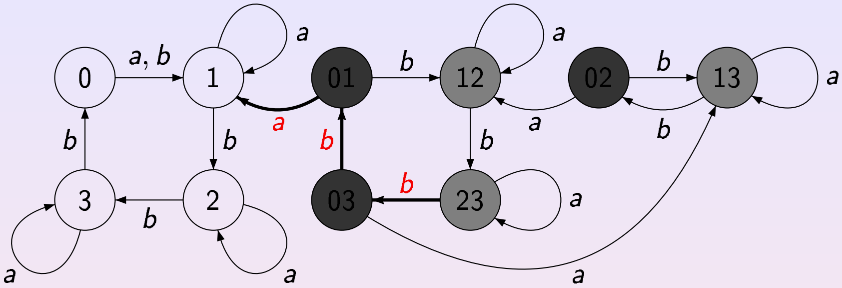
Thus, recognizing synchronizability reduces to a reachability problem in the automaton whose states are the 2-subsets and the 1-subsets of Q. The latter can be solved by BFS in $O(n^2 \cdot |\Sigma|)$ time where $n = |Q|$.
If one also wants to produce a reset word, one need $O(n^3 + n^2 \cdot |\Sigma|)$ time.

Clearly, the resulting reset word has length $O(n^3)$: the algorithm makes at most $n - 1$ steps and the length of the segment added in the step when k states are still to be compressed ($n \geq k \geq 2$) is at most $1 + \#$ of dark-grey 2-subsets, i.e. $1 + \binom{n}{2} - \binom{k}{2}$. This gives the upper bound $\frac{n^3 - n}{3}$. Can we do better? What is the exact bound?

8. A Resource for Improvement



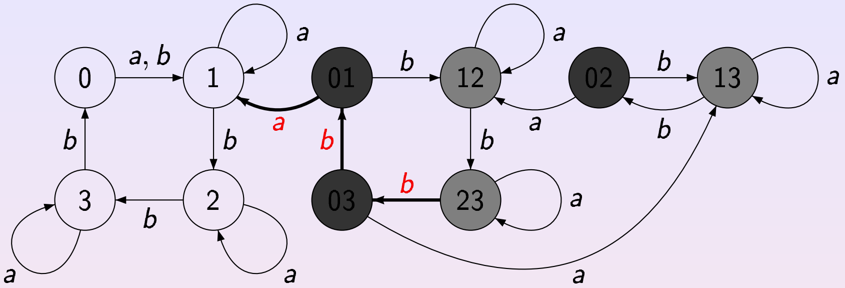
We see that the shortest path from a light-grey 2-subset to a singleton do not necessarily pass through all dark-grey 2-subsets. Consider a generic step of the algorithm at which states to be compressed form a set P with $|P| = k > 1$ and let $v = a_1 \cdots a_\ell$ with $a_i \in \Sigma$, $i = 1, \ldots, \ell$, be a word of minimum length such that $|P\,.\,v| < k$.



We see that the shortest path from a light-grey 2-subset to a singleton do not necessarily pass through all dark-grey 2-subsets.

Consider a generic step of the algorithm at which states to be compressed form a set P with $|P| = k > 1$ and let $v = a_1 \cdots a_\ell$ with $a_i \in \Sigma$, $i = 1, \ldots, \ell$, be a word of minimum length such that $|P\,.\,v| < k$.

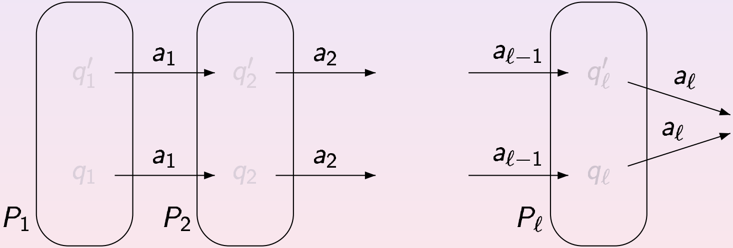
8. A Resource for Improvement



We see that the shortest path from a light-grey 2-subset to a singleton do not necessarily pass through all dark-grey 2-subsets. Consider a generic step of the algorithm at which states to be compressed form a set P with $|P| = k > 1$ and let $v = a_1 \cdots a_\ell$ with $a_i \in \Sigma$, $i = 1, \ldots, \ell$, be a word of minimum length such that $|P \,.\, v| < k$.

9. Studying Generic Step

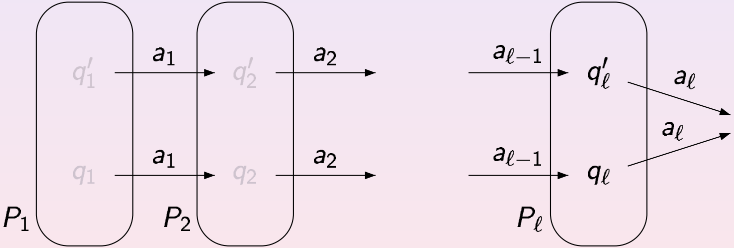
The sets $P_1 = P,\ P_2 = P_1 \, . \, a_1,\ \ldots,\ P_\ell = P_{\ell-1} \, . \, a_{\ell-1}$ are k-subsets of Q. Since $|P_\ell \, . \, a_\ell| < |P_\ell|$, there exist two states $q_\ell, q'_\ell \in P_\ell$ such that $\delta(q_\ell, a_\ell) = \delta(q'_\ell, a_\ell)$. Now define 2-subsets $R_i = \{q_i, q'_i\} \subseteq P_i$, $i = 1, \ldots, \ell$, such that $\delta(q_i, a_i) = q_{i+1}$, $\delta(q'_i, a_i) = q'_{i+1}$ for $i = 1, \ldots, \ell - 1$.



The condition that v is a word of minimum length with $|P \, . \, v| < |P|$ implies $R_i \not\subseteq P_j$ for $1 \le j < i \le \ell$.

9. Studying Generic Step

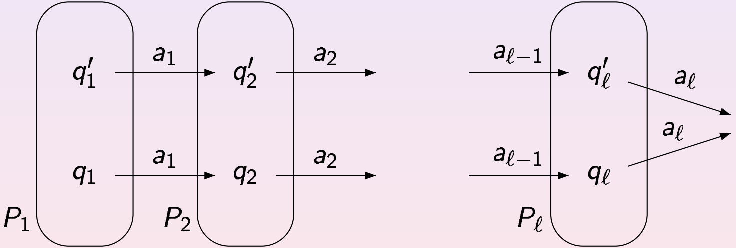
The sets $P_1 = P, \ P_2 = P_1 \,.\, a_1, \ \ldots, \ P_\ell = P_{\ell-1} \,.\, a_{\ell-1}$ are k-subsets of Q. Since $|P_\ell \,.\, a_\ell| < |P_\ell|$, there exist two states $q_\ell, q'_\ell \in P_\ell$ such that $\delta(q_\ell, a_\ell) = \delta(q'_\ell, a_\ell)$. Now define 2-subsets $R_i = \{q_i, q'_i\} \subseteq P_i$, $i = 1, \ldots, \ell$, such that $\delta(q_i, a_i) = q_{i+1}$, $\delta(q'_i, a_i) = q'_{i+1}$ for $i = 1, \ldots, \ell - 1$.



The condition that v is a word of minimum length with $|P \,.\, v| < |P|$ implies $R_i \not\subseteq P_j$ for $1 \le j < i \le \ell$.

9. Studying Generic Step

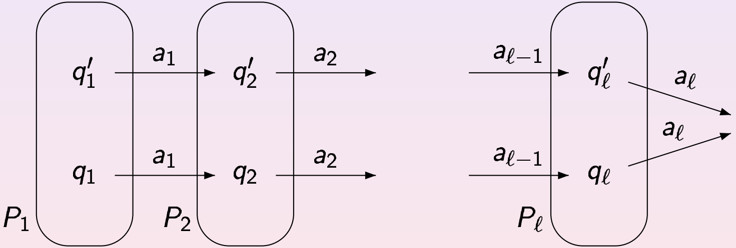
The sets $P_1 = P$, $P_2 = P_1 \,.\, a_1$, $\dots$, $P_\ell = P_{\ell-1} \,.\, a_{\ell-1}$ are k-subsets of Q. Since $|P_\ell \,.\, a_\ell| < |P_\ell|$, there exist two states $q_\ell, q'_\ell \in P_\ell$ such that $\delta(q_\ell, a_\ell) = \delta(q'_\ell, a_\ell)$. Now define 2-subsets $R_i = \{q_i, q'_i\} \subseteq P_i$, $i = 1, \dots, \ell$, such that $\delta(q_i, a_i) = q_{i+1}$, $\delta(q'_i, a_i) = q'_{i+1}$ for $i = 1, \dots, \ell - 1$.



The condition that v is a word of minimum length with $|P \,.\, v| < |P|$ implies $R_i \not\subseteq P_j$ for $1 \le j < i \le \ell$.

9. Studying Generic Step

The sets $P_1 = P$, $P_2 = P_1 . a_1$, ..., $P_\ell = P_{\ell-1} . a_{\ell-1}$ are k-subsets of Q. Since $|P_\ell . a_\ell| < |P_\ell|$, there exist two states $q_\ell, q'_\ell \in P_\ell$ such that $\delta(q_\ell, a_\ell) = \delta(q'_\ell, a_\ell)$. Now define 2-subsets $R_i = \{q_i, q'_i\} \subseteq P_i$, $i = 1, \ldots, \ell$, such that $\delta(q_i, a_i) = q_{i+1}$, $\delta(q'_i, a_i) = q'_{i+1}$ for $i = 1, \ldots, \ell - 1$.



The condition that v is a word of minimum length with $|P . v| < |P|$ implies $R_i \nsubseteq P_j$ for $1 \leq j < i \leq \ell$.

10. Combinatorial Configuration

Our question reduces to the following problem in combinatorics of finite sets:

Let Q be an n-set, $P_1, \dots, P_\ell$ a sequence of its k-subsets ($k > 1$) such that each P_i, $1 < i \leq \ell$, includes a "fresh" 2-subset that does not occur in any previous P_j ($1 \leq j < i$). How long can such refreshing sequences be?

A construction: fix a $(k-2)$-subset W of Q, list all $\binom{n-k+2}{2}$ 2-subsets of $Q \setminus W$ and let T_i be the union of W with the i^{th} 2-subset in the list. This gives the refreshing sequence $T_1, \dots, T_s$ of length $s = \binom{n-k+2}{2}$. Is this the maximum?

Our question reduces to the following problem in combinatorics of finite sets:

Let Q be an n-set, $P_1, \dots, P_\ell$ a sequence of its k-subsets ($k > 1$) such that each P_i, $1 < i \leq \ell$, includes a "fresh" 2-subset that does not occur in any previous P_j ($1 \leq j < i$). How long can such refreshing sequences be?

A construction: fix a $(k-2)$-subset W of Q, list all $\binom{n-k+2}{2}$ 2-subsets of $Q \setminus W$ and let T_i be the union of W with the i^{th} 2-subset in the list. This gives the refreshing sequence $T_1, \dots, T_s$ of length $s = \binom{n-k+2}{2}$. Is this the maximum?

Our question reduces to the following problem in combinatorics of finite sets:

Let Q be an n-set, $P_1, \dots, P_\ell$ a sequence of its k-subsets ($k > 1$) such that each P_i, $1 < i \leq \ell$, includes a "fresh" 2-subset that does not occur in any previous P_j ($1 \leq j < i$). How long can such refreshing sequences be?

A construction: fix a $(k-2)$-subset W of Q, list all $\binom{n-k+2}{2}$ 2-subsets of $Q \setminus W$ and let T_i be the union of W with the i^{th} 2-subset in the list. This gives the refreshing sequence $T_1, \dots, T_s$ of length $s = \binom{n-k+2}{2}$. Is this the maximum?

Our question reduces to the following problem in combinatorics of finite sets:

Let Q be an n-set, $P_1, \dots, P_\ell$ a sequence of its k-subsets $(k > 1)$ such that each P_i, $1 < i \leq \ell$, includes a "fresh" 2-subset that does not occur in any previous P_j $(1 \leq j < i)$. How long can such refreshing sequences be?

A construction: fix a $(k-2)$-subset W of Q, list all $\binom{n-k+2}{2}$ 2-subsets of $Q \setminus W$ and let T_i be the union of W with the i^{th} 2-subset in the list. This gives the refreshing sequence $T_1, \dots, T_s$ of length $s = \binom{n-k+2}{2}$. Is this the maximum?

10. Combinatorial Configuration

Our question reduces to the following problem in combinatorics of finite sets:

Let Q be an n-set, $P_1, \ldots, P_\ell$ a sequence of its k-subsets ($k > 1$) such that each P_i, $1 < i \leq \ell$, includes a "fresh" 2-subset that does not occur in any previous P_j ($1 \leq j < i$). How long can such refreshing sequences be?

A construction: fix a $(k-2)$-subset W of Q, list all $\binom{n-k+2}{2}$ 2-subsets of $Q \setminus W$ and let T_i be the union of W with the i^{th} 2-subset in the list. This gives the refreshing sequence $T_1, \ldots, T_s$ of length $s = \binom{n-k+2}{2}$. Is this the maximum?

10. Combinatorial Configuration

Our question reduces to the following problem in combinatorics of finite sets:

Let Q be an n-set, $P_1, \ldots, P_\ell$ a sequence of its k-subsets ($k > 1$) such that each P_i, $1 < i \leq \ell$, includes a “fresh” 2-subset that does not occur in any previous P_j ($1 \leq j < i$). How long can such refreshing sequences be?

A construction: fix a $(k-2)$-subset W of Q, list all $\binom{n-k+2}{2}$ 2-subsets of $Q \setminus W$ and let T_i be the union of W with the i^{th} 2-subset in the list. This gives the refreshing sequence $T_1, \ldots, T_s$ of length $s = \binom{n-k+2}{2}$. Is this the maximum?

The question turned out to be very difficult and was solved (in the affirmative) by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127).

The proof uses linearization techniques which is quite common in combinatorics of finite sets. One reformulates the problem in linear algebra terms and then uses the corresponding machinery.

We identify Q with $\{1, 2, \ldots, n\}$ and assign to each k-subset $I = \{i_1, \ldots, i_k\}$ the following polynomial $D(I)$ in variables $x_{i_1}, \ldots, x_{i_k}$ over the field of rationals.

The question turned out to be very difficult and was solved (in the affirmative) by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127).

The proof uses linearization techniques which is quite common in combinatorics of finite sets. One reformulates the problem in linear algebra terms and then uses the corresponding machinery.

We identify Q with $\{1, 2, \ldots, n\}$ and assign to each k-subset $I = \{i_1, \ldots, i_k\}$ the following polynomial $D(I)$ in variables $x_{i_1}, \ldots, x_{i_k}$ over the field of rationals.

The question turned out to be very difficult and was solved (in the affirmative) by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127).

The proof uses linearization techniques which is quite common in combinatorics of finite sets. One reformulates the problem in linear algebra terms and then uses the corresponding machinery.

We identify Q with $\{1, 2, \ldots, n\}$ and assign to each k-subset $I = \{i_1, \ldots, i_k\}$ the following polynomial $D(I)$ in variables $x_{i_1}, \ldots, x_{i_k}$ over the field of rationals.

The question turned out to be very difficult and was solved (in the affirmative) by Peter Frankl (An extremal problem for two families of sets, Eur. J. Comb., 3 (1982) 125–127).

The proof uses linearization techniques which is quite common in combinatorics of finite sets. One reformulates the problem in linear algebra terms and then uses the corresponding machinery.

We identify Q with $\{1, 2, \ldots, n\}$ and assign to each k-subset $I = \{i_1, \ldots, i_k\}$ the following polynomial $D(I)$ in variables $x_{i_1}, \ldots, x_{i_k}$ over the field of rationals.

12. Linearization

$$I = \{i_1, \dots, i_k\} \mapsto D(I) = \begin{vmatrix} 1 & i_1 & i_1^2 & \cdots & i_1^{k-3} & x_{i_1} & x_{i_1}^2 \\ 1 & i_2 & i_2^2 & \cdots & i_2^{k-3} & x_{i_2} & x_{i_2}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & i_k & i_k^2 & \cdots & i_k^{k-3} & x_{i_k} & x_{i_k}^2 \end{vmatrix}_{k \times k}$$

Then one proves that:

• the polynomials $D(P_1), \dots, D(P_\ell)$ are linearly independent whenever the k-subsets $P_1, \dots, P_\ell$ form a refreshing sequence;

• the polynomials $D(T_1), \dots, D(T_s)$ (derived from the "standard" sequence) generate the linear space spanned by all polynomials of the form $D(I)$.

12. Linearization

$$I = \{i_1, \ldots, i_k\} \mapsto D(I) = \begin{vmatrix} 1 & i_1 & i_1^2 & \cdots & i_1^{k-3} & x_{i_1} & x_{i_1}^2 \\ 1 & i_2 & i_2^2 & \cdots & i_2^{k-3} & x_{i_2} & x_{i_2}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & i_k & i_k^2 & \cdots & i_k^{k-3} & x_{i_k} & x_{i_k}^2 \end{vmatrix}_{k \times k}$$

Then one proves that:

• the polynomials $D(P_1), \ldots, D(P_\ell)$ are linearly independent whenever the k-subsets $P_1, \ldots, P_\ell$ form a refreshing sequence;

• the polynomials $D(T_1), \ldots, D(T_s)$ (derived from the "standard" sequence) generate the linear space spanned by all polynomials of the form $D(I)$.

12. Linearization

$$I = \{i_1, \dots, i_k\} \mapsto D(I) = \begin{vmatrix} 1 & i_1 & i_1^2 & \cdots & i_1^{k-3} & x_{i_1} & x_{i_1}^2 \\ 1 & i_2 & i_2^2 & \cdots & i_2^{k-3} & x_{i_2} & x_{i_2}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & i_k & i_k^2 & \cdots & i_k^{k-3} & x_{i_k} & x_{i_k}^2 \end{vmatrix}_{k \times k}$$

Then one proves that:

• the polynomials $D(P_1), \dots, D(P_\ell)$ are linearly independent whenever the k-subsets $P_1, \dots, P_\ell$ form a refreshing sequence;

• the polynomials $D(T_1), \dots, D(T_s)$ (derived from the "standard" sequence) generate the linear space spanned by all polynomials of the form $D(I)$.

13. Results

Thus, in the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$.

Summing up over $k = n, \ldots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

$$\binom{2}{2}+\binom{3}{2}+\binom{4}{2}+\cdots+\binom{n-1}{2}+\binom{n}{2} =$$

$$\binom{3}{3}+\binom{3}{2}+\binom{4}{2}+\cdots+\binom{n-1}{2}+\binom{n}{2} =$$

$$\binom{4}{3}+\binom{4}{2}+\cdots+\binom{n-1}{2}+\binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

13. Results

Thus, in the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$.
Summing up over $k = n, \dots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{3}{3} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

Thus, in the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$.
Summing up over $k = n, \dots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{3}{3} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

13. Results

Thus, in the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$.
Summing up over $k = n, \dots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{3}{3} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

Thus, in the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$.
Summing up over $k = n, \dots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{3}{3} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

Thus, in the step when k states are still to be compressed, the compression can always be achieved by applying a suitable word of length $\leq \binom{n-k+2}{2}$.
Summing up over $k = n, \dots, 2$, we see that the greedy algorithm always returns a reset word of length $\leq \frac{n^3-n}{6}$:

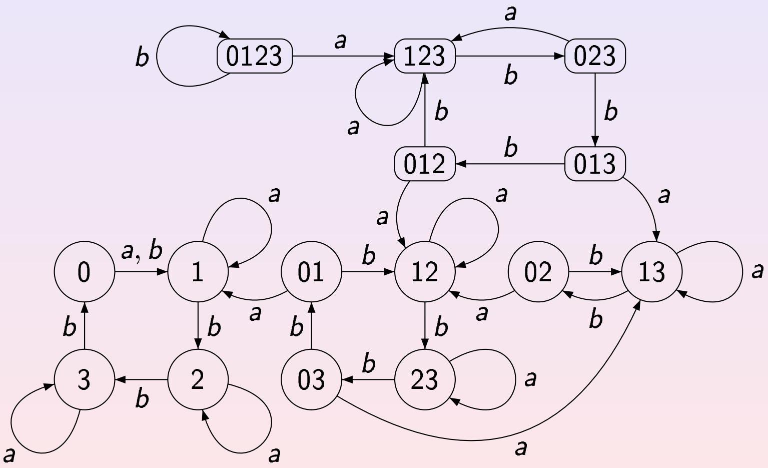
$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{3}{3} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} =$$

$$\binom{4}{3} + \binom{4}{2} + \cdots + \binom{n-1}{2} + \binom{n}{2} = \cdots = \binom{n+1}{3} = \frac{n^3-n}{6}.$$

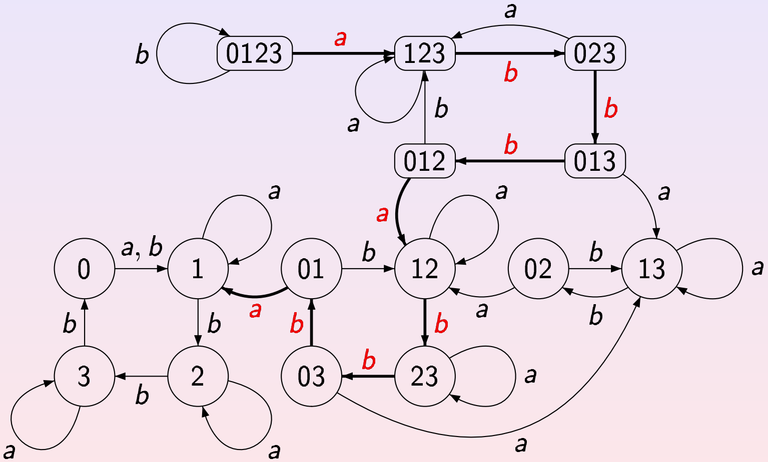
14. Example Revisited

We have already seen that the greedy algorithm fails to find a reset word of minimum length.



14. Example Revisited

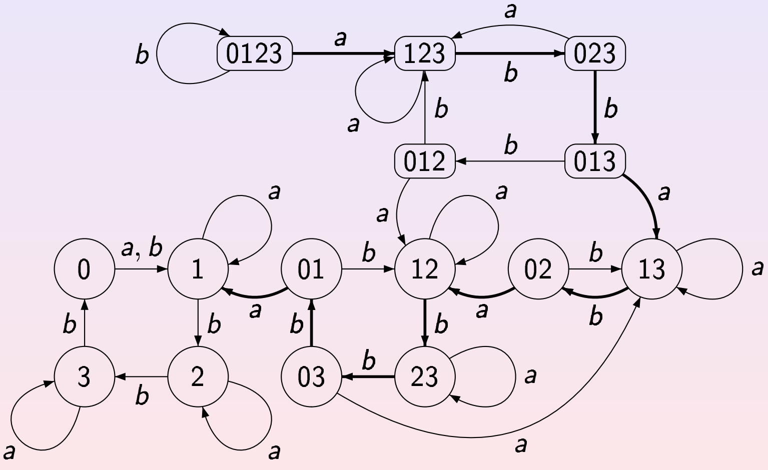
We have already seen that the greedy algorithm fails to find a reset word of minimum length.



CSClub, St Petersburg, November 13, 2010

14. Example Revisited

We have already seen that the greedy algorithm fails to find a reset word of minimum length.



15. Short Reset Words are Hard to Find

Actually, the gap between the minimum length of a reset word and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exists a synchronizing automaton with n states whose shortest reset word has length $(n-1)^2$ while the greedy algorithm produces a reset word of length $\Omega(n^2 \log n)$.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well.

Now we aim to prove that under standard assumptions (like $NP \neq coNP$) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata.

15. Short Reset Words are Hard to Find

Actually, the gap between the minimum length of a reset word and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exists a synchronizing automaton with n states whose shortest reset word has length $(n-1)^2$ while the greedy algorithm produces a reset word of length $\Omega(n^2 \log n)$.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well.

Now we aim to prove that under standard assumptions (like $NP \neq coNP$) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata.

Actually, the gap between the minimum length of a reset word and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exists a synchronizing automaton with n states whose shortest reset word has length $(n-1)^2$ while the greedy algorithm produces a reset word of length $\Omega(n^2 \log n)$.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well.

Now we aim to prove that under standard assumptions (like $NP \neq coNP$) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata.

15. Short Reset Words are Hard to Find

Actually, the gap between the minimum length of a reset word and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exists a synchronizing automaton with n states whose shortest reset word has length $(n-1)^2$ while the greedy algorithm produces a reset word of length $\Omega(n^2 \log n)$.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well.

Now we aim to prove that under standard assumptions (like $\mathrm{NP} \neq \mathrm{coNP}$) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata.

15. Short Reset Words are Hard to Find

Actually, the gap between the minimum length of a reset word and the length of the word produced by the greedy algorithm may be arbitrarily large: for each $n > 1$ there exists a synchronizing automaton with n states whose shortest reset word has length $(n-1)^2$ while the greedy algorithm produces a reset word of length $\Omega(n^2 \log n)$.

The behaviour of the greedy algorithm on average is not yet understood; practically it behaves rather well.

Now we aim to prove that under standard assumptions (like NP $\neq$ coNP) no polynomial algorithm, even non-deterministic, can find the minimum length of reset words for synchronizing automata.

16. Short Reset Words are Hard to Decide

Consider the following decision problem:

SHORT-RESET-WORD: *Given a synchronizing automaton* $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ *and a positive integer* ℓ*, is it true that* $\mathscr{A}$ *has a reset word of length* ℓ*?*

Clearly, SHORT-RESET-WORD belongs to NP: one can non-deterministically guess a word $w \in \Sigma^*$ of length ℓ and then check if w is a reset word for $\mathscr{A}$ in time $\ell|Q|$.

Several authors have observed that SHORT-RESET-WORD is NP-hard by a transparent reduction from SAT.

16. Short Reset Words are Hard to Decide

Consider the following decision problem:

SHORT-RESET-WORD: *Given a synchronizing automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ and a positive integer ℓ, is it true that $\mathscr{A}$ has a reset word of length ℓ?*

Clearly, SHORT-RESET-WORD belongs to NP: one can non-deterministically guess a word $w \in \Sigma^*$ of length ℓ and then check if w is a reset word for $\mathscr{A}$ in time $\ell|Q|$.

Several authors have observed that SHORT-RESET-WORD is NP-hard by a transparent reduction from SAT.

16. Short Reset Words are Hard to Decide

Consider the following decision problem:

SHORT-RESET-WORD: *Given a synchronizing automaton* $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$ *and a positive integer* ℓ*, is it true that* $\mathscr{A}$ *has a reset word of length* ℓ*?*

Clearly, SHORT-RESET-WORD belongs to NP: one can non-deterministically guess a word $w \in \Sigma^*$ of length ℓ and then check if w is a reset word for $\mathscr{A}$ in time $\ell|Q|$.

Several authors have observed that SHORT-RESET-WORD is NP-hard by a transparent reduction from SAT.

17. Reduction from SAT

Given an instance ψ of SAT with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$, one constructs $\mathscr{A}(\psi)$ with 2 input letters a and b and the state set $\{z, q_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq n+1\}$.

The transitions are defined by:

17. Reduction from SAT

Given an instance ψ of SAT with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$, one constructs $\mathscr{A}(\psi)$ with 2 input letters a and b and the state set $\{z, q_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq n+1\}$. The transitions are defined by:

17. Reduction from SAT

Given an instance ψ of SAT with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$, one constructs $\mathscr{A}(\psi)$ with 2 input letters a and b and the state set $\{z, q_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq n+1\}$. The transitions are defined by:

$$q_{i,j} \,.\, a = \begin{cases} z \text{ if } x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \qquad \text{for } 1 \leq i \leq m,\ 1 \leq j \leq n;$$

17. Reduction from SAT

Given an instance ψ of SAT with n variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$, one constructs $\mathscr{A}(\psi)$ with 2 input letters a and b and the state set $\{z, q_{i,j} \mid 1 \le i \le m,\ 1 \le j \le n+1\}$. The transitions are defined by:

$$q_{i,j} \, . \, a = \begin{cases} z \text{ if } x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \qquad \text{for } 1 \le i \le m,\ 1 \le j \le n;$$

$$q_{i,j} \, . \, b = \begin{cases} z \text{ if } \neg x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \qquad \text{for } 1 \le i \le m,\ 1 \le j \le n;$$

17. Reduction from SAT

Given an instance ψ of SAT with n variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$, one constructs $\mathscr{A}(\psi)$ with 2 input letters a and b and the state set $\{z, q_{i,j} \mid 1 \le i \le m,\ 1 \le j \le n+1\}$. The transitions are defined by:

$$q_{i,j} \,.\, a = \begin{cases} z \text{ if } x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \qquad \text{for } 1 \le i \le m,\ 1 \le j \le n;$$

$$q_{i,j} \,.\, b = \begin{cases} z \text{ if } \neg x_j \text{ occurs in } c_i, \\ q_{i,j+1} \text{ otherwise} \end{cases} \qquad \text{for } 1 \le i \le m,\ 1 \le j \le n;$$

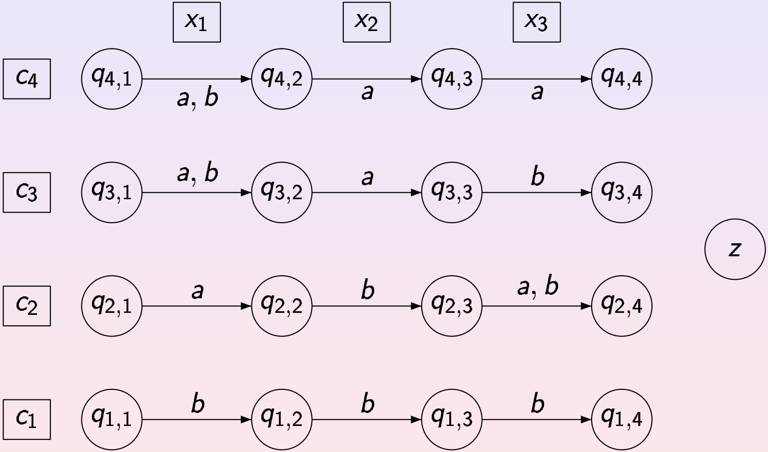
$$q_{i,n+1} \,.\, a = q_{i,n+1} \,.\, b = z \qquad \text{for } 1 \le i \le m;$$

$$z \,.\, a = z \,.\, b = z.$$

For $\psi = \{x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}$:

18. Reduction from SAT

For $\psi = \{x_1 \vee x_2 \vee x_3,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}$:



19. Reduction from SAT

It is easy to see that $\mathscr{A}(\psi)$ is reset by every word of length $n+1$ and is reset by a word of length n if and only if ψ is satisfiable.

Thus, assigning the instance $(\mathscr{A}(\psi), n)$ of SHORT-RESET-WORD to an arbitrary n-variable instance ψ of SAT, one gets a polynomial reduction which is in fact parsimonious.

19. Reduction from SAT

It is easy to see that $\mathscr{A}(\psi)$ is reset by every word of length $n+1$ and is reset by a word of length n if and only if ψ is satisfiable. In the above example the truth assignment $x_1 = x_2 = 0$, $x_3 = 1$ satisfies ψ and the word bba resets $\mathscr{A}(\psi)$.

Thus, assigning the instance $(\mathscr{A}(\psi), n)$ of SHORT-RESET-WORD to an arbitrary n-variable instance ψ of SAT, one gets a polynomial reduction which is in fact parsimonious.

19. Reduction from SAT

It is easy to see that $\mathscr{A}(\psi)$ is reset by every word of length $n+1$ and is reset by a word of length n if and only if ψ is satisfiable. In the above example the truth assignment $x_1 = x_2 = 0$, $x_3 = 1$ satisfies ψ and the word bba resets $\mathscr{A}(\psi)$.

If we change ψ to $\{x_1 \vee x_2, \neg x_1 \vee x_2, \neg x_2 \vee x_3, \neg x_2 \vee \neg x_3\}$, it becomes unsatisfiable and $\mathscr{A}(\psi)$ is reset by no word of length 3. Thus, assigning the instance $(\mathscr{A}(\psi), n)$ of SHORT-RESET-WORD to an arbitrary n-variable instance ψ of SAT, one gets a polynomial reduction which is in fact parsimonious.

It is easy to see that $\mathscr{A}(\psi)$ is reset by every word of length $n+1$ and is reset by a word of length n if and only if ψ is satisfiable.

Thus, assigning the instance $(\mathscr{A}(\psi), n)$ of SHORT-RESET-WORD to an arbitrary n-variable instance ψ of SAT, one gets a polynomial reduction which is in fact parsimonious.

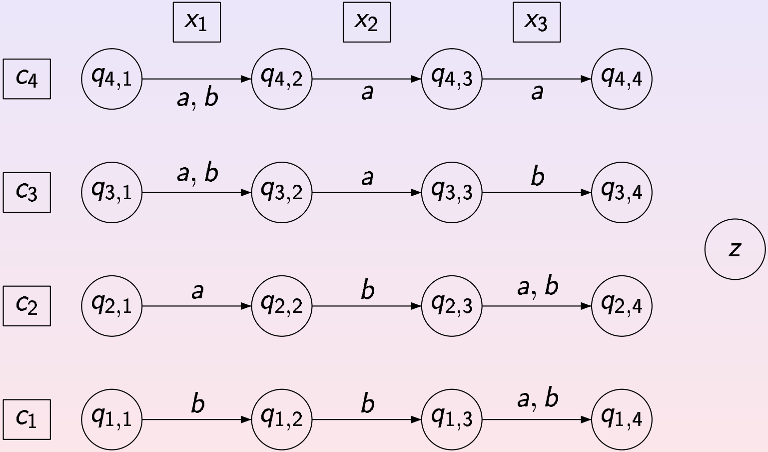
It is easy to see that $\mathscr{A}(\psi)$ is reset by every word of length $n+1$ and is reset by a word of length n if and only if ψ is satisfiable.

Thus, assigning the instance $(\mathscr{A}(\psi), n)$ of SHORT-RESET-WORD to an arbitrary n-variable instance ψ of SAT, one gets a polynomial reduction which is in fact parsimonious.

For $\psi = \{x_1 \vee x_2, \neg x_1 \vee x_2, \neg x_2 \vee x_3, \neg x_2 \vee \neg x_3\}$:

20. Reduction from SAT

For $\psi = \{x_1 \vee x_2,\ \neg x_1 \vee x_2,\ \neg x_2 \vee x_3,\ \neg x_2 \vee \neg x_3\}$:



21. Shortest Reset Words are Even Harder to Decide

Now consider the following decision problem:

SHORTEST-RESET-WORD: *Given a synchronizing automaton $\mathscr{A}$ and a positive integer ℓ, is it true that the minimum length of a reset word for $\mathcal{A}$ is equal to ℓ?*

Assigning the instance $(\mathcal{A}(\psi), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system ψ of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if ψ is not satisfiable. This is a polynomial reduction from the negation of SAT to SHORTEST-RESET-WORD whence the latter problem is coNP-hard. As a corollary, SHORTEST-RESET-WORD cannot belong to NP unless NP $=$ coNP.

Recently, SHORTEST-RESET-WORD has shown to be complete for DP (Difference Polynomial-Time).

21. Shortest Reset Words are Even Harder to Decide

Now consider the following decision problem:

SHORTEST-RESET-WORD: *Given a synchronizing automaton $\mathscr{A}$ and a positive integer ℓ, is it true that the minimum length of a reset word for $\mathcal{A}$ is equal to ℓ?*

Assigning the instance $(\mathcal{A}(\psi), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system ψ of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if ψ is not satisfiable. This is a polynomial reduction from the negation of SAT to SHORTEST-RESET-WORD whence the latter problem is coNP-hard. As a corollary, SHORTEST-RESET-WORD cannot belong to NP unless NP = coNP.

Recently, SHORTEST-RESET-WORD has shown to be complete for DP (Difference Polynomial-Time).

21. Shortest Reset Words are Even Harder to Decide

Now consider the following decision problem:

SHORTEST-RESET-WORD: *Given a synchronizing automaton $\mathscr{A}$ and a positive integer ℓ, is it true that the minimum length of a reset word for $\mathcal{A}$ is equal to ℓ?*

Assigning the instance $(\mathcal{A}(\psi), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system ψ of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if ψ is not satisfiable. This is a polynomial reduction from the negation of SAT to SHORTEST-RESET-WORD whence the latter problem is coNP-hard. As a corollary, SHORTEST-RESET-WORD cannot belong to NP unless NP = coNP.

Recently, SHORTEST-RESET-WORD has shown to be complete for DP (Difference Polynomial-Time).

21. Shortest Reset Words are Even Harder to Decide

Now consider the following decision problem:

SHORTEST-RESET-WORD: *Given a synchronizing automaton $\mathscr{A}$ and a positive integer ℓ, is it true that the minimum length of a reset word for $\mathcal{A}$ is equal to ℓ?*

Assigning the instance $(\mathcal{A}(\psi), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system ψ of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if ψ is not satisfiable. This is a polynomial reduction from the negation of SAT to SHORTEST-RESET-WORD whence the latter problem is coNP-hard. As a corollary, SHORTEST-RESET-WORD cannot belong to NP unless NP = coNP.

Recently, SHORTEST-RESET-WORD has shown to be complete for DP (Difference Polynomial-Time).

21. Shortest Reset Words are Even Harder to Decide

Now consider the following decision problem:

SHORTEST-RESET-WORD: *Given a synchronizing automaton $\mathscr{A}$ and a positive integer ℓ, is it true that the minimum length of a reset word for $\mathcal{A}$ is equal to ℓ?*

Assigning the instance $(\mathcal{A}(\psi), n+1)$ of SHORTEST-RESET-WORD to an arbitrary system ψ of clauses on n variables, one sees that the answer to the instance is "Yes" if and only if ψ is not satisfiable. This is a polynomial reduction from the negation of SAT to SHORTEST-RESET-WORD whence the latter problem is coNP-hard. As a corollary, SHORTEST-RESET-WORD cannot belong to NP unless NP = coNP.

Recently, SHORTEST-RESET-WORD has shown to be complete for DP (Difference Polynomial-Time).

$P^{NP[\log]}$ is the class of all problems that can be solved by a deterministic polynomial-time Turing machine that has an access to an oracle for an NP-complete problem, with the number of queries being logarithmic in the size of the input.

DP is contained in $P^{NP[\log]}$ (for every problem in DP two oracle queries suffice) and the inclusion is believed to be strict.

The problem of computing the minimum length of reset words is complete for the functional analogue $FP^{NP[\log]}$ of $P^{NP[\log]}$.

Finding the shortest reset words may be even harder than computing their length but the exact complexity is not yet known.

22. Computing is Harder than Deciding

$P^{NP[\log]}$ is the class of all problems that can be solved by a deterministic polynomial-time Turing machine that has an access to an oracle for an NP-complete problem, with the number of queries being logarithmic in the size of the input.
DP is contained in $P^{NP[\log]}$ (for every problem in DP two oracle queries suffice) and the inclusion is believed to be strict.

The problem of computing the minimum length of reset words is complete for the functional analogue $FP^{NP[\log]}$ of $P^{NP[\log]}$.

Finding the shortest reset words may be even harder than computing their length but the exact complexity is not yet known.

22. Computing is Harder than Deciding

$P^{NP[\log]}$ is the class of all problems that can be solved by a deterministic polynomial-time Turing machine that has an access to an oracle for an NP-complete problem, with the number of queries being logarithmic in the size of the input.
DP is contained in $P^{NP[\log]}$ (for every problem in DP two oracle queries suffice) and the inclusion is believed to be strict.

The problem of computing the minimum length of reset words is complete for the functional analogue $FP^{NP[\log]}$ of $P^{NP[\log]}$.

Finding the shortest reset words may be even harder than computing their length but the exact complexity is not yet known.

22. Computing is Harder than Deciding

$P^{NP[\log]}$ is the class of all problems that can be solved by a deterministic polynomial-time Turing machine that has an access to an oracle for an NP-complete problem, with the number of queries being logarithmic in the size of the input.
DP is contained in $P^{NP[\log]}$ (for every problem in DP two oracle queries suffice) and the inclusion is believed to be strict.

The problem of computing the minimum length of reset words is complete for the functional analogue $FP^{NP[\log]}$ of $P^{NP[\log]}$.

Finding the shortest reset words may be even harder than computing their length but the exact complexity is not yet known.

23. Non-approximability

However, all known results were consistent with the existence of very good polynomial approximation algorithms for the problem!

Recently, Mikhail Berlinkov, a PhD student of mine, has shown that under NP $\neq$ P, for no k, there may exist a polynomial algorithm that, given a synchronizing automaton, produces a reset word whose length is less than $k \times$minimum possible length of a reset word (CSR-2010).

Open problem: a similar non-approximation result for non-deterministic polynomial algorithms.

Open problem: is approximating within a logarithmic factor possible?

23. Non-approximability

However, all known results were consistent with the existence of very good polynomial approximation algorithms for the problem!

Recently, Mikhail Berlinkov, a PhD student of mine, has shown that under NP $\neq$ P, for no k, there may exist a polynomial algorithm that, given a synchronizing automaton, produces a reset word whose length is less than $k \times$minimum possible length of a reset word (CSR-2010).

Open problem: a similar non-approximation result for non-deterministic polynomial algorithms.

Open problem: is approximating within a logarithmic factor possible?

23. Non-approximability

However, all known results were consistent with the existence of very good polynomial approximation algorithms for the problem!

Recently, Mikhail Berlinkov, a PhD student of mine, has shown that under NP $\neq$ P, for no k, there may exist a polynomial algorithm that, given a synchronizing automaton, produces a reset word whose length is less than $k \times$minimum possible length of a reset word (CSR-2010).

Open problem: a similar non-approximation result for non-deterministic polynomial algorithms.

Open problem: is approximating within a logarithmic factor possible?

23. Non-approximability

However, all known results were consistent with the existence of very good polynomial approximation algorithms for the problem!

Recently, Mikhail Berlinkov, a PhD student of mine, has shown that under NP $\neq$ P, for no k, there may exist a polynomial algorithm that, given a synchronizing automaton, produces a reset word whose length is less than $k \times$minimum possible length of a reset word (CSR-2010).

Open problem: a similar non-approximation result for non-deterministic polynomial algorithms.

Open problem: is approximating within a logarithmic factor possible?