# A New Evolutionary Algorithm for Synchronization

*Jakub Kowalski*, Adam Roman

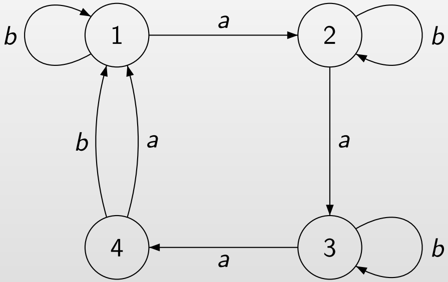
University of Wrocław, Poland

EvoApplications

21.04.2017

Suppose that:

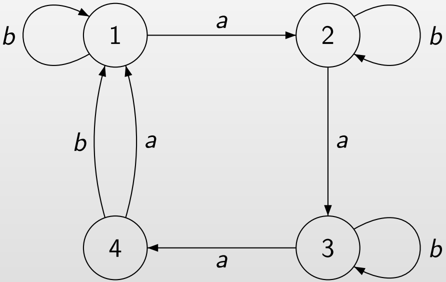
- We have a deterministic finite (semi)automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.




- We do not know in which state the automaton is left.
- Question: is it possible to synchronize the automaton by some word, so that we will know for sure its current state?

Suppose that:

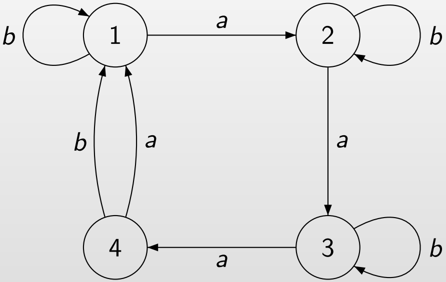
- We have a deterministic finite (semi)automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.




- We do not know in which state the automaton is left.
- Question: is it possible to synchronize the automaton by some word, so that we will know for sure its current state?

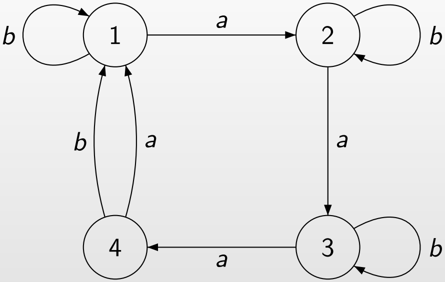
Suppose that:

- We have a deterministic finite (semi)automaton $\mathscr{A} = \langle Q, \Sigma, \delta \rangle$.




- We do not know in which state the automaton is left.
- Question: is it possible to synchronize the automaton by some word, so that we will know for sure its current state?

# Greedy compression algorithm

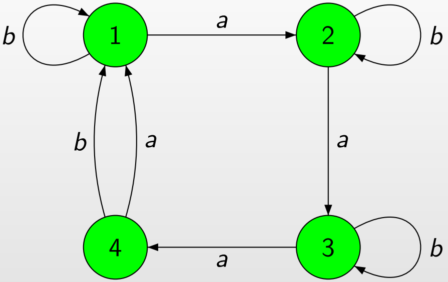



$$w = b\ aab\ abaaab$$

$$Qw =$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.



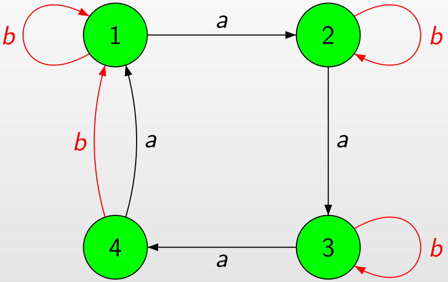

$$w = b\ aab\ abaaab$$

$$Qw =$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



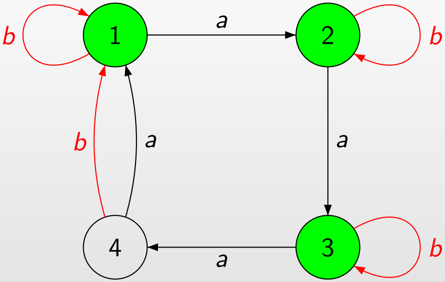

$$w = b\ aab\ abaaab$$

$$Qw = \{1, 2, 3, 4\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



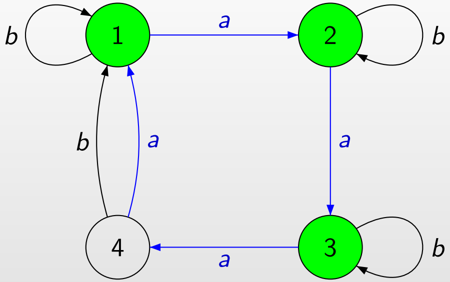

$w = b$ $aab\ abaaab$

$Qw = \{1, 2, 3\}$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



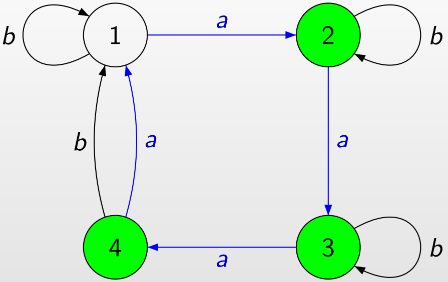

$$w = b\ aab\ abaaab$$

$$Qw = \{1, 2, 3\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



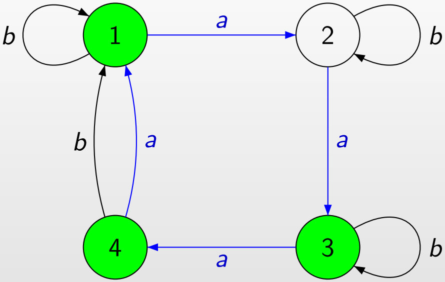

$$w = b\ aab\ abaaab$$

$$Qw = \{2, 3, 4\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



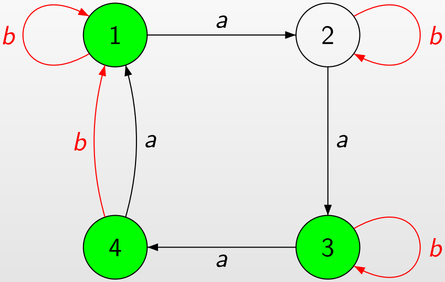

$$w = b\ aab\ abaaab$$

$$Qw = \{1, 3, 4\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



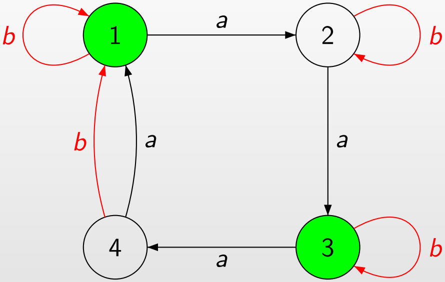

$$w = b\ aab\ abaaab$$

$$Qw = \{1, 3, 4\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



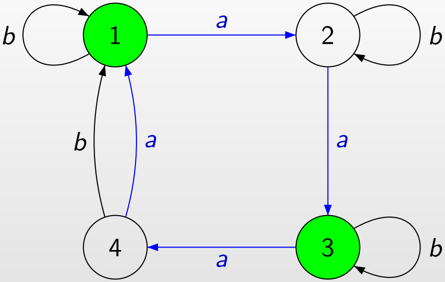

$$w = b\ aab\ abaaab$$

$$Qw = \{1, 3\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



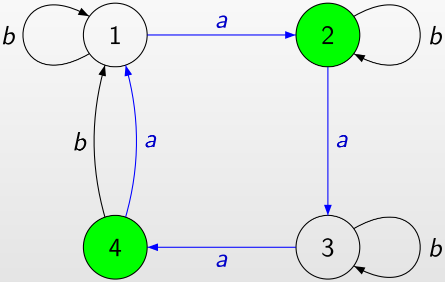

$$w = b\ aab\ abaaab$$

$$Qw = \{1, 3\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



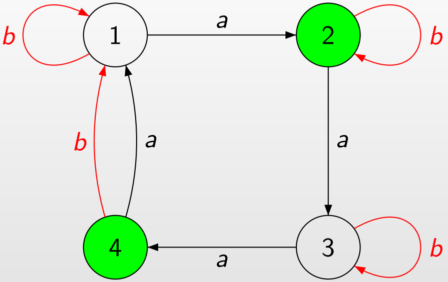

$$w = b\ aab\ abaaab$$

$$Qw = \{2, 4\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



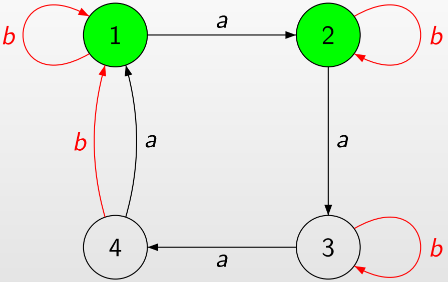

$$w = b\ aab\ abaaab$$

$$Qw = \{2, 4\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm

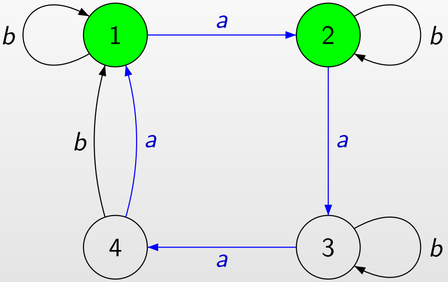



$$w = b\ aab\ ab\color{red}{b}\color{gray}{aaab}$$

$$Qw = \{1, 2\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.



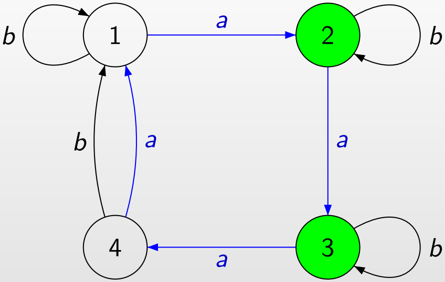

$$w = b\ aab\ ab aaab$$

$$Qw = \{1, 2\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



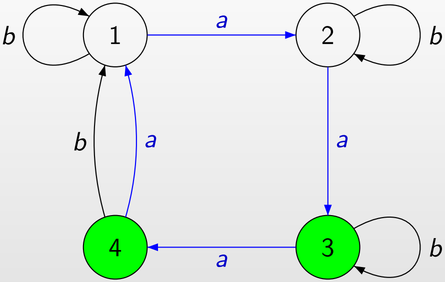

$$w = b\ aab\ aba\textcolor{lightgray}{aab}$$

$$Qw = \{2, 3\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



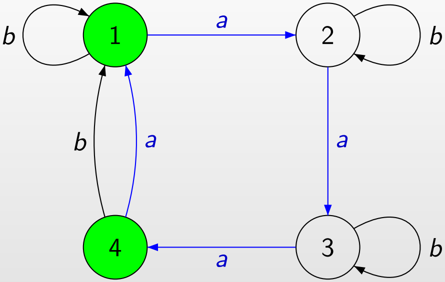

$$w = b\ aab\ abaaab$$

$$Qw = \{3, 4\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



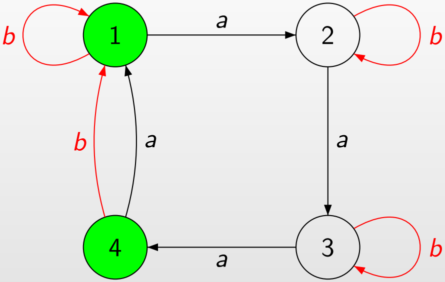

$$w = b\ aab\ abaaab$$

$$Qw = \{1, 4\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



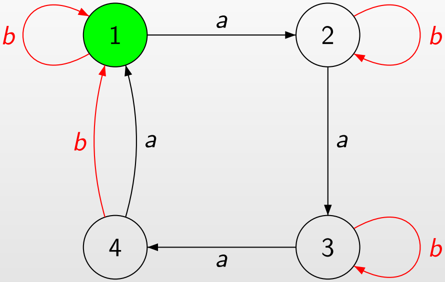

$$w = b\ aab\ abaaab$$

$$Qw = \{1, 4\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm



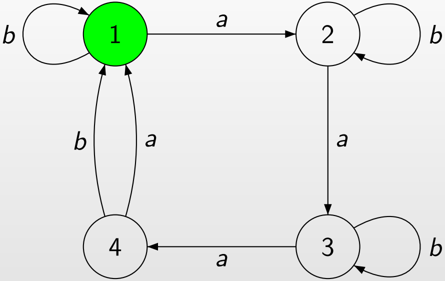

$$w = b\ aab\ abaaab$$

$$Qw = \{1\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Greedy compression algorithm




$$w = b\ aab\ abaaab$$
$$Qw = \{1\}$$

Word $w$ is a *reset word* (*synchronizing word*).

Thus, the automaton $\mathscr{A}$ is *synchronizing*.

# Computing reset words

Given an $n$-state automaton, it is easy to check if it is synchronizing and, if so, find some reset word.
The found reset words can have length $O(n^3)$.

---

The problem of finding a shortest reset word is difficult:

- Deciding the existence of a reset word $\leq k$ is NP-complete (Eppstein 1990).
- Computing the length of the shortest reset words is $\mathrm{FP}^{\mathrm{NP}[\log]}$-complete (Olschewski, Ummels 2010).
- No polynomial algorithm exists for approximating the length of the shortest reset words within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$ (assuming $\mathrm{P} \neq \mathrm{NP}$) (Gawrychowski, Straszak 2015).

# Computing reset words

Given an $n$-state automaton, it is easy to check if it is synchronizing and, if so, find some reset word.
The found reset words can have length $O(n^3)$.

---

The problem of finding a shortest reset word is difficult:

- Deciding the existence of a reset word $\leq k$ is $\mathrm{NP}$-complete (Eppstein 1990).
- Computing the length of the shortest reset words is $\mathrm{FP}^{\mathrm{NP}[\log]}$-complete (Olschewski, Ummels 2010).
- No polynomial algorithm exists for approximating the length of the shortest reset words within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$ (assuming $\mathrm{P} \neq \mathrm{NP}$) (Gawrychowski, Straszak 2015).

How large can be the length of the *shortest* reset words of an $n$-state synchronizing automaton?

**The Černý conjecture (Černý 1969)**

Every synchronizing automaton has a reset word of length at most

$$(n-1)^2$$

The bound can be met for every $n$ by the *Černý automata*:

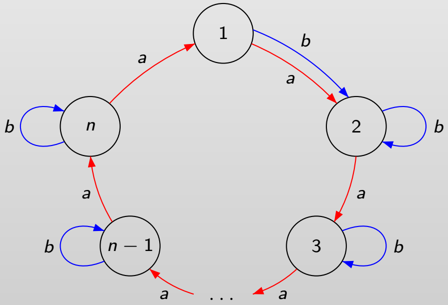
How large can be the length of the *shortest* reset words of an $n$-state synchronizing automaton?

**The Černý conjecture (Černý 1969)**

Every synchronizing automaton has a reset word of length at most

$$(n-1)^2$$

The bound can be met for every $n$ by the *Černý automata*:



- proved for various classes of automata:
  *Oriented*, *Eulerian*, *One-cluster*, *Aperiodic*, $|\Sigma| = 2 \wedge |Q| \leq 12$, ...

- general upper bound:

**Pin–Frankl, 1983**

The length of the shortest reset words is at most

$$(n^3 - n)/6 - 1 \qquad (n \geq 4)$$

**Szykuła, 2017**

The length of the shortest reset words is at most

$$(15617n^3 + 7500n^2 + 9375n - 31250)/93750 \qquad (n \geq 4)$$

(coefficient $1/6$ at $n^3$ improved by $4/46875$).

# The Černý conjecture

- proved for various classes of automata:
  *Oriented*, *Eulerian*, *One-cluster*, *Aperiodic*, $|\Sigma| = 2 \wedge |Q| \leq 12$, ...
- general upper bound:

**Pin–Frankl, 1983**

The length of the shortest reset words is at most

$$(n^3 - n)/6 - 1 \qquad (n \geq 4)$$

**Szykuła, 2017**

The length of the shortest reset words is at most

$$(15617n^3 + 7500n^2 + 9375n - 31250)/93750 \qquad (n \geq 4)$$

(coefficient $1/6$ at $n^3$ improved by $4/46875$).

# The Černý conjecture

- proved for various classes of automata:
  *Oriented*, *Eulerian*, *One-cluster*, *Aperiodic*, $|\Sigma| = 2 \wedge |Q| \leq 12, \ldots$
- general upper bound:

**Pin–Frankl, 1983**

The length of the shortest reset words is at most

$$(n^3 - n)/6 - 1 \qquad (n \geq 4)$$

**Szykuła, 2017**

The length of the shortest reset words is at most

$$(15617n^3 + 7500n^2 + 9375n - 31250)/93750 \qquad (n \geq 4)$$

(coefficient $1/6$ at $n^3$ improved by $4/46875$).

# Extremal examples: infinite series

| Shortest reset words | 2-letter automata | 3-letter automata |
|---|---|---|
| $n^2 - 2n + 1 = (n-1)^2$ | $\mathscr{C}_n$ (Černý automaton) | |
| ... | | |
| $n^2 - 3n + 4$ | $\mathscr{D}'_n$ | |
| $n^2 - 3n + 3$ | $\mathscr{W}_n$ $\mathscr{F}_{n\ \text{(odd)}}$ | $\mathscr{M}_n$ |
| $n^2 - 3n + 2$ | $\mathscr{D}''_n$ $\mathscr{E}_n$ $\mathscr{B}_{n\ \text{(odd)}}$ | $\mathscr{M}'_n$ |
| ... | | |
| $n^2 - 4n + 7$ | $\mathscr{G}_{n\ \text{(odd)}}$ | |
| ... | ... | |

# Applications

- Part orienters;
- Finding location on a map/graph;
- Resetting biocomputers;
- Test generation for sequential circuits;
- Model-based testing of reactive systems;
- Error corrections of compressed data.

# Algorithm

## Exact algorithms (exponential) finding a shortest reset word

- Standard BFS in the power automaton (Hennie 1964);
- Using Binary Decision Diagrams (Rho, Somenzi, Pixley 1993);
- Semigroup algorithm (Trahtman 2006);
- Reduction to SAT (Skvortsov, Tipikin 2011);
- Using Answer Sets Programming (Güniçen, Erdem, Yenigün 2013);
- The bidirectional algorithm (Kisielewicz, Kowalski, Szykuła 2013).

## Inexact algorithms (polynomial) finding a (short) reset word

- An algorithm in $O(n^4)$ time (Natarajan 1986);
- The Eppstein algorithm in $O(n^3)$ time (Eppstein 1990);
- Semigroup and cycle algorithms (Trahtman 2006);
- A genetic algorithm (Roman 2009);
- $\lceil \frac{n-1}{p-1} \rceil$-approximation algorithm in $O(pn^p + n^4/p)$ time (Gerbush, Heeringa 2011);
- SynchroP and SynchroPL algorithms in $O(n^5)$ time, FastSynchro in $O(n^4)$ time (Roman 2005; Kudłacik, Roman, Wagner 2012);
- The Cut-Off IBFS algorithm in $O(cn^4)$ time (Roman, Szykuła 2015).

## Exact algorithms (exponential) finding a shortest reset word

- Standard BFS in the power automaton (Hennie 1964);
- Using Binary Decision Diagrams (Rho, Somenzi, Pixley 1993);
- Semigroup algorithm (Trahtman 2006);
- Reduction to SAT (Skvortsov, Tipikin 2011);
- Using Answer Sets Programming (Güniçen, Erdem, Yenigün 2013);
- The bidirectional algorithm (Kisielewicz, Kowalski, Szykuła 2013).

## Inexact algorithms (polynomial) finding a (short) reset word

- An algorithm in $O(n^4)$ time (Natarajan 1986);
- The Eppstein algorithm in $O(n^3)$ time (Eppstein 1990);
- Semigroup and cycle algorithms (Trahtman 2006);
- A genetic algorithm (Roman 2009);
- $\lceil \frac{n-1}{p-1} \rceil$-approximation algorithm in $O(pn^p + n^4/p)$ time (Gerbush, Heeringa 2011);
- SynchroP and SynchroPL algorithms in $O(n^5)$ time, FastSynchro in $O(n^4)$ time (Roman 2005; Kudłacik, Roman, Wagner 2012);
- The Cut-Off IBFS algorithm in $O(cn^4)$ time (Roman, Szykuła 2015).

# Genetic Algorithm (Roman 2009)

## SynchroGA

- chromosomes are various length words over the $\Sigma$;
- probability distribution of letters based on the construction of automaton;
- probabilities of mutation and crossover increasing when in stagnation;
- one-point crossover;
- three types of mutations: letter flip, insert random subword, delete random subword;
- fitness function:

$$f(w) = \frac{(|Q| - |Qw|)^4}{\sqrt[4]{|w|}}.$$

# Knowledge-based approach

## Rank-based model

Instead of returning only $rk(w) = |Qw|$ (the word's rank), the evaluation process returns the rank of every prefix of $w$, forming a vector $r_1, r_2, \ldots, r_{|w|}$.

- this enhancement do not increase the complexity of evaluation;
- it allows to trivially improve words $w$ such that $rk(w) = 1$, $w = vx$ and $rk(v) = 1$.

## FI-2POP (Kimbrough, et al. 2008)

Keep separate populations for best of feasible and infeasible individuals.

- Feasible population (synchronizing words):

  minimize the length of the word

- Infeasible population (non-synchronizing words):

  minimize the rank of the word

# Knowledge-based approach

## Rank-based model

Instead of returning only $rk(w) = |Qw|$ (the word's rank), the evaluation process returns the rank of every prefix of $w$, forming a vector $r_1, r_2, \ldots, r_{|w|}$.

- this enhancement do not increase the complexity of evaluation;
- it allows to trivially improve words $w$ such that $rk(w) = 1$, $w = vx$ and $rk(v) = 1$.

## FI-2POP (Kimbrough, et al. 2008)

Keep separate populations for best of feasible and infeasible individuals.

- Feasible population (synchronizing words):

  minimize the length of the word
- Infeasible population (non-synchronizing words):

  minimize the rank of the word

# Operators

## Initialization

We generate twice the size of the population random words.

- *uniform(l)* $(\frac{1}{|\Sigma|})$
- *rank-based(l)* $(|Qa|)$
- *reverse-rank-based(l)* $(|Q| - |Qa| + 1)$

## Selection

- *tournament(s)*
- **uniform** $(\textit{tournament(1)})$

## Replication

Best of childern and parents for both feasible and infeasible population. Keep both populations equal.

# Operators

## Initialization

We generate twice the size of the population random words.

- *uniform(l)* $(\frac{1}{|\Sigma|})$
- *rank-based(l)* $(|Qa|)$
- *reverse-rank-based(l)* $(|Q| - |Qa| + 1)$

## Selection

- *tournament(s)*
- **uniform** (*tournament(1)*)

## Replication

Best of childern and parents for both feasible and infeasible population. Keep both populations equal.

# Operators

## Initialization

We generate twice the size of the population random words.

- *uniform(l)* $(\frac{1}{|\Sigma|})$
- *rank-based(l)* $(|Qa|)$
- *reverse-rank-based(l)* $(|Q| - |Qa| + 1)$

## Selection

- *tournament(s)*
- **uniform** (*tournament(1)*)

## Replication

Best of childern and parents for both feasible and infeasible population.
Keep both populations equal.

# Operators

## Crossover

We introduce *rank-based* operators as operating on compressing words instead of single letters.

- *one-point* and *rank-based one-point*;
- *two-point* and *rank-based two-point*;
- *uniform* and *rank-based uniform*;

## Mutation

Most designed operators aim to the specific population.

- (IF/FI) *letter-exchange(p)*
- (IF) *letter-insertion(p)* / *adaptive letter-insertion(p)*
- (IF) *lastwords(p)*
- (IF) *compressing-word-insertion(p)*
- (FI) *letter-deletion(p)* / *adaptive letter-deletion(p)*

# Operators

## Crossover

We introduce *rank-based* operators as operating on compressing words instead of single letters.

- *one-point* and *rank-based one-point*;
- *two-point* and *rank-based two-point*;
- *uniform* and *rank-based uniform*;

## Mutation

Most designed operators aim to the specific population.

- (IF/FI) *letter-exchange(p)*
- (IF) *letter-insertion(p)* / *adaptive letter-insertion(p)*
- (IF) *lastwords(p)*
- (IF) *compressing-word-insertion(p)*
- (FI) *letter-deletion(p)* / *adaptive letter-deletion(p)*

# Experiments and Results

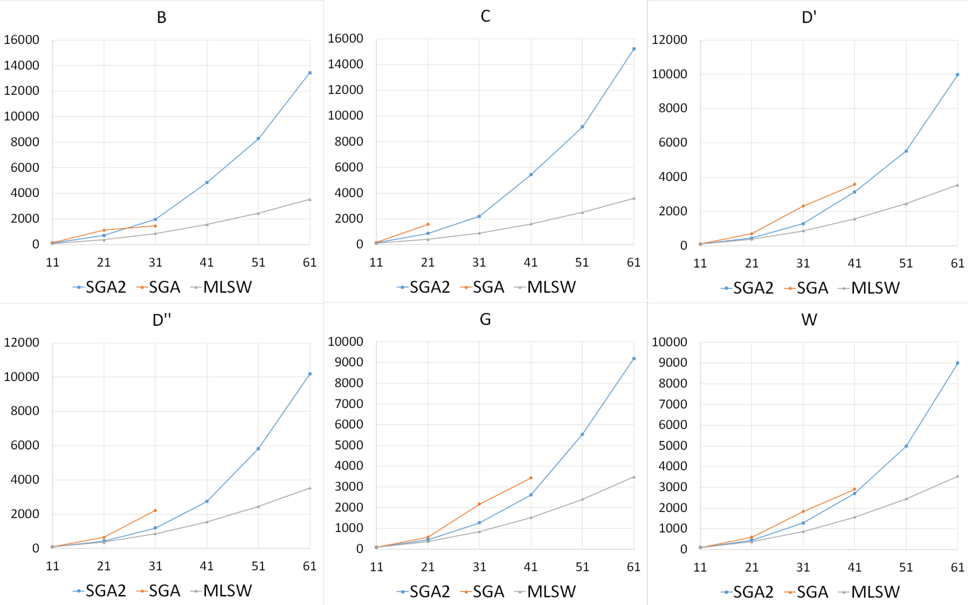
# Experiments

1. Selecting the best settings for operators
   - random automata: $|\Sigma| = 2, n \in \{25, 50, 75, 100\}$; $|\Sigma| \in \{3, 4\}, n \in \{25, 50, 75\}$;
   - population size: 30+30; maximum generations: 500;
   - hill-climbing search for settings improvement ($>$ 110 tested).
2. Extremal automata and comparison with SynchroGA
   - Extremal series of automata: $B, C, D', D'', E, F, G, H, W$; $n \in \{11, 21, 31, 41, 51, 61\}$;
   - population size: 20+20 / 40; maximum generations: 1000.
3. Large random automata
   - binary automata with $n \in \{100, 200, 300, 400, 500, 600\}$;
   - population size: 30+30; maximum generations: 500;
   - Eppstein and Cut-Off IBFS algorithms for comparison.

# Operators selection

| % MLSW | avg. gen. | Ratio: LSW /: MLSW | EPPLSW | Operators: Init | $C_{FI}$ | $C_{IF}$ | $M_{FI}$ | $M_{IF}$ |
|---|---|---|---|---|---|---|---|---|
| 75.68 | 87.54 | 1.0233 | 0.6880 | rb(1.0) | 1pL | 2pRB | ald(0.065) | lw |
| 75.67 | 86.19 | 1.0229 | 0.6878 | uni(2.0) | 1pL | 2pRB | ald(0.065) | lw |
| 75.52 | 86.14 | 1.0231 | 0.6879 | uni(2.5) | 1pL | 2pRB | ald(0.065) | lw |
| 75.50 | 87.14 | 1.0232 | 0.6880 | rb(2.0) | 1pL | 2pRB | ald(0.065) | lw |
| 75.50 | 85.76 | 1.0231 | 0.6879 | uni(1.0) | 1pL | 2pRB | ald(0.065) | lw |
| 75.46 | 84.41 | 1.0234 | 0.6881 | uni(1.0) | 1pL | 2pRB | ald(0.050) | lw |
| 75.45 | 86.71 | 1.0231 | 0.6879 | rb(0.5) | 1pL | 2pRB | ald(0.065) | lw |
| 75.36 | 87.60 | 1.0231 | 0.6880 | uni(1.0) | 1pL | 2pRB | ald(0.080) | lw |
| 75.36 | 85.81 | 1.0232 | 0.6880 | uni(1.5) | 1pL | 2pRB | ald(0.065) | lw |
| 75.27 | 84.76 | 1.0233 | 0.6880 | uni(0.5) | 1pL | 2pRB | ald(0.065) | lw |
| 75.16 | 83.55 | 1.0239 | 0.6884 | uni(1.0) | 1pL | 2pRB | ald(0.040) | lw |
| 75.06 | 85.41 | 1.0234 | 0.6881 | rrb(2.0) | 1pL | 2pRB | ald(0.065) | lw |
| 75.03 | 85.17 | 1.0234 | 0.6882 | rrb(1.0) | 1pL | 2pRB | ald(0.065) | lw |
| 74.93 | 88.96 | 1.0236 | 0.6883 | uni(2.0) | 2pRB | 2pRB | ald(0.065) | lw |
| 74.87 | 89.11 | 1.0236 | 0.6883 | uni(1.0) | 2pRB | 2pRB | ald(0.065) | lw |
| 74.87 | 90.50 | 1.0239 | 0.6884 | rb(1.0) | 2pRB | 2pRB | ald(0.065) | lw |
| 74.82 | 90.25 | 1.0239 | 0.6884 | rb(2.0) | 2pRB | 2pRB | ald(0.065) | lw |
| 74.82 | 91.39 | 1.0237 | 0.6883 | uni(1.0) | 2pRB | 2pRB | ald(0.080) | lw |
| 74.79 | 88.85 | 1.0249 | 0.6891 | uni(1.5) | 1pL | 1pRB | ald(0.065) | cwi |
| 74.78 | 88.85 | 1.0249 | 0.6891 | uni(1.0) | 1pL | 1pRB | ald(0.065) | cwi |
| | | | | ...52: | | | | |
| 73.75 | 97.08 | 1.0260 | 0.6898 | uni(1.0) | 1pL | 1pL | ald(0.065) | ali(0.04) |

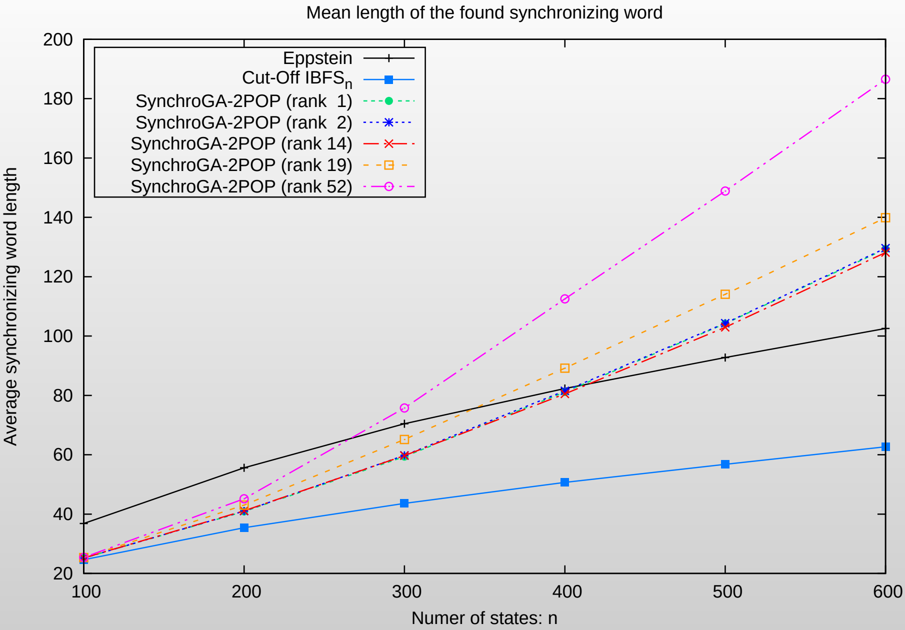
# Extremal automata – synchronizing word lenghts



# Extremal automata – rank comparison

| $\mathcal{A}$ | best rk SGA2 | best rk SGA | mean rk SGA2 | mean rk SGA | $\mathcal{A}$ | best rk SGA2 | best rk SGA | mean rk SGA2 | mean rk SGA |
|---|---|---|---|---|---|---|---|---|---|
| $B_{11}$ | 1 | 1 | 1.45 | 1 | $C_{11}$ | 1 | 1 | 1 | 1 |
| $B_{21}$ | 1 | 1 | 1.05 | 1 | $C_{21}$ | 1 | 1 | 1 | 1 |
| $B_{31}$ | 1 | 1 | 1 | 1.90 | $C_{31}$ | 1 | 2 | 1 | 2.35 |
| $B_{41}$ | 1 | 2 | 1 | 2.75 | $C_{41}$ | 1 | 4 | 1 | 5.30 |
| $B_{51}$ | 1 | - | 1 | - | $C_{51}$ | 1 | - | 1 | - |
| $B_{61}$ | 1 | - | 1 | - | $C_{61}$ | 1 | - | 1 | - |
| $D'_{11}$ | 1 | 1 | 1.05 | 1 | $D''_{11}$ | 1 | 1 | 1.45 | 1 |
| $D'_{21}$ | 1 | 1 | 1 | 1 | $D''_{21}$ | 1 | 1 | 1.05 | 1 |
| $D'_{31}$ | 1 | 1 | 1 | 1.1 | $D''_{31}$ | 1 | 1 | 1 | 1.35 |
| $D'_{41}$ | 1 | 1 | 1 | 1.9 | $D''_{41}$ | 1 | 2 | 1 | 2 |
| $D'_{51}$ | 1 | - | 1 | - | $D''_{51}$ | 1 | - | 1 | - |
| $D'_{61}$ | 1 | - | 1 | - | $D''_{61}$ | 1 | - | 1 | - |
| $G_{11}$ | 1 | 1 | 1.05 | 1 | $W_{11}$ | 1 | 1 | 1 | 1 |
| $G_{21}$ | 1 | 1 | 1 | 1 | $W_{21}$ | 1 | 1 | 1 | 1 |
| $G_{31}$ | 1 | 1 | 1 | 1.1 | $W_{31}$ | 1 | 1 | 1 | 1.1 |
| $G_{41}$ | 1 | 1 | 1 | 2 | $W_{41}$ | 1 | 1 | 1 | 1.75 |
| $G_{51}$ | 1 | - | 1 | - | $W_{51}$ | 1 | - | 1 | - |
| $G_{61}$ | 1 | - | 1 | - | $W_{61}$ | 1 | - | 1 | - |

# Large random automata



# Thank you