Learning Finite-State Machines: Conserving Fitness Evaluations by Marking Used Transitions

Daniil Chivilikhin and Vladimir Ulyantsev

National Research University of IT, Mechanics and Optics

St. Petersburg, Russia

Machine Learning Applications in Software Engineering @ ICMLA'13

December 6, 2013

Outline

- Motivation and scope
- Proposed idea
- Experiments
  - Artificial Ant problem
  - Test-based EFSM induction
- Conclusion

Motivation and scope

Motivation: Reliable software

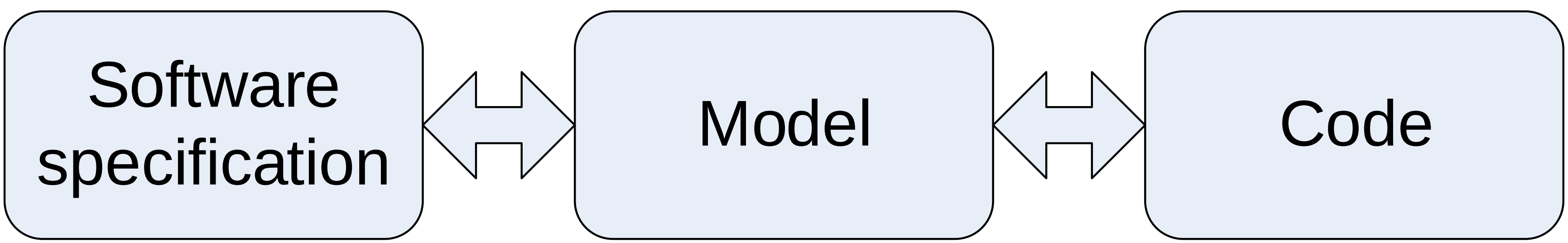
- Systems with high cost of failure
  - Energy industry
  - Aircraft industry
  - Space industry
  - …
- We want to have **reliable software**
  - Testing is not enough
  - **Verification** is needed

Verification

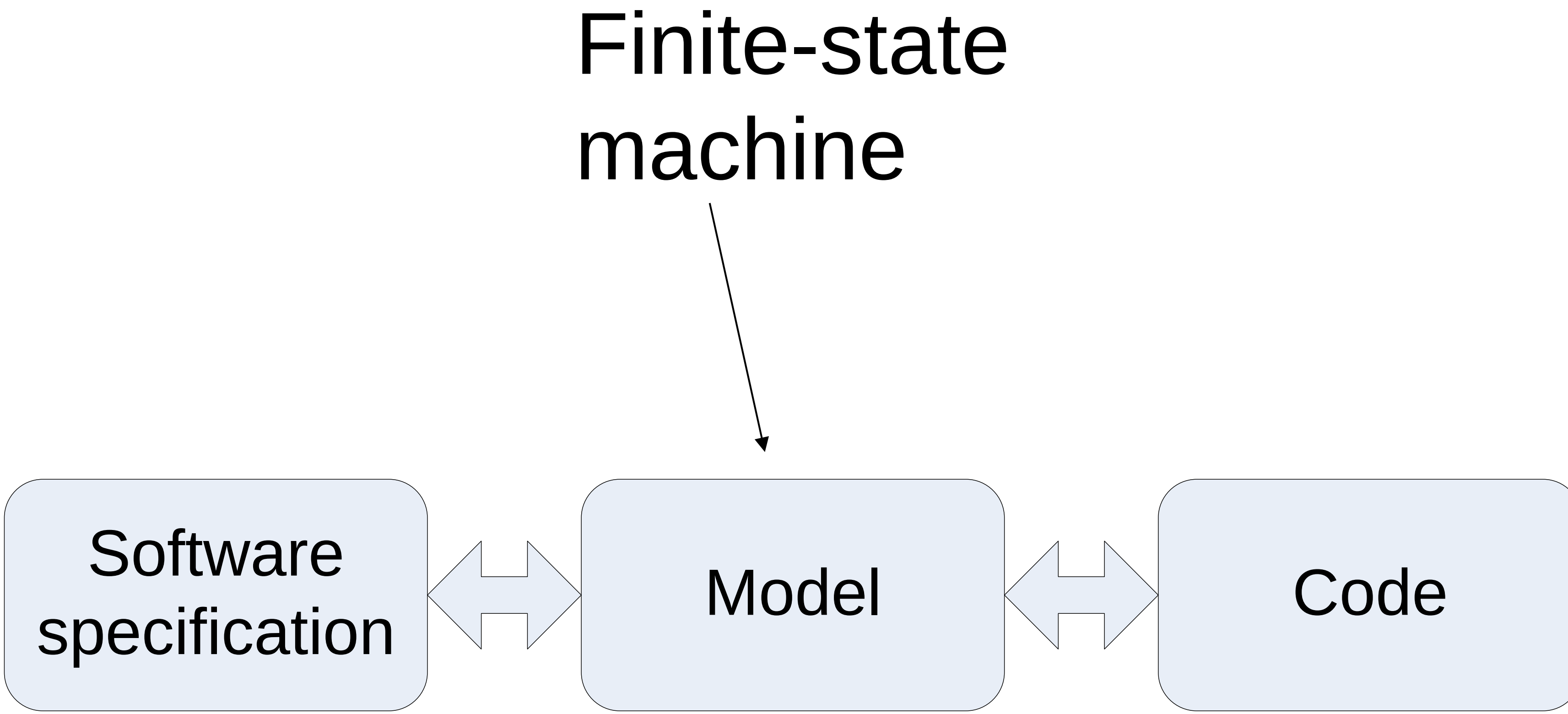
- Checking temporal rules (e.g. *LTL*)
- Software verification can be harder than software development
- Need to make software that satisfies *LTL*-specification by design
- How?

Model-driven development

- Automated software engineering
- Model-driven development

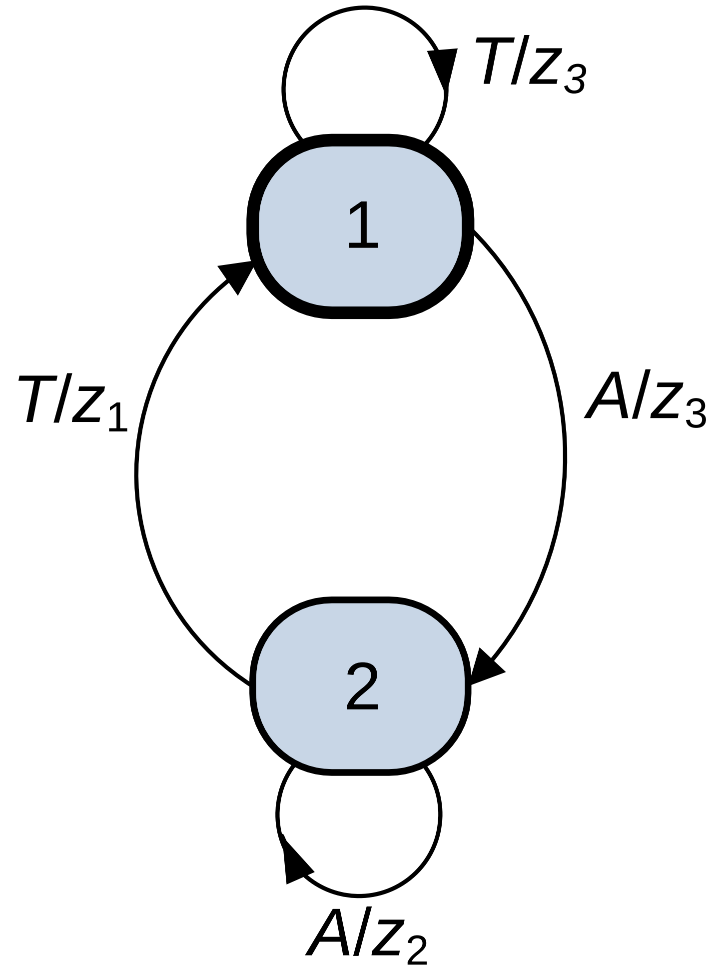


Model-driven development



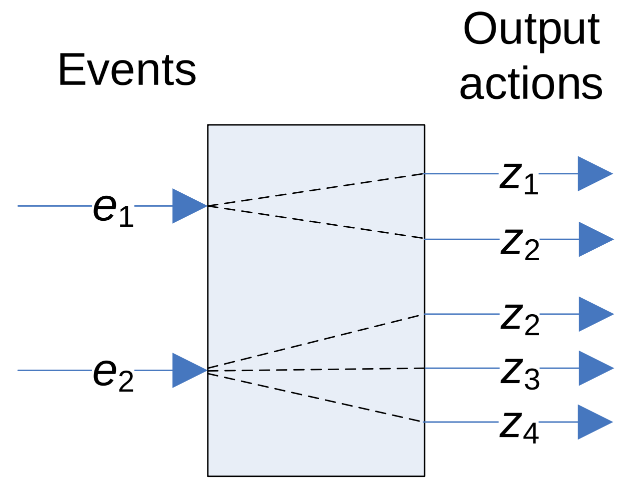
Finite-State Machine

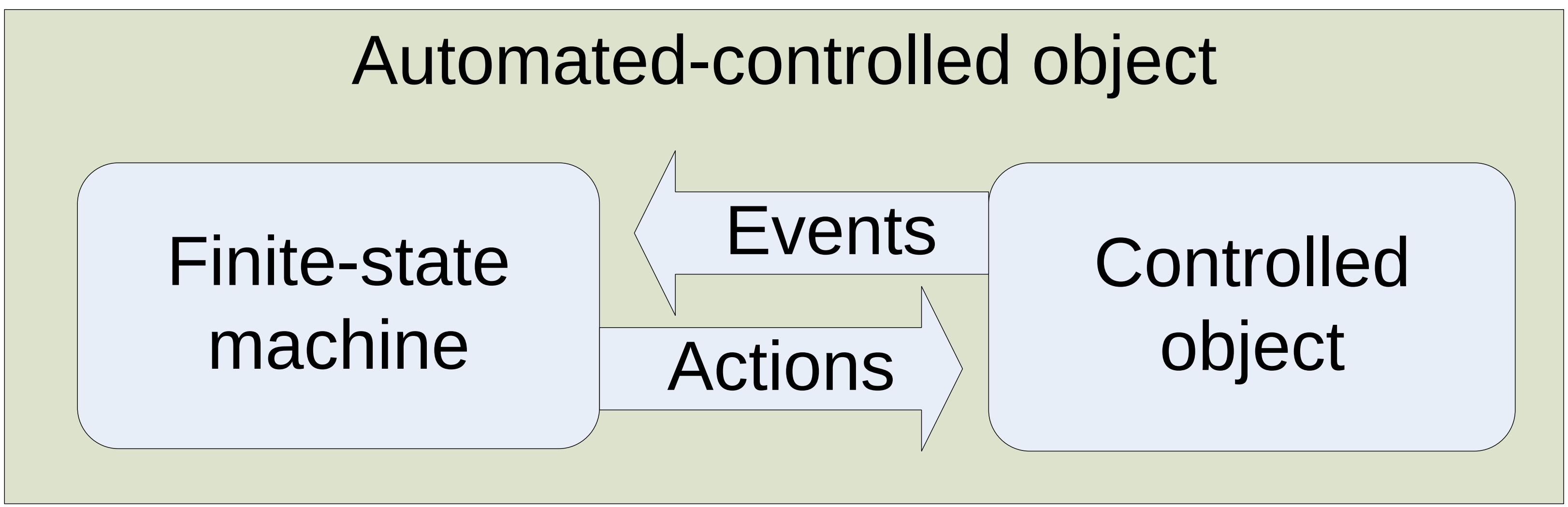
- Σ – set of input events
- Δ – set of output actions
- δ: $S \times \Sigma \to S$ – transition function
- λ: $S \times \Sigma \to \Delta$ – actions function



Automata-based programming

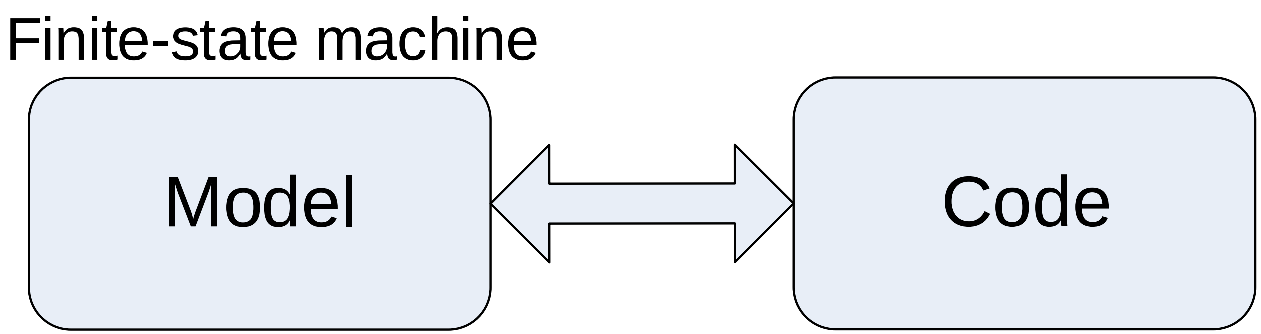
Design programs with complex behavior as automated-controlled objects





Automata-based programming: advantages

- Model before programming code, not vice versa



- Possibility of program verification using *Model Checking*
- You can check temporal properties (LTL)

Issues

- Hard to build an FSM with desired structure and behavior
- One of approaches – mutation-based metaheuristics

Mutation-based FSM learning

- Evolution Strategies (ES)
- Genetic Algorithms (GA)
- Mutation-based Ant Colony Optimization (MuACO)

All these algorithms use FSM **mutations**

Mutation-Based Ant Colony Optimization

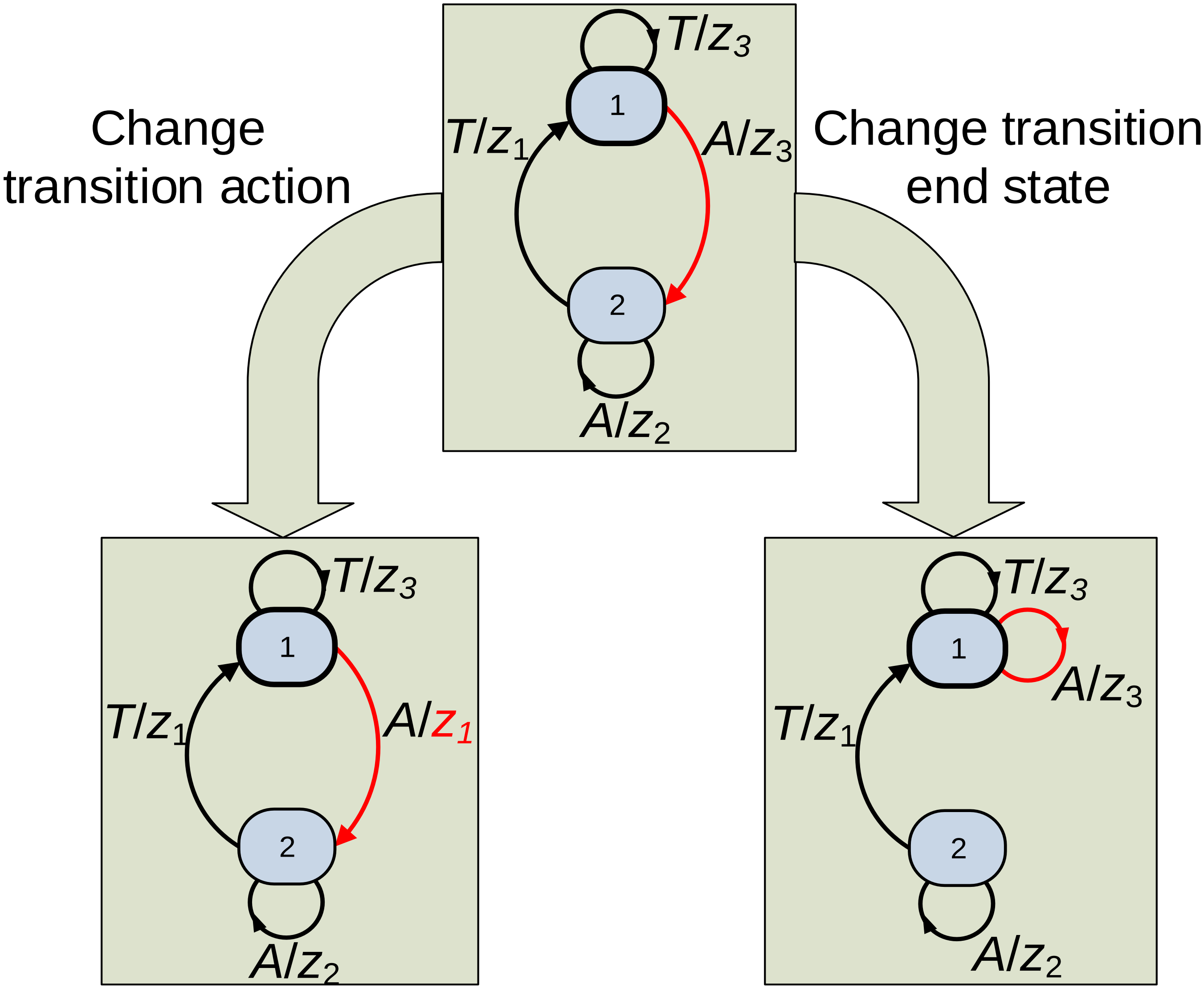
- Proposed by the authors of this work in 2012
- Based on Ant Colony Optimization
- Can be thought of as an ES “with memory”
- No time to go into detail ☹

Solution representation

Transition table		
δ	**Event**	
State	*A*	*T*
1	1	2
2	2	1

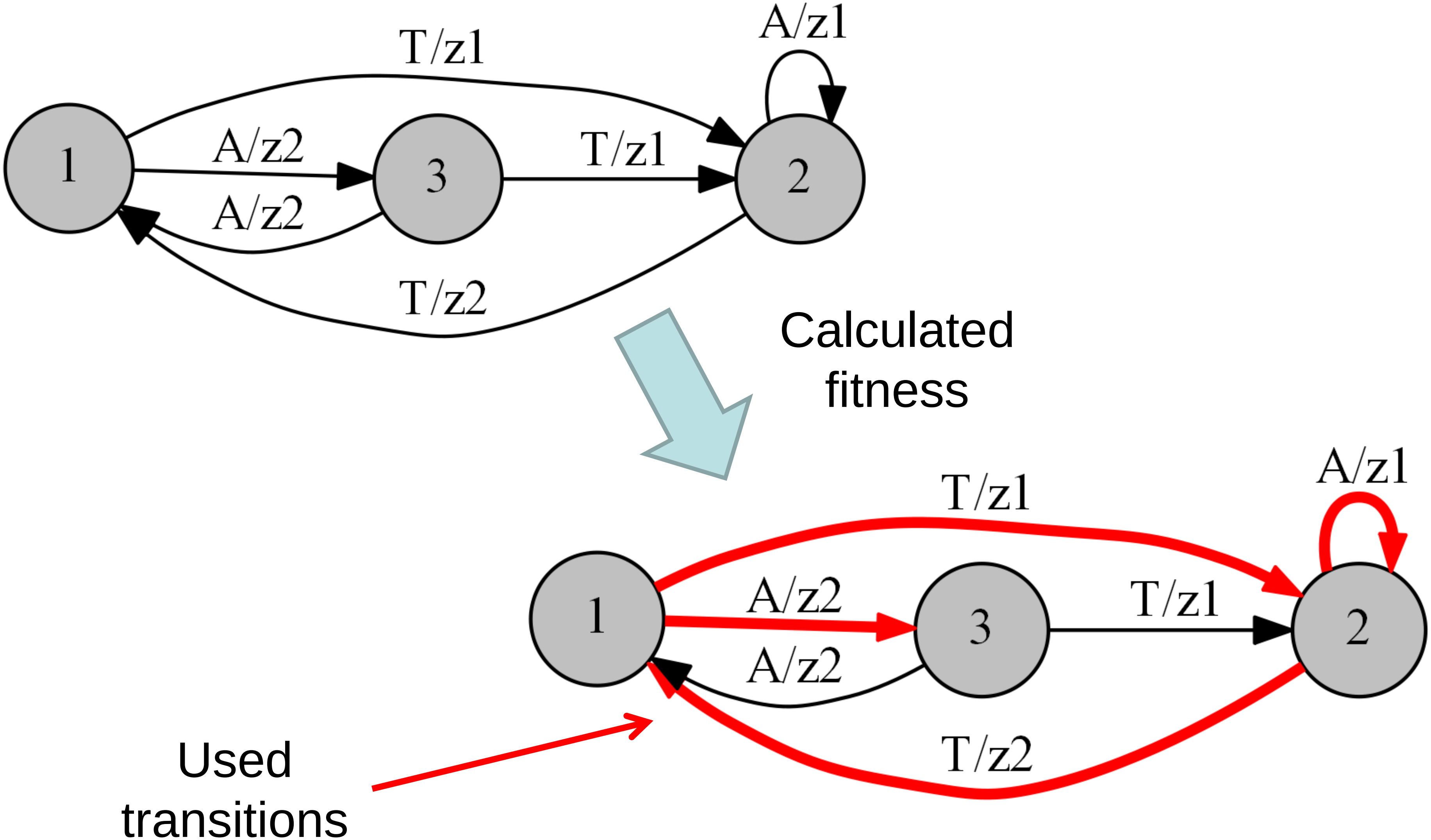
Output table		
λ	**Event**	
State	*A*	*T*
1	z_1	z_2
2	z_2	z_3

FSM Mutations



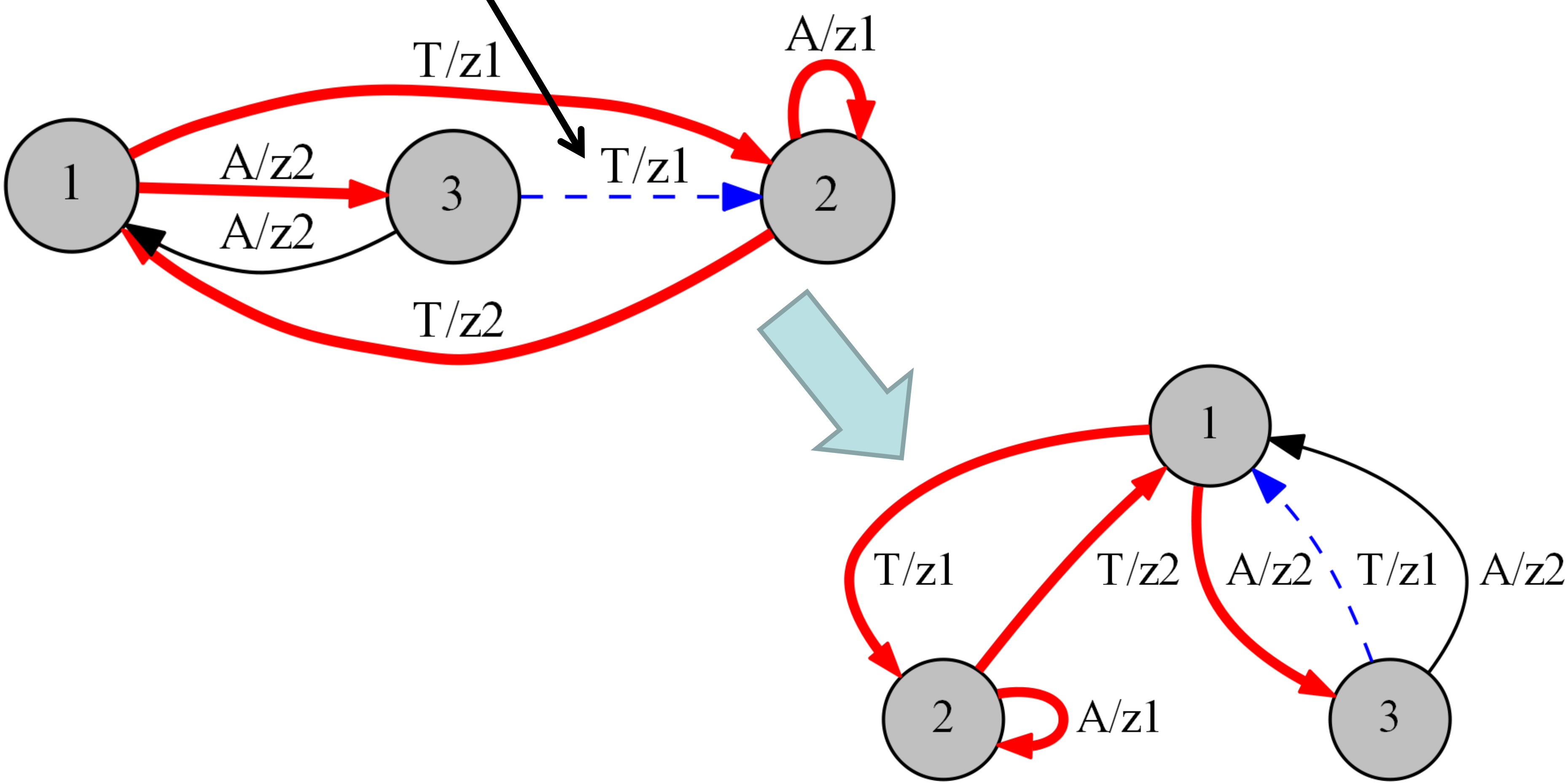
Proposed idea

Fitness evaluation: used transitions

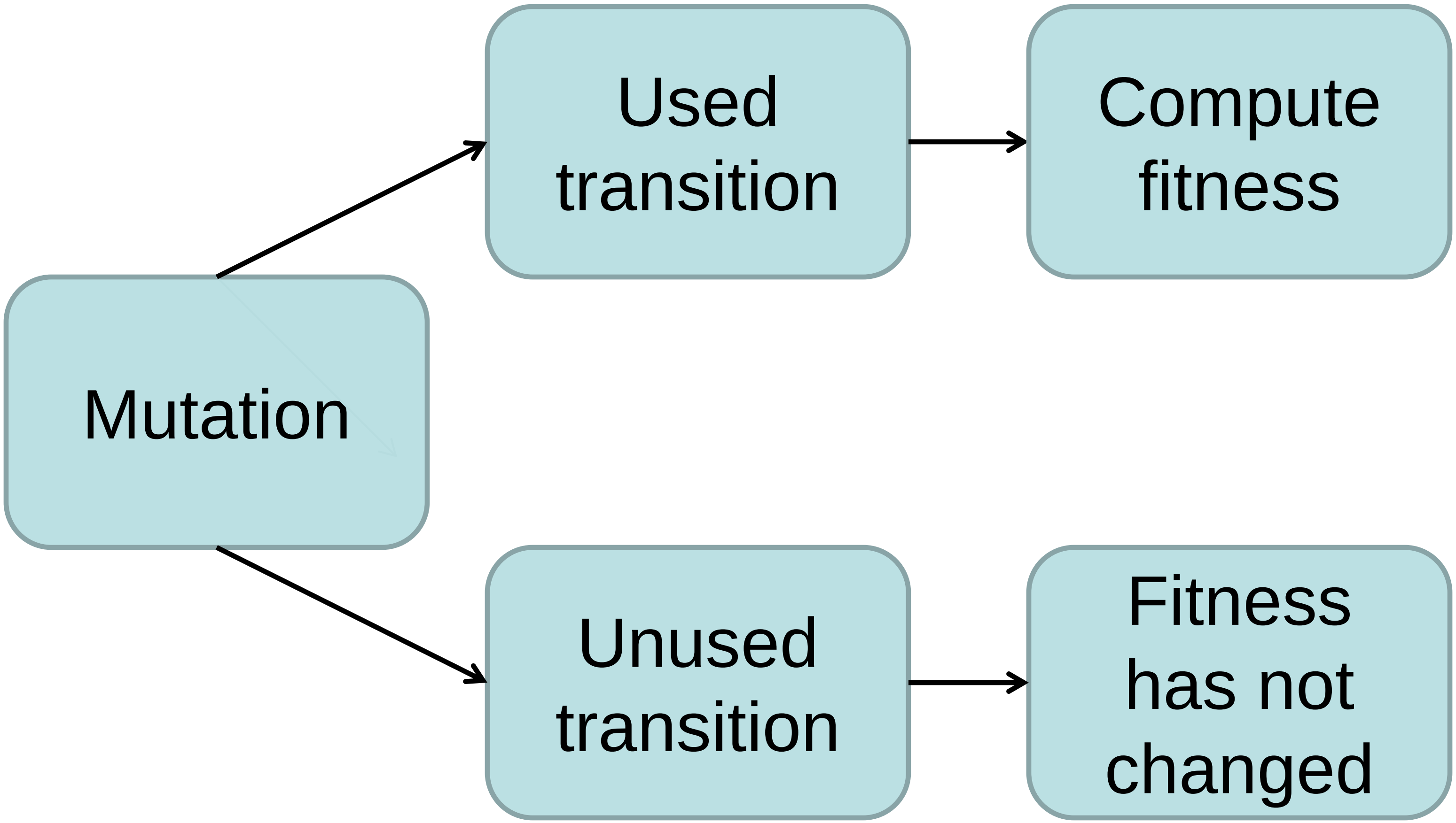


Mutation

Mutate this transition



Observation



Challenge

OK, found a way not to calculate fitness of FSMs in some cases

1. Can we design an efficient **implementation**?
2. Will it make a difference in **performance**?
3. **Limits** of applicability?

Implementation

- Store an array of transition usage marks for each FSM
- Mark used transitions during fitness evaluation
- Copy marks when not calculating fitness

Domain knowledge

- Black box – no domain knowledge
- General domain knowledge about FSMs
- Problem-specific domain knowledge

Experiments

Experiments: Purpose

- Does it make a difference?
- How much resources does it require?

Experiments: Algorithms

- Evolutionary strategy (ES)
- Genetic algorithm (GA)
- Mutation-Based Ant Colony Optimization for learning FSMs (MuACOsm)

Experiments: Problems

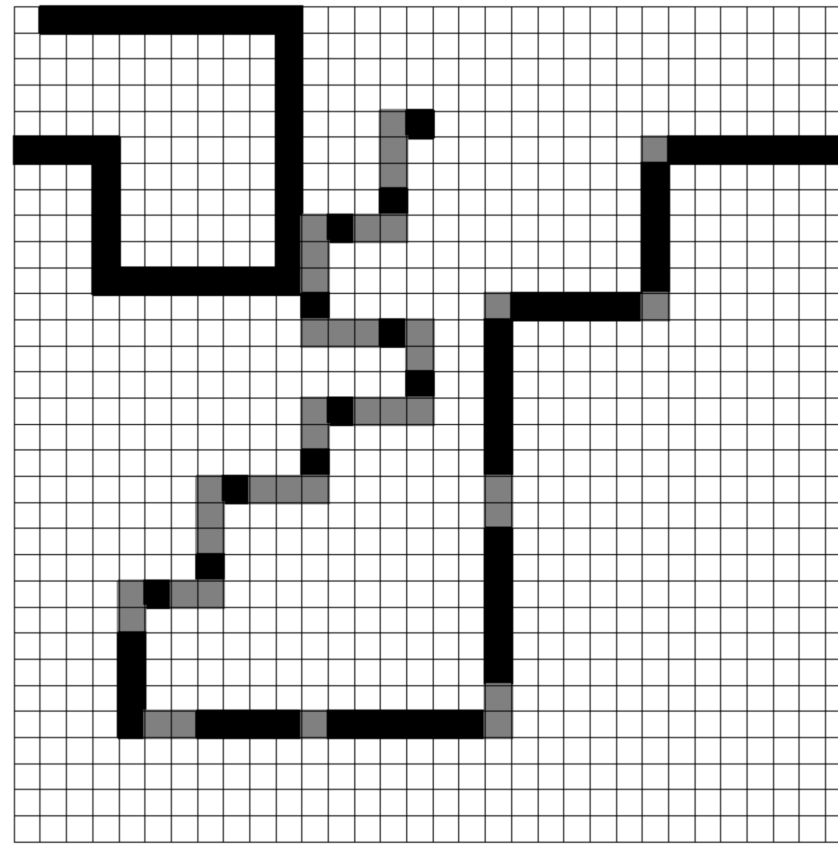
1. Artificial Ant Problem
2. Learning EFSMs from tests

General experimental setup

1. Tune each algorithm for time t_{tune}
   – Using full factorial design of experiment
2. Run each algorithm with tuned parameters

Artificial Ant Problem

- Toroidal field $N \times N$
- M pieces of food
- K time steps
- Fixed position of food and the ant
- Goal – build an FSM, such that the ant will eat all food in K steps

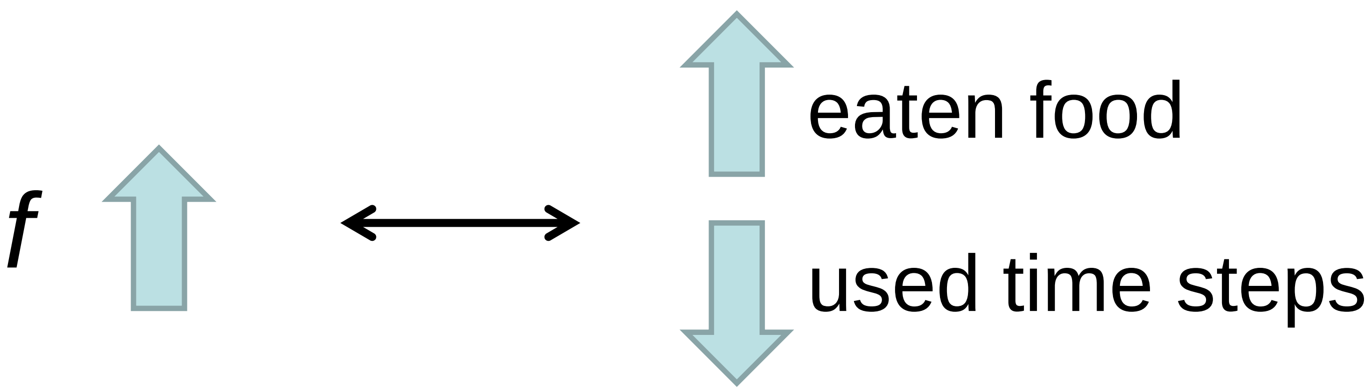


Field example: John Muir Trail

Artificial Ant: Fitness function

$$f = n_{\text{food}} + \frac{K - s_{\text{last}} - 1}{K}$$

- n_{food} – number of eaten food pieces
- K – max number of allotted steps
- s_{last} – number of used steps



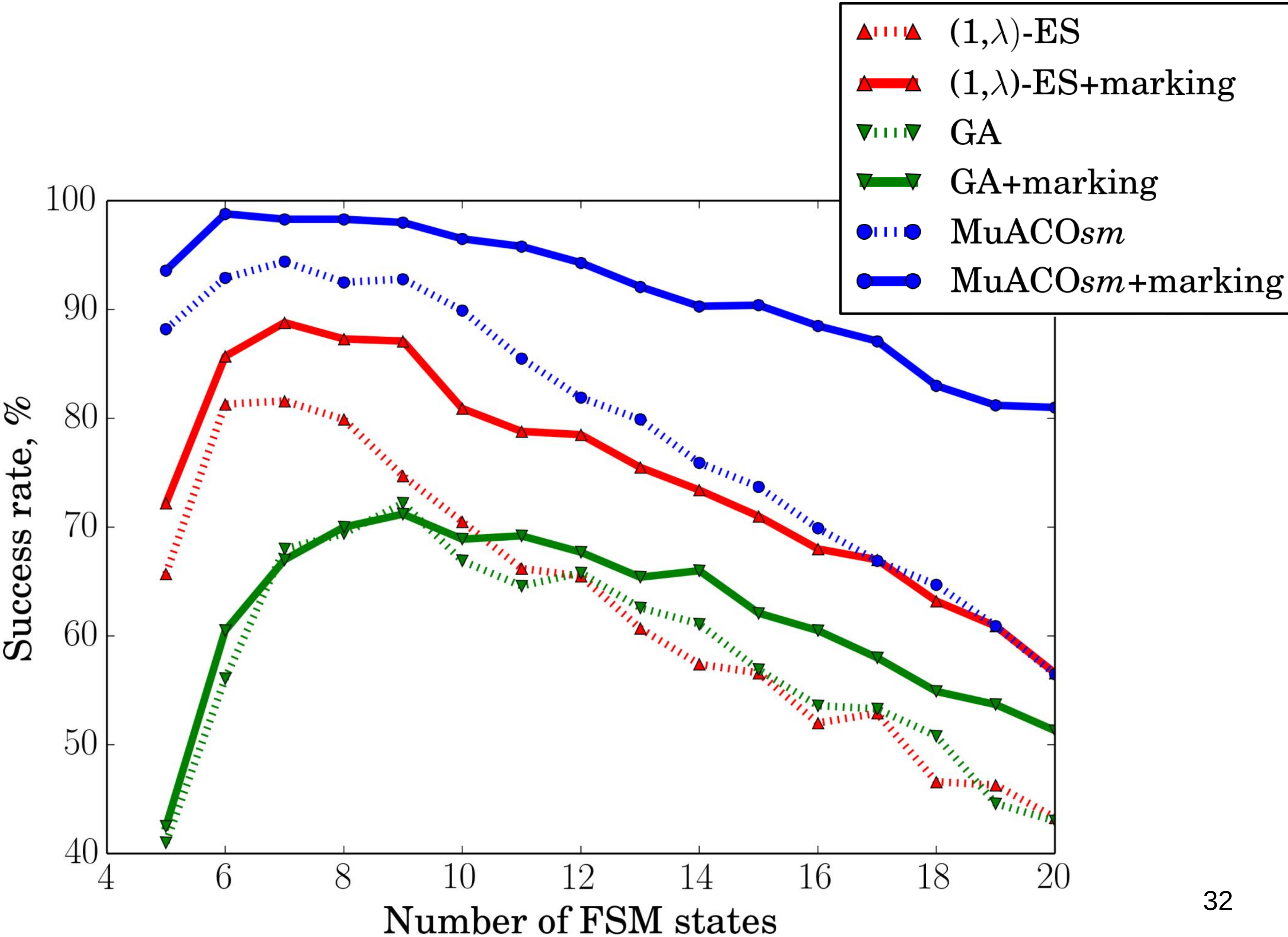
Success rate

- Successful run: fitness ≥ 89
- Success rate = $N_{\text{succesful runs}} / N_{\text{runs}}$

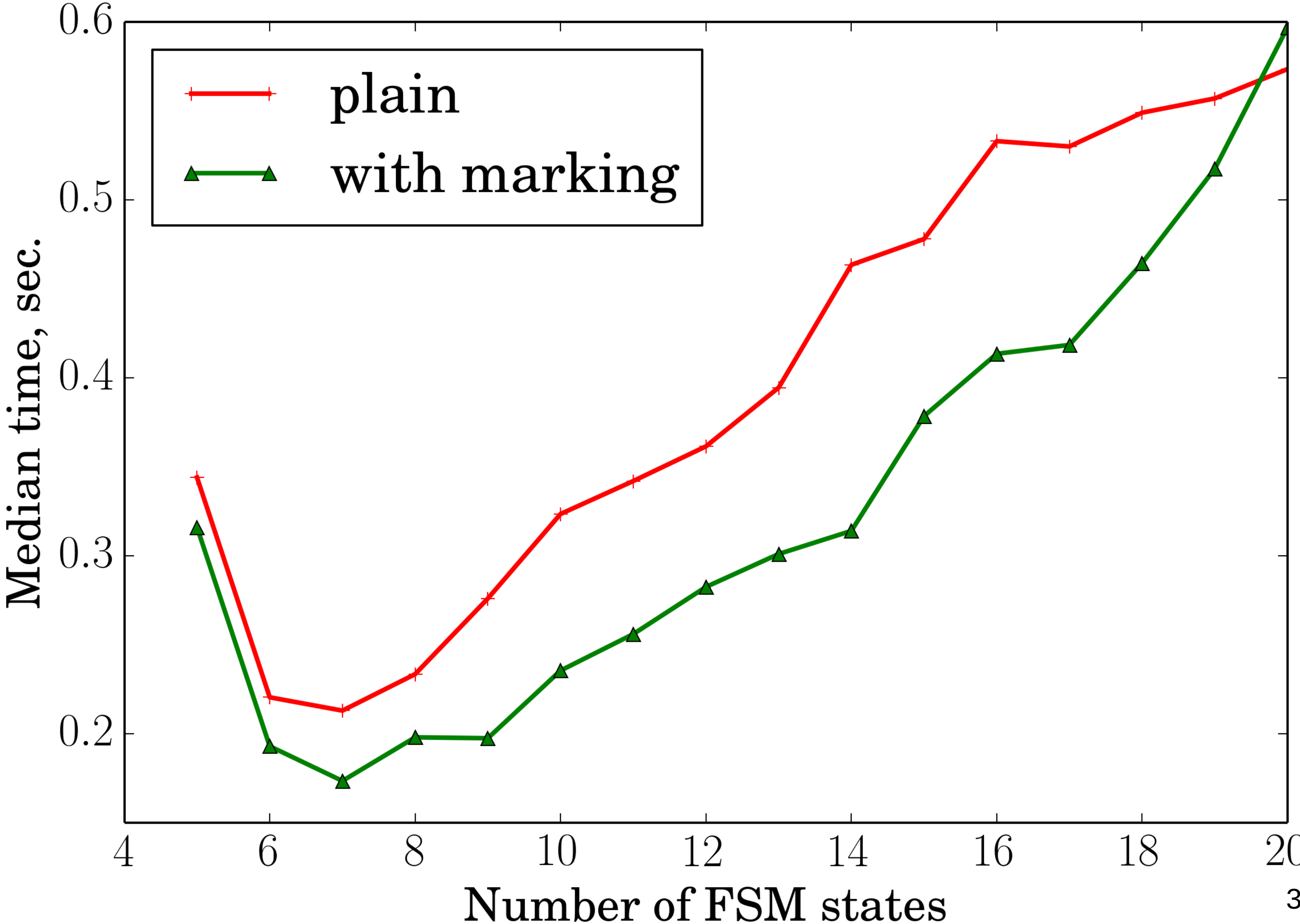
Experiment design

- Vary number of states
- Limited number of fitness evaluations
- Measure:
  - Success rate
  - Time

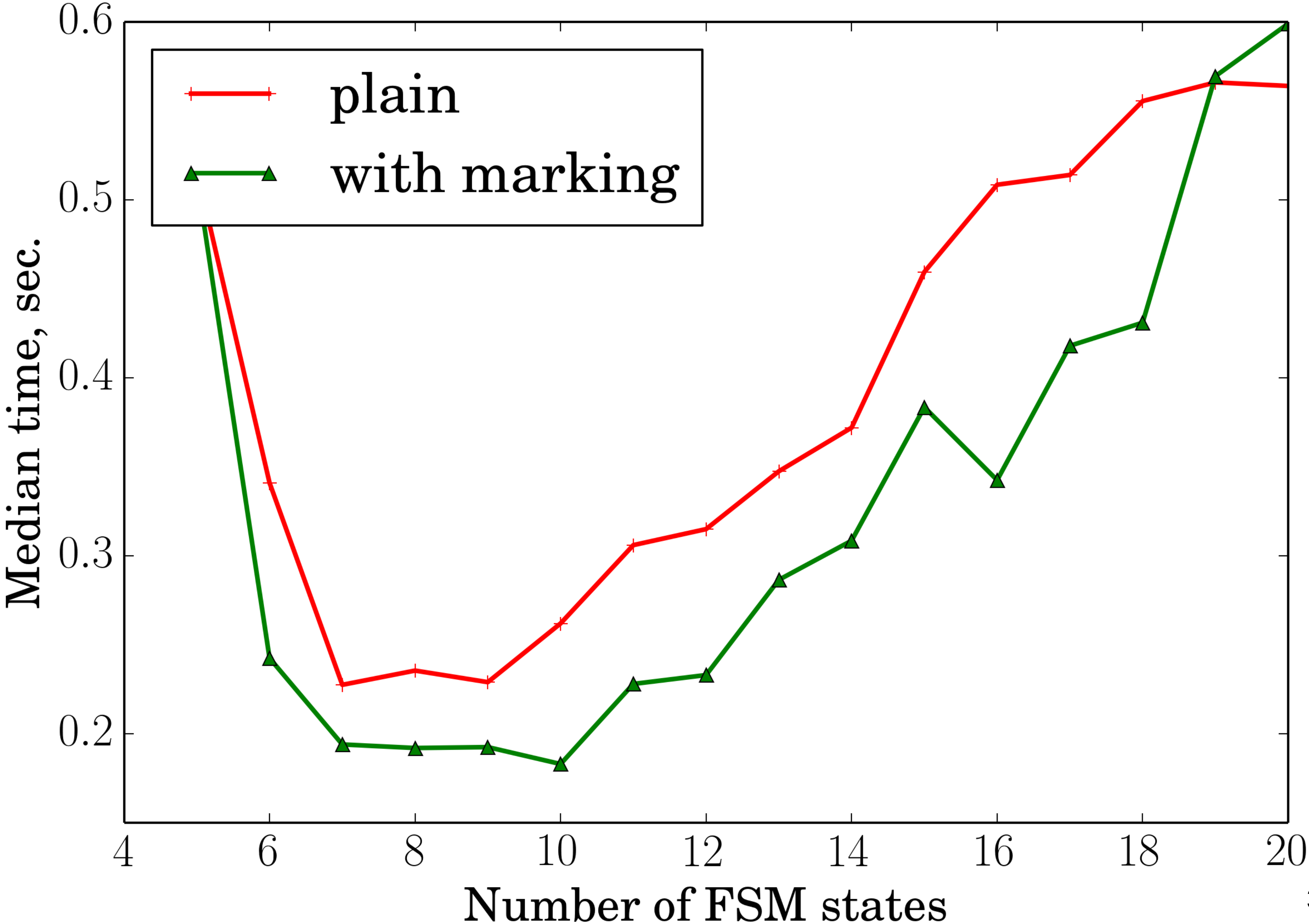
Success rate



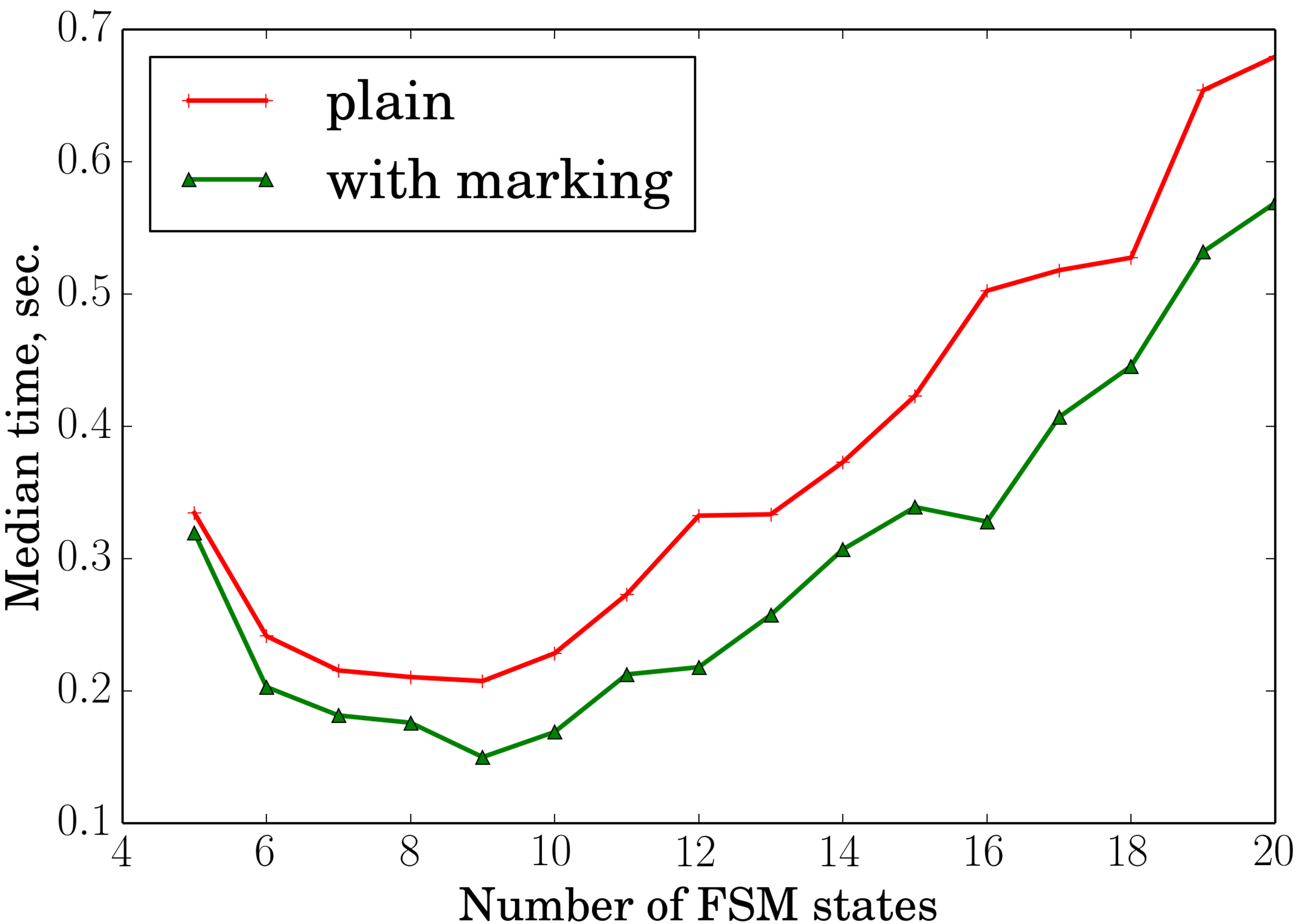
ES median time



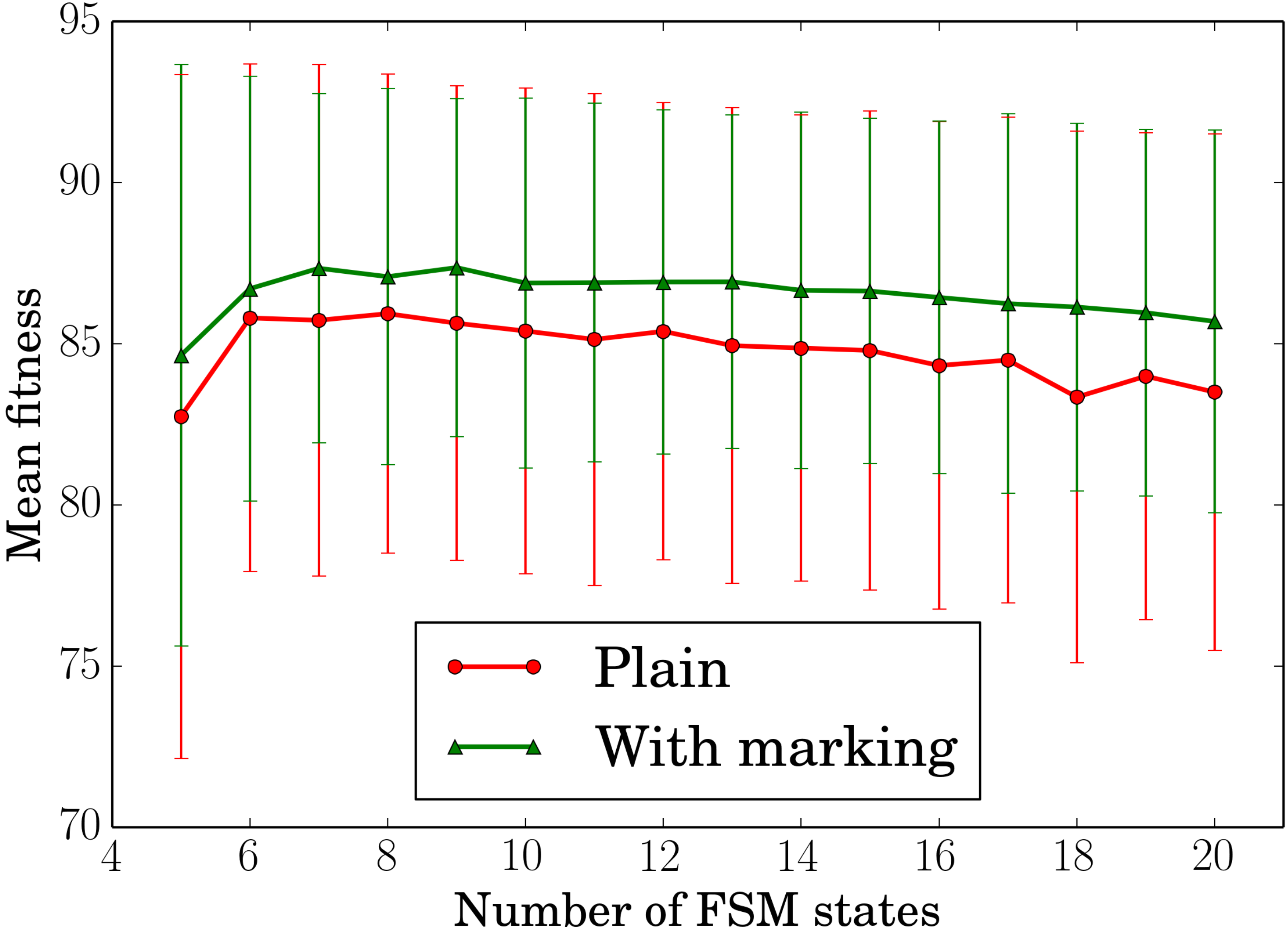
GA median time



MuACO median time

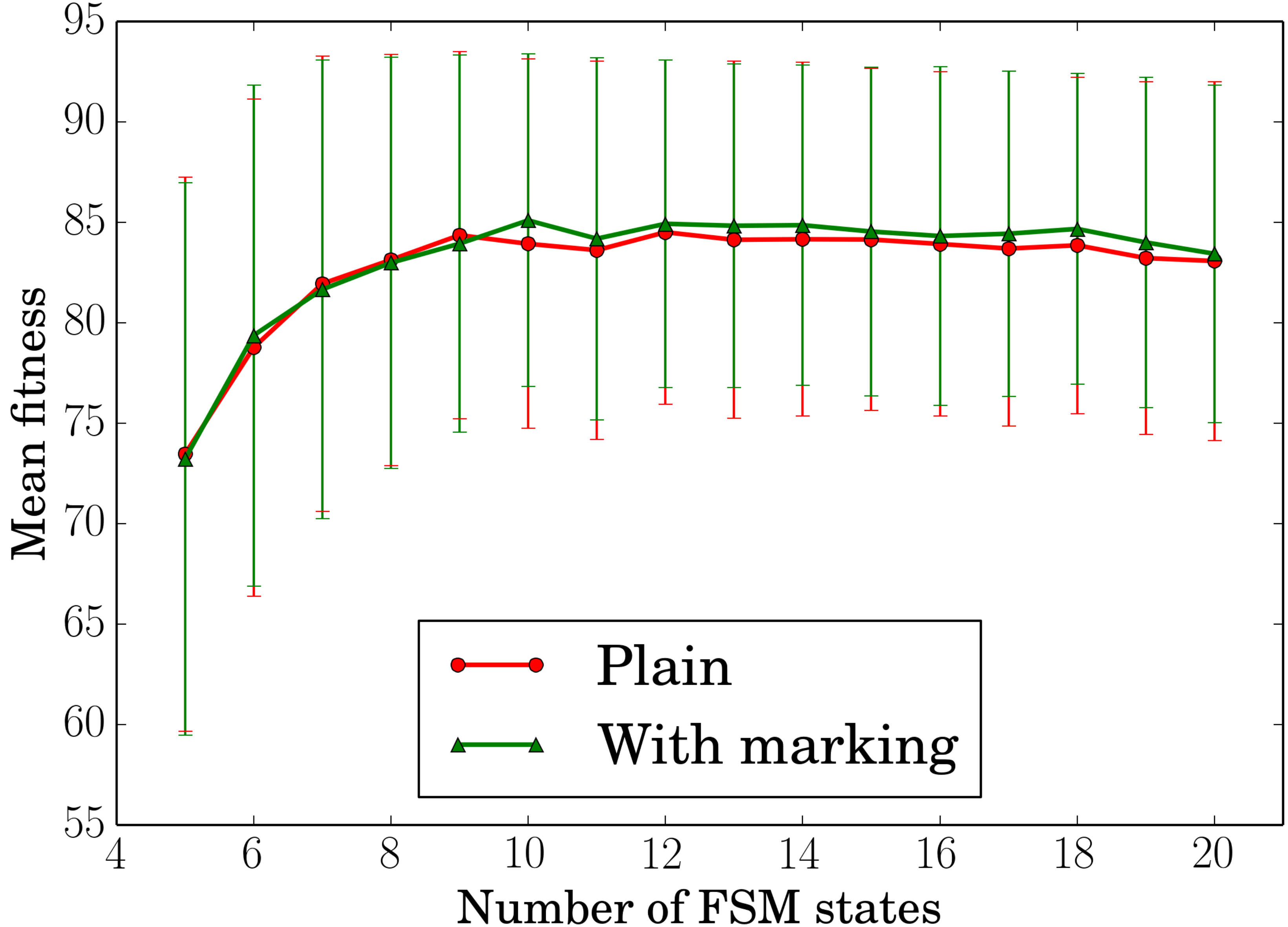


ES Fitness



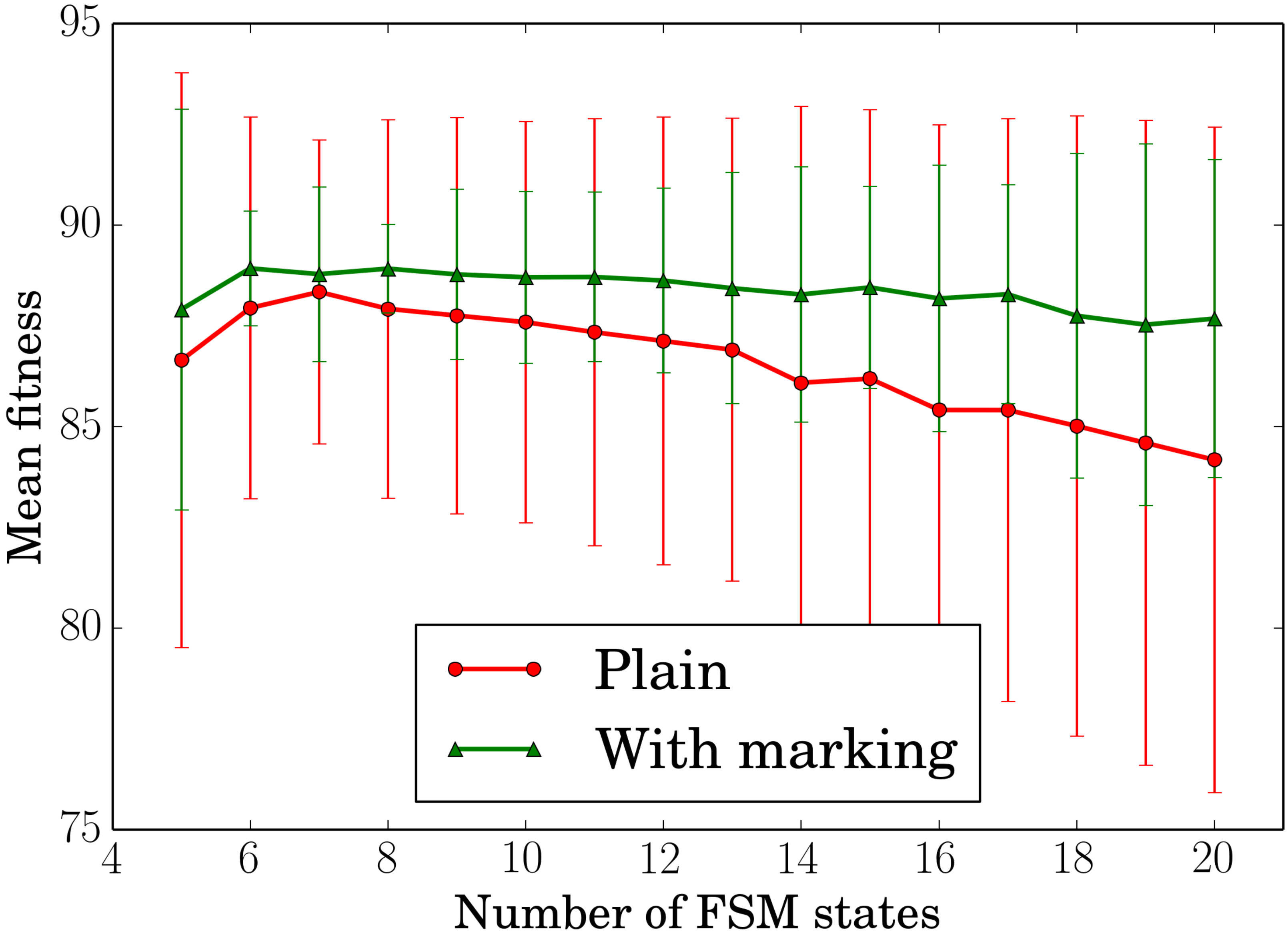
95
90
85
80
75
70
4
6
8
10
12
14
16
18
20
Mean fitness
Number of FSM states
Plain
With marking

GA Fitness

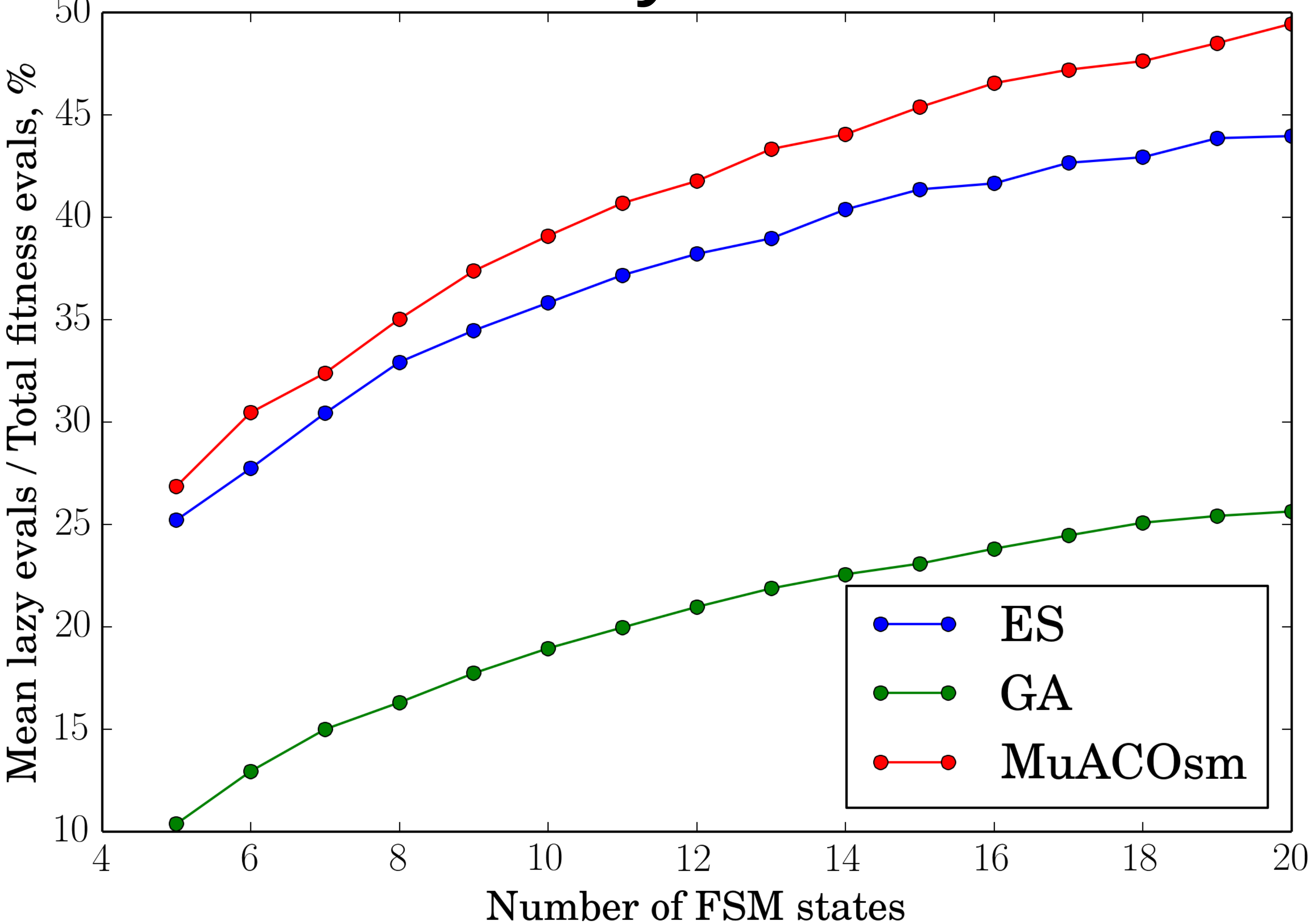


Mean fitness
Number of FSM states
95
90
85
80
75
70
65
60
55
4
6
8
10
12
14
16
18
20
Plain
With marking

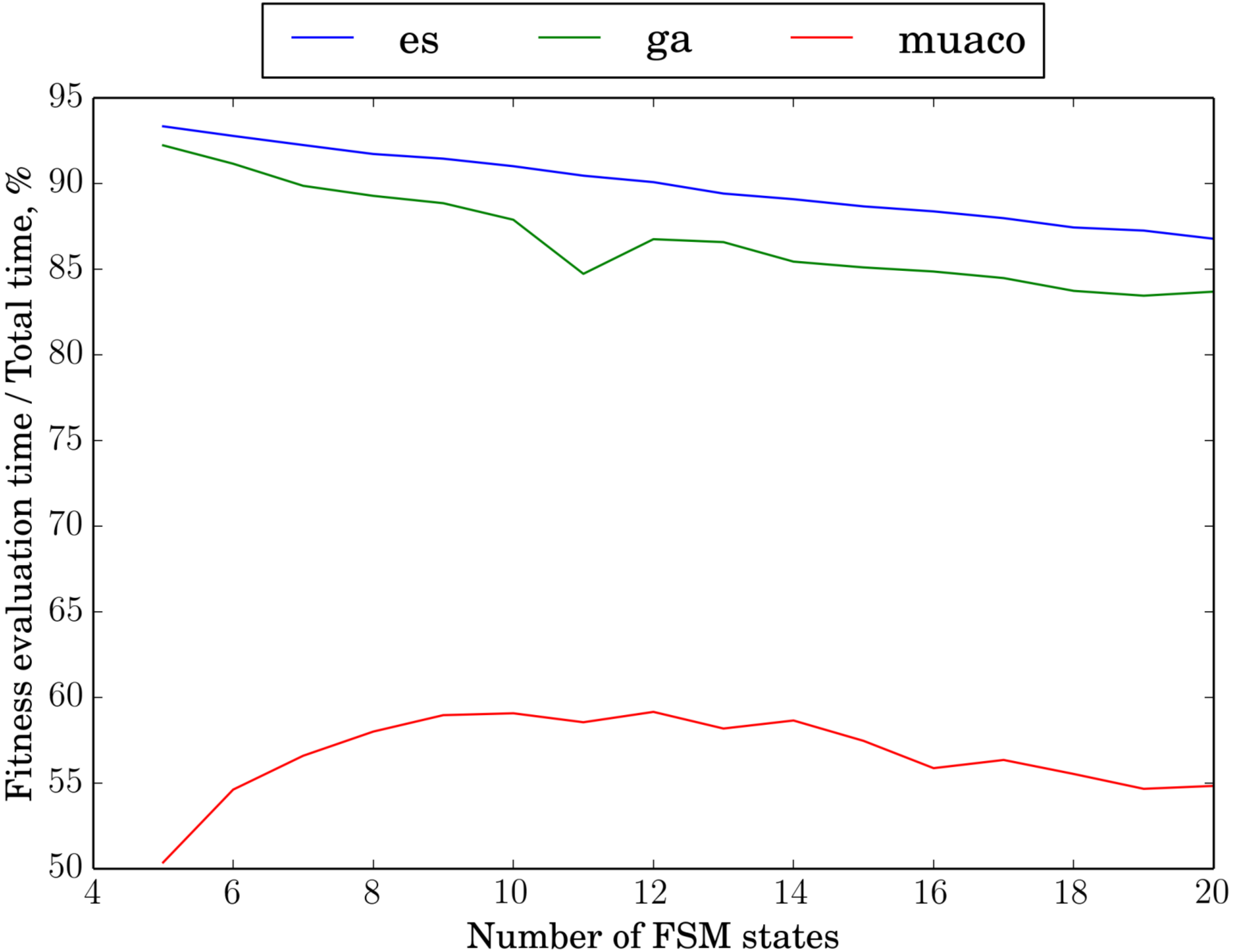
MuACO Fitness



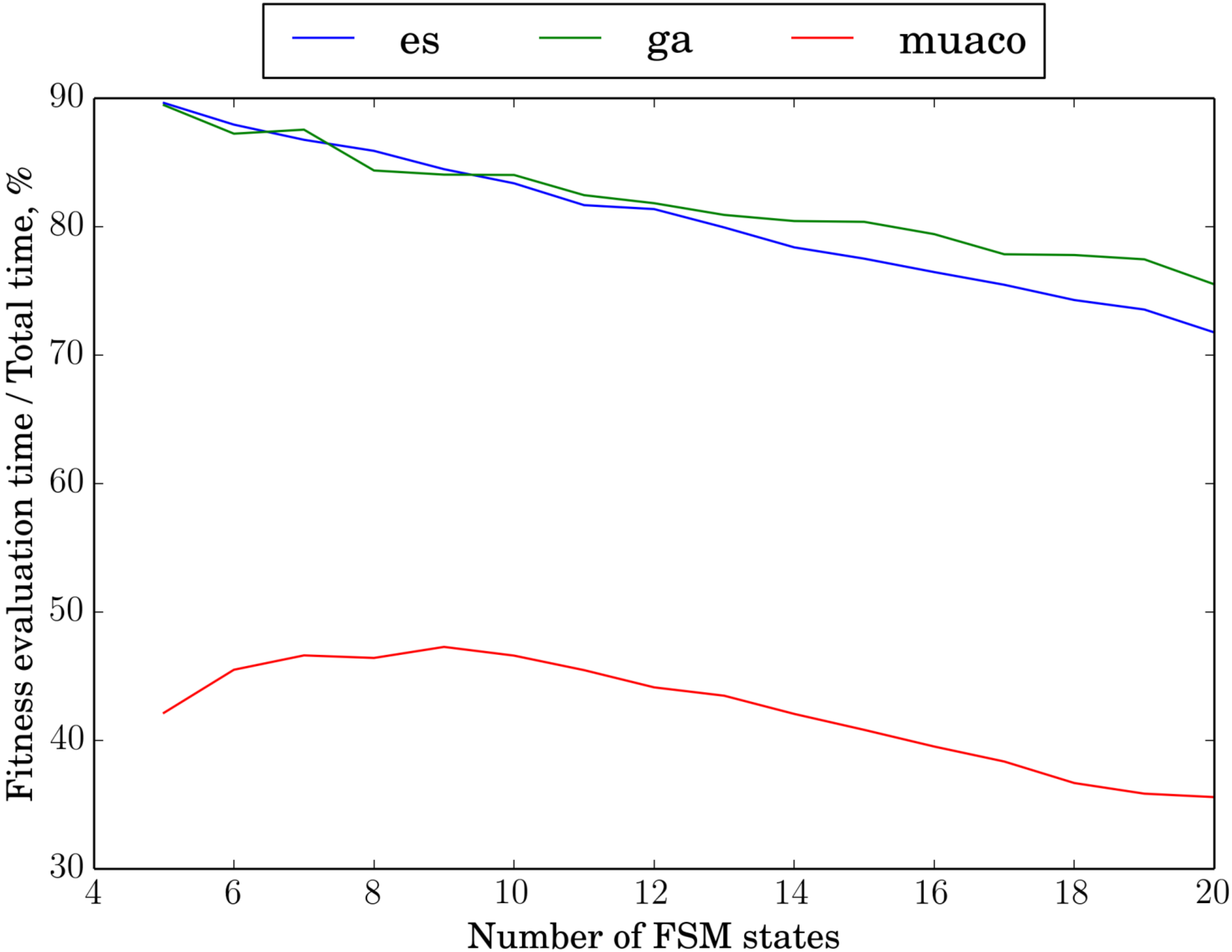
Ratio of lazy evaluations



Fitness evaluation time: plain algorithms



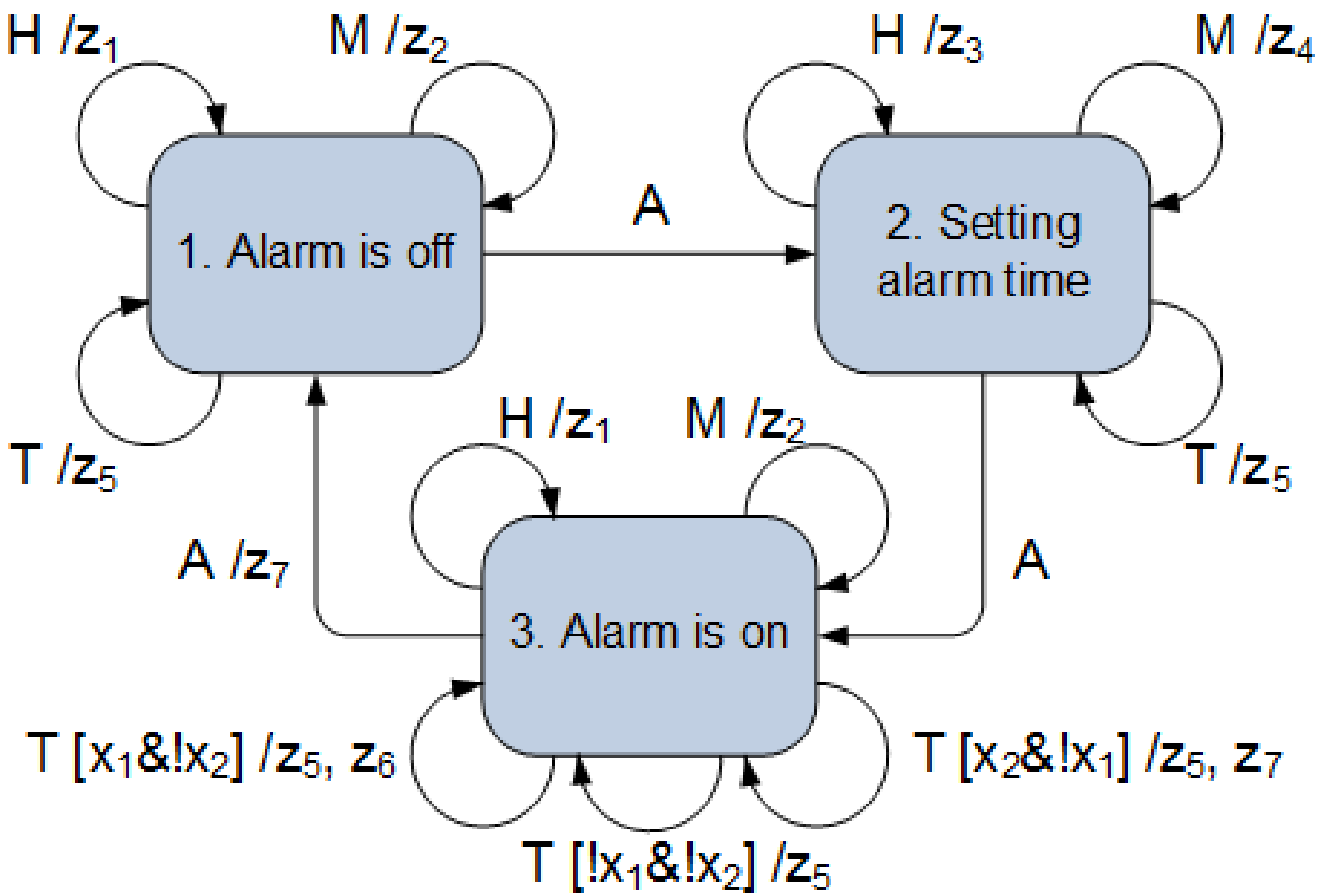
Fitness evaluation time: with marking



Statistical Significance

- ANOVA test
- Fitness distributions significantly different for ES and MuACOsm
- Insignificant for GA

Learning Extended Finite-State Machines from tests (1)



Learning Extended Finite-State Machines from tests (2)

Input data:

- Number of states C and sets Σ and Δ
- Set of test examples T
- T_i =<input sequence $I_{j,}$ output sequence O_j>

NP-hard problem: build an EFSM with *C* states compliant with tests *T*

Example of a test

A H T[x_1] T[x1 & x2] $\rightarrow$ z_1 z_1 z_2 z_5 z_7

Learning EFSMs: Fitness function

- Pass inputs to EFSM, record outputs
- Compare generated outputs with references
- Fitness = string similarity measure (edit distance)

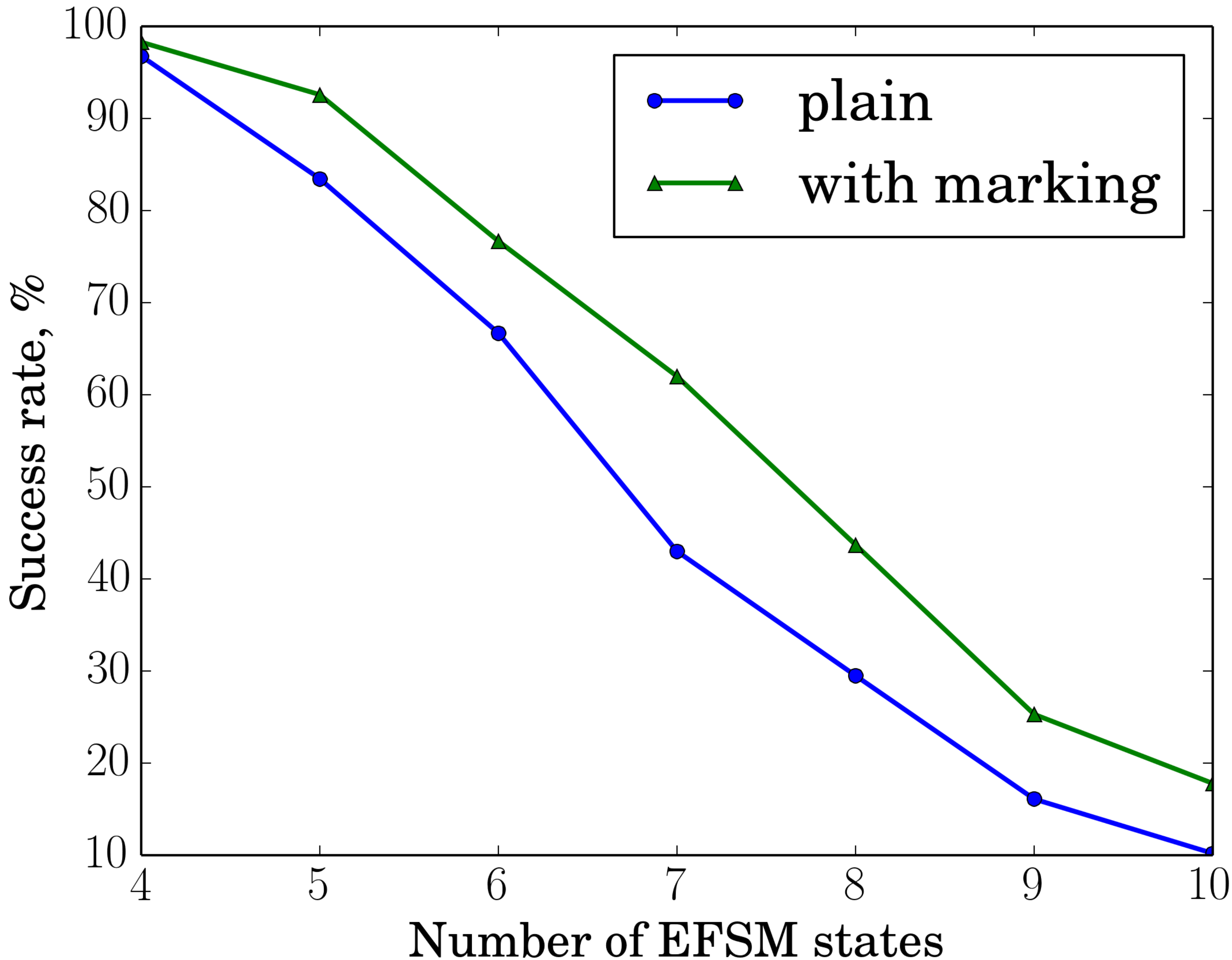
$$f' = \frac{1}{|T|} \sum_{j=1}^{|T|} \left(1 - \frac{ED(O_j, A_j)}{\max\left(len(O_j), len(A_j)\right)} \right)$$

$$f = 100 \cdot f' + \frac{1}{100} \cdot (100 - n_{trans})$$

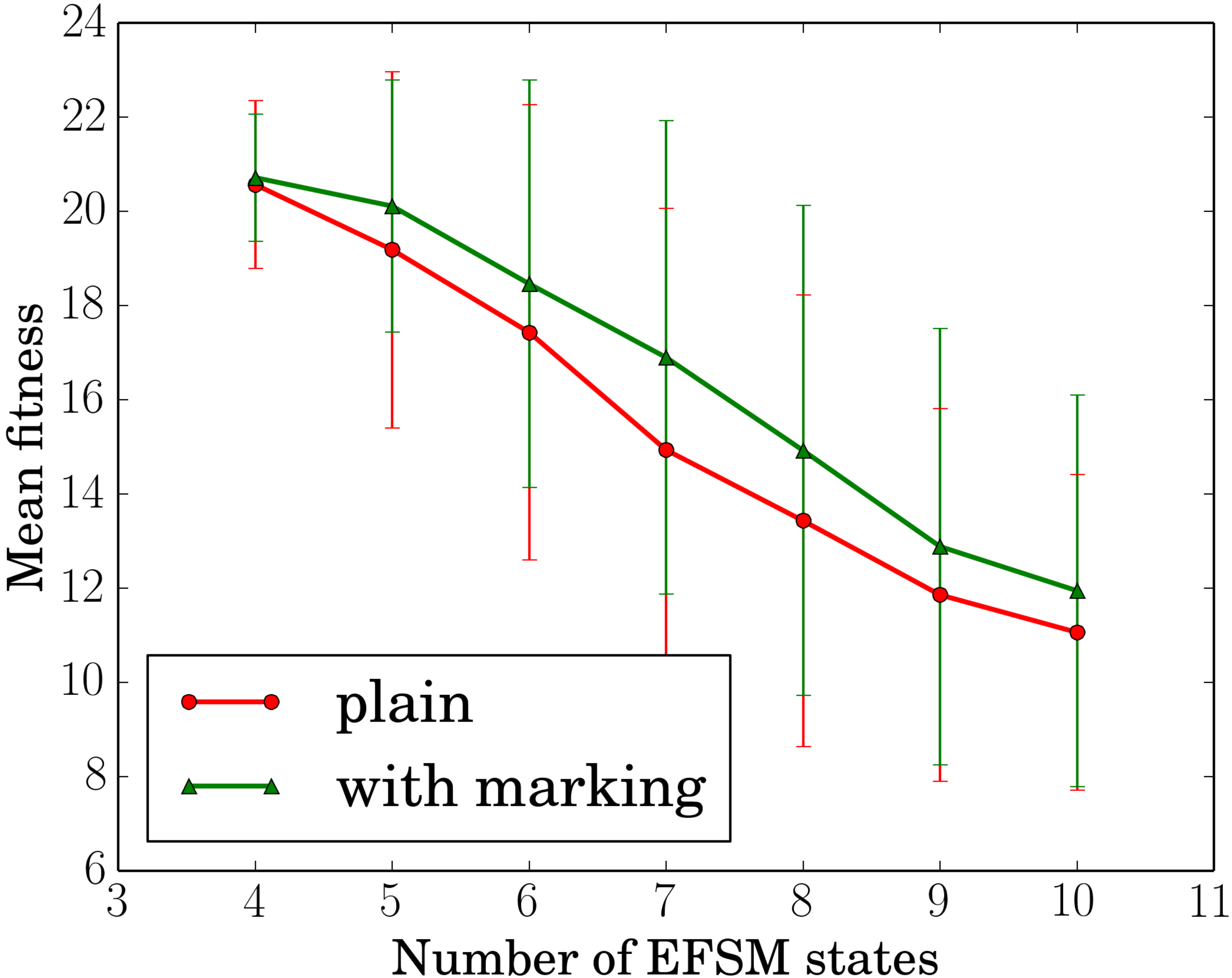
Experimental setup

1. Generate random EFSM with *C* states
2. Generate set of tests of total length $C \times 150$
3. Learn EFSM from tests
4. Experiment for each *C* repeated 100 times
5. Limited number of fitness evaluations

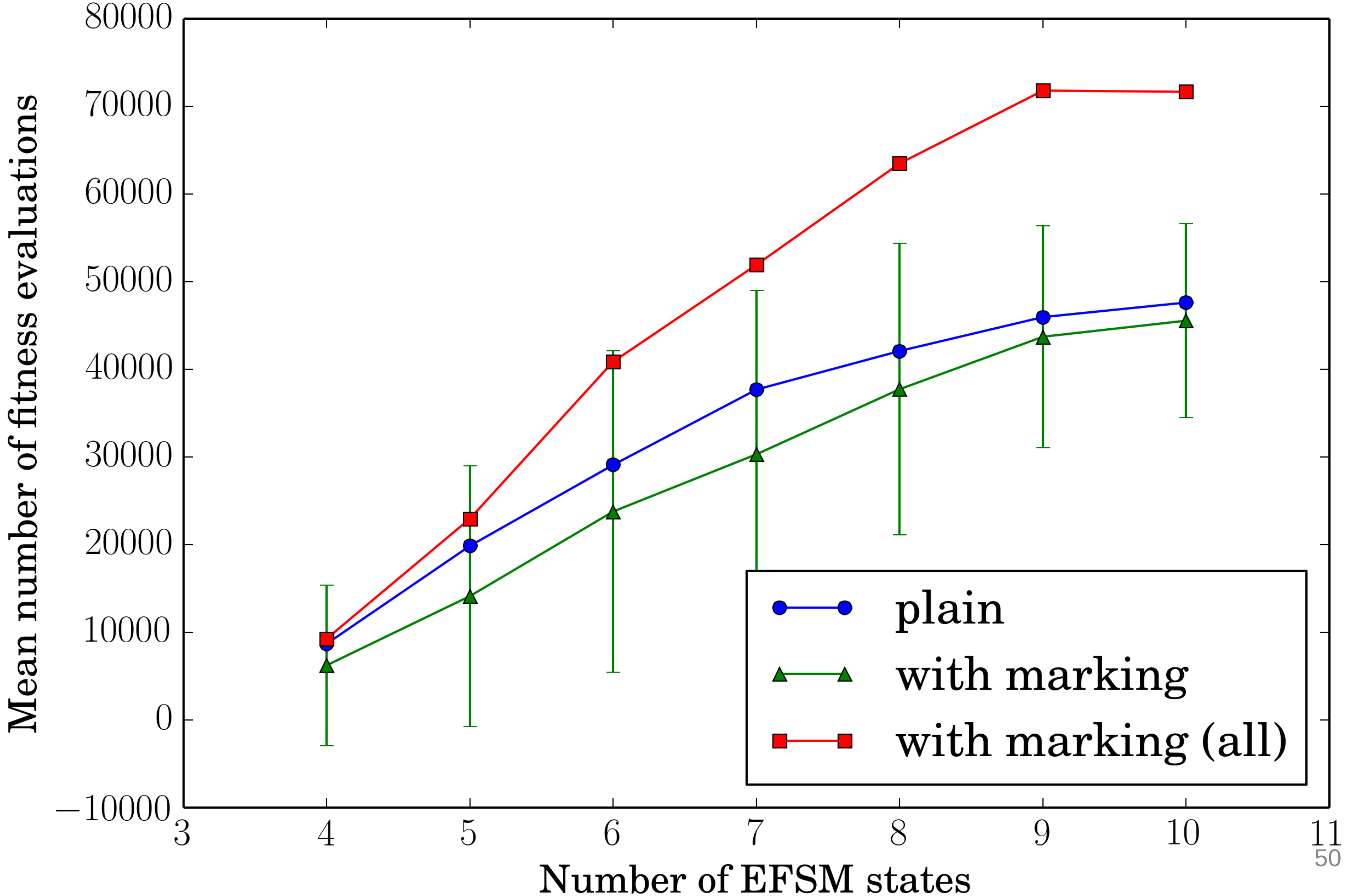
Success rate



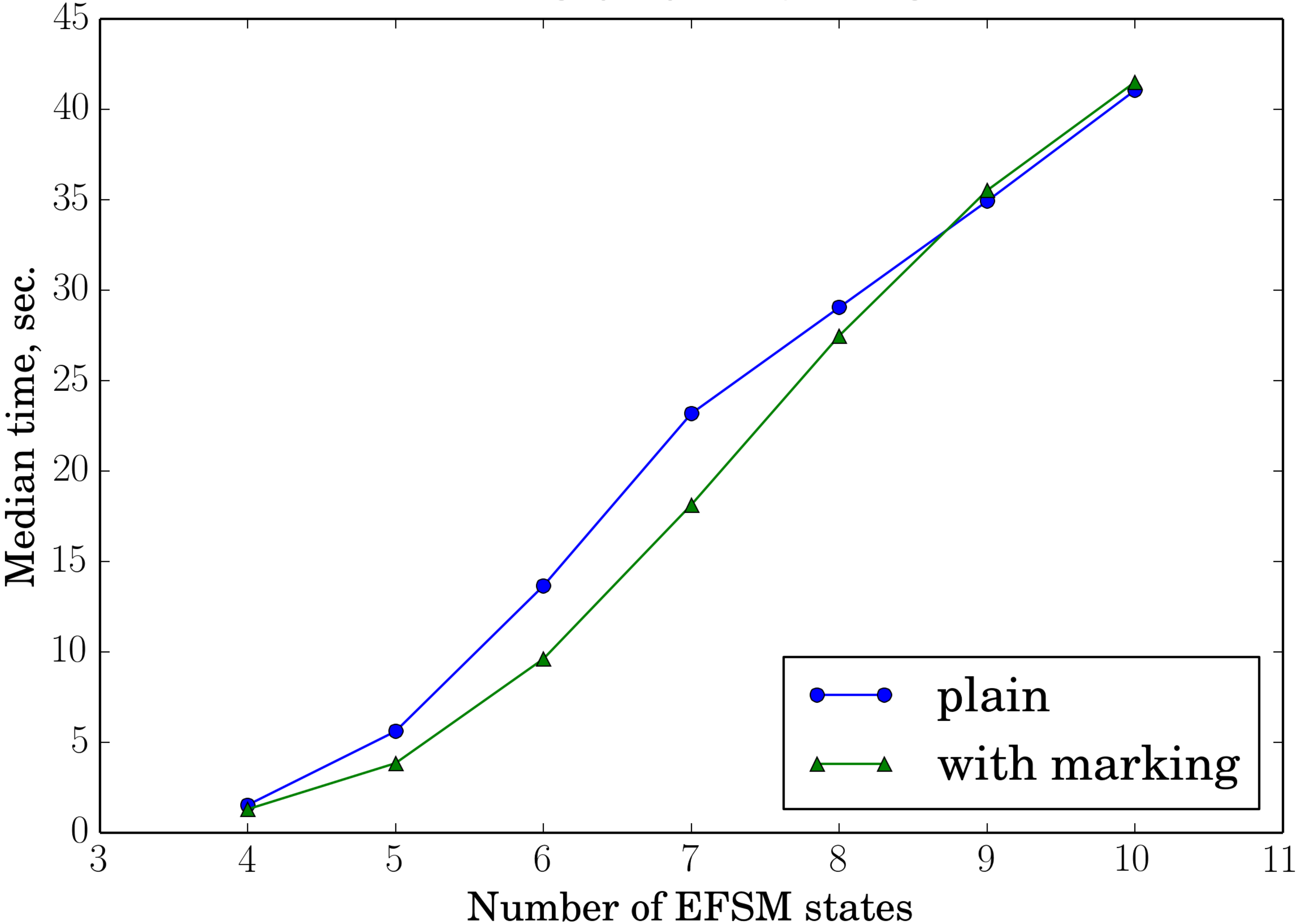
Mean fitness



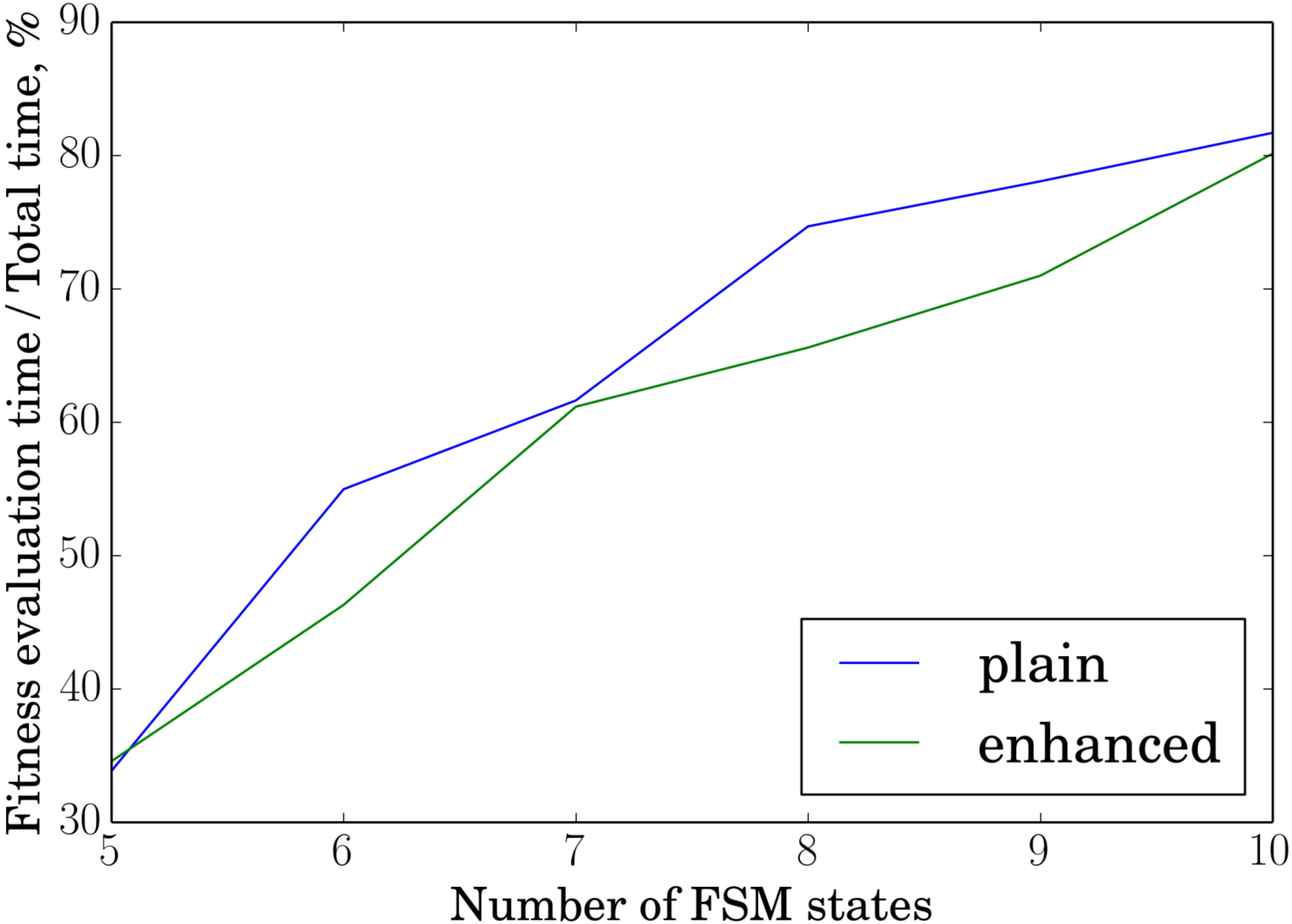
Fitness evaluations



Median time


45
40
35
30
25
20
15
10
5
0
3
4
5
6
7
8
9
10
11
Median time, sec.
Number of EFSM states
plain
with marking

Time for fitness evaluation



Conclusion

Developed approach

– Applicable to all FSM learning algorithms that use mutations

– Allows to explore more FSMs with the same number of fitness evaluations

– Effectively improves fitness and time

Limitations

- Makes sense to use if cost of fitness computation is relatively high

Future work

Explore more ways of using domain knowledge in FSM learning algorithms

Acknowledgements

- University ITMO research project 610455
- Ministry of Education and Science of Russian Federation in the framework of the federal program “Scientific and scientific-pedagogical personnel of innovative Russia in 2009-2013” (contract 16.740.11.0455, agreement 14.B37.21.0397)

Thank you for your attention!

Learning FSMs: Conserving Fitness Evaluations by Marking Used Transitions

Daniil Chivilikhin
Vladimir Ulyantsev

{chivdan,ulyantsev}@rain.ifmo.ru

This presentation online:
http://rain.ifmo.ru/~chivdan/papers/2013/icmla-2013-presentation.pdf