MuACO*sm* – A New Mutation-Based Ant Colony Optimization Algorithm for Learning Finite-State Machines

Daniil Chivilikhin and Vladimir Ulyantsev

National Research University of IT, Mechanics and Optics

St. Petersburg, Russia

Evolutionary and Combinatorial Optimization Track @ GECCO 2013

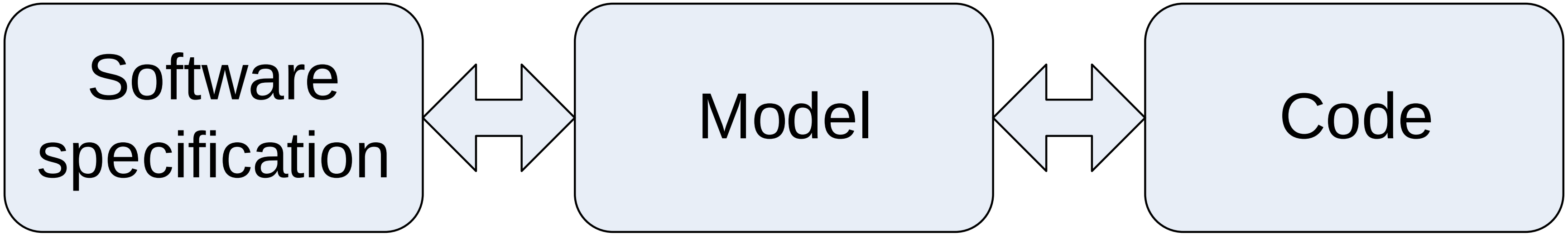
July 8, 2013

Motivation: Reliable software

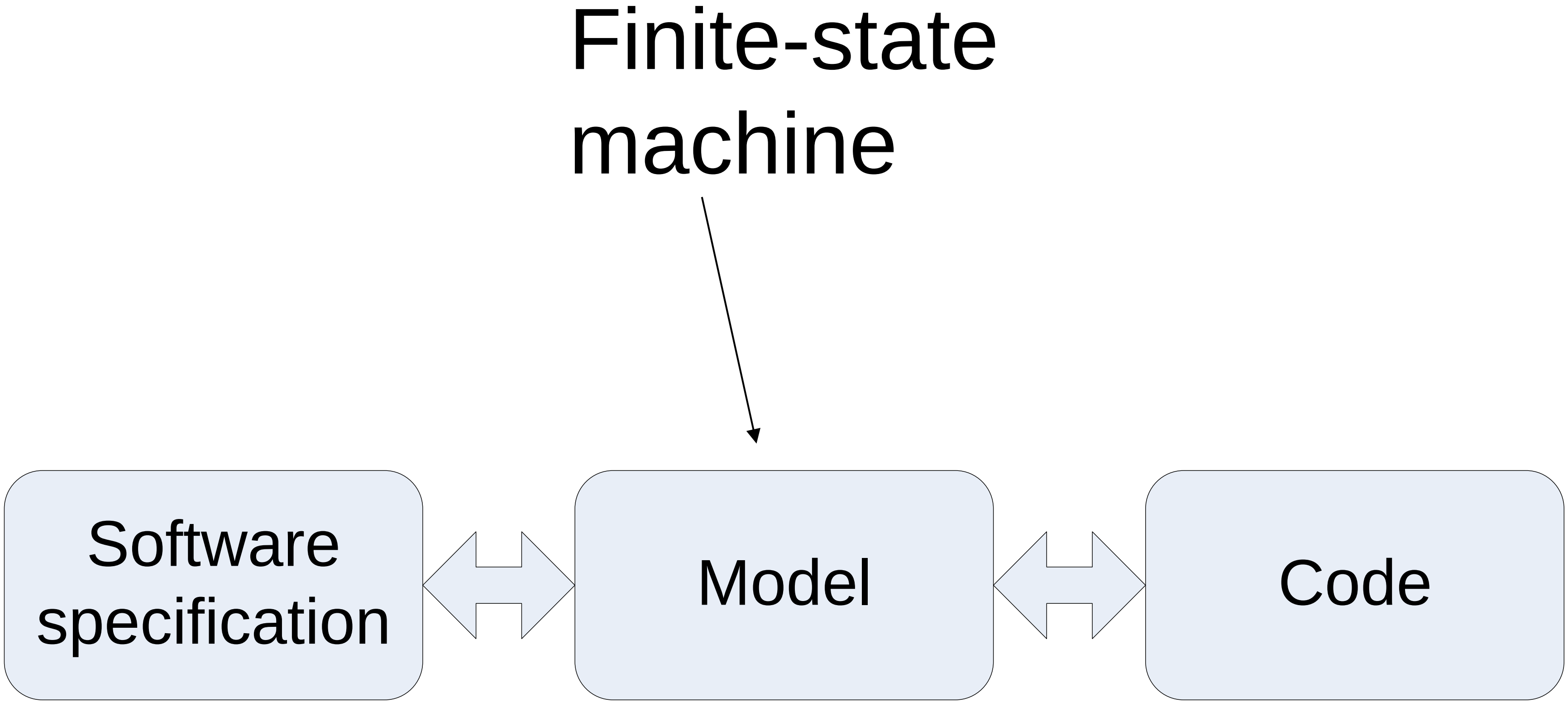
- Systems with high cost of failure
  - Energy industry
  - Aircraft industry
  - Space industry
  - …
- We want to have **reliable software**
  - Testing is not enough
  - **Verification** is needed

Introduction (1)

- Automated software engineering
- Model-driven development
- Automata-based programming



Introduction (2)

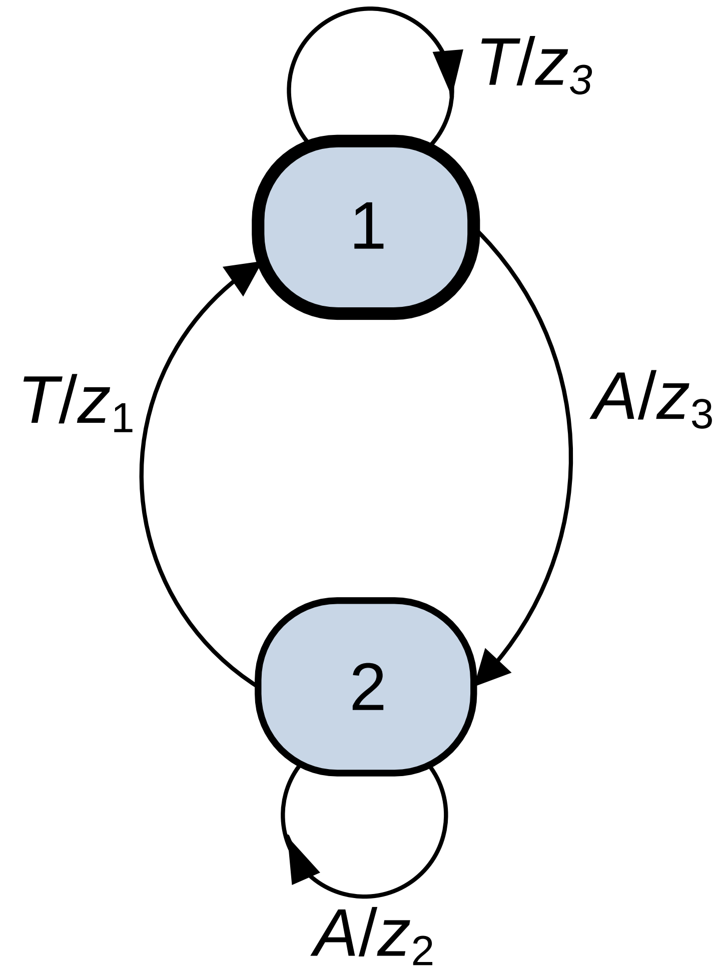


Finite-State Machine

- S – set of states
- $s_0 \in S$ – initial state
- Σ – set of input events
- Δ – set of output actions
- δ: $S \times \Sigma \rightarrow S$ – transition function
- λ: $S \times \Sigma \rightarrow \Delta$ – actions function

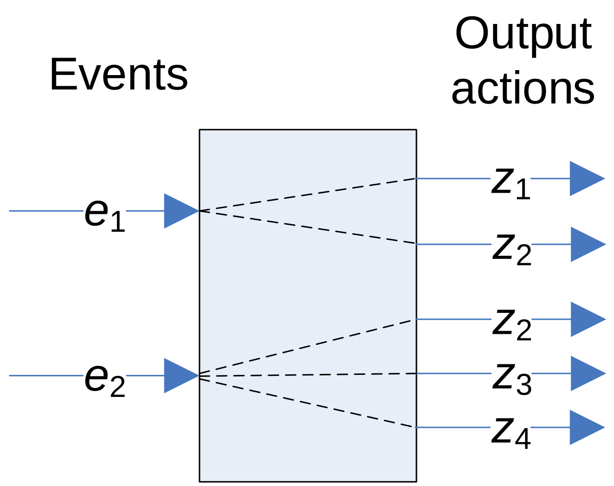
Example:

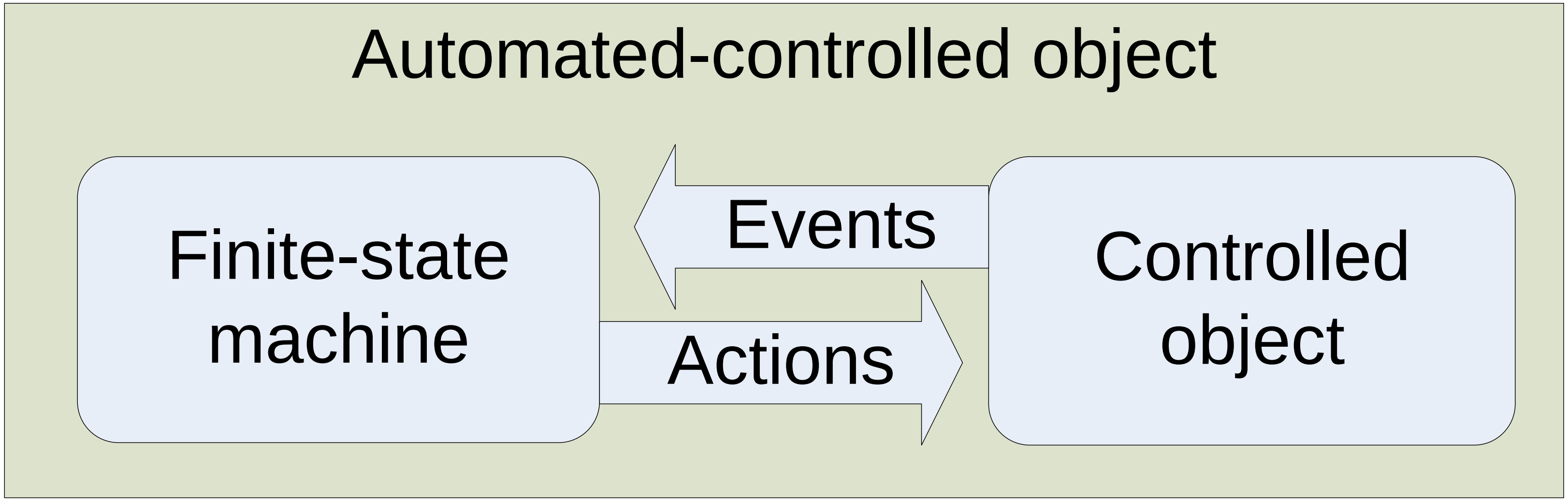
- two states
- events = {A, T}
- actions = {z_1, z_2, z_3, z_4}



Automata-based programming

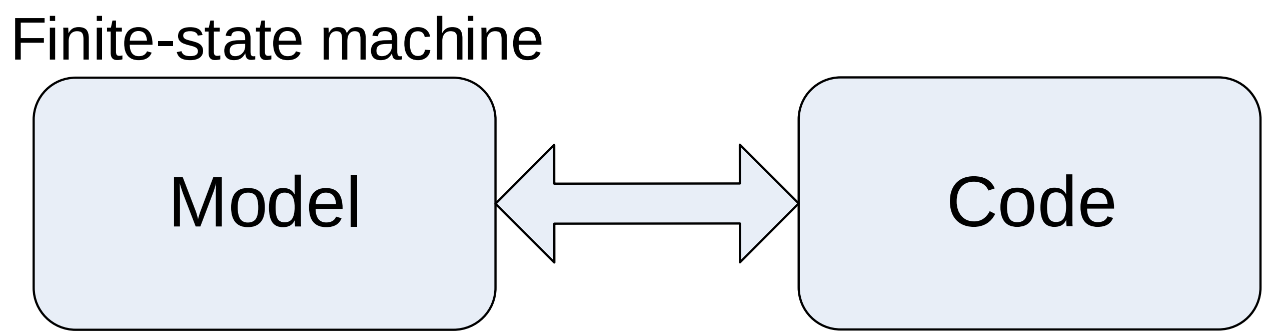
Design programs with complex behavior as automated-controlled objects





Automata-based programming: advantages

- Model before programming code, not vice versa



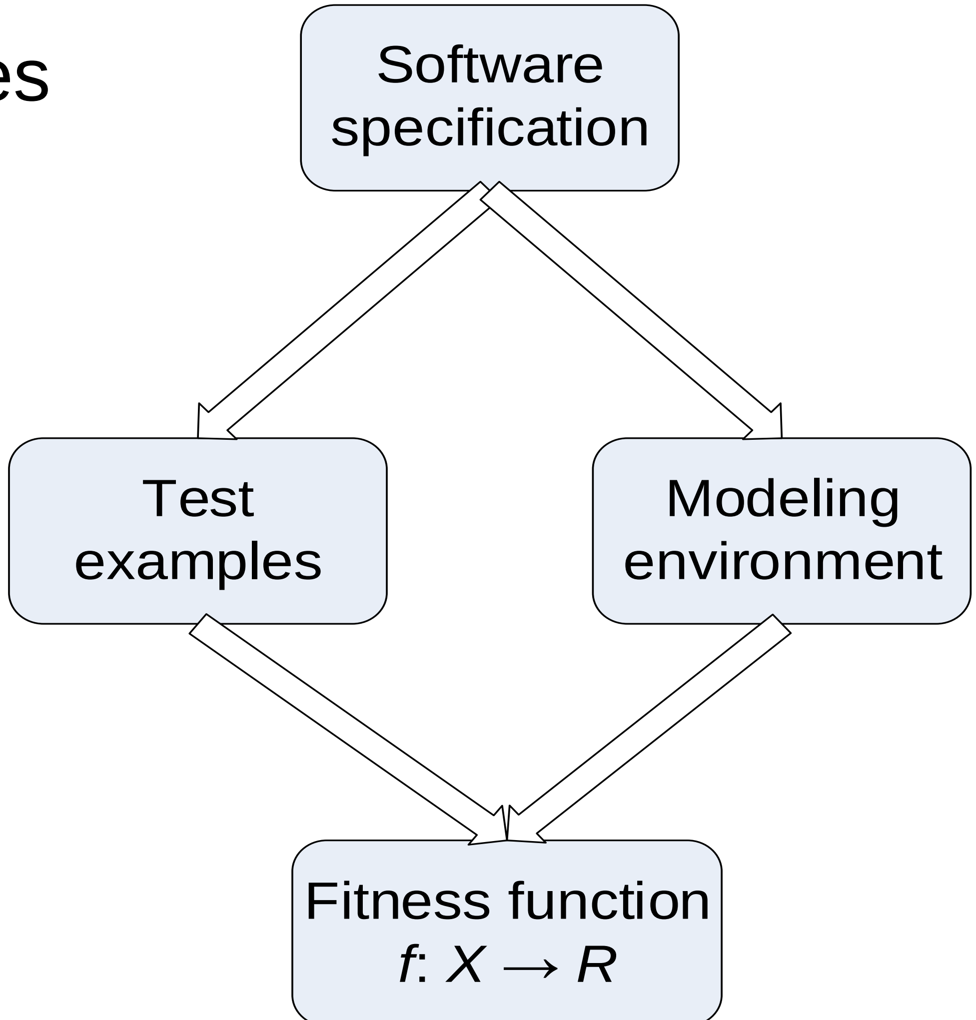
- Possibility of program verification using *Model Checking*
- You can check temporal properties (LTL)

Issues

- Hard to build an FSM with desired structure and behavior
- Several problems of learning FSMs were proven to be NP-hard
- One of the solutions – metaheuristics

Learning finite-state machines with metaheuristics

- N_{states} – number of states
- Σ – input events
- Δ – output actions
- $X = (N_{\text{states}}, \Sigma, \Delta)$ – search space



Approaches to learning FSMs

- Greedy heuristics
  - problem-specific
- Reduction to *SAT* and *CSP* problems
  - fast
  - problem-specific
- Evolutionary algorithms (general)
  - slow

Proposed approach

- Based on Ant Colony Optimization (ACO)
- Non-standard problem reduction
- Modified ACO algorithm

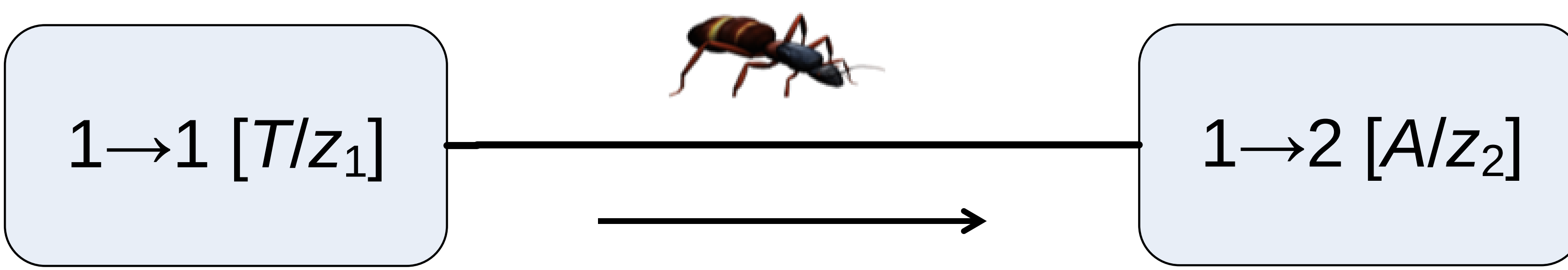
Solution representation

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

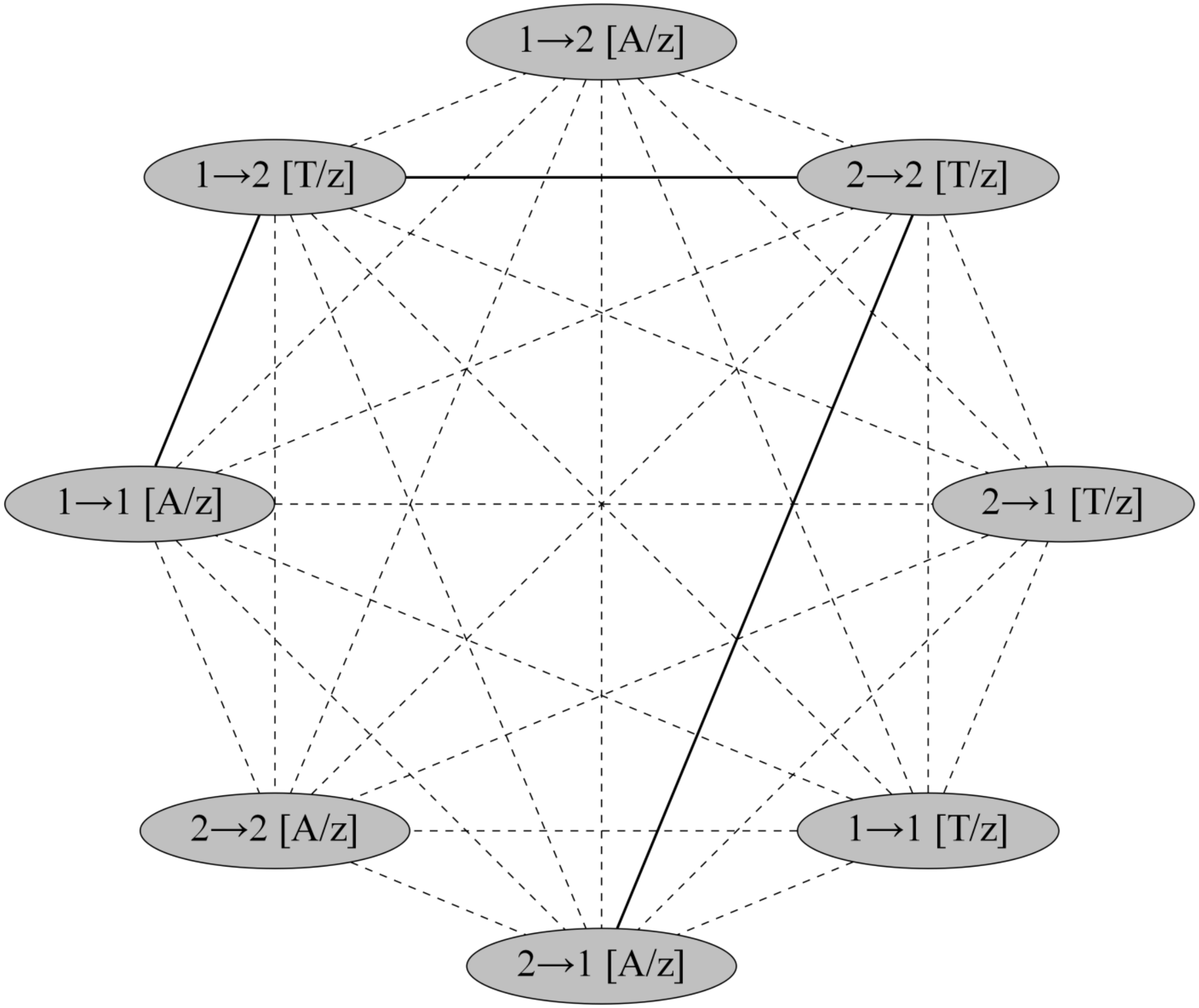
"Canonical" way to apply ACO

- Reduce problem to finding a minimum cost path in some complete graph
- Vertices – FSM transitions:
  - $\langle i \in S, j \in S, e \in \Sigma, a \in \Delta \rangle$
- Each ant adds transitions to its FSM



“Canonical” ACO: example

- 2 states
- 2 events
- 1 action



"Canonical" ACO: issues

- Number of vertices in the construction graph grows as $(N_{\text{states}})^2 \times |\Sigma| \times |\Delta|$
- No meaningful way to define heuristic information
- Later we show that "canonical" ACO is ineffective for FSM learning

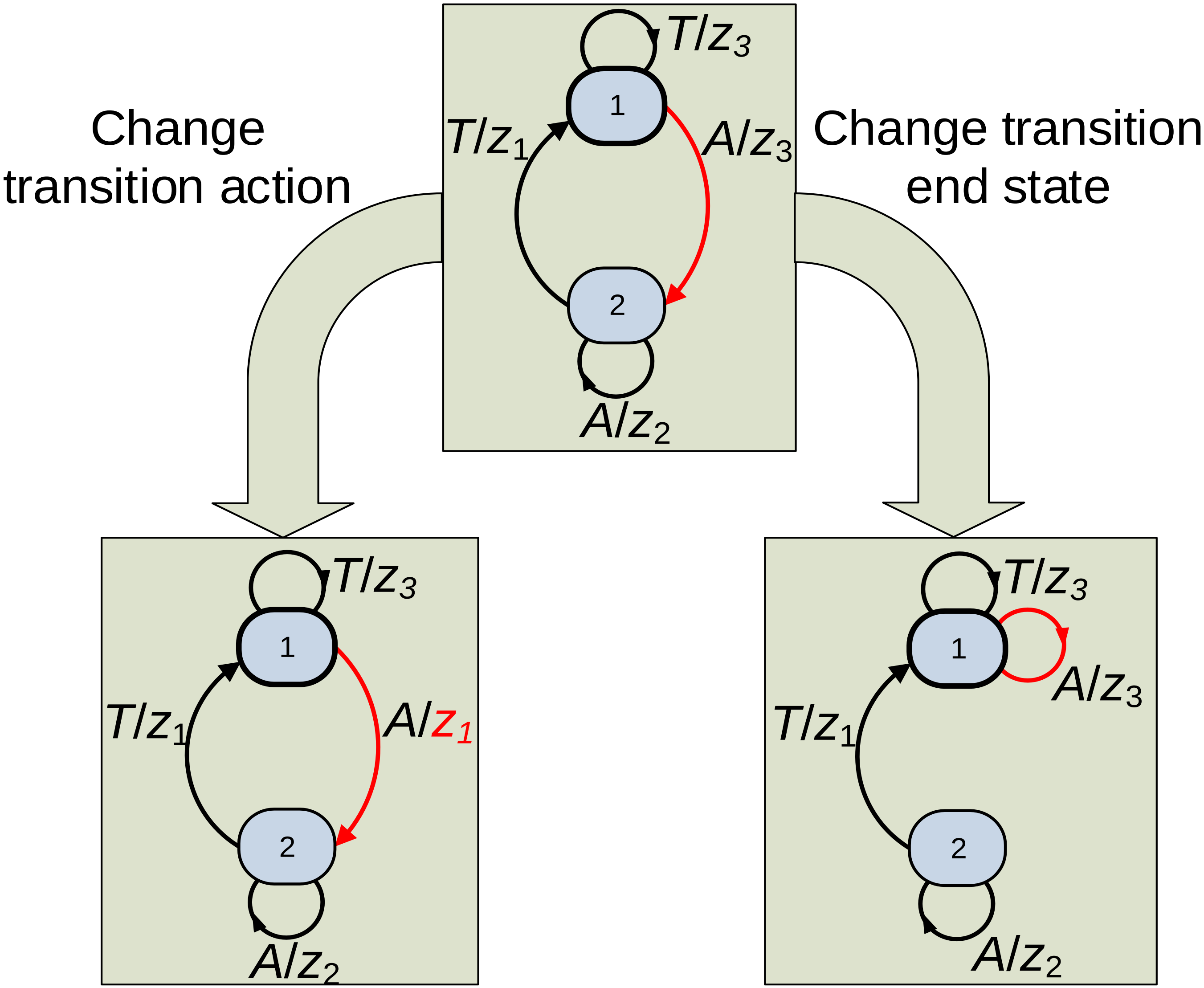
Proposed algorithm: MuACO*sm*

- Mutation-Based ACO for learning FSMs
- Uses a non-standard problem reduction
- Modified ACO

Problem reduction: MuACO*sm* vs. “canonical”

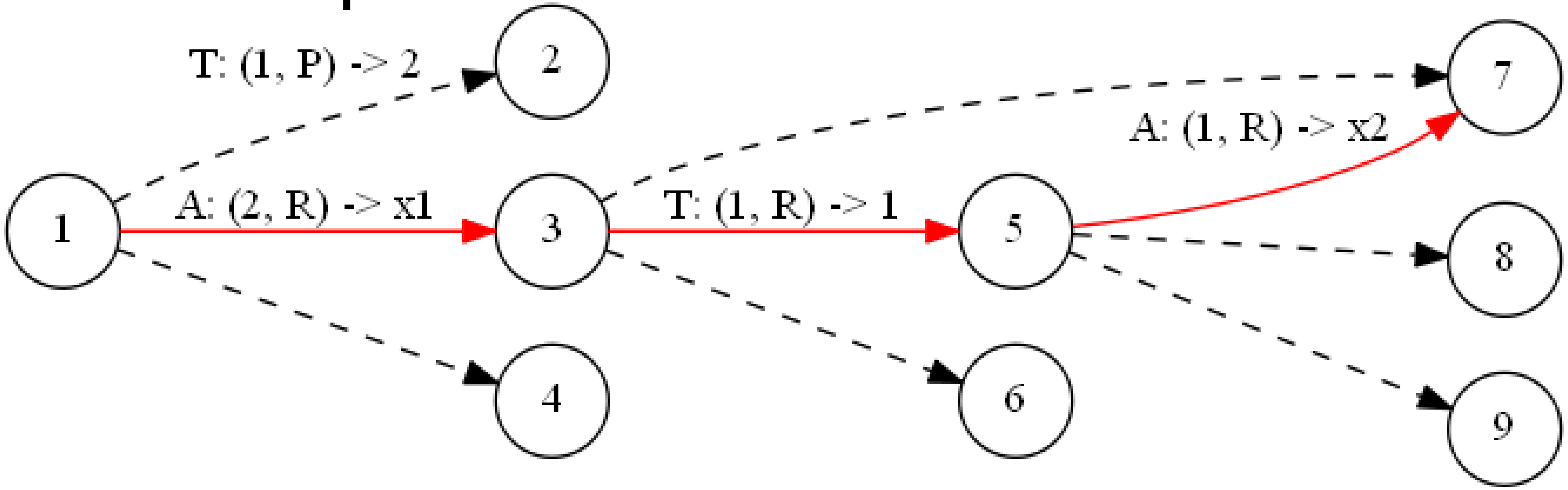
- “Canonical” ACO
  - Nodes are solution components
  - Full solutions are built by ants
- Proposed MuACO*sm* algorithm
  - Nodes are full solutions (FSMs)
  - Ants travel between full solutions

FSM Mutations



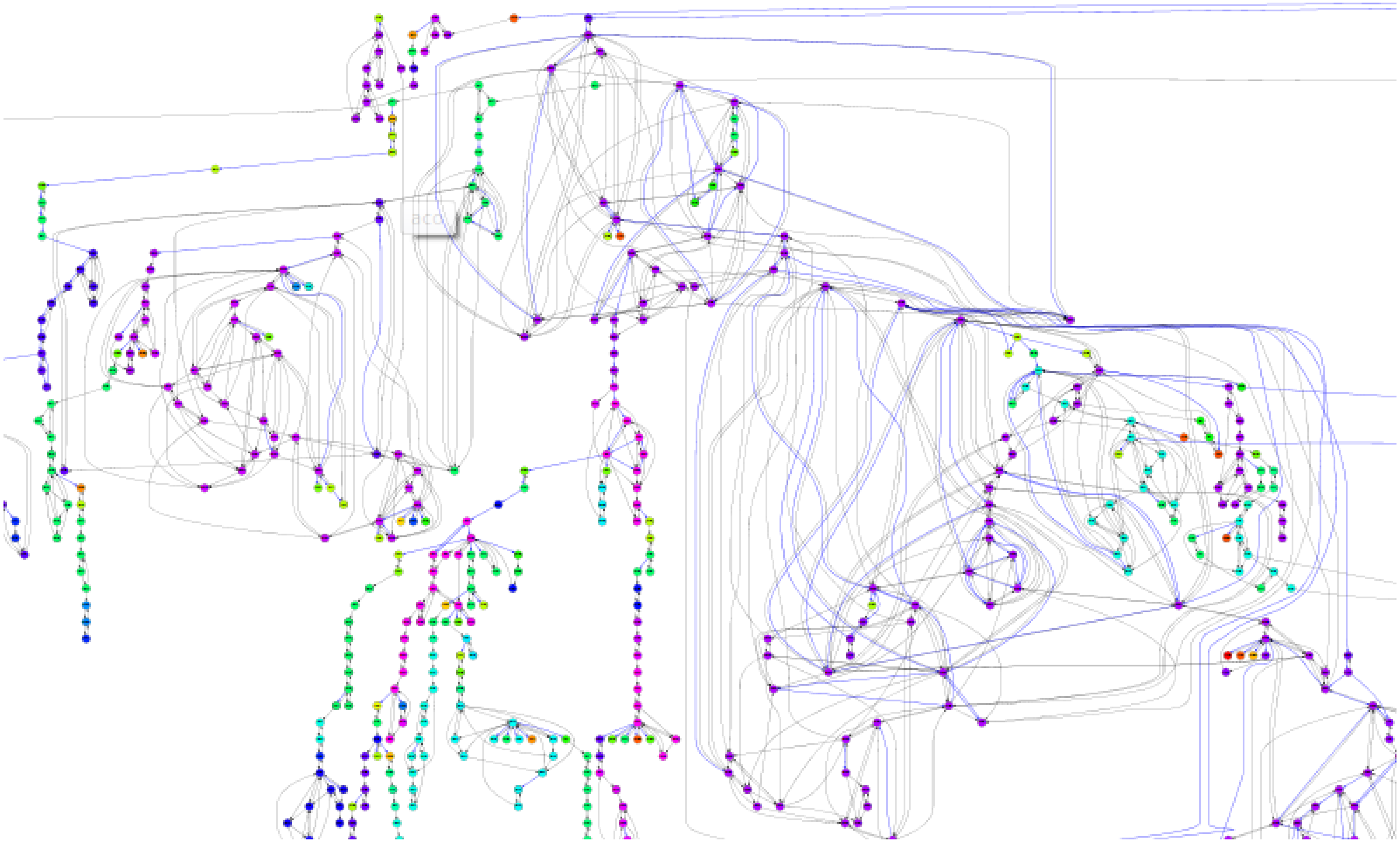
MuACOsm problem reduction

- Construction graph
  - nodes are FSMs
  - edges are mutations of FSMs
- Example

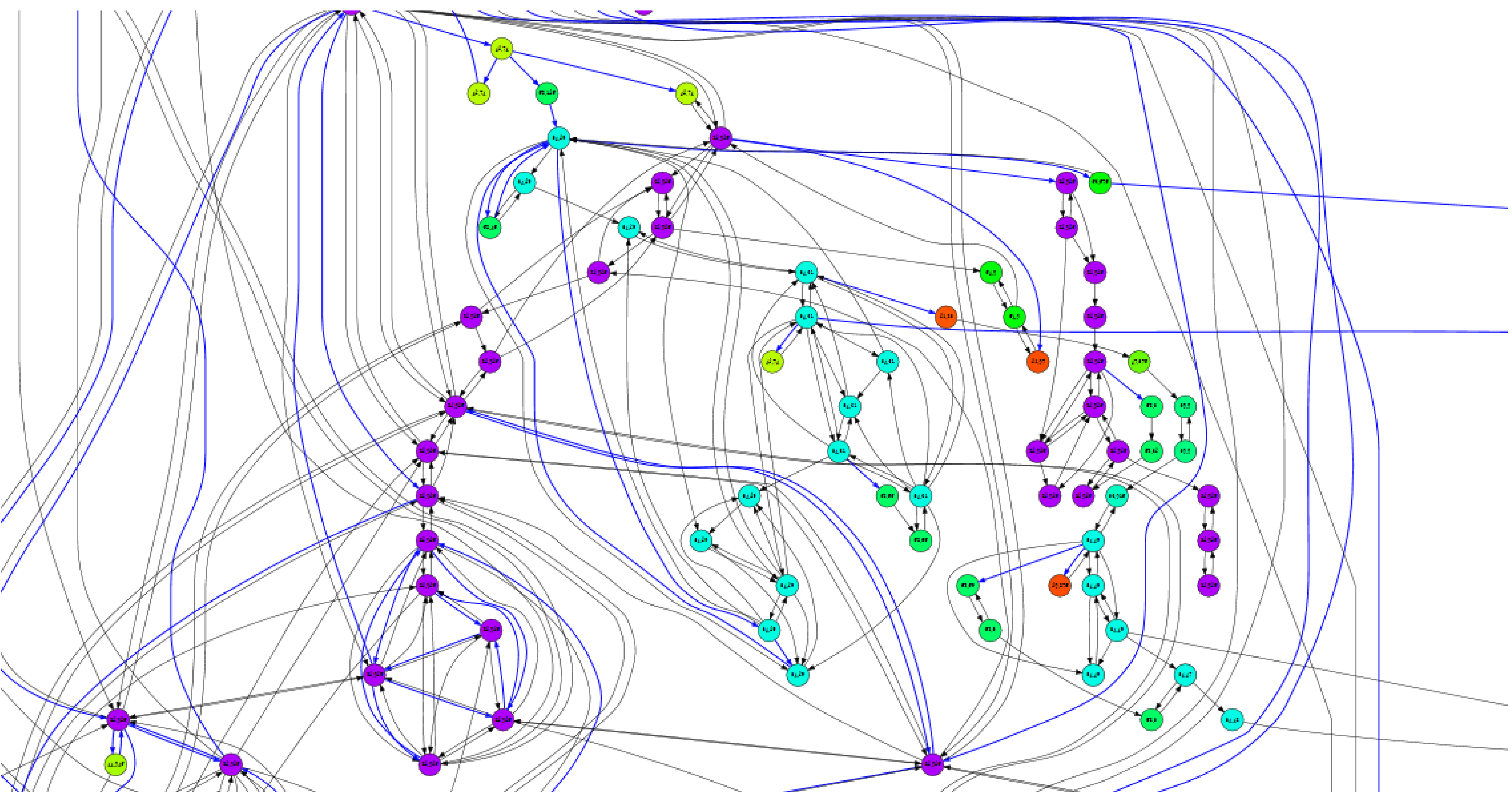


Real search space graph

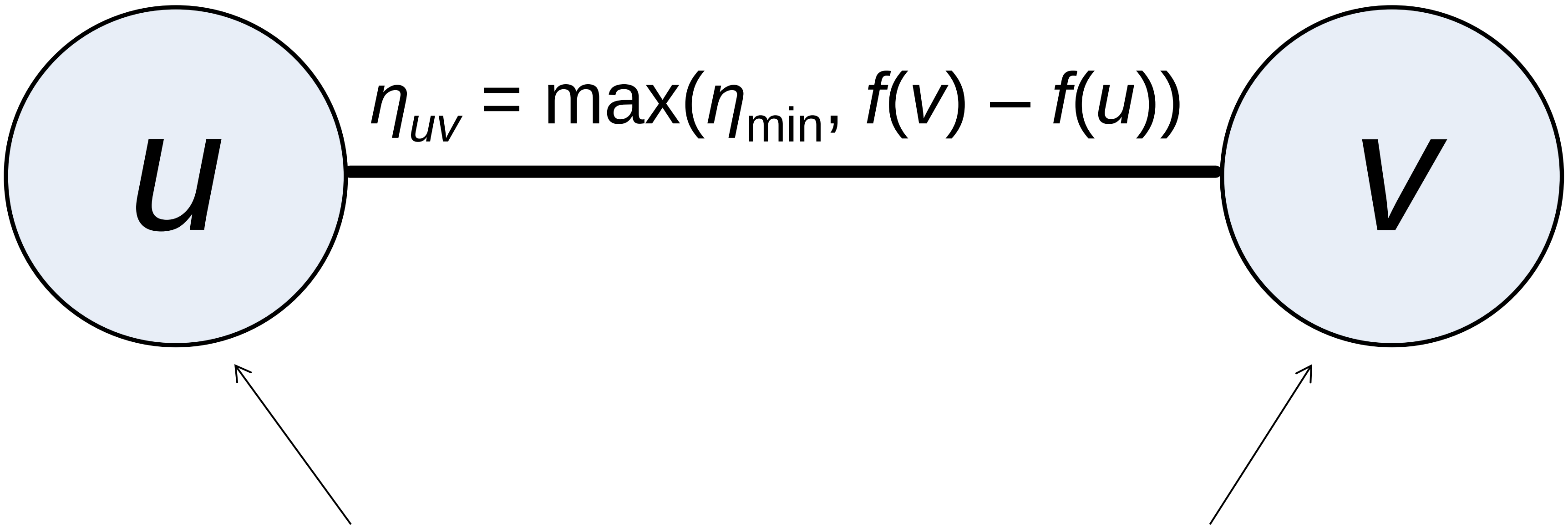
Part of real search space (1)



Part of real search space (2)



Heuristic information



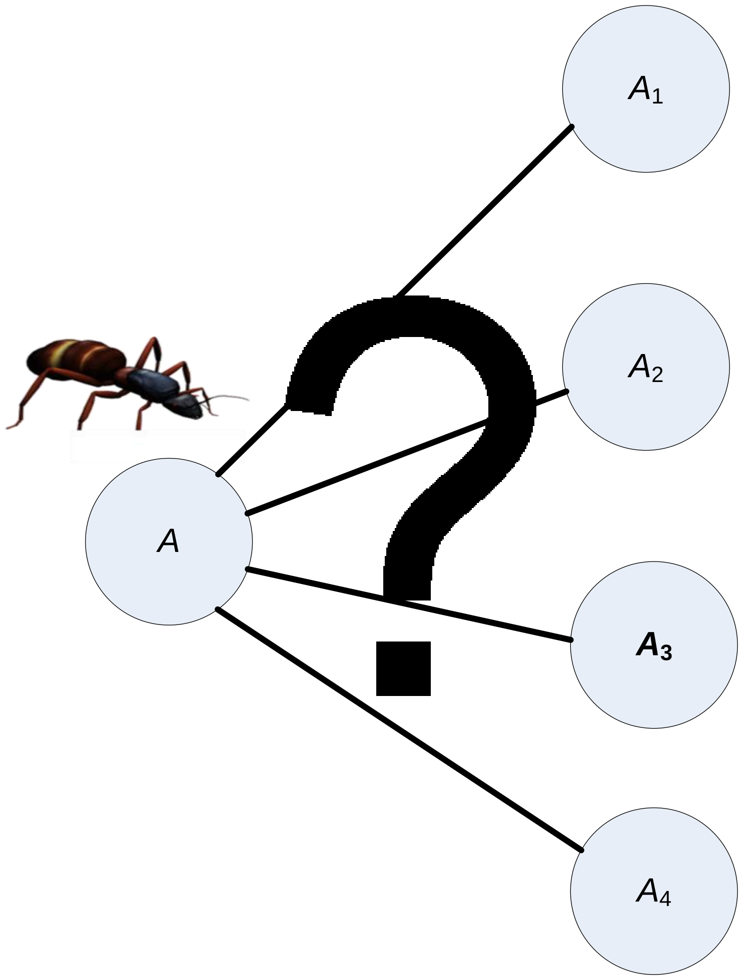
Finite-state machines

ACO algorithm

```
A_0 = random FSM
Improve A_0 with (1+1)-ES
Graph = {A_0}
while not stop() do
        ConstructAntSolutions
        UpdatePheromoneValues
        DaemonActions
```

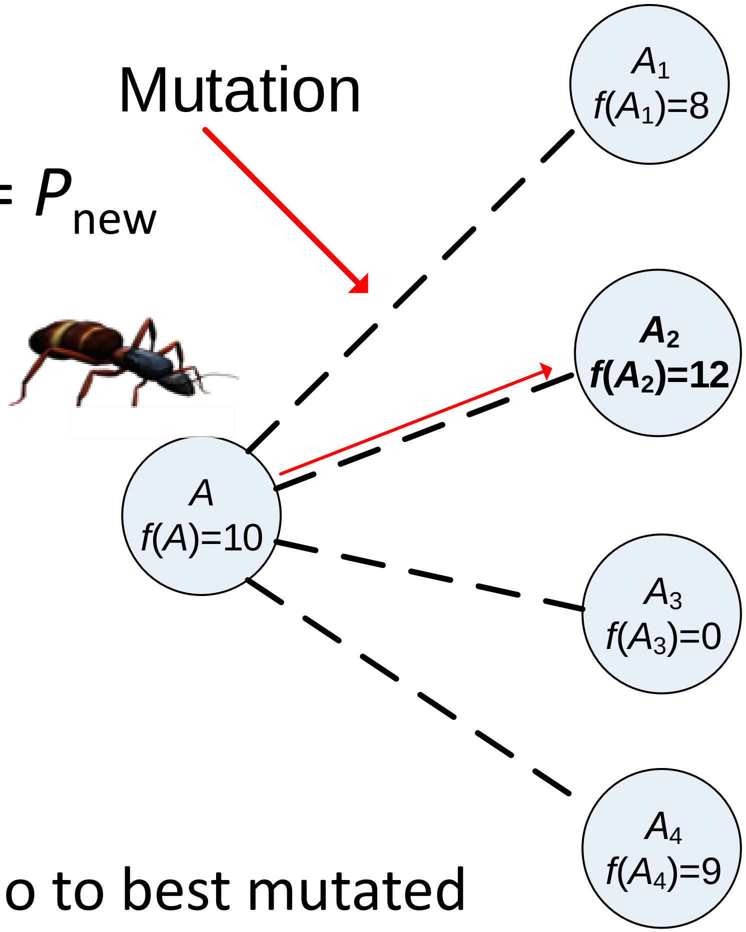
Constructing ant solutions

- Use a colony of ants
- An ant is placed on a graph node
- Each ant has a limited number of steps
- On each step the ant moves to the next node



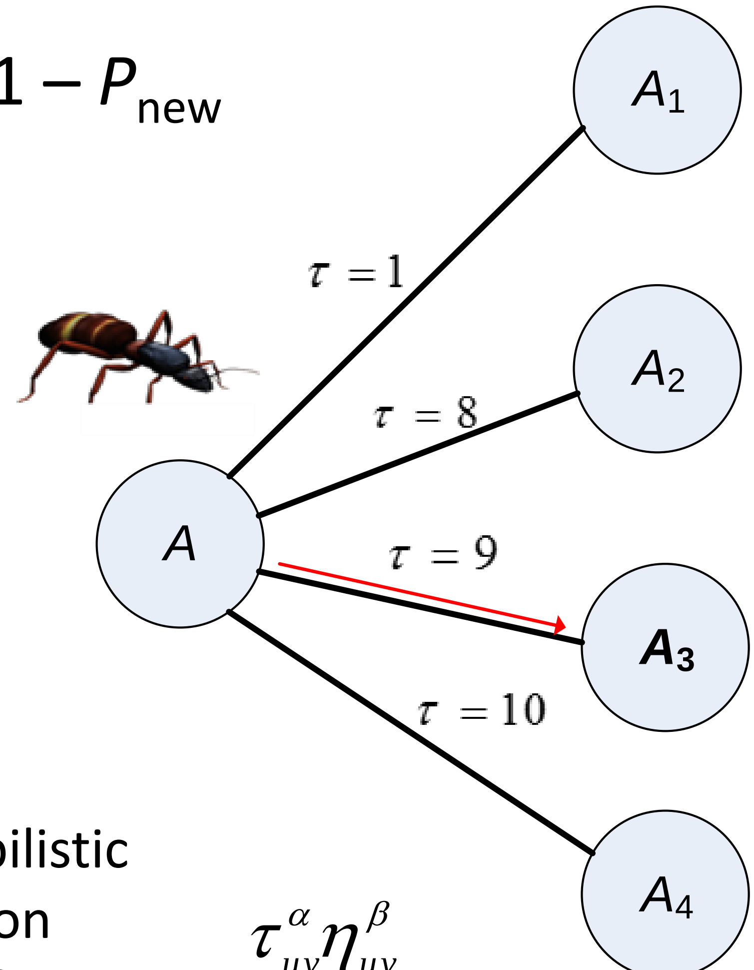
Ant step: selecting the next node

$P = P_{\text{new}}$



Go to best mutated FSM

$P = 1 - P_{\text{new}}$



Probabilistic selection

$$p_{Av} = \frac{\tau_{uv}^{\alpha}\eta_{uv}^{\beta}}{\sum_{w \in \{A1, A2, A3, A4\}} \tau_{uw}^{\alpha}\eta_{uw}^{\beta}}$$

Pheromone update

- Ant path quality = max fitness value on a path
- Update τ_{uv}^{best} – largest pheromone value deployed on edge (u, v)
- Update pheromone values:

$$\tau_{uv} = (1-\rho)\tau_{uv} + \tau_{uv}^{best}$$

- $\rho \in [0,1]$ – pheromone evaporation rate

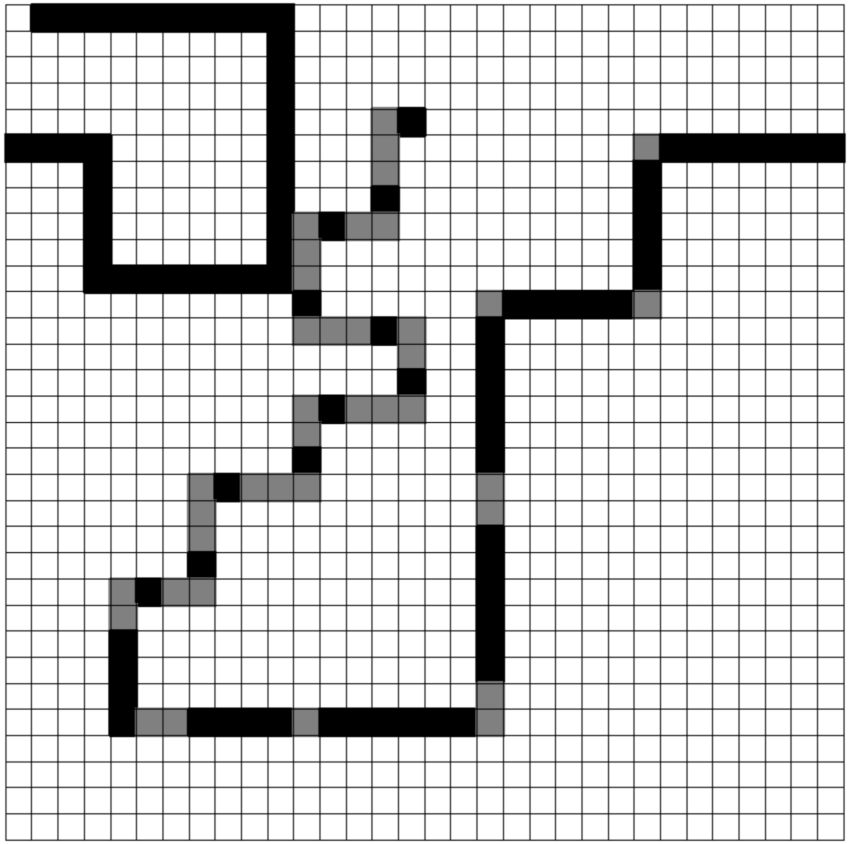
Differences from previous work

- Added heuristic information
- Changed start node selection for ants
- Coupling with (1+1)-ES

- More experiments (later)
- More comparisons with other authors
- Harder problem

“Simple” problem: Artificial Ant

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

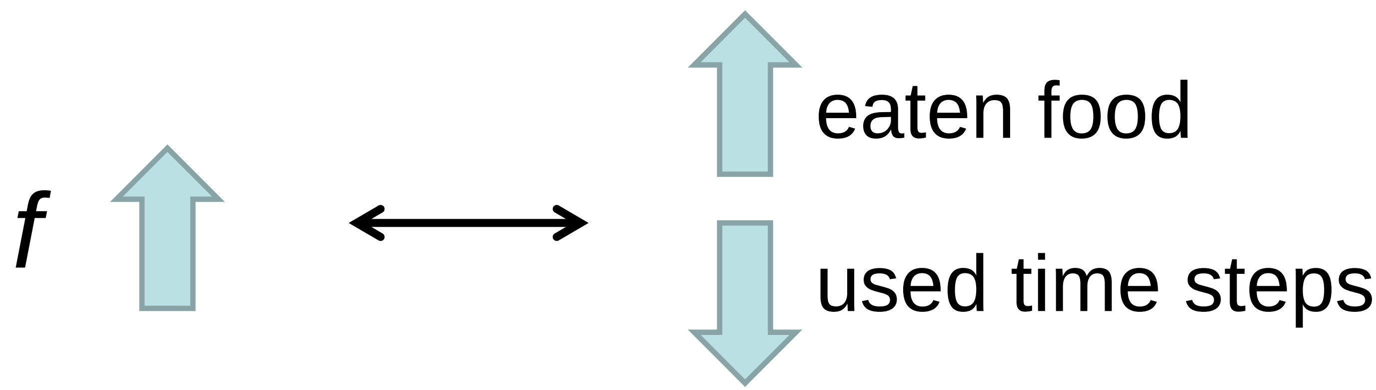


Field example: John Muir Trail

Artificial Ant: Fitness function

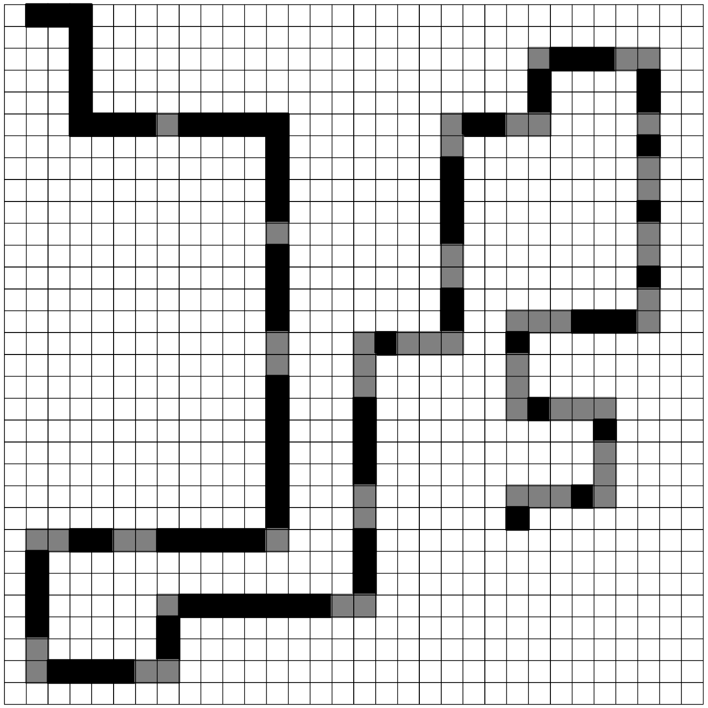
$$f = n_{\text{food}} + \frac{s_{\max} - s_{\text{last}} - 1}{s_{\max}}$$

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



“Simple” problem: Artificial Ant

- Two fields:
  - Santa Fe Trail
  - John Muir Trail
- Comparison:
  - **“Canonical” ACO**
  - **Christensen et al. (2007)**
  - **Tsarev et al. (2007)**
  - Chellapilla et al. (1999)

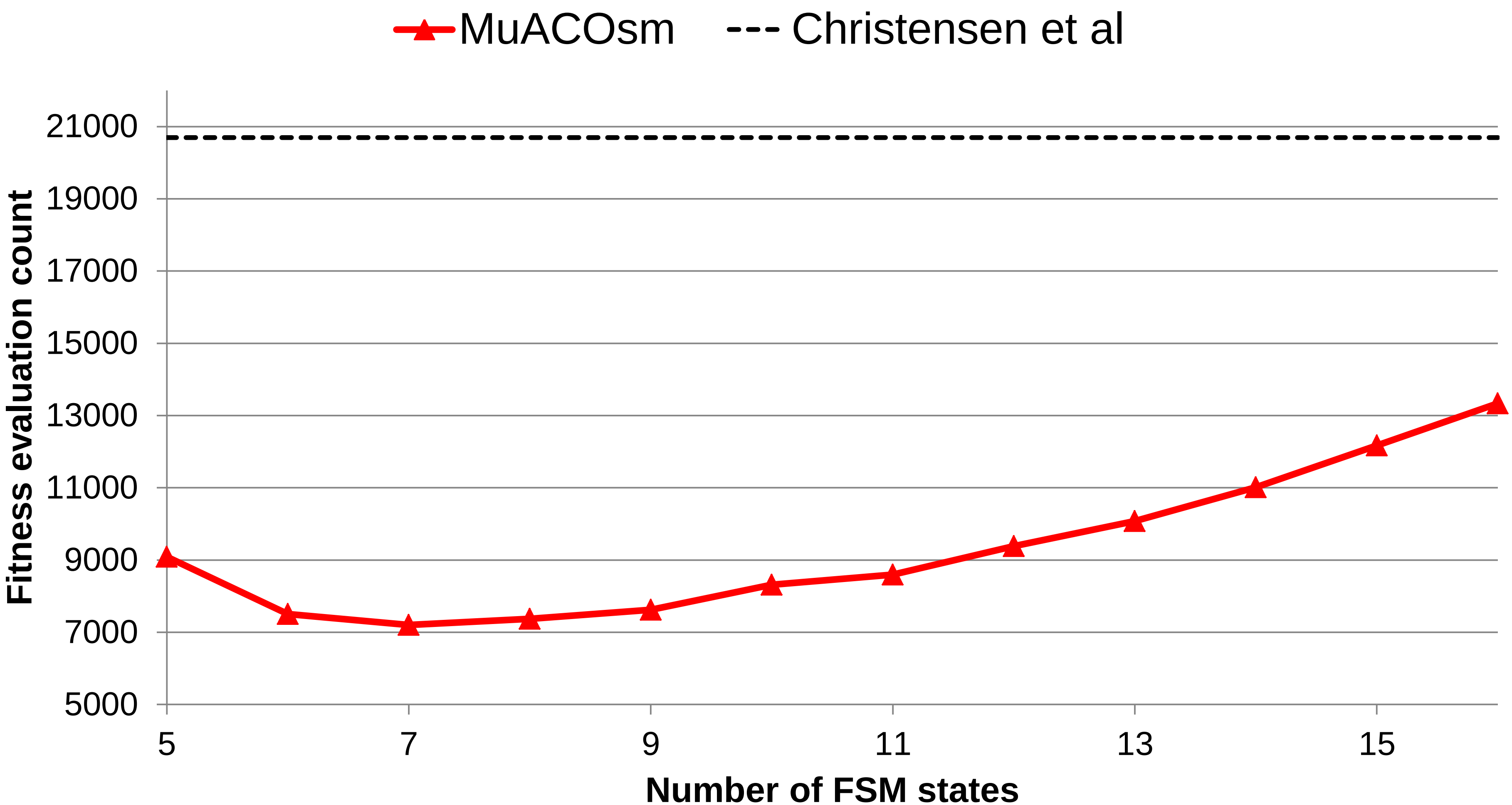


Santa Fe Trail

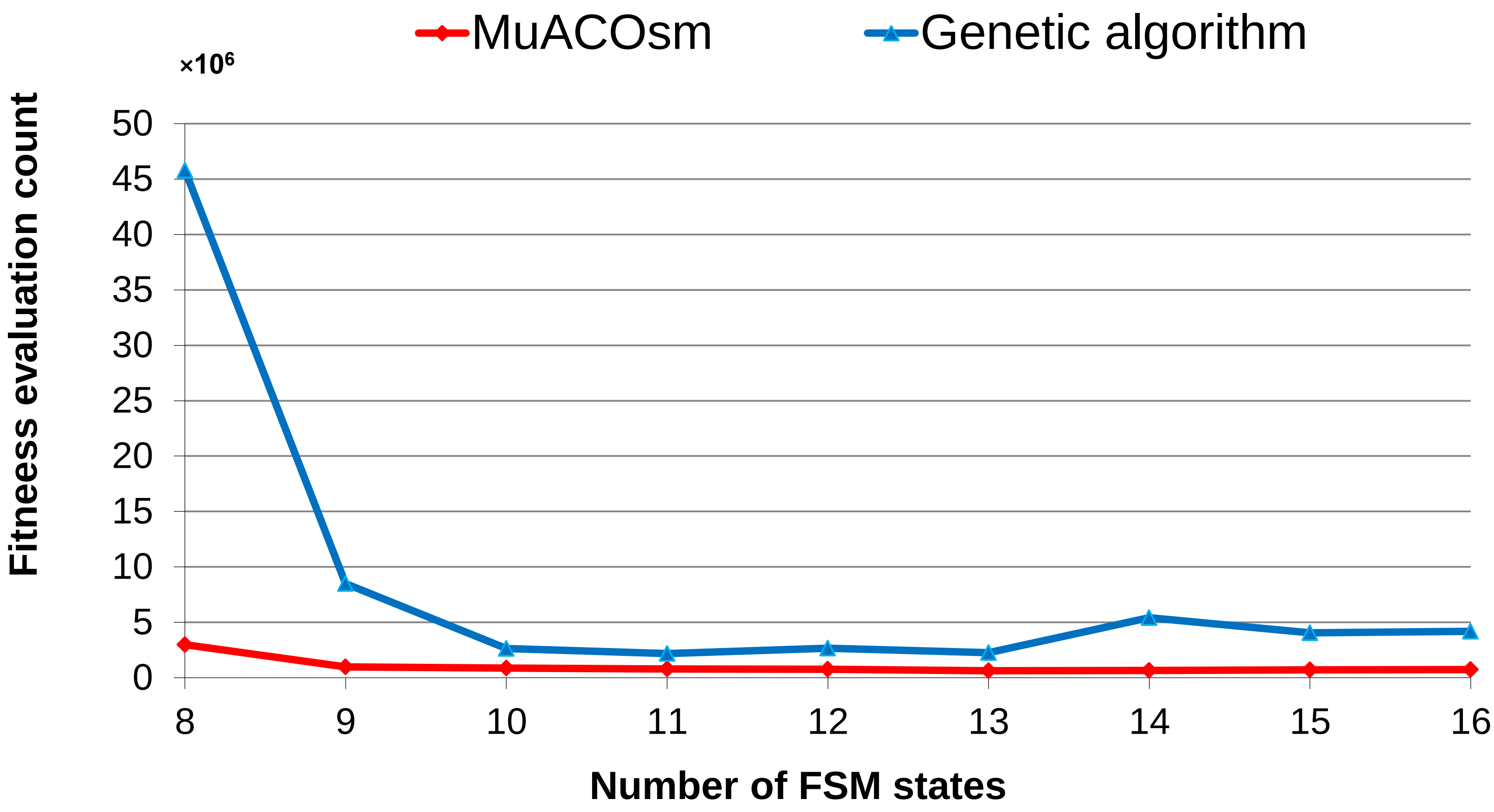
“Canonical” ACO

	“Canonical” ACO	**MuACO*sm***
State count	**Success rate, %**	**Success rate, %**
5	18	**87**
10	10	**91**

Santa Fe Trail (Christensen et al., 600 steps)

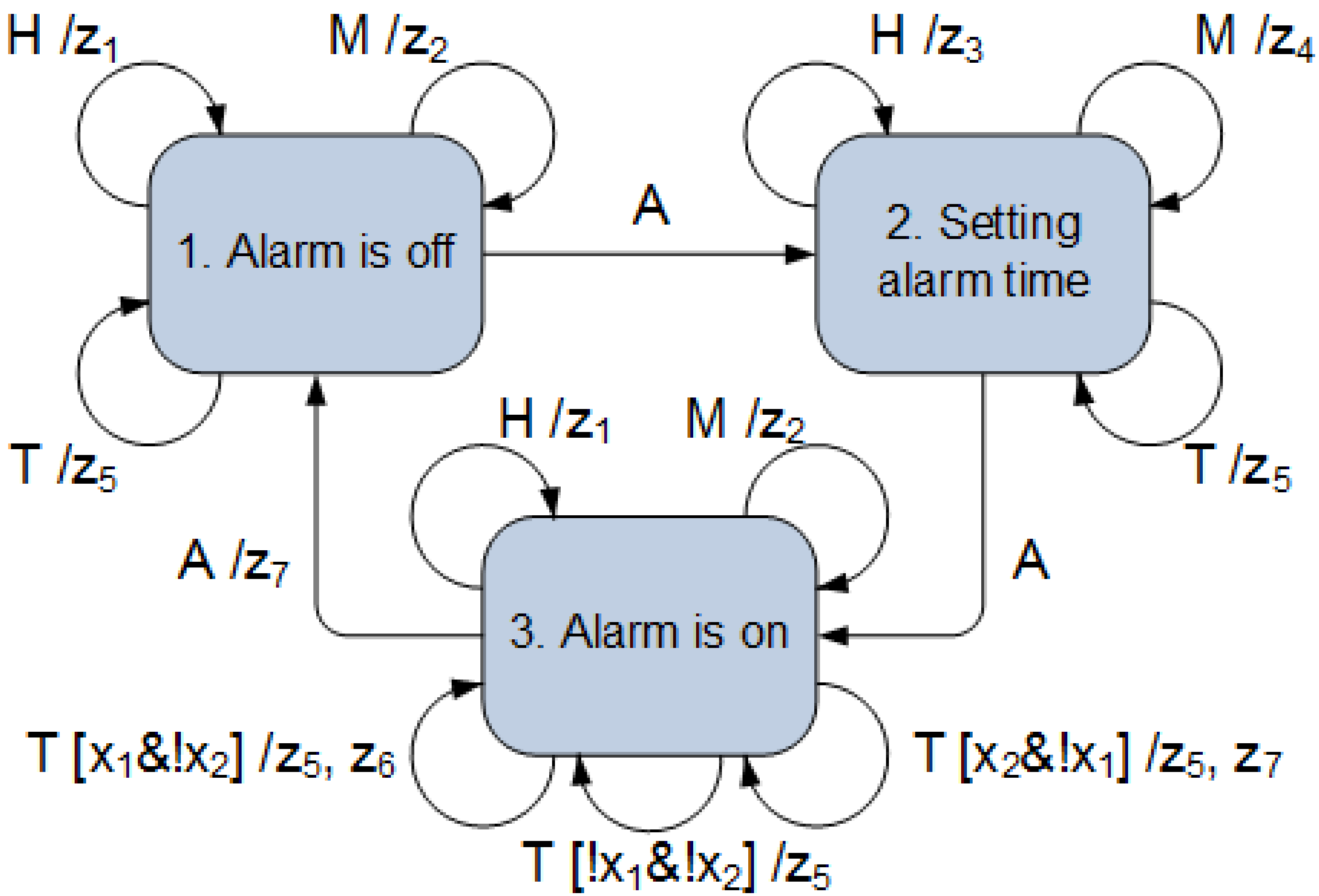


John Muir Trail (Tsarev et al., 2007): 200 steps



- MuACO*sm* is 30 times faster for FSMs with 7 states

“Harder” problem: learning Extended Finite-State Machines (1)



“Harder” problem: learning Extended Finite-State Machines (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

Learning EFSMs: Fitness function

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

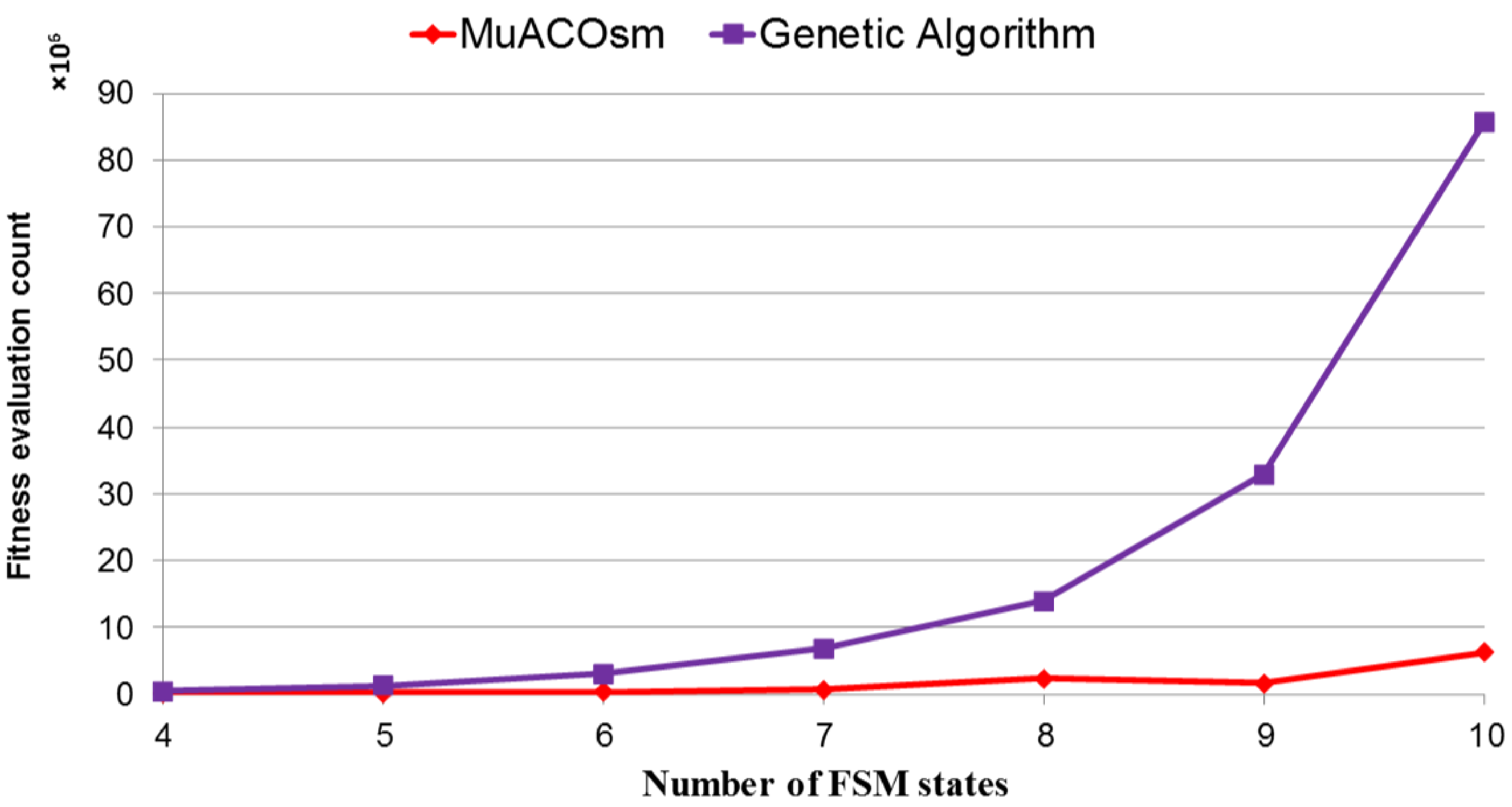
$$f' = \frac{1}{|T|} \sum_{j=1}^{|T|} \left(1 - \frac{ED(O_j, A_j)}{\max\left(len(O_j), len(A_j) \right)} \right)$$

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

Experimental setup

1. Generate random EFSM with C states
2. Generate set of tests of total length $C \times 150$
3. Learn EFSM
4. Experiment for each C repeated 100 times
5. Run until perfect fitness
6. Record mean number of fitness evaluations

Learning random EFSMs

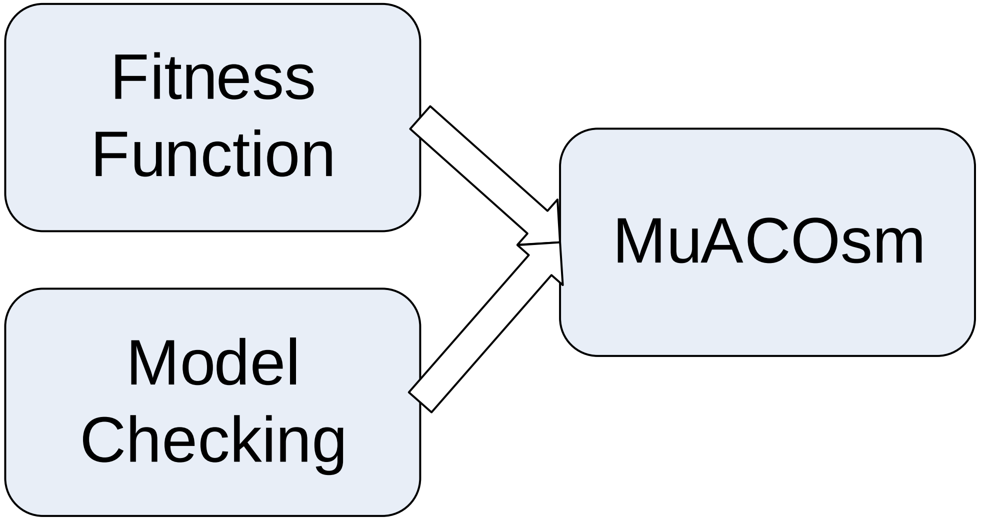


Conclusion

- Developed new ACO-based algorithm for learning FSMs and EFSMs
- MuACO*sm* greatly outperforms GA on considered problems
- Generated programs can be verified with *Model Checking*

Future work

- Better FSM representation to deal with isomorphism
- Use novelty search
- Employ verification in learning process



Acknowledgements

- We thank the ACM for the student travel grants

Thank you for your attention!

MuACOsm – A New Mutation-Based Ant Colony Optimization Algorithm for Learning Finite-State Machines

Daniil Chivililikhin
Vladimir Ulyantsev

chivdan@gmail.com

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