MuACOsm – 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

Software specification ➔ Model ➔ Code
Finite-State Machine

- $S$ – set of states
- $s_0 \in S$ – initial state
- $\Sigma$ – set of input events
- $\Delta$ – set of output actions
- $\delta: S \times \Sigma \rightarrow S$ – transition function
- $\lambda: S \times \Sigma \rightarrow \Delta$ – 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

Automated-controlled object

Finite-state machine

Events

Controlled object

Actions

Output actions

ACO for Learning FSMs
Automata-based programming: advantages

• Model before programming code, not vice versa

![Finite-state machine](image)

• 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

- $N_{\text{states}}$ – number of states
- $\Sigma$ – input events
- $\Delta$ – 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

<table>
<thead>
<tr>
<th>δ</th>
<th>Event</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>A</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Output table

<table>
<thead>
<tr>
<th>λ</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>$z_1$</td>
</tr>
<tr>
<td>2</td>
<td>$z_2$</td>
</tr>
</tbody>
</table>

ACO for Learning FSMs
“Canonical” way to apply ACO

• Reduce problem to finding a minimum cost path in some complete graph
• Vertices – FSM transitions:
  – \(<i \in S, j \in S, e \in \Sigma, a \in \Delta>\)
• Each ant adds transitions to its FSM

1 \rightarrow 1 [T/z_1] \quad \text{Ant} \quad 1 \rightarrow 2 [A/z_2]
“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\textsuperscript{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

Change transition action

Change transition end state

ACO for Learning FSMs 18
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

\[ \eta_{uv} = \max(\eta_{\min}, f(v) - f(u)) \]

Finite-state machines

ACO for Learning FSMs
ACO algorithm

$A_0 = \text{random FSM}$

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

\[ P = 1 - P_{\text{new}} \]

\[ P = P_{\text{new}} \]

Go to best mutated FSM

**ACO for Learning FSMs**
Pheromone update

• Ant path quality = max fitness value on a path

• Update $\tau_{uv}^{\text{best}}$ – largest pheromone value deployed on edge (u, v)

• Update pheromone values:

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

• $\rho \in [0, 1]$ – 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
- $s_{\text{max}}$ 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

\[ f = n_{food} + \frac{s_{max} - s_{last} - 1}{s_{max}} \]

- \( n_{food} \) – 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)
### “Canonical” ACO

<table>
<thead>
<tr>
<th>State count</th>
<th>“Canonical” ACO</th>
<th>MuACOsm</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>18</td>
<td>87</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>91</td>
</tr>
</tbody>
</table>
Santa Fe Trail (Christensen et al., 600 steps)

Fitness evaluation count

Number of FSM states

ACO for Learning FSMs
John Muir Trail (Tsarev et al., 2007): 200 steps

- MuACOsm is 30 times faster for FSMs with 7 states
“Harder” problem: learning Extended Finite-State Machines (1)

ACO for Learning FSMs
“Harder” problem: learning Extended Finite-State Machines (2)

Input data:

• Number of states \( C \) and sets \( \Sigma \) and \( \Delta \)
• Set of test examples \( T \)
• \( T_i = \langle \text{input sequence } I_j, \text{output sequence } O_j \rangle \)

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)

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

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

- MuACOsm
- Genetic Algorithm

Fitness evaluation count vs. Number of FSM states
Conclusion

• Developed new ACO-based algorithm for learning FSMs and EFSMs
• MuACO\(\text{sm}\) 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

This presentation online: