Preprints of the 2013 IFAC Conference on Manufacturing Modelling, Management, and Control, Saint Petersburg State University and Saint Petersburg National Research University of Information Technologies, Mechanics, and Optics, Saint Petersburg, Russia, June 19-21, 2013

Solving Five Instances of the Artificial Ant Problem with Ant Colony Optimization

Daniil S. Chivilikhin * **Vladimir I. Ulyantsev** ** **Anatoly A. Shalyto** ***

* *Computer Technologies Department, St. Petersburg National Research University of Information Technologies, Mechanics and Optics, St. Petersburg, Russia (e-mail: chivilikhin.daniil@gmail.com).*
** *Computer Technologies Department, St. Petersburg National Research University of Information Technologies, Mechanics and Optics, St. Petersburg, Russia (e-mail: ulyantsev@rain.ifmo.ru).*
*** *Computer Technologies Department, St. Petersburg National Research University of Information Technologies, Mechanics and Optics, St. Petersburg, Russia (e-mail: shalyto@mail.ifmo.ru).*

Abstract: The Artificial Ant problem is a common benchmark problem often used for metaheuristic algorithm performance evaluation. The problem is to find a strategy controlling an agent (called an Artificial Ant) in a game performed on a square toroidal field. Some cells of the field contain "food" pellets, which are distributed along a certain trail. In this paper we use Finite-State Machines (FSM) for strategy representation and present a new algorithm – MuACO*sm* – for learning finite-state machines. The new algorithm is based on an Ant Colony Optimization algorithm (ACO) and a graph representation of the search space. We compare the new algorithm with a genetic algorithm (GA), evolutionary strategies (ES), a genetic programming related approach and reinforcement learning on five instances of the Artificial Ant Problem.

Keywords: ant colony optimization, automata-based programming, finite-state machine, learning, induction, artificial ant problem

1. INTRODUCTION

Automata-based programming (Polykarpova and Shalyto (2009)) is a programming paradigm that proposes the use of FSMs as key components of software systems. A software system in the paradigm consists of a finite-state machine and an *automated-controlled object*. The FSM listens to input events from the environment. When an event is received, the FSM makes a transition to another state and produces a sequence of output actions which control the automated-controlled object (e.g. call its methods and functions). Although a complex software system may contain many automated-controlled objects, in this work for simplicity we focus on software systems with only one automated-controlled object.

The problem of learning FSMs received a significant amount of research over the past years. For instance, in (Spears and Gordon (2000)), evolutionary strategies were used to learn FSM controllers for the Competition for Resources problem. A random mutation hill climber (RMHC) was used in (Lucas and Reynolds (2007)) to learn finite-state transducers from test examples. Genetic algorithms were used in (Tsarev and Shalyto (2007)) to build FSMs for the Artificial Ant problem with the John Muir trail.

The Artificial Ant problem was first introduced in (Jefferson et al. (1991)) and later discussed in (Koza (1992)). The problem is to find a strategy controlling an agent (called an Artificial Ant) in a game performed on a square toroidal field. Some cells of the field contain "food" pellets, which are distributed along a certain trail that contains turns and gaps. There are three common trails (i.e. problem instances) for this problem: John Muir Trail, Santa Fe trail and Los Altos Hills trail. The goal in the problem is to build a strategy that would allow the ant to collect all pellets of food in a limited amount of steps.

Various strategy representations and learning algorithms were used for tackling this problem. The original approach by Koza used LISP S-expressions as a strategy representation and genetic programming for learning an optimal strategy. This approach was further developed in (Christensen and Oppacher (2007)), (Chellapilla (1997)), (Janikow and Mann (2005)) and, most recently, in (Karim and Ryan (2012)).

Another possible strategy representation is a finite-state machine. In (Kim (2006)) Mealy FSMs and S-experessions were evolved with GP. Experimental results demostrated that FSMs prove to be better strategy representations for the Artificial Ant problem. The authors of (Mesot et al. (2002)) developed algorithms based on reinforcement learning and evolution to learn Moore FSMs for the Santa Fe trail and the harder Los Altos Hills trail.

We have developed an FSM learning algorithm based on an Ant Colony Optimization algorithm (Dorigo (1992), Dorigo et al. (1996), Stützle and Hoos (2000), Dorigo and Gambardella (1997)) – the MuACO*sm* algorithm (Chivilikhin and Ulyantsev (2012)). In MuACO*sm* the problem of learning FSMs in stated in the following way. The target FSM is defined by the maximum number of accessible states N_{states}, a set of events Σ and a set of actions Δ. The target model problem that the FSM has to solve (e.g. the Artificial Ant problem) is formalized by a real-valued fitness function f which is defined on the set of all finite-state machines with parameters $(N_{\text{states}}, \Sigma, \Delta)$. The goal is to find an FSM A with a value of the fitness function f greater than or equal to some predefined boundary value.

In this work we present an improved version of our algorithm and experimental results of applying it to the Artificial Ant problem with five trails – John Muir trail, Santa Fe Trail, Los Altos Hills trail and two trails of our own design. MuACO*sm* is compared with a GA, a GP-related approach, a $(1+1)$-ES and reinforcement learning in terms of efficiency.

An example of applying the proposed approach to an automation-related problem is (Buzhinsky and Ulyantsev (2012)) where MuACO*sm* is used to tackle the problem of ummanned aircraft control. Examples of applying metaheuristics to automation include (Precup et al. (2012), Yoshida et al. (2000)).

2. FINITE-STATE MACHINES

A finite-state machine is a six-tuple $\langle S, s_0, \Sigma, \Delta, \delta, \lambda \rangle$, where S is a set of states, $s_0 \in S$ is the start state, Σ is a set of input events and Δ is a set of output actions. $\delta : S \times \Sigma \to S$ is the *transition* function and $\lambda : S \times \Sigma \to \Delta$ is the *action* function.

A mutation of an FSM is a rather small change in its structure. For example, a mutation can change a transition's output action or destination state. In this work we consider two following types of FSM mutations.

- **Change transition end state**. For a random transition in the FSM, the transition's end state s is set to another state selected uniformly randomly from the set $S \setminus \{s\}$.
- **Change transition action**. For a random transition in the FSM, the transition's output action a is set to another action selected uniformly randomly from the set $\Delta \setminus \{a\}$.

Excluding the current values of transition end state an action from the set of possible values ensures that a mutation causes exactly one change.

3. ACO OVERVIEW

Ant Colony Optimization algorithms are metaheuristics inspired by the foraging behavior of real ants. The first ACO algorithm called Ant System was developed in (Dorigo (1992), Dorigo et al. (1996)) and was used to tackle the traveling salesman problem. Various ACO algorithms were applied to a number of NP-hard combinatorial optimization problems, for example the sequential ordering problem, knapsack problem, bin packing, quadratic assignment problem, etc.

In order to apply ACO to a combinatorial problem, a graph representation of the search space must be devised. Solutions in ACO are built by a colony of artificial ants which use a probabilistic strategy to traverse the graph called the *construction graph*. Nodes of the construction graph represent solution components while complete solutions are represented by paths in the graph.

Each edge (u, v) of the graph (u and v are nodes of the graph) has an assigned *pheromone value* τ_{uv} and can also have an associated *heuristic information* η_{uv}. Pheromone values are modified by the ants in the process of solution construction, while the heuristic information is assigned initially and is not changed.

An ACO algorithm consists of three operations that are repeated until a viable solution is found or a stop criterion is met.

(1) `ConstructAntSolutions`. Each ant traverses the graph following a certain path. It chooses the next edge to visit according to the pheromone value and heuristic information of this edge. When an edge has been selected, the ant appends it to its path and moves to the next node. The ants continue exploring the graph until each of them has built a complete solution to the problem.
(2) `PheromoneUpdate`. Pheromone values of all graph edges are modified. A particular pheromone value can increase if the edge it is associated with has been traversed by an ant or it can decrease due to *pheromone evaporation*. The amount of pheromone that each ant deposits on a graph edge depends on the quality of the solution built by this ant measured by the fitness function value of this solution.

4. MUTATION-BASED ACO FOR LEARNING FINITE-STATE MACHINES

In this section we provide a full description of the new algorithm. First we describe the representation of the search space in the form of a directed graph and discuss the key difference of the proposed ACO algorithm from other ACO algorithms. Then we describe the ACO-related steps of our algorithm, including the initial solution generation procedure, heuristic information definition, ant solution constuction procedure, pheromone update rule and daemon actions. The section ends with a highlight of the key differences of the proposed algorithm and its previous version.

4.1 Search Space Representation

The search space, which is a set of all FSMs with the specified parameters $(N_{\text{states}}, \Sigma, \Delta)$, is represented in the form of a directed graph G, where nodes are associated with FSMs and edges are associated with FSM mutations. The following two properties hold for graph G.

(1) Let u be a node associated with FSM A_1 and v be a node associated with FSM A_2. If machine A_2 lays within one mutation from A_1, then a fully constructed

graph G contains edges $u \to v$ and $v \to u$. Otherwise, nodes u and v are not connected with an edge.

(2) Consequently, for each pair of FSMs A_1 and A_2 and the corresponding pair of nodes u and v, a fully constructed graph G contains paths from u to v and also from v to u.

Note that a fully constructed graph G should contain all FSMs with parameters $(N_{\text{states}}, \Sigma, \Delta)$. Constructing such a graph would effectively mean performing a brute force search on the search space.

The described representation of the search space, or problem reduction, is very different from the classical problem reduction usually performed in ACO. Typically, in order to solve a combinatorial problem with ACO, it is mapped on some problem where graph nodes represent solution components while full solutions are represented by sequences of components. The goal is then to find a minimum or maximum cost path in this graph.

In our problem mapping described above, graph nodes represent *complete* solutions themselves. The ants travel between solutions, which are actually built by an external local search procedure – FSM mutation.

4.2 Initial Solution Generation

In the beginning of the algorithm a random initial solution is generated by filling both the transition and action tables of the FSM with values selected uniformly randomly from sets of all possible values. Next, a simple RMHC is applied to the random solution for one hundred fitness evaluations.

The RMHC, or a (1+1)-ES, stores one current solution. On each step it performs one random mutation of the current solution and calculates the fitness function value of the new solution. The current solution is replaced by the new solution if the new solutions's fitness function value is greater than or equal to the current solution's fitness value.

4.3 Heuristic Information

As discussed before in Section 3, each edge of the construction graph can have an associated value called heuristic information. Consider an edge $(u \to v)$, where u and v are nodes of the construction graph G. The heuristic information η_{uv} is calculated as follows:

$$\eta_{uv} = max\left(\eta_{\min}, f(v) - f(u)\right),$$

where $\eta_{\min}$ is a small positive constant value. This ensures that the heuristic information is always positive.

4.4 Constructing Ant Solutions

On this step all ants traverse the graph and construct solutions. First, a start node is selected for each ant. We have experimented with several ways to select start nodes. The most efficient way we found was to select a single start node which is associated with the current best solution – solution with the largest fitness value. All ants use this best node as a start node.

Second, all ants explore the graph while searching for good solutions. On each colony iteration all ants traverse the graph until they stop. Let the artificial ant be located in a node u associated with FSM A. If this node has adjacent nodes, then the ant selects the next node v to visit according to the rules discussed below.

(1) **Expansion.** The ant attempts to construct new solutions by making N_{mut} mutations of A. The procedure of processing a single mutation of machine A is as follows. First, a mutated FSM A_{mut} is constructed. Then we try to find a node t in graph G associated with A_{mut}. If G does not contain such a node, we construct a new node and associate it with A_{mut}. Finally we add an edge (u, t) to G. After all N_{mut} mutations have been performed, the ant selects the best newly constructed node v and moves to that node.

(2) **ACO path selection.** The ant probabilistically selects the next node from the existing successors set N_u of node u. Node $v \in N_u$ is selected with a probability defined by the classical ACO formula:

$$p_{uv} = \frac{\tau_{uv}^{\alpha} \cdot \eta_{uv}^{\beta}}{\sum_{w \in N_u} \tau_{uw}^{\alpha} \cdot \eta_{uw}^{\beta}},$$

where $\alpha, \beta \in [0, 1]$ define the influence of pheromone values and heuristic information on path selection.

The ant uses the expansion rule with a probability of p_{new} and the ACO path selection rule with a probability of $1 - p_{\text{new}}$. If node u does not have adjacent nodes, then the next node is always selected using the expansion rule.

Each ant in the colony is given at most n_{stag} steps to make without an increase in its best fitness value; when the ant exceeds this number, it is stopped. Ants take turns to make one move until all the ants stop. The whole colony is given at most N_{stag} iterations to run without an increase of the best fitness value before the algorithm is restarted.

4.5 Pheromone Update

We use our own pheromone update rule we call the *global elitist min-bound pheromone update* based on the MAX MIN rule (Stützle and Hoos (2000)) and the elitist rule (Dorigo (1992)), that fits our problem better than any other.

For each graph edge (u, v) we store τ_{uv}^{best} – the best pheromone value that any ant has ever deposited on edge (u, v). For each ant path, a sub-path is selected that spans from the start of the path to the best node in the path. The values of τ_{uv}^{best} are updated for all edges along this sub-path. Next, for each graph edge (u, v), the pheromone value is updated according to the formula:

$$\tau_{uv} = max(\tau_{\min}, \rho\tau_{uv} + \tau_{uv}^{\text{best}}),$$

where $\rho \in [0, 1]$ is the evaporation rate and $\tau_{\min}$ is an empirically selected minimum pheromone bound (we use a value of 0.001).

4.6 Differences from Previous Version

The MuACO*sm* algorithm described above differs from the original version from Chivilikhin and Ulyantsev (2012) in several key points. First, on the step of initial solution generation the original algorithm used the random solution as the root node of the construction graph. Later it was found

that the use of a random mutation hill climber described in section 4.2 for about a hundred fitness evaluations boosts up performance by several percent.

Second, the original algorithm did not use heuristic information and, consequently, search was biased by pheromone trails only. Introduction of heuristic information typically increases performance of ACO algorithms, which is also the case for our algorithm. And third, in the last version of MuACO*sm* we introduced a lower bound on pheromone values $\tau_{\min}$, which helped us overcome the issue of unbounded pheromone evaporation.

5. SOLVING THE ARTIFICIAL ANT PROBLEM

The problem is to find a strategy represented by an FSM for controlling an agent (called an Artificial Ant) in a game performed on a square toroidal field. Some cells of the field contain "food" pellets, which are distributed along a certain trail.

The ant's initial position is the leftmost upper cell and it is initially "looking" east. It can determine whether the next cell contains a piece of food or not. On each step it can turn left, turn right or move forward, eating a piece of food if the next cell contains one. The goal is to build an FSM that would allow the ant to eat all food in a limited number of steps. Examples of fields are shown on Fig. 1-4.

In this problem there are two input events – F (the next cell contains food) and $!F$ (the next cell does not contain food) – and three output actions: L (turn left), R (turn right) and M (move forward). The fitness function we use is defined in the following way:

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

where n_{food} is the number of food pellets eaten by the ant, $s_{\max}$ is the maximum number of steps the ant is allowed to make and s_{last} is the number of the step on which the ant ate the last piece of food.

5.1 Santa Fe Trail

The Santa Fe trail contains 89 food pellets, 21 turns and has a total length of 144. Many published results are available for the Artificial Ant problem with the Santa Fe Trail. The best results were achieved in (Christensen and Oppacher (2007)) by two approaches: Memorized-Random-Tree-Search with GP + Subroutines (43,000 fitness evaluations) and Memorized-Random-Tree-Search with Random Search + Subroutines (20,696 fitness evaluations).

These approaches use LISP S-expression representation of programs instead of finite-state machines. However, the comparison we perform is still feasible because an S-expression can be easily transformed into a finite-state machine (Kim (2006)). Furthermore, we know of no approaches that would build FSMs for the Santa Fe trail faster than the approach in (Christensen and Oppacher (2007)).

In our experimental setup we varied both the number of accessible states N_{states} in the target FSM and the maximum number of steps $s_{\max}$ the artificial ant is allowed to make. We used the following values of parameters:

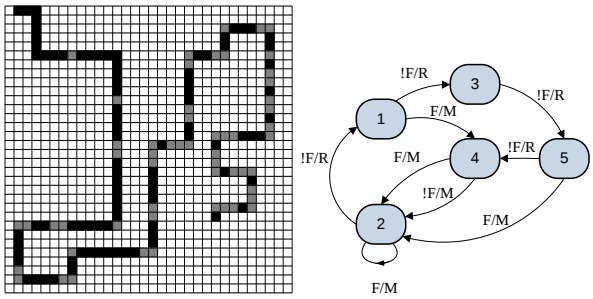

Fig. 1. The Santa Fe trail (left) an FSM that allows the ant to eat all food in 394 steps.

$N_{\text{ants}} = 5$, $\rho = 0.5$, $n_{\text{stag}} = 50$, $N_{\text{stag}} = 100$, $N_{\text{mut}} = 20$, $p_{\text{new}} = 0.6$, $\alpha = \beta = 1$.

The experiment was run 10,000 times for each case to achieve statistically meaningful results. MuACO*sm* was able to find an FSM capable of eating all pellets of food in each run. Table 1 shows the mean number of fitness evaluations for each number of FSM states and maximum number of ant steps $s_{\max}$

Table 1. Mean fitness evaluations for the Santa Fe trail

	$s_{\max}$		
N_{states}	**400**	**500**	**600**
5	10975	9538	9087
6	9313	7722	7503
7	9221	7469	7203
8	9882	7668	7371
9	10665	8172	7625
10	11717	8785	8318
11	12832	9365	8598
12	14146	10215	9388
13	16000	10869	10075
14	17591	12047	11015
15	19775	13050	12174
16	21505	14362	13337

Results in Table 1 indicate that MuACO*sm* was able to achieve a computational effort less than any other published algorithm. In particular, the smallest computational effort of 7203 mean fitness evaluations for $s_{\max} = 600$ was achieved for target FSMs with seven states. It is approximately three times less than the mean effort of 20,696 published in (Christensen and Oppacher (2007)). A state diagram of one of the found FSM with five states is shown on Fig. 1. The start state in all diagrams in this paper is always state "1".

5.2 John Muir Trail

The John Muir trail is the original trail used in Jefferson et al. (1991). It has a total length of 121, contains 89 food pellets and 33 turns. For the John Muir trail we compare our results with those achived with GA in (Tsarev and Shalyto (2007)). Both the MuACO*sm* and the GA were run 100 times until completion. The number of FSM states N_{states} was varied from 7 to 16. FSMs with less then seven states were not considered due to the fact that no FSM with less then seven states can solve the problem in $s_{\max} = 200$ steps as it was proven in (Tsarev and Shalyto (2007)).

We used the same parameter values as for the Santa Fe trail, except for setting $N_{\text{ants}} = 10$, $N_{\text{stag}} = 200$, $N_{\text{mut}} = 60$.

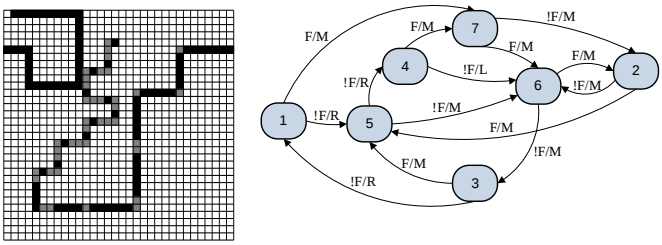

Fig. 2. The John Muir trail (left) and an FSM that allows the ant to eat all food in 189 time steps

Results presented in Table 2 state that MuACO*sm* is always several times better than GA and, in particular, 60 times better on minimal FSMs with seven states. One of the generated FSMs with seven states is shown on Fig. 2. Another result was acquired with the use of a (1+1)-ES provided by our colleague. The ES was restarted every 10,000 fitness evaluations regardless of the progress. The ES was launched for target FSMs with seven states and achieved an average effort of 46.961×10^6 fitness evaluations which is close to the best effort achieved by our algorithm.

Table 2. Mean fitness evaluations ($\times 10^6$) for the John Muir trail

	Algorithm	
N_{states}	**MuACO*sm***	**GA**
7	31.826	1799.630
8	2.978	45.785
9	0.989	8.503
10	0.864	2.626
11	0.800	2.173
12	0.776	2.666
13	0.628	2.273
14	0.654	5.431
15	0.703	4.061
16	0.736	4.200

5.3 Los Altos Hills Trail

The Los Altos Hills trail shown on Fig. 3 has a total length of 221, contains 157 food pellets and 29 turns. This is probably the hardest common instance for the Artificial Ant problem. This trail has been tackled by quite a few researchers.

Koza's original GP (Koza (1992)) generated a solution that achieved the perfect score using 1808 ant steps. A variation of Grammatical Evolution was developed in (Karim and Ryan (2012)) that was able to solve the Santa Fe trail, but never solved the Los Altos Hills trail – the best evolved programs achieved a mean fitness of only about 139 out of 157. The authors of (Mesot et al. (2002)) acquired a Moore FSM that achieved a perfect fitness value of 157 in 679 steps with the use of a hybrid algorithm based on reinforcement learning, adaptive reward and evolution. However, since no statistical data was provided it is not clear if the presented approach performs well on average. Furthermore, to our knowledge there are no published

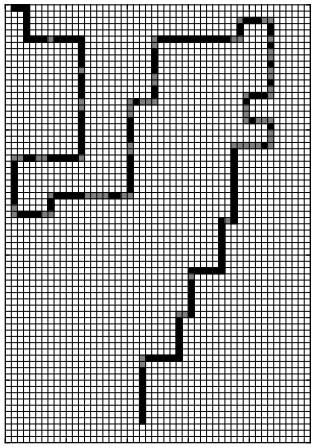

Fig. 3. The Los Altos Hills trail – a 69×50 upper left part of the whole 100×100 field.

results on learning Mealy FSMs for the Los Altos Hills problem.

Our experimental setup allowed the ant to make a maximum of $s_{\max} = 800$ steps. Results of experimental launches with N_{states} varied from 7 to 16 are presented in Table 3. We used the same parameter values as for the John Muir trail. Experiments for each number of FSM states were performed 50 times. It is interesting that the best mean s_{last} was reached for the smallest FSMs with seven states while the best minimal s_{last}=397 was reached for FSMs with twelve states.

Table 3. Experimental results for the Los Altos Hills trail with $s_{\max}$

N_{states}	**Mean evaluations**	**Mean s_{last}**	**Min s_{last}**
7	10.026×10^6	606.2	517
8	3.184×10^6	657.4	504
9	2.818×10^6	661	517
10	2.337×10^6	651.8	471
11	1.984×10^6	667	493
12	2.304×10^6	668.6	397
13	2.000×10^6	668.6	484
14	2.093×10^6	661.4	497
15	1.821×10^6	659.8	404
16	1.686×10^6	624.6	460

5.4 Auxiliary Trails

The auxiliary trails that are shown on Fig. 4 were introduced by our colleague during his research and proved to be on par with the common trails in terms of required computational effort. Both trails are located on 20×20 toroidal fields. The first trail has a total length of 119, contains 91 food pellets and 45 turns. We searched for target FSMs with six states and limited the number of ant steps to $s_{\max} = 283$. The second trail has a total length of 104, contains 69 food pellets and 33 turns. The target FSMs also contained six nominal states and the number of ant steps was limited to $s_{\max} = 266$. Each experiment was repeated 100 times. The same parameter values as for the Santa Fe trail were used.

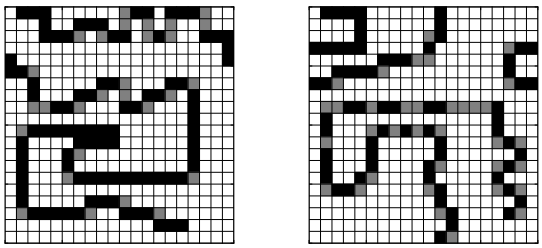

Fig. 4. First (on the left) and second (on the right) auxiliary trails

For the first trail MuACO*sm* achieved a mean of 107.33×10^6 while the ES scored 500.23×10^6. For the second trail MuACO*sm* scored 29.78×10^6 and the ES scored 187.14×10^6. These experimental results indicate that on average MuACO*sm* is about five times faster than the ES on these trails.

6. CONCLUSION

Experimental results presented in the previous section indicate that the new MuACO*sm* algorithm significantly outperforms all other published algorithms on five presented instances of the Artificial Ant problem, including GA, (1+1)-ES, reinforcement learning and a GP-related approach.

One direction of our current work is generalization of the presented MuACO*sm* algorithm for applying it to learning other automata types as well as solving other combinatorial optimization problems apart from learning automata.

7. ACKNOWLEDGEMENTS

We are grateful to Maxim Buzdalov from St. Petersburg National Research University of ITMO for providing the auxilary ant trails and results of the (1+1)-ES.

REFERENCES

Buzhinsky, I. and Ulyantsev, V. (2012). Use of ant colony optimization for inducing finite-state machines from tests for controlling systems with complex behavior. In *Proceedings of the XV International conference on soft computations and measurements (SCM'12)*, volume 1, 250–253. In Russian.

Chellapilla, K. (1997). Evolutionary programming with tree mutations: Evolving computer programs without crossover. In *Koza, J.R. et al., eds., Genetic Programming 1997: Proceedings of the Second Annual Conference*. Morgan Kaufmann.

Chivilikhin, D. and Ulyantsev, V. (2012). Learning finite-state machines with ant colony optimization. In *Proceedings of the 8th international conference on Swarm Intelligence*, ANTS'12, 268–275. Springer-Verlag, Berlin, Heidelberg.

Christensen, S. and Oppacher, F. (2007). Solving the artificial ant on the santa fe trail problem in 20,696 fitness evaluations. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, 1574–1579. ACM, New York, NY, USA.

Dorigo, M. (1992). Optimization, learning and natural algorithms. PhD thesis.

Dorigo, M. and Gambardella, L. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1), 53–66.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(1), 29–41.

Janikow, C.Z. and Mann, C.J. (2005). Cgp visits the santa fe trail: effects of heuristics on gp. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, 1697–1704. ACM, New York, NY, USA.

Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C., and Wang, A. (1991). Evolution as a theme in artificial life. *Artificial Life II*.

Karim, M.R. and Ryan, C. (2012). Sensitive ants are sensible ants. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, 775–782. ACM, New York, NY, USA.

Kim, D. (2006). Memory analysis and significance test for agent behaviours. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, 151–158. ACM, New York, NY, USA.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press. Cambridge, MA, USA.

Lucas, S.M. and Reynolds, T.J. (2007). Learning finite-state transducers: Evolution versus heuristic state merging. *IEEE Transactions on Evolutionary Computation*, 11(3), 308–325.

Mesot, B., Sanchez, E., Pena, C., and Perez-Uribe, A. (2002). SOS++: Finding Smart Behaviors Using Learning and Evolution. In *Artificial Life 8*, 264–273.

Polykarpova, N. and Shalyto, A. (2009). *Automata-based programming*. Piter. In Russian.

Precup, R., David, R.C., Petriu, E., Preitl, S., and Radac, M. (2012). Fuzzy control systems with reduced parametric sensitivity based on simulated annealing. *Industrial Electronics, IEEE Transactions on*, 59(8), 3049–3061.

Spears, W.M. and Gordon, D.F. (2000). Evolving finite-state machine strategies for protecting resources. In *Proceedings of the 12th International Symposium on Foundations of Intelligent Systems*, ISMIS '00, 166–175. Springer-Verlag, London, UK, UK.

Stützle, T. and Hoos, H.H. (2000). Max-min ant system. *Future Generation Computer Systems*, 16(9), 889–914.

Tsarev, F. and Shalyto, A. (2007). Use of genetic programming for finite-state machine generation in the smart ant problem. *Proceedings of the IV International Scientific-practical conference "Integrated models and soft calculations in artificial intelligence"*., (2), 590–597. In Russian.

Yoshida, H., Kawata, K., Fukuyama, Y., Takayama, S., and Nakanishi, Y. (2000). A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *Power Systems, IEEE Transactions on*, 15(4), 1232–1239.