ISSN 1064-2307, Journal of Computer and Systems Sciences International, 2014, Vol. 53, No. 2, pp. 256–266. © Pleiades Publishing, Ltd., 2014.
Original Russian Text © I.P. Buzhinsky, V.I. Ulyantsev, D.S. Chivilikhin, A.A. Shalyto, 2014, published in Izvestiya Akademii Nauk. Teoriya i Sistemy Upravleniya, 2014, No. 2, pp. 111–121.

ARTIFICIAL INTELLIGENCE

Inducing Finite State Machines from Training Samples Using Ant Colony Optimization

I. P. Buzhinsky, V. I. Ulyantsev, D. S. Chivilikhin, and A. A. Shalyto

ITMO University, St. Petersburg, Russia

e-mail: buzhinsky@rain.ifmo.ru

Received June 18, 2013; in final form, October 25, 2013

Abstract—A method for control finite state machine (FSM) induction in which an ant colony optimization algorithm is used for search optimization is proposed. The efficiency of this method is estimated using the generation of FSMs for controlling a model of an unmanned aerial vehicle (UAV). It is shown that the proposed method outperforms the method based on genetic algorithms both in terms of performance and quality.

DOI: 10.1134/S106423071402004X

INTRODUCTION

Automata-based programming is a paradigm in which programs are designed as sets of automated controlled objects [1]. An automated controlled object consists of a plant (object under control) and a control system, which can include one or several finite state machines (FSMs) or automata. One of the domains where the use of automata-based programming is reasonable is the control of plants with complex behavior; these are the plants that may exhibit different behavior under identical control inputs. Examples of the application of automata-based programming for this class of problems can be found in [2–4]. In [3, 4], the FSMs are constructed automatically because it is difficult to design them manually. There are problems in which FSMs cannot be constructed manually at all.

To automate the construction of FSMs, a performance measure must be defined. One way of doing this is to specify a *fitness function*, which assigns a real number to each FSM. An FSM with a prescribed value of the fitness function can be found using search optimization algorithms (e.g., evolutionary algorithms [5–9]). An alternative approach to defining the performance measure is to describe a set of constraints that must be satisfied by the FSM. This approach was used in [10, 11], where the generation of the FSM is reduced to the Boolean satisfiability problem (SAT) [12].

In [13], a genetic algorithm was used as a search optimization algorithm for constructing an FSM with discrete outputs intended for controlling a model of an unmanned aerial vehicle (UAV). The fitness function was evaluated using a flight simulator. The complete FSM generation cycle took as long as *several weeks* on two dual-core desktops. In [14], a genetic algorithm was also used, but the fitness function was evaluated based on training samples rather than using simulation. As a result, the FSM generation time was reduced to *several hours* on a desktop. A distinctive feature of [14] is the use of continuous outputs in addition to discrete ones.

In the present paper, we propose an FSM generation method that is an elaboration of the method proposed in [14]. As a search optimization algorithm, we use a modification of the ant colony optimization (ACO) algorithm (see [15, 16]) that was proposed in [17]. This enabled us to reduce the FSM generation time to about 15 min on a quad-core computer.

1. PROBLEM STATEMENT

The Mealy FSM is defined as a sextuple$(S, E, A, \delta, \lambda, s_0)$, where S is the finite set of states, E is the set of input events, A is the set of output actions, δ: $S \times E \to S$ is the transition function, λ: $S \times E \to A$ is the output function, and s_0 is the initial state.

An example of an FSM is shown in Fig. 1. The circles in the diagram correspond to the states of the FSM and the numbers within them indicate the state indexes. The initial state 1 is marked by the incoming arrow on the left. Each of the other arrows, which indicate transitions, is labeled by an event triggering the

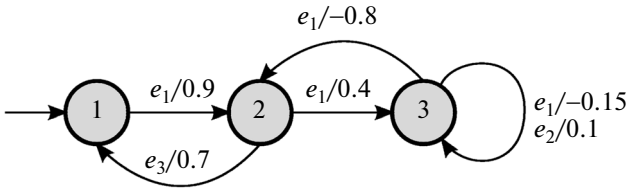

Fig. 1. Example of a control FSM.

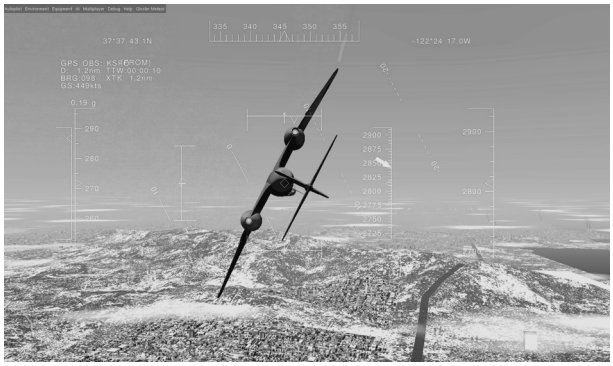

Fig. 2. *FlightGear* flight simulator (screenshot).

transition (to the left of the slash) and the output (a real number) produced when the transition is performed. The arrow on the right corresponds to two transitions simultaneously.

The problem that we solve in this paper is formulated as follows. A set of training samples is given. Based on these samples, generate an FSM with discrete and continuous outputs to control a complex plant. The training samples are prepared by a man and provide examples of the desired behavior of the plant.

In this paper, as in [14], the plant is a model of a UAV, and its desired behavior is the execution of an aerobatic maneuver. A flight simulator is used to execute the maneuvers. This simulator must be able to register the aircraft parameters (velocity, bank angle, etc.) and positions of the aircraft controllers (characterized by numbers) in the course of the flight. *FlightGear* (see [18]) is used as the flight simulator (see Fig. 2).

1.1. Plant Control

Here, we describe how the FSM interacts with the plant. An *input tuple* of the FSM is an ordered set of real numbers describing the state of the plant; the input tuple is fed to the input of the FSM. In the case of the UAV, the input tuple consists of flight parameters—altitude, velocity, heading, pitch, and bank angles, etc.

The plant has a set of *control devices* whose positions can be changed by the FSM. The positions of the control devices are specified by numbers. The control devices with a finite number of positions are called *discrete control devices*. An example is the starter, which can be on or off. The set of values of the lth discrete controller ($l = \overline{1,d}$) is denoted by V_l. The positions of other control devices can be specified by real numbers belonging to an interval of possible values; such devices are said to be continuous. For example, the position of the rudder varying from the leftmost to the rightmost position can be specified by numbers in the interval $[-1, 1]$. The lower and the upper bounds of the interval of possible values of the kth continuous control device ($k = \overline{1,c}$) are denoted by m_k and M_k, respectively.

An *output tuple* of an FSM is defined as a pair of two ordered sets of numbers—d integers and c real numbers, where d is the number of discrete control devices and c is the number of continuous control devices. The output tuple is a *snapshot* of the positions of all control devices at a certain point

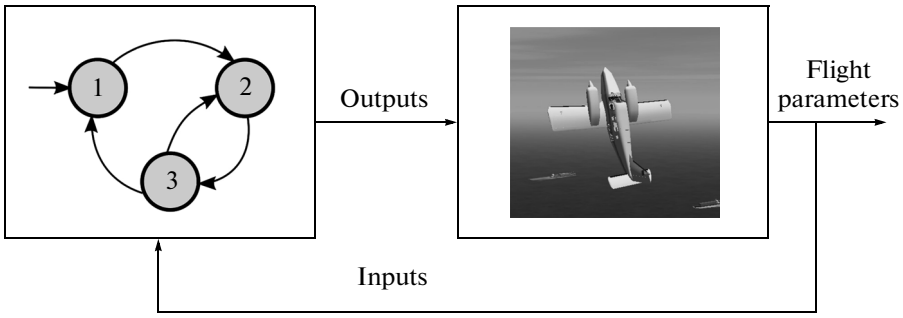

Fig. 3. Interaction between the FSM and the plant.

in time. The set of all possible output tuples forms the set of outputs of the FSM: $A = V_1 \times \ldots \times V_d \times [m_1, M_1] \times \ldots \times [m_c, M_c]$.

A *predicate* is a Boolean variable depending on the state of the plant or the state of the environment in which the plant resides. Examples of predicates are such assertions as *the vertical speed is positive* or *the pitch angle is greater than* 5°. We assume that all the information needed to evaluate predicates can be obtained from the input tuples. Let us fix a set of m predicates P_1, ..., P_m. The predicates for each set of training samples are chosen manually.

We will construct a synchronous FSM—the cycles of its operation are equally spaced in time (the time interval between the cycles is 0.1 s). The set of events for the FSM is formed by assertions P_i *is true* or P_i *is false* for each predicate P_i. For each event in each state of the FSM, we store the transition for which the final state and the output tuple (more precisely, the tuple of *changes* in the outputs) are specified.

Each cycle consists of m transitions—the ith transition (in the order they are executed) is triggered by the event corresponding to the value of the predicate P_i. Suppose that the transitions associated with the output tuples with continuous components represented by c-dimensional vectors z_1, ..., z_m were executed in a certain cycle. Then, the continuous part of the output tuple z' of the whole cycle is formed by the rule

$$z' = z + \sum_{i=1}^{m} z_i, \tag{1.1}$$

where z is the continuous part of the output tuple in the preceding cycle. From now on, it is convenient to assume that the continuous outputs are not restricted by the intervals $[m_1, M_1], \ldots, [m_c, M_c]$. If a component of z' is outside such an interval, it is assumed to be equal to the corresponding boundary of the interval. The discrete outputs in the cycle are set to the result of the last executed transition.

The diagram of the interaction between the FSM and the aircraft is shown in Fig. 3. The input tuples fed to the input of the FSM depend on the current state of the aircraft (on the flight parameters).

1.2. Training Samples

In this subsection, we formalize the concept of training sample. Denote the number of points in time registered in the ith training sample by L_i ($i = \overline{1, N}$, where N is the number of training samples). This number is said to be the *length* of the ith training sample. Each training sample consists of two parts. The first part is the set of input tuples I_i, which consists of numbers $I_{i,t,j}$, where $t = \overline{1, L_i}$ is the time and $j = \overline{1, p}$ is the index of the input in the tuple. The second part—the sequence of output tuples O_i—consists of numbers $D_{i,t,l}$ and $C_{i,t,k}$, where i and t determine the training sample and the time, $l = \overline{1, d}$ is the index of the discrete control device, and $k = \overline{1, c}$ is the index of the continuous control device. The differences between the adjacent points in time registered in the training sample are equal to the time interval between the FSM cycles. An example of a training sample is shown in Table 1.

Table 1. Example of a training sample ($p = 4$, $d = 1$, $c = 3$, $L_i = 235$)

Parameters	Description	$t = 1$	...	$t = 10$	...	$t = 20$	...	$t = 235$
$I_{i,t,1}$	Pitch angle, deg	3.078	...	3.544	...	4.112	...	2.412
$I_{i,t,2}$	Bank angle, deg	−0.076	...	0.351	...	3.413	...	1.759
$I_{i,t,3}$	Heading angle, deg	198.03	...	198.11	...	198.41	...	205.64
$I_{i,t,4}$	Velocity, knots	251.42	...	252.29	...	253.20	...	289.40
$D_{i,t,1}$	Starter	0	...	0	...	0	...	0
$C_{i,t,1}$	Position of the ailerons (from −1 to 1)	0.000	...	0.032	...	0.073	...	−0.003
$C_{i,t,2}$	Position of the rudder (from −1 to 1)	0.000	...	0.016	...	0.037	...	−0.001
$C_{i,t,3}$	Position of the elevation rudder (from −1 to 1)	−0.035	...	−0.039	...	−0.037	...	−0.011

2. FSM GENERATION METHOD

In the method proposed in this paper, FSMs are generated using a modification of the ant colony optimization algorithm [17]. However, only FSMs with discrete outputs were considered in that paper.

2.1. Fitness Function

The fitness function used in this paper is based on the similarity of the behavior demonstrated by the FSM when it is fed with input tuples from the training samples and the reference behavior registered in those samples. Let us fix an FSM. Denote by $\tilde{O}_i$ the sequence of output tuples produced by the FSM when the input tuples were taken from the ith training sample, and denote its discrete and continuous components by $\tilde{D}_{i,t,l}$ and $\tilde{C}_{i,t,k}$, respectively ($t = \overline{1, L_i}$, $l = \overline{1, d}$, $k = \overline{1, c}$). We have $\tilde{D}_{i,1,l} = D_{i,1,l}$ and $\tilde{C}_{i,1,k} = C_{i,1,k}$ (the initial outputs of the FSM are identical to the initial outputs registered in the training sample). At the end of the t th cycle, the FSM produces the outputs $\tilde{D}_{i,t+1,l}$ and $\tilde{C}_{i,t+1,k}$.

Consider the distance between the sequences O_i and $\tilde{O}_i$ as a penalty for the FSM:

$$\rho\left(O_i, \tilde{O}_i\right) = \sqrt{\frac{1}{L_i}\sum_{t=2}^{L_i}\frac{1}{d+c}\left(\sum_{l=1}^{d}\left[\tilde{D}_{i,t,l} \neq D_{i,t,l}\right] + \sum_{k=1}^{c}\left(\frac{\tilde{C}_{i,t,k} - C_{i,t,k}}{M_k - m_k}\right)^2\right)}.$$

Here, the square brackets (Iverson brackets) in the formula above evaluate to unity if the condition inside them is satisfied and to zero, otherwise.

Define the fitness function by combining the penalties $\rho\left(O_i, \tilde{O}_i\right)$ calculated for all the training samples:

$$f = 1 - \sqrt{\frac{1}{N}\sum_{i=1}^{N}\rho^2\left(O_i, \tilde{O}_i\right)}.$$

Thus, f takes into account the differences in the behavior of the FSM and the reference behavior on all the training samples. For the FSMs with the behavior close to that registered in the training samples, f is close to unity. A similar fitness function (differing from the one used in this paper by the type of normalization) was used in [14].

2.2. Individual in the Search Optimization Algorithm

The *framework* of an FSM is the FSM for which no output function is specified (no output tuples are assigned to the FSM transitions). The use of the framework as an individual for the search optimization algorithm makes the search space discrete.

We will assign outputs on the FSM framework so as to maximize the fitness function for the given framework using the algorithm proposed in [14]. This is possible due to the form of the fitness function

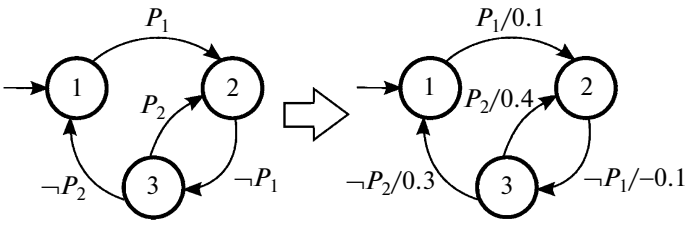

Fig. 4. Assigning outputs to the edges of the FSM framework.

which allows one to reduce the maximization problem to solving several systems of linear equations. The procedure of assigning outputs is schematically illustrated in Fig. 4. This figure illustrates the case when the plant has two predicates, one continuous control device and no discrete control devices. The output assignment algorithm is executed before every evaluation of the fitness function.

Let us describe the output assignment procedure. To maximize the fitness function on the given framework, the following optimization problem, which can be considered independently for each control device, should be solved:

$$1-\sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{1}{L_i}\sum_{t=2}^{L_i}\frac{1}{d+c}\left(\sum_{l=1}^{d}\left[\tilde{D}_{i,t,l}\neq D_{i,t,l}\right]+\sum_{k=1}^{c}\left(\frac{\tilde{C}_{i,t,k}-C_{i,t,k}}{M_k-m_k}\right)^2\right)}\to\max. \tag{2.1}$$

The unknowns in this problem are the discrete and continuous outputs at each transition of the FSM, which do not yet explicitly appear in (2.1) but on which the quantities $\tilde{D}_{i,t,l}$ and $\tilde{C}_{i,t,k}$ depend. It is clear that this problem can be simplified to form the problem

$$\sum_{i=1}^{N}\frac{1}{L_i}\sum_{t=2}^{L_i}\left(\sum_{l=1}^{d}\left[\tilde{D}_{i,t,l}\neq D_{i,t,l}\right]+\sum_{k=1}^{c}\left(\frac{\tilde{C}_{i,t,k}-C_{i,t,k}}{M_k-m_k}\right)^2\right)\to\min. \tag{2.2}$$

First, consider discrete control devices. Let l be the index of a discrete control device, and let $v_j\in V_l$ ($j=\overline{1,n}$) be the discrete output that must be assigned to the transition with the index j and n be the number of transitions in the FSM. By eliminating the constant terms in problem (2.2), we reduce it to the problem

$$\sum_{i=1}^{N}\frac{1}{L_i}\sum_{t=2}^{L_i}\left[\tilde{D}_{i,t,l}\neq D_{i,t,l}\right]\to\min_{v_1,\ldots,v_n}.$$

Let $w_{j,i,t}$ ($t=\overline{2,L_i}$) be equal to unity if transition j was performed at the end of the cycle $t-1$ while operating on the ith training sample and be zero otherwise (recall that the outputs $\tilde{D}_{i,t,l}$ for $t>1$ are determined by the last transition of the cycle $t-1$). We have

$$\left[\tilde{D}_{i,t,l}\neq D_{i,t,l}\right]=\sum_{j=1}^{n}\left[v_j\neq D_{i,t,l}\right]w_{j,i,t},$$

$$\sum_{j=1}^{n}\sum_{i=1}^{N}\frac{1}{L_i}\sum_{t=2}^{L_i}\left[v_j\neq D_{i,t,l}\right]w_{j,i,t}\to\min_{v_1,\ldots,v_n}.$$

The terms in the outer sum are independent of each other; therefore, the numbers v_j can be determined for each transition independently. The solution of the problem is reduced to finding $v_j\in V_l$ that minimizes the corresponding term. These values can be found in the amount of time

$$O\left(\sum_{i=1}^{N}L_i+n|V_l|\right).$$

Now, consider continuous control devices. Let k be the index of a continuous control device, and let $u_j \in \mathbb{R}$ ($j = \overline{1,n}$) be the continuous output that must be assigned to the transition with the index j. The optimization problem for the kth continuous output can be written as

$$g_k = \sum_{i=1}^{N} \frac{1}{L_i} \sum_{t=2}^{L_i} \left(\tilde{C}_{i,t,k} - C_{i,t,k}\right)^2 \to \min_{u_1,\ldots,u_n}. \tag{2.3}$$

Return to rule (1.1). According to this rule, each continuous output at a cycle changes by the sum of the outputs assigned to the transitions performed during this cycle. Let $\beta_{i,t,j}$ be the number of executions of transition j during the cycle $t-1$ while operating on the ith training sample, and let $\alpha_{i,t,j}$ be the number of times it was executed from the start of the FSM operation to the beginning of the cycle t; that is,

$$\alpha_{i,t,j} = \sum_{t'=2}^{t} \beta_{i,t',j}.$$

Applying the notation introduced above, we can rewrite (1.1) using the sum over all transitions of the FSM rather than over the transitions executed in the current cycle:

$$\tilde{C}_{i,t,k} = \tilde{C}_{i,t-1,k} + \sum_{j=1}^{n} \beta_{i,t,j} u_j.$$

Since $\tilde{C}_{i,1,k} = C_{i,1,k}$, this can be written in closed form as

$$\tilde{C}_{i,t,k} = C_{i,1,k} + \sum_{j=1}^{n} \alpha_{i,t,j} u_j.$$

Substitute this expression into (2.3) to obtain

$$g_k = g_k(u_1,\ldots,u_n) = \sum_{i=1}^{N} \frac{1}{L_i} \sum_{t=2}^{L_i} \left(C_{i,1,k} + \sum_{j=1}^{n} \alpha_{i,t,j} u_j - C_{i,t,k} \right)^2 \to \min_{u_1,\ldots,u_n}.$$

To find the minimum of the function g_k, we equate its derivatives with respect to $u_1,\ldots,u_n$ to zero:

$$\frac{\partial g_k}{\partial u_{j_0}} = 2 \sum_{i=1}^{N} \frac{1}{L_i} \sum_{t=2}^{L_i} \alpha_{i,t,j_0} \left(C_{i,1,k} + \sum_{j=1}^{n} \alpha_{i,t,j} u_j - C_{i,t,k} \right) = 0, \quad j_0 = \overline{1,n}.$$

These conditions written for different j_0 form the system of linear equations

$$\sum_{j=1}^{n} \left(\sum_{i=1}^{N} \frac{1}{L_i} \sum_{t=1}^{L_i} \alpha_{i,t,j_0} \alpha_{i,t,j} \right) u_j = \sum_{i=1}^{N} \frac{1}{L_i} \sum_{t=2}^{L_i} \alpha_{i,t,j_0} \left(C_{i,t,k} - C_{i,1,k} \right), \quad j_0 = \overline{1,n}. \tag{2.4}$$

We have shown that the optimal continuous outputs for a given continuous control device k can be found by solving system (2.4). Its coefficients and solution by the Gauss elimination method can be found in the amount of time

$$O\left(n^2 \sum_{i=1}^{N} L_i + n^3 \right).$$

For different control devices k, the left-hand side of this system is the same; therefore, the assignment of outputs can be done for all the control devices simultaneously without deteriorating the asymptotic estimate of the complexity. More detailed estimates of the time needed to execute the assignment procedures for discrete and continuous control devices can be found in [14].

2.3. Ant Colony Optimization Algorithm

As has already been mentioned, in this paper we generate FSMs using the modification of the ant colony optimization (ACO) algorithm proposed in [17]. The distinctive feature of this modification is that the solutions of the problem are represented by the vertices of a graph, while in the classical ACO algo-

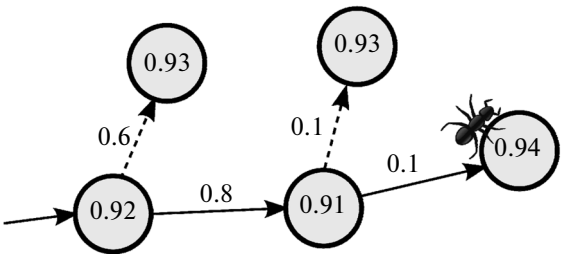

Fig. 5. Part of the graph *G*.

rithm they are represented by paths in a graph. In [17], this algorithm was used to construct FSMs with discrete outputs only based on training samples. However, the use of continuous outputs does not change the essence of the algorithm. Consider a directed graph *G* in which the vertices correspond to FSMs (individuals or elements of the search space) and the arcs correspond to *mutations* of the FSMs. In this problem, a mutation is a change in one transition of the FSM framework. The graph will be constructed step by step beginning with a single vertex representing a randomly generated FSM.

The algorithm executes a sequence of iteration steps. At the start of each iteration step, N_{ants} ants are placed in certain vertices of the graph; then, the ants travel through the graph in search of the best individual (vertex). Each ant memorizes the best individual it visited on its path. The ants are placed randomly in the vertices of the path made by the ant that found the best individual among those considered by all the ants in the course of all iterations of the algorithm.

The path of an ant is formed as follows (the ant is assumed to be in a vertex *v*).

With the probability p_{new}, N_{mut} new arcs leading from the vertex *v* to the vertices that differ from *v* in one mutation are added to the graph *G*. If the vertex at the endpoint of such an arc was not yet included in the graph, then it is added along with the corresponding FSM framework. The choice of N_{mut} vertices to be added to the graph among all the vertices that differ from *v* in one mutation and are not yet connected with *v* by an arc is made randomly.

Otherwise or in the case when N_{mut} new arcs emanating from the vertex *v* cannot be added, the ant chooses an adjacent vertex that is already included in the graph and goes to this vertex. The probability of choosing the next vertex is proportional to the amount of *pheromone* on the arc connecting *v* with this vertex (roulette wheel selection is used).

Pheromone is a substance that the ants deposit on the arcs when passing through them. When an arc of *G* is created, the amount of pheromone on it is set to a certain value τ_0 and is then changed in the course of the algorithm operation. The amount of pheromone on an arc is updated at the end of each iteration step when all N_{ants} ants stop. The ant stops moving if the last N_{stag} vertices visited by it did not improve the best individual found by this ant.

The amount of pheromone is updated as follows. For each arc *uv*, we specify two quantities—the current amount of pheromone τ_{uv} and the maximum amount of pheromone τ_{uv}^{best} that has ever been deposited on *uv*. The quantity τ_{uv}^{best} is updated only on the prefix of the path of the ant that ends at the best individual in the ant's path (rather than on all the arcs in this path). The new (updated) value of τ_{uv}^{best} monotonically increases with increasing value of the fitness function of the best individual in the path. Upon updating τ_{uv}^{best}, the current amount of pheromone on each arc in *G* is recalculated using the classical ACO formula

$$\tau'_{uv} = \rho\tau_{uv} + \tau_{uv}^{\text{best}},$$

where ρ is the pheromone evaporation rate ($0 < \rho < 1$). If the amount of pheromone becomes less than τ_0 as a result of evaporation, then it is increased to τ_0. In this work, we use the following parameters of the ACO algorithm: $N_{\text{ants}} = 4$, $N_{\text{stag}} = 40$, $N_{\text{mut}} = 35$, $p_{\text{new}} = 0.25$, $\rho = 0.35$, and $\tau_0 = 0.005$.

A small fragment of the graph *G* and an example of the last part of an ant path in it are shown in Fig. 5. The values of the fitness function of the FSMs are shown within the circles indicating the vertices; the amount of pheromone is shown on the arcs. The path of the ant is shown in solid arrows, and the other arcs of the graph are shown by dotted arrows.

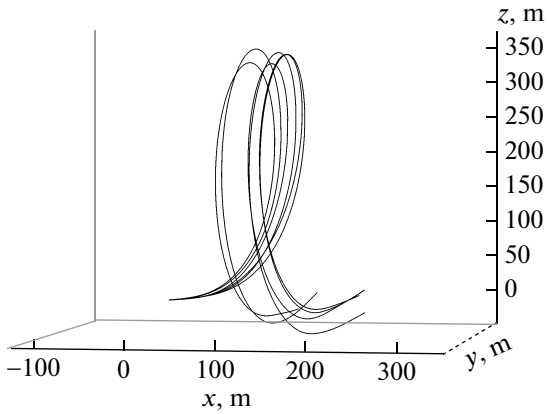

Fig. 6. Examples of trajectories from the set of training examples for the loop maneuver.

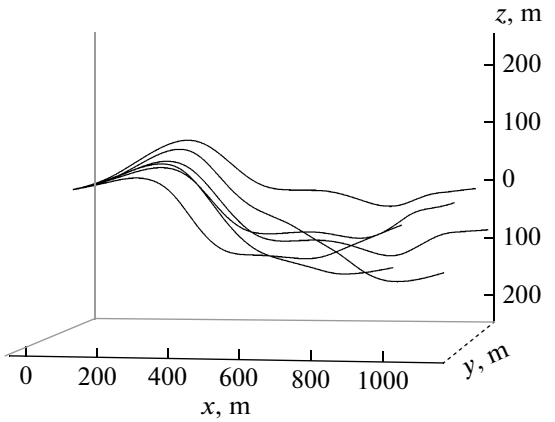

Fig. 7. Examples of trajectories from the set of training examples for the barrel roll maneuver.

3. EXPERIMENTAL STUDY

In this section, we describe the results of application of the proposed FSM generation method using two aerobatic maneuvers and compare the performance of the ACO algorithm with the performance of the genetic algorithm. The genetic algorithm was implemented as described in [14].

3.1. Aerobatic Maneuvers

Using the *FlightGear* flight simulator, two sets of training samples were recorded. The first set consisted of 23 samples describing the execution of the inside loop maneuver. The second set consisted of 28 samples describing the execution of the barrel roll maneuver (rotation by 360° about the principal axis of the aircraft). To execute the loop maneuver, a model of the civil aircraft *Piper PA34-200T* was used, while the barrel roll was executed on a model of the jet fighter *Gloster Meteor* because civil aircraft cannot execute such a maneuver.

Figures 6 and 7 show examples of aircraft trajectories corresponding to training samples in each set. The initial points and heading of all the trajectories are superposed for the convenience of perception. It is seen that the trajectories for each maneuver are similar but not identical. The reason is in random factors affecting the aircraft flight and lack of professionalism of the person who registered the training samples. However, small deviations of the trajectories do not present a problem for the proposed FSM generation method.

Table 2. Average number of fitness function evaluations in the execution of search optimization algorithms

Set of training samples	Ant colony optimization algorithm	Genetic algorithm
Loop	27000	44000
Barrel roll	22000	41000

Table 3. Number of runs in which a prescribed value of the fitness function was achieved for the loop training samples

Value of the fitness function	Ant colony optimization algorithm	Genetic algorithm
0.9890	3	0
0.9887	8	3
0.9884	15	15
0.9881	16	18
0.9878	17	19

Table 4. Number of runs in which a prescribed value of the fitness function was achieved for the barrel roll training samples

Value of the fitness function	Ant colony optimization algorithm	Genetic algorithm
0.9882	11	4
0.9880	22	14
0.9878	23	19
0.9876	24	23
0.9874	25	24

3.2. Results of the Experimental Study

To compare the performance of the ACO and genetic algorithms, we ran both algorithms 25 times on sets of training samples describing the execution of the loop and barrel roll maneuvers. Each run was terminated if the maximum found value of the fitness function did not increase in the course of 10000 evaluations. The number of states of the FSM was fixed and equal to four. A preliminary investigation showed that this number of states is sufficient for solving the problems.

Table 2 shows the number of evaluations of the fitness function performed by the algorithms on different sets of training samples averaged over all the runs. Tables 3 and 4 present the results of the runs. The left columns of Tables 3 and 4 contain the reference values of the fitness function and the right columns show the number of runs in which these values were achieved by the compared algorithms. Recall that the closer the value of the fitness function to unity, the better the behavior of the FSM matches the training samples. It is seen from the tables that high values of the fitness function were more often achieved by the ACO algorithm than by the genetic algorithm. This fact and the data presented in Table 2 suggest that the ACO algorithm outperforms the genetic algorithm in terms of performance on the examined aerobatic maneuvers. Furthermore, for the loop maneuver, the value 0.9890 of the fitness function was achieved only by the ACO algorithm. This indicates that the quality of the FSMs produced by this algorithm is higher.

Computations were performed on a 2.66 GHz *Intel Core 2 Quad Q9400* computer. In both algorithms, the fitness function was evaluated for groups consisting of several tens or hundreds of individuals, which allowed us to evaluate the fitness function concurrently for different individuals.

The average run time of the ACO algorithm is about 5 min. To obtain a higher value of the fitness function, it is reasonable to run the algorithm two or three times. Therefore, the time needed for the ACO algorithm to produce an FSM is about 15 min. Note that it takes the genetic algorithm about 30 min to produce an FSM; this is better than in [14] even if we take into account the differences in the computers used in this study and in [14] (one core of *Intel Core 2 Duo T7250*, 2 GHz).

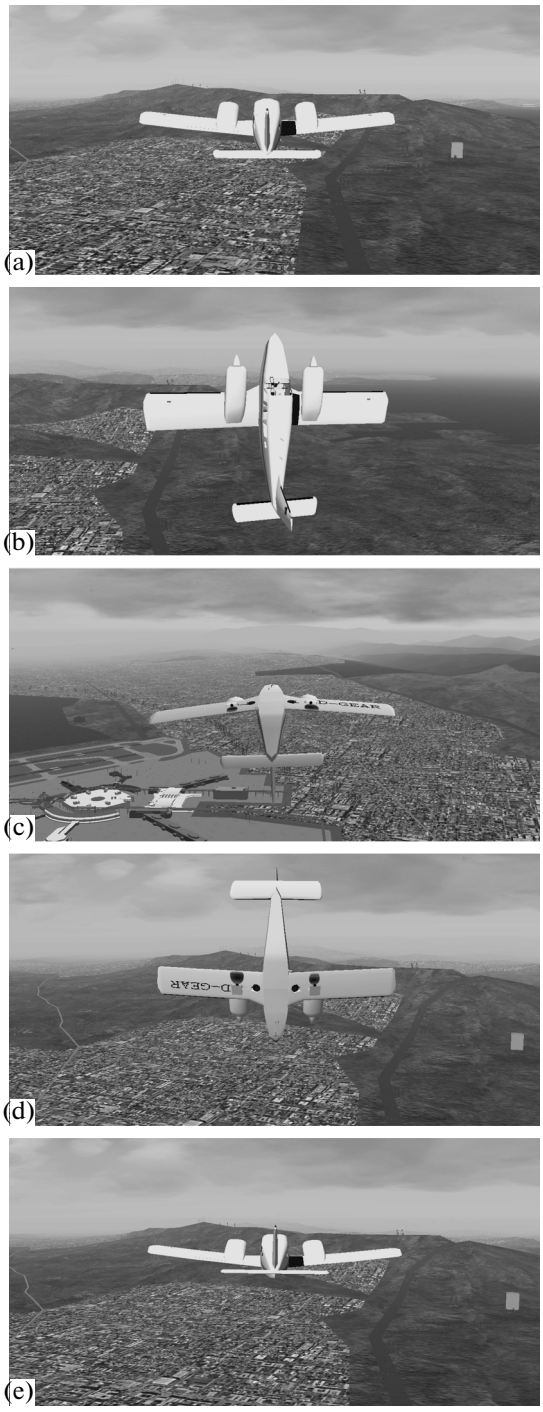

Fig. 8. Screenshots demonstrating various stages of executing the loop maneuver controlled by one of the FSMs in *FlightGear*.

3.3. Analysis of Results

The FSMs produced by the ACO algorithm were tested on the *FlightGear* flight simulator under the conditions similar to those under which the training samples were recorded. Each of the FSMs had to execute the aerobatic maneuver five times. More than 90% of FSMs executed the maneuvers successfully. Typical noncritical errors of the FSMs were in the final stage of the maneuvers—after the execution of the maneuver, the attitude of the aircraft must be leveled and the aircraft must fly evenly. We see the following ways of resolving this problem:

the final stages of the training samples could be recorded by a human pilot more accurately;

the set of predicates could be better;

the stabilization of the aircraft could be performed by a special FSM that should get control automatically.

Figures 8a–8e show screenshots of the *FlightGear* screen representing various stages of the execution of the loop maneuver under the control of an FSM produced by the ACO algorithm. The FSM has four states. It turned out to be difficult to visualize the FSM itself because of the large number of transitions (56 transitions).

CONCLUSIONS

In this paper, an improved method for inducing FSMs was proposed. Similarly to its prototype, this method makes it possible to generate FSMs with discrete and continuous outputs based on training samples. The improvement in performance and quality of the method for the examined training samples was achieved due to the application of a modification of the ant colony optimization algorithm for search optimization instead of the genetic algorithm.

REFERENCES

1. N. I. Polikarpova and A. A. Shalyto, *Automata-Based Programming* (Piter, St. Petersburg, 1991) [in Russian], http://is.ifmo.ru/books/_book.pdf
2. V. O. Kleban and A. A. Shalyto, "Development of control system for a small-size helicopter," Nauchno-tekhnicheskii Vestn. St. Petersburg Gos. Univ. Inform. Tekhnologii, Mekhaniki i Optiki, No. 2, 12–16 (2011). http://is.ifmo.ru/works/2011/Vestnik/72-2/02-Kleban-Shalyto.pdf
3. F. N. Tsarev and A. A. Shalyto, "The use of genetic programming for generating a finite state machine in the smart ant problem," in *Proc. of the 4th Int. Conf. on Integrated Models and Soft Computations in Artificial Intelligence* (Fizmatlit, Moscow, 2007), pp. 590–597. http://is.ifmo.ru/genalg/_ant_ga.pdf
4. F. N. Tsarev, "Combined use of genetic programming, finite state machines, and artificial neural networks for designing a control system for an unmanned aerial vehicle," Nauchno-tekhnicheskii Vestn. St. Petersburg Gos. Univ. Inform. Tekhnologii, Mekhaniki i Optiki, No. 2, 12–16 (2011). http://is.ifmo.ru/works/2008/Vestnik/53/03-genetic-neuro-automata-flying-plates.pdf
5. L. A. Gladkov, V. V. Kureichik, and V. M. Kureichik, *Genetic Algorithms* (Fizmatlit, Moscow, 2006) [in Russian].
6. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach* (Prentice Hall, Upper Saddle River, N.J., 2003; Vil'yams, Moscow, 2003).
7. J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, (1992).
8. V. M. Kureichik, "Genetic Algorithms: State of the Art, Problems, and Perspectives," J. Comput. Syst. Sci. Int. **38**, 137–152 (1999).
9. V. M. Kureichik and S. I. Rodzin, "Evolutionary Algorithms: Genetic Programming," J. Comput. Syst. Sci. Int. **41**, 123–132 (2002).
10. M. Heule and S. Verwer, "Exact DFA identification using SAT solvers," in *Grammatical Inference: Theoretical Results and Applications, 10th Int. Colloquium (ICCGI 2010)*, Lect. Notes Comput. Sci. **6339**, pp. 66–79 (2012).
11. V. Ulyantsev and F. Tsarev, "Extended finite-state machine induction using SAT-solver," in *Proc. of the 14th IFAC Symp. on Information Control Problems in Manufacturing (INCOM'12), 2012*, pp. 512–517.
12. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms* (MIT Press, Cambridge, Mass., 1990; Vil'yams, Moscow, 1999).
13. N. I. Polikarpova, V. N. Tochilin, and A. A. Shalyto, "Method of reduced tables for generation of automata with a large number of input variables based on genetic programming," J. Comput. Syst. Sci. Int. **49**, 265–282 (2010).
14. A. V. Aleksandrov, S. V. Kazakov, A. A. Sergushichev, F. N. Tsarev, and A. A. Shalyto, "The use of evolutionary programming based on training examples for the generation of finite state machines for controlling objects with complex behavior," J. Comput. Syst. Sci. Int. **52**, 410–425 (2013).
15. M. Dorigo, "Optimization, learning and natural algorithms," *PhD Thesis* (Dipartimento di Elettronica, Politechnico di Milano, Milano, 1992).
16. M. Dorigo and T. Stützle, *Ant Colony Optimization* (MIT Press, Cambridge, Mass., 2004).
17. D. Chivilikhin and V. Ulyantsev, "Learning finite-state machines with ant colony optimization," Lect. Notes Comput. Sci. **7461**, 268–275 (2012).
18. FlightGear. http://www.flightgear.org/. Accessed February 14, 2013.

Translated by A. Klimontovich