ISSN 1064-2307, Journal of Computer and Systems Sciences International, 2010, Vol. 49, No. 2, pp. 265–282. © Pleiades Publishing, Ltd., 2010.
Original Russian Text © N.I. Polikarpova, V.N. Tochilin, A.A. Shalyto, 2010, published in Izvestiya Akademii Nauk. Teoriya i Sistemy Upravleniya, 2010, No. 2, pp. 100–117.

ARTIFICIAL INTELLIGENCE

Method of Reduced Tables for Generation of Automata with a Large Number of Input Variables Based on Genetic Programming

N. I. Polikarpova, V. N. Tochilin, and A. A. Shalyto

St. Petersburg State University of Information Technologies, Mechanics and Optics, ul. Sablinskaya 14, St. Petersburg, 197101 Russia

Received August 12, 2008; in final form, September 8, 2009

Abstract—Known methods of automatic generation of finite automata based on genetic programming are inefficient in the case of a large number of input variables of the automaton. A method free from this disadvantage is proposed. The preference of this method for a large number of input variables is theoretically substantiated and experimentally proved. The method was used for automation of development of an aircraft control system on a high level of abstraction.

DOI: 10.1134/S1064230710020127

INTRODUCTION

A number of methods for construction of finite automata using genetic algorithms [1, 2], genetic programming [3, 4], and other evolutionary approaches have been considered in literature. Most papers in this field are devoted to the generation of *parser* automata describing the grammar of a language. The task of such an automaton is to verify whether a given string belongs to the language. The recognizer does not perform output actions—the result is determined by the state of the automaton after processing the input sequence. The example of the parser is a syntactic analyzer establishing the program code fits the grammar of the programming language. Papers [5–12] should be mentioned as the most important ones in this field. A more complex form of the finite automaton, the *transducer*, maps the set of input strings into the set of output strings, possibly over another alphabet. An example of the transducer is a compiler. The evolutionary construction of transducers is considered in [13–15].

For both classes of automata considered above, the input actions are symbols of a given (most often small) alphabet. In other words, the comparison of the current input symbol with one of a small set of given symbols is used as the condition of transition of the automaton from one state to another.

In the field of *genetic programming*, the problem of construction of finite automata is not formulated explicitly. However, in this field calculation models in the form of graphs which can be interpreted as transition graphs of finite automata are conventionally optimized. A large number of papers, for example [16–20], have been devoted to construction of programs in the form of graphs. The application of automata in games was described in [21–23].

A large number of papers have been devoted to the problems of construction of *control automata* describing the logics of complex behavior of an entity or a system. For example, in [24–26] the automatic construction of components of software for logical controllers in the form of automata was considered, and in [27] the efficiency of automaton models was estimated, as applied to different problems.

Practically in all above studies the automaton at each time instant processes just one input variable. The exceptions are only [22] (four parallel ternary inputs) and [27] (comparison of values of two registers is admitted as the transition condition). Theoretically, any number of parallel inputs can be reduced to one input for which the combinations of signals of initial parallel inputs serve as the actions. However, the size of the alphabet of thus obtained input exponentially increases with increasing number of initial parallel inputs. In these publications parallel inputs do not result in an admissibly large alphabet, but for real control systems this problem is extremely topical. In the papers mentioned above the automaton at each step can generate not more than one output variable. This results in the exponential growth of the output alphabet, since in real problems the automaton often has to implement arbitrary combinations of elementary actions on each step. Note that [27] allows actions with arguments and concurrently executing automata, responsible for different actions, which considerably weakens the problem.

In this paper the problem of automatic construction of control automata based on genetic programming is studied. Boolean functions over an arbitrary number of input variables are used as transition conditions in control automata. Among the sources mentioned above, the best results in the field of construction of control automata were obtained in [25, 26]. However, these publications are not free from the

above disadvantages connected with the dimensions of input and output signals.

In this paper, the *method of reduced tables* is proposed for construction of control automata. The greatest attention in development of this method was paid to the possibility of construction of automata with an arbitrary number of parallel inputs and outputs. Unlike existing genetic programming methods in which the program is described at a low level of abstraction and is subject to optimization as a whole, in the developed approach the program is represented in the form of the *automated control object* [28]: the logics of complex behavior is expressed by the automaton or the system of automata at a high level of abstraction and is optimized, while the control object is implemented manually (in a hardware or software way), and does not assume optimization. The control object can be arbitrary (generally speaking, of any complexity). Thus, the proposed method solves the problem of application of complex data structures in the framework of the genetic programming formulated by the founder of genetic programming J. Koza in [29].

1. STATEMENT OF THE PROBLEM

Let us formulate the *problem of construction of the control automaton*. We determine the following control object: $O = \langle V, v_0, X, Z \rangle$, where V is the set of *computational states*; v_0 is the initial computational state; $X = \{x_i : V \to \{0,1\}\}_{i=1}^{n}$ is the set of predicates; and $Z = \{z_i : V \to V\}_{i=1}^{m}$ is the set of signals. The criterion function (*fitness function*) on the set of numerical states $\varphi : V \to \mathbf{R}^+$ and the natural number k are also known.

The object O can be controlled by the automaton of the form $A = \langle S, s_0, \Delta \rangle$, where S is the finite number of *control states*; s_0 is the *initial state*, $\Delta : S \times \{0,1\}^n \to S \times Z^*$ is the *control function* matching the control state and the *input signal* the new state and the *output signal*. The control function can be decomposed into two components: the output function $\zeta : S \times \{0,1\}^n \to Z^*$ and the transition function $\delta : S \times \{0,1\}^n \to S$. Separate components of the input signal corresponding to predicates of the control object are called *input variables*. Separate components of the output signal corresponding to signals of the control object are called *output variables*.

Let before the beginning of operation the control object be situated in the computational state v_0, and let the automaton be in the control state s_0. The following sequence of operations makes the *step of operation* of the automated object:

(1) the control object calls all predicates from the set X and forms the vector of *input signal* $in \in \{0,1\}^n$ from their values.

(2) the automaton calculates the value of the vector of *output signal* $out = \zeta(s, in)$, where s is the current state of the automaton, and passes into the new control state $s_{new} = \delta(s, in)$;

(3) the control object implements the actions $z \in out$, changing the current computational state [28].

The problem of design of the control automaton consists in finding the automaton of the given form, such that for k steps the object O, controlled by this automaton, passes to the numerical state with the maximal fitness ($\varphi(v) \to \max$).

In relation with the application of genetic programming, the following subproblems occur for this problem: the choice of representation of the finite automaton in the form of the set of chromosomes; adaptation of genetic operators (mutation and crossover) for this representation; and adjustment of the parameters of genetic optimization. For solution of the first subproblem, it is convenient to interpret the automaton as the set of control states each of which is determined by the projection of the control function $\Delta_s : \{0,1\}^n \to S \times Z^*$, $s \in S$. Thus, the problem is reduced to the description of each separate control state in the form of a chromosome.

2. STATE REPRESENTATION IN THE FORM OF A CHROMOSOME

The natural way of representing the control function in a state is the *table of inputs and outputs* in which each possible combination of the values of input variables corresponds to the set of values of the output variables and the new state. The main problem arising in this case is the exponential growth of the chromosome size with increasing number of predicates of the control object, since the number of rows in the table is 2^n, where n is the number of predicates.

Experience has shown that in real problems the number of transitions in manually formed control automata does not increase exponentially with increasing number of predicates of the control object. The reason is probably that in most problems predicates have a "local nature" with respect to control states. In each state, just a certain small subset of predicates is *significant*, the other do not affect the value of the control function. This property makes it possible to considerably reduce the size of state description. Moreover, the application of this property in the course of optimization provides the result which resembles a manually constructed automaton, and consequently, is clearer to a person.

The locality property of predicates can be used for reducing the size of description of the control state using different methods. We chose one of the

x_0	x_1	x_2	x_3	x_4	x_5
0	1	0	1	0	0

x_1	x_3	s	z_0	z_1	z_2
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	2	1	1	1

Fig. 1. State chromosome: reduced table ($n = 6$, $r = 2$).

approaches for which the number of significant state predicates is limited by a constant r. In this case, the automaton state is represented in the form of a *reduced table* of transitions and outputs in which each combination of the values of *significant* input variables corresponds to the set of values of output variables and the new state. This representation also contains the bit vector describing the set of significant predicates (Fig. 1).

The number of rows of the reduced table is 2^r. In manually constructed automata, the number of significant state predicates usually varies from 1 to 5. This provides the assumption that good optimization results can be achieved even for small values of the constant r. Thus, the memory volume necessary for the optimization process for the method of reduced tables can be estimated as $O(g|S|(|Z| + |X|))$, where g is the number of specimens in the generation. Note that unlike the required memory volume the performance of the algorithm of genetic optimization is difficult to estimate theoretically, since it is probabilistic. Therefore, the performance was estimated experimentally (Section 4).

3. GENETIC OPERATORS

Let us describe genetic operators acting on chromosomes of states represented in the form of reduced tables.

Algorithm 1. Mutation of reduced tables. In the case of the state mutation the value in each row of the reduced table and the set of significant predicates can change with some probability. In this case, each of the significant predicates is replaced with a given probability by another one which does not belong to the set (Fig. 2). Thus, the number of significant state predicates remains constant. The description of the mutation algorithm is given in the listing in Fig. 3.

Algorithm 2. Crossover of reduced tables. The main steps of the crossover algorithm are shown in the listing in Fig. 4. Since parent chromosomes represented by reduced tables can have different sets of significant predicates, first it is necessary to choose which of these predicates will be present in child chromosomes. The function *ChoosePreds* making this choice is shown in the listing in Fig. 5. As a result of operation of the function *ChoosePreds* the predicates which are significant for both parents are inherited by both children, and each of the predicates present in just one parent specimen goes to any of the children with equal probability. The example of operation of the function for parent chromosomes shown in Fig. 6 is illustrated in Fig. 7. After choosing significant predicates tables for both children are formed. The filling algorithm is shown in the listing in Fig. 8. The illustration of the example of filling the first row of the table of one of the children is shown in Fig. 9. In this implementation of the crossover operator, the values of each row of the child table are influenced by the values of several rows of the parent tables. In this case, the particular quantity placed in the cell of the child table is determined by "voting" of all cells of parent tables which influence this cell.

In the above variant of the algorithm, all automaton states have the equal number of significant predicates (r is the constant for the whole optimization process). However, the proposed crossover algorithm can easily be extended to the case of the different number of significant predicates for the pair of parents.

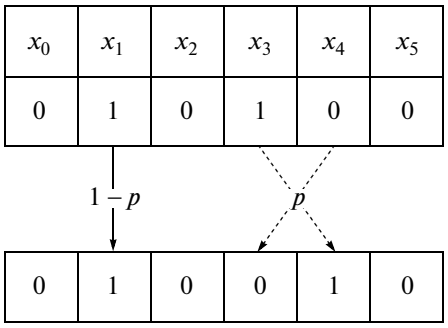


Fig. 2. Example of mutation of the set of significant predicates.

```
State Mutate(State state)
{
     State mutant = state;
     if (with the probability p) {
          int from, to;
          randomly choose from and to in such a way that
               (mutant.predicates[from] == 1) && (mutant.predicates[to] == 0)
          mutant.predicates[from] = 0;
          mutant.predicates[to] = 1;
     }
     for (for all i: rows of table) {
          if (with the probability p1) {
               mutant.table[i].targetState = random from 0 to nStates - 1;
          }
          if (with the probability p2) {
               int nActsPresent = number of units in mutant[i].table.output;
               if ((nActsPresent == 0) || (nActsPresent == nActions)) {
                    Index j = random number from 0 to nActions - 1;
                    mutant[i].table.output[j] = !mutant[i].table.output[j];
               } else {
                    for (for all j: action numbers) {
                         mutant[i].table.output[j] = 1 with the probability
                         nActsPresent/nActions and 0 otherwise;
                    }
               }
          }
     }
     return mutant;
}
```

Fig. 3. Mutation of reduced tables (listing).

4. EXPERIMENTAL ESTIMATE OF THE METHOD OF REDUCED TABLES

For estimating the characteristics of the method of reduced tables a number of experiments were performed in which this method was compared with the representation of control automata in the form of complete tables of transitions and outputs [30]. The comparison was performed according to the following criteria: occupied memory volume, time spent for processing each generation, growth rate of the fitness function (of the generation number and time).

It was already mentioned above that the method of reduced tables solves the problem of exponential growth of the size of automaton description with increasing number of input variables (predicates of the control object). This property of the method was proved in the course of experimental verification. During the experiment the automata descriptions were stored in the computer memory; therefore, the occupied memory volume in this case is proportional to the size of the automaton description. The results of experiment are shown in Fig. 10a. It follows from the plot that the occupied memory volume in the case of application of the complete tables exponentially grows, while for the method of reduced tables this volume practically does not change with increasing number of predicates. In reality it linearly depends on the number of predicates, however, the proportionality coefficient is so small that it is not observed in experiments. The dependence of time required for processing each generation on the number of predicates has the same character (Fig. 10b).

Now let us estimate the growth rate of the fitness function. Figure 10c shows the dependences of the value of the estimator on the generation number for the methods of complete and reduced tables (the measurements were performed for a small number of predicates). It follows from this plot that the optimization using the method of complete tables requires the calculation of a smaller number of generations. This means that in the case of a small number of predicates

```
pair<State, State> Cross(State statel, State state2)
{
      State child1 = statel;
      State child2 = childl;

      ChoosePreds(statel.predicates, state2.predicates,
            childl.predicates, child2.predicates);

      int crossPoint = random number from 0 to tableSize;
      FillChildTable(statel, state2, childl, crossPoint);
      FillChildTable(statel, state2, child2, crossPoint);
      return make_pair(childl, child2);
}
```

Fig. 4. Crossover of reduced tables (listing).

```
void ChoosePreds(Predicates p1, Predicates p2,
                 Predicates chl, Predicates ch2)
{
     for (for all i: predicate numbers) {
           if (p1[i] && p2[i]) {         // Predicate from both parents
                 chl[i] = ch2[i] = true; // goes to both children
                 //remembering that the sets of child predicates
                 // have one space less;
           }
     }
     for (for all i: predicate numbers) {
           if (pl[i] != p2[i]) {
                 Predicates* pCh;
                 if (both children have space) {
                       pCh = any child with equal probability;
                 } else {
                       pCh = the child who still has space;
                 }
                 (*pCh)[i] = true;
                 remembering who has less space;
           }
     }
}
```

Fig. 5. Choice of significant predicates of children upon crossover reduced tables (listing).

the method of complete tables possesses higher performance. However, with increasing number of predicates the time of processing one generation using this method sharply increases. Moreover, due to the exponential growth of the required memory volume the application of the method of complete tables, beginning from some number of predicates, becomes not only inefficient, but practically impossible. In experiments we could not obtain the automaton with more than 14 input variables using the method of complete tables. Note also that it is practically impossible to depict and understand automata constructed using the method of complete tables, since they contain a large number of excessive transitions, and conditions for these transitions are cumbersome. On the contrary, automata based on the method of reduced tables can be relatively easily understood.

In order to adequately estimate the performance of both methods, it is necessary to establish the dependence of the value of the estimator achieved using each of them during a particular time on the number of predicates. This dependence for the time interval equal to 5 min is shown in Fig. 10d. As expected, these dependences demonstrate that in the case of a small number of predicates the method of complete tables has higher performance; however, with increasing number of predicates its performance sharply drops. At the same time, the performance of the method of reduced tables insignificantly decreases with increasing number of predicates.

x_0	x_1	x_2	x_3	x_4	x_5
0	1	0	1	0	0

x_0	x_1	x_2	x_3	x_4	x_5
0	1	0	0	0	0

x_1	x_3	s	z_0	z_1	z_2
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	2	1	1	1

x_1	x_3	s	z_0	z_1	z_2
0	0	1	1	1	0
0	1	2	0	0	1
1	0	2	0	1	0
1	1	0	0	1	0

Fig. 6. Parent chromosomes represented as reduced tables.

The following conclusions can be made from the investigation of the characteristics of the methods of complete and reduced tables:

(i) in the case of a small number of predicates the optimization using the method of complete tables requires less time; however, thus constructed automata are incomprehensible;

(ii) beginning with a certain number of predicates, the application of the method of complete tables is practically impossible, while the method of reduced tables yields rather good results with respect to the required time and memory.

A tool based on the method described above was developed by us for automata generation [30, 31].

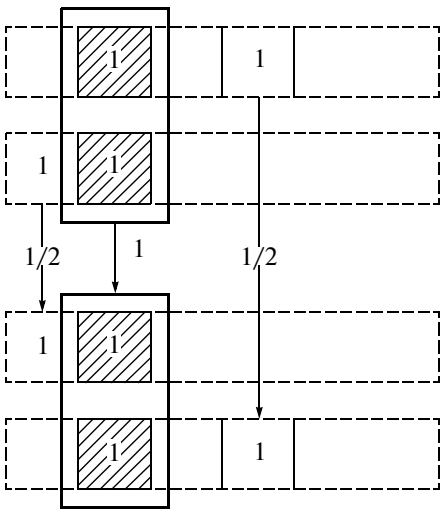


Fig. 7. Example of choosing significant predicates of children.

5. APPLICATION OF THE METHOD OF REDUCED TABLES FOR AUTOMATIC CONTROL OF AN AIRCRAFT

The proposed method was applied to designing the automatic control system for an aircraft model on a high level of abstraction. It is known that at present autopilot systems are widely used in aircraft control; however, they do not provide a completely automatic flight. In particular, the switching between autopilot regimes and the adjustment of navigation devices is performed manually. In this study the following task was formulated: to construct an aircraft controlling finite automaton which completely automates the flight. In this case, the automaton can use the standard autopilot as the control object. The developed system should not produce nonstandard requirements to the ground equipment.

5.1. Method of Generation of Aircraft Controlling Automaton

For solving the problem formulated above using genetic programming based on the method of reduced tables the following steps were made:

(i) choice of a spacecraft simulator;

(ii) implementation of control object interface;

(iii) determination of fitness function;

(iv) construction of control automaton;

(v) analysis of results of experiment.

5.2. Choice of a Spacecraft Simulator

In the considered problem, the control object is an aircraft which is situated in some medium, including air (possibly, moving), runways, landscape, flight control service, radio beacons, and so on. It is necessary to emulate aerodynamics, mechanics, and operation of

```
void FillChildTable(State s1, State s2, States child, int crossPoint) {
      for (for all i: rows of table child) {
            vector<int> lines1 = choose rows of table s1 in which predicates
                  significant for child have the same values as in row i,
                  and if the predicate is significant for both parents and i >=
                  crossPoint, its value is not taken into account;
            vector<int> lines2 = choose rows of table s2 in which predicates
                  significant for child have the same values as in row i,
                  and if the predicate is significant for both parents and i <
                  crossPoint, its value is not taken into account;
            vector<Probability> p1(nStates);
            vector<Probability> p2(nStates);
            for (for all j from lines1) {
                  p1[target state for s1 in row j] += 1.0;
            }
            for (for all j from lines2) {
                  p2[target state for s2 in row j] += 1.0;
            }
            Divide values of p1 by the number of rows from lines1;
            Divide values of p2 by the number of rows from lines2;
            vector<Probability> p = p1 + p2;
            child[i].targetState = random with probability distribution p;
            for (for all k: action numbers) {
                  Probability q1, q2;
                  for (for all j from lines1) {
                        q1 += s1[j].output[k];
                  }
                  for (for all j from lines2) {
                        q2 += s2[j].output[k];
                  }
                  Divide q1 by the number of rows from lines1;
                  Divide q2 by the number of rows from lines2;
                  child[i].output[k] = 1 with the probability (q1 + q2)/2
                  and 0 otherwise;
            }
      }
}
```

Fig. 8. Filling tables of children for crossover of reduced tables (listing).

the aircraft equipment. Since it is rather cumbersome to develop the necessary emulator, it is natural to use a ready emulator. For this purpose the aviation simulator *X-Plane* [32] was chosen; the specific features of this simulator are the precision of physical simulation and documented interface of interaction with other programs (API), which made it possible to use this simulator in the experiment.

Table 1. Input actions of automaton

Identifier	Value description
x_1	Aircraft is moving
x_2	Aircraft velocity is sufficient for flying with extended flaps
x_3	Aircraft velocity is sufficient for flying with retracted flaps
x_4	Aircraft is flying
x_5	Aircraft is near the ground
x_6	Flight altitude corresponds to the recommended altitude level
x_7	Aircraft is near the reference point of GPS receiver

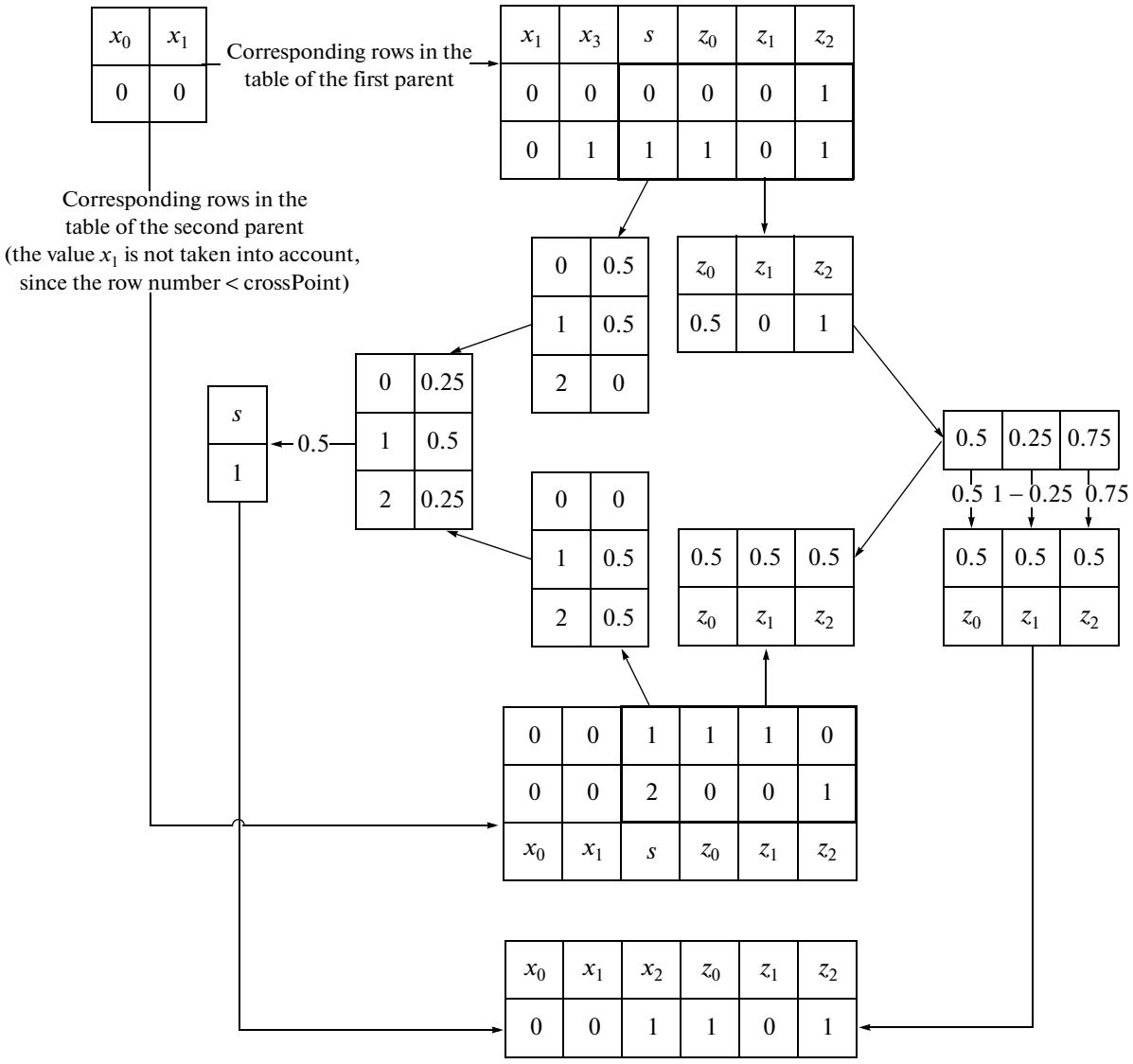


Fig. 9. Example of filling the row of the child table upon crossover of reduced tables.

5.3. Choice of Control Object Interface

The simulator *X-Plane* provides getting from the aircraft and transmitting to the aircraft of a large number of data. In this experiment, it was impossible to use all these data in the control automaton, since in this case the optimization process would be too long. It was already mentioned above that the realistic character of the proposed method of generation of control automata is provided by the high level of abstraction of the logics. In this case, predicates and actions of the control object are manually chosen and implemented.

In the studied problem 7 input (Table 1) and 24 output variables (Table 2) were used; we assumed that this amount was sufficient for the aircraft control on a high level of abstraction. For each of the chosen variables the subprogram corresponding to the predicate or action of the control object was developed. These subprograms interact with the spacecraft simulator, reading the device indications and acting on the control organs of the aircraft. The composition of devices and control organs is shown in Fig. 11.

5.4. Fitness Function

Let us reformulate the problem in the form of the fitness function. The autopilot should guide the aircraft along the route and not crash it. Therefore, two important factors can be separated: deviation from the route and the aircraft safety. It is also important that after passing along the route the aircraft should stop (or decelerate to a safe velocity) on the runway. The combined deviation from the route both with respect to position and velocity is calculated in the form of the time integrals of the absolute values of corresponding deviations,

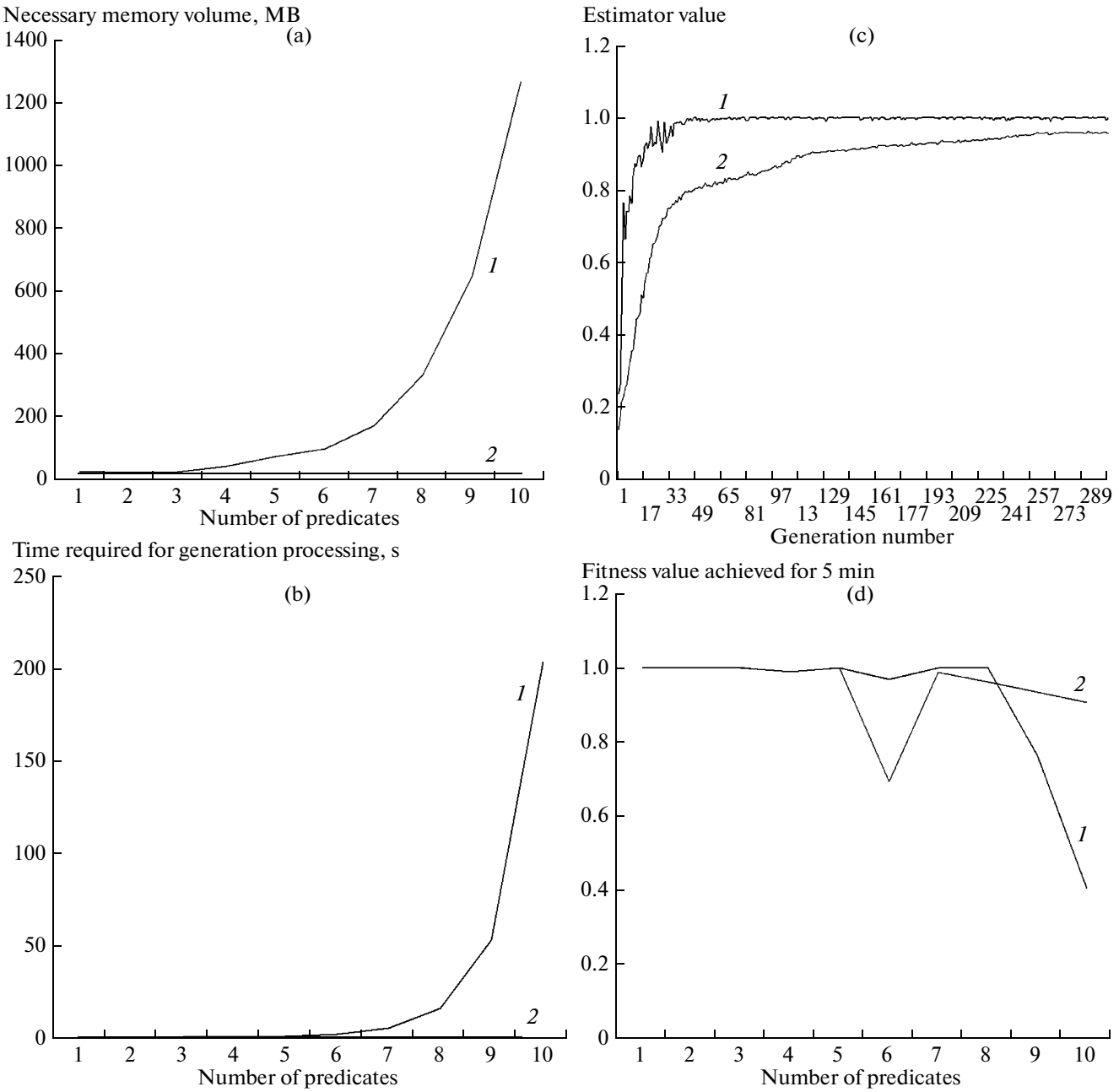


Fig. 10. The dependence of the memory amount (a), the processing time of generation (b). The value of estimation function for the given duration of the method (d) on the number of predicates and the value of estimation function on the number of generation (c); *1*—full tables, *2*—reduced tables.

$$P = \int_{t_0}^{t_1} |p(t)|\,dt,$$

where P is the combined deviation from the route with respect to position, t_0, is the time of beginning of the emulation, t_1 is the time of the end of emulation, and $p(t)$ is the position deviation at the time instant t;

$$V = \int_{t_0}^{t_1} |v(t)|\,dt,$$

where V is the combined deviation from the optimal velocity and $v(t)$ is the deviation from the optimal velocity at the time instant t. Let us denote by C_α, P_α, and V_α the weighting coefficients of a crash, position, and velocity deviations. The coefficients determine the relative importance of the corresponding defects of the autopilot behavior. The following values were taken for these coefficients: $C_\alpha = 9$, $P_\alpha = 0.01$ 1/m, and $V_\alpha = 0.01$ s/m. As a result, we chose the fitness function of the form

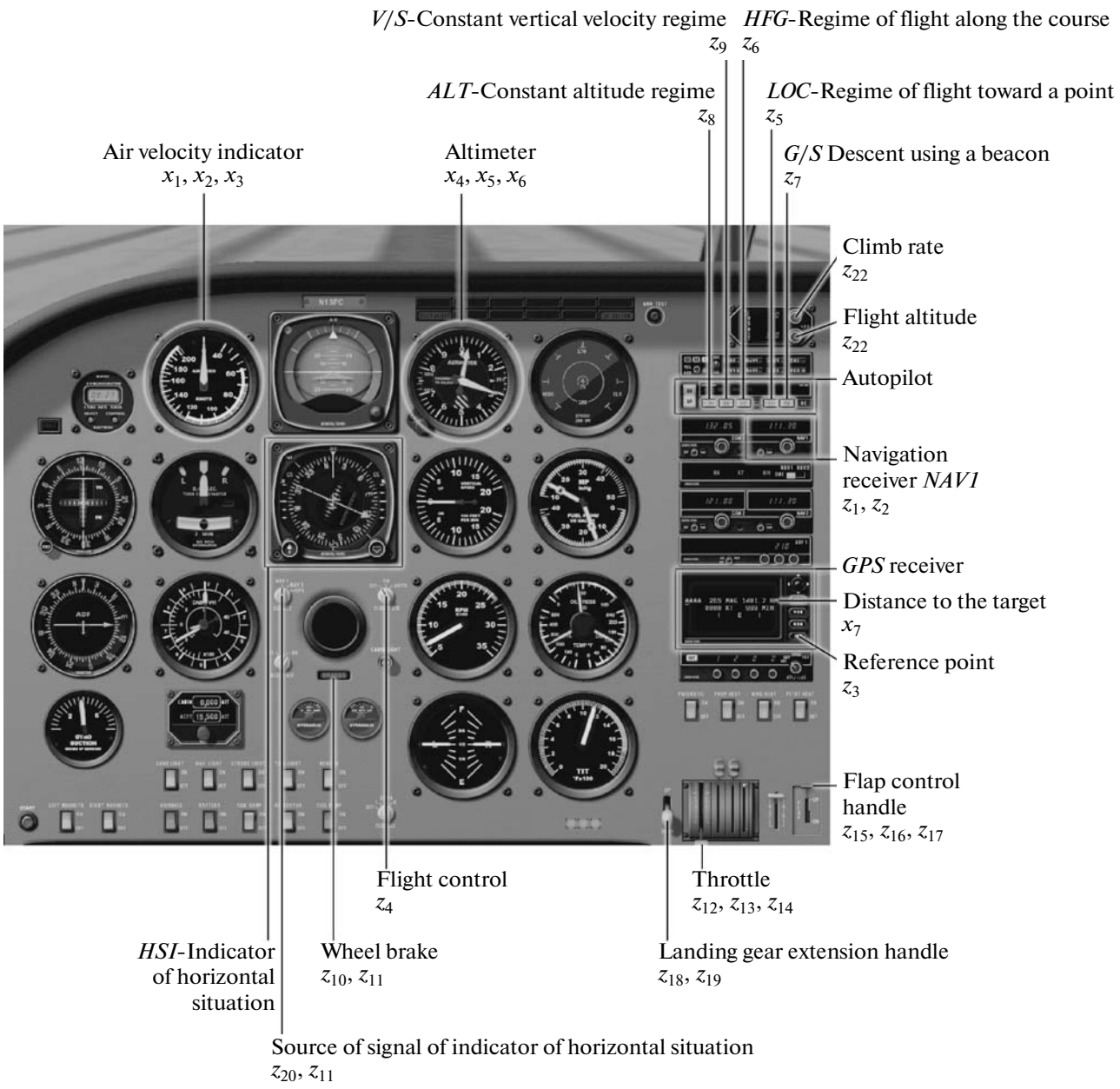


Fig. 11. Applied devices and control organs of the aircraft.

$$f = \frac{t_1 - t_0}{t_i\left(1 + \dfrac{PP_\alpha + VV_\alpha}{t_1 - t_0}\right)(1 + bC_\alpha)},$$

where b is the variable which is equal to zero if the aircraft is safe, and unity if it is broken; $t_i = 77$ s is the time during which in the case of exact following the route (which is physically impossible) autopilot would get the same value of the fitness function as the "ideal" physically possible autopilot gets during the whole flight. The function is normalized to the interval (0, 1), as compared with the "ideal" autopilot. Thus, if the function is equal to unity, this means that the goal of optimization has been achieved.

5.5. Construction of Control Automaton

The algorithm of construction of the control automaton consists of the following steps:

(1) initial population of automata is generated in a random way;

(2) each automaton in the population is estimated. For this purpose, the flight is simulated in the spacecraft simulator controlled by the estimated automaton, and the data on the route and state of the aircraft

are recorded. Then the value of the fitness function is calculated based on these data;

(3) if there is one automaton with the fitness function equal to unity in the population it is chosen as the result, and the process of optimization is completed;

(4) in the opposite case, a new generation is constructed using the operators of mutation and crossover; after that the process returns to step 2.

The plot of variation of the fitness function with increasing generation number in the course of operation of the above algorithm is shown in Fig. 12. The maximal value of the estimator equal to unity for two last generations testifies the successful completion of the process of optimization.

The constraint on the number of significant predicates in each state $r = 1$ was imposed upon the automaton generation. Even for this constraint the optimization lasted about one month. Note that most of the time was spent for the calculation of the fitness function: the emulation necessary for estimation of one automaton required about 5 min. The automaton obtained as a result of optimization has a small number of states and transitions. However, it is rather complicated for comprehension due to a large number of actions at transitions. In order to simplify the automaton, the actions rejecting each other, which do not influence the aircraft behavior in this state, were manually eliminated. The automaton states were enumerated and named. The final variant of the automaton is shown in Fig. 13. Since the number of significant predicates in each state of the control automaton was

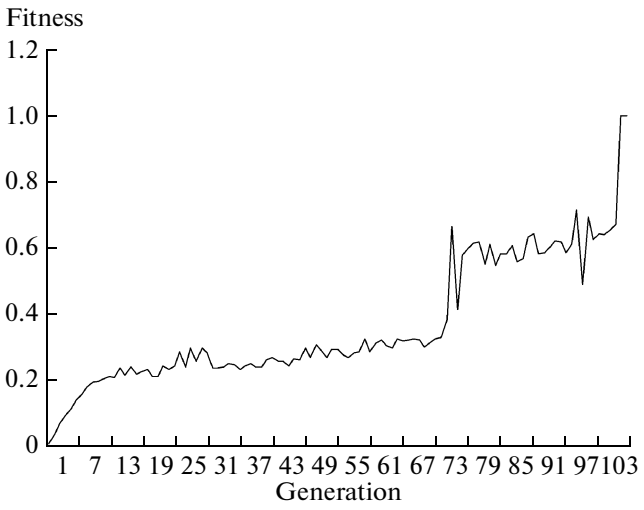


Fig. 12. Variation of fitness in the course of evolution.

Table 2. Output actions of automaton

Identifier	Action description
z_1	Adjusting navigation receiver to the frequency of landing beacon of the departure airport
z_2	Adjusting navigation receiver to the frequency of landing beacon of the destination airport
z_3	Switching the GPS receiver in the following regime to the reference point
z_4	Switching the flight control switch into the position “AUTO”
z_5	Switching autopilot to the flight regime to the point in the horizontal plane
z_6	Switching autopilot to the flight regime along the course in the horizontal plane
z_7	Switching autopilot to the descent regime using the beacon
z_8	Switching autopilot to the constant altitude regime
z_9	Switching autopilot to the constant vertical velocity regime
z_{10}	Switching on wheel brake
z_{11}	Switching off wheel brake
z_{12}	Setting maximal fuel supply
z_{13}	Setting intermediate fuel supply
z_{14}	Setting minimal fuel supply
z_{15}	Retracting flaps
z_{16}	Setting flaps into takeoff position
z_{17}	Setting flaps into descent position
z_{18}	Extending landing gear
z_{19}	Retracting landing gear
z_{20}	Using navigation receiver as the source of signal of indicator of horizontal situation
z_{21}	Using GPS receiver as the source of signal of indicator of horizontal situation
z_{22}	Setting climb rate and flight altitude for autopilot
z_{23}	Controlling rudder according to autopilot signals
z_{24}	Setting rudder into the central position

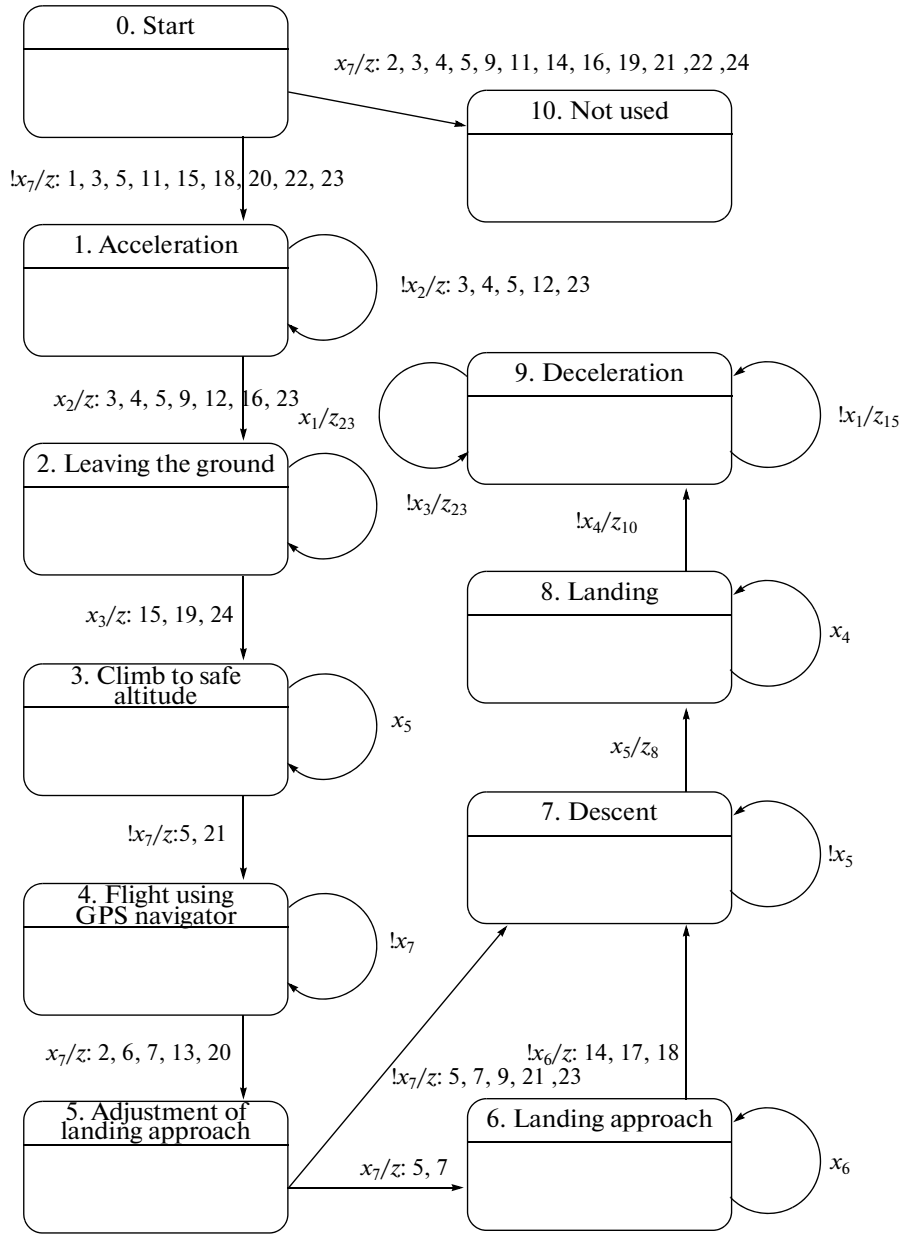
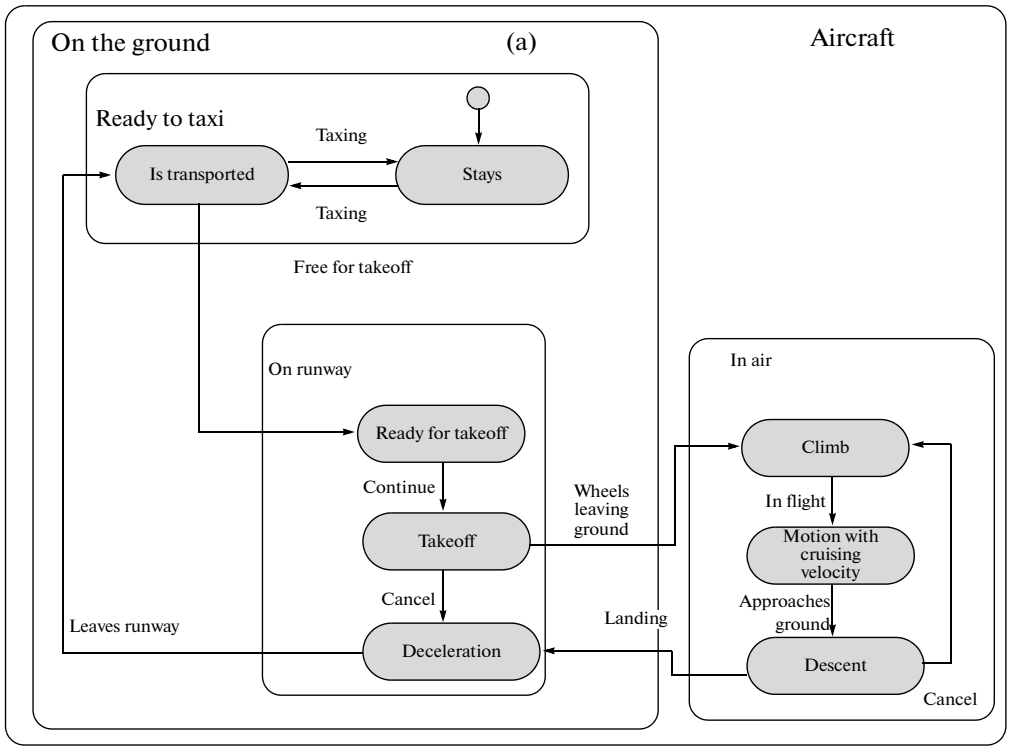


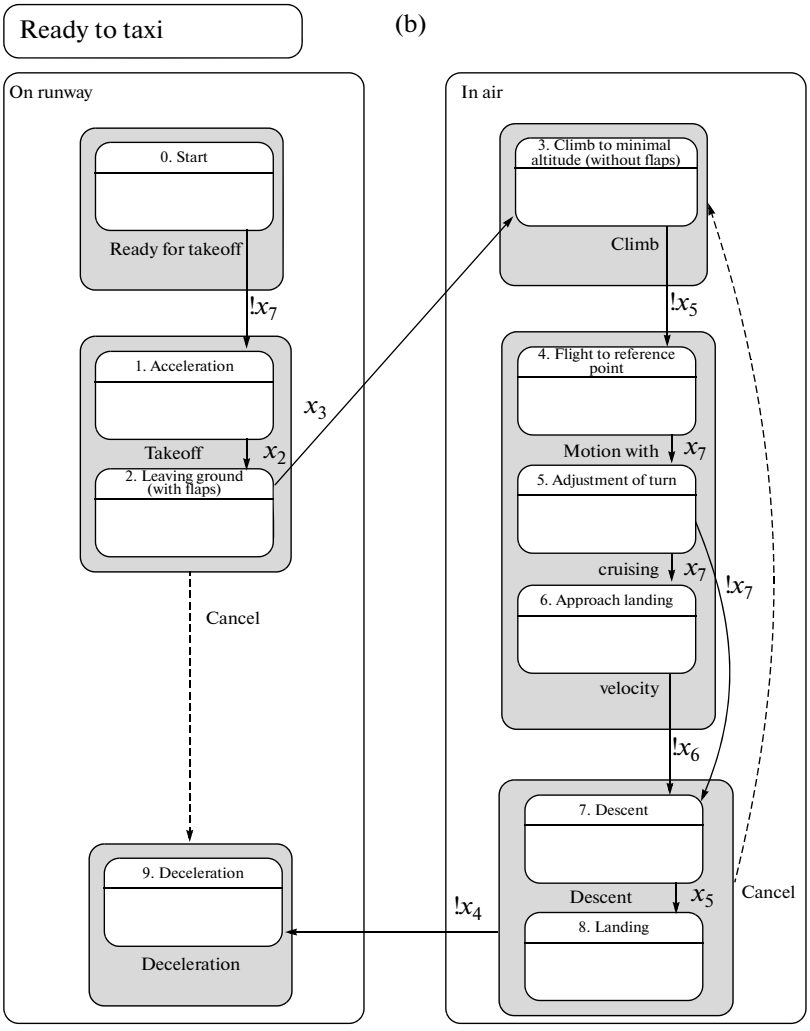
Fig. 13. Transition graph of the obtained automaton.

taken equal to unity, not more than two transitions from each state were possible. Due to the fact that the predicates and actions of the control object were chosen on a rather high level of abstraction, the automaton with such a simple structure was sufficient for control.

The automaton shown in Fig. 13 can be rather easily heuristically (without the use of genetic programming) constructed. For example, in [33] a very similar transition graph describing the aircraft flight (Fig. 14a) was given as the example of the state diagram. In Fig. 14b the states and transitions of this automaton are correlated with the states and transitions of the automaton obtained using genetic optimization in this paper. The difference between the two transition graphs is explained by the difference in the problem formulation. However, often it is not easy to construct the control automaton manually. This automaton in most cases is nonoptimal and unjustified complex. Besides, the optimization approach is characterized by better scalability.

Fig. 14. (a) State diagram for the aircraft flight [33] and (b) its comparison with automatically constructed automaton.


On the ground
(a)
Aircraft
Ready to taxi
Taxing
Is transported
Stays
Taxing
Free for takeoff
On runway
Ready for takeoff
Continue
Takeoff
Cancel
Deceleration
Leaves runway
Wheels leaving ground
Landing
In air
Climb
In flight
Motion with cruising velocity
Approaches ground
Descent
Cancel


Ready to taxi
(b)
On runway
0. Start
Ready for takeoff
!x7
1. Acceleration
Takeoff
x2
2. Leaving ground (with flaps)
x3
Cancel
9. Deceleration
Deceleration
In air
3. Climb to minimal altitude (without flaps)
Climb
!x5
4. Flight to reference point
Motion with
x7
5. Adjustment of turn
cruising
x7
6. Approach landing
velocity
!x6
7. Descent
Descent
x5
8. Landing
!x4
Cancel
!x7

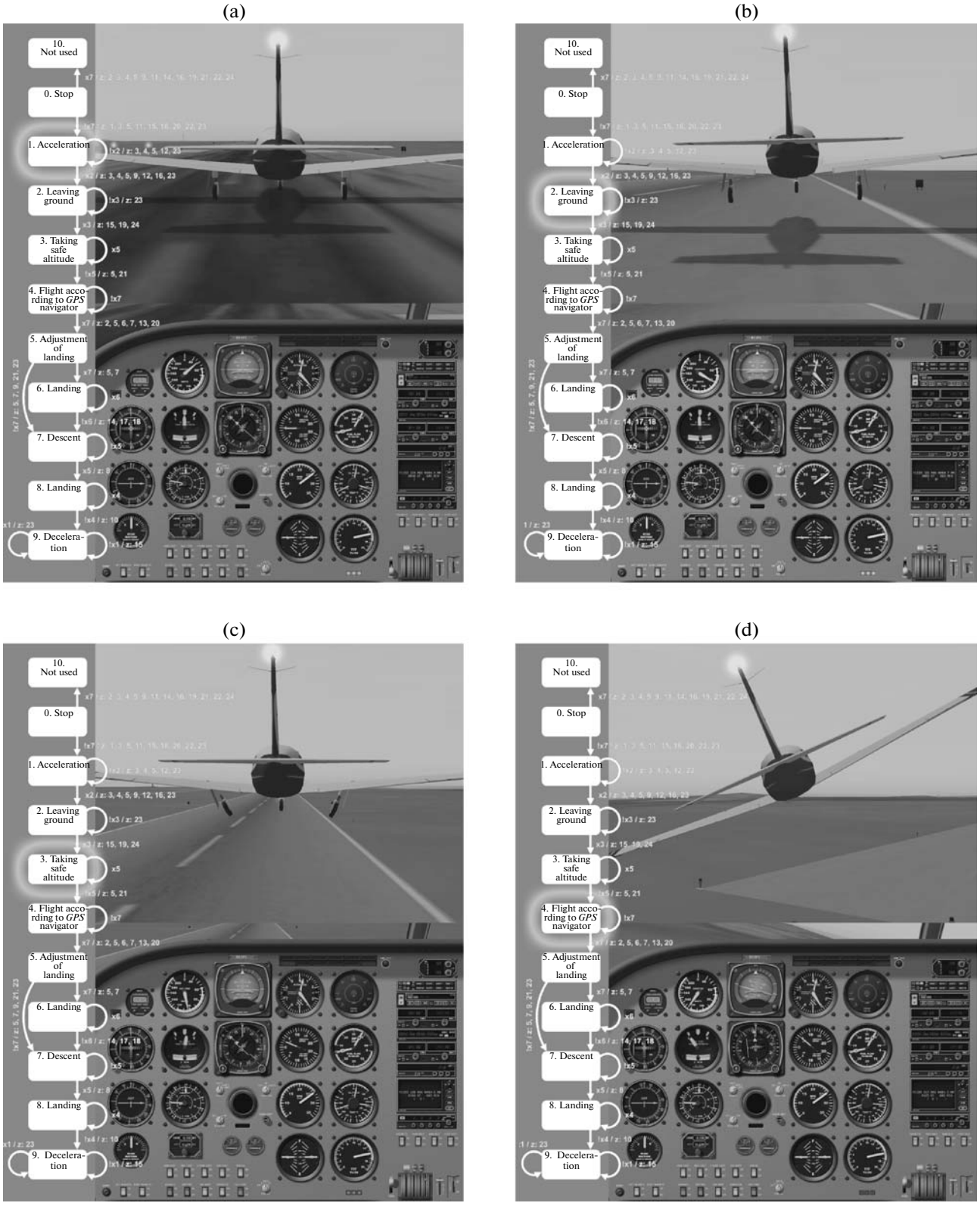
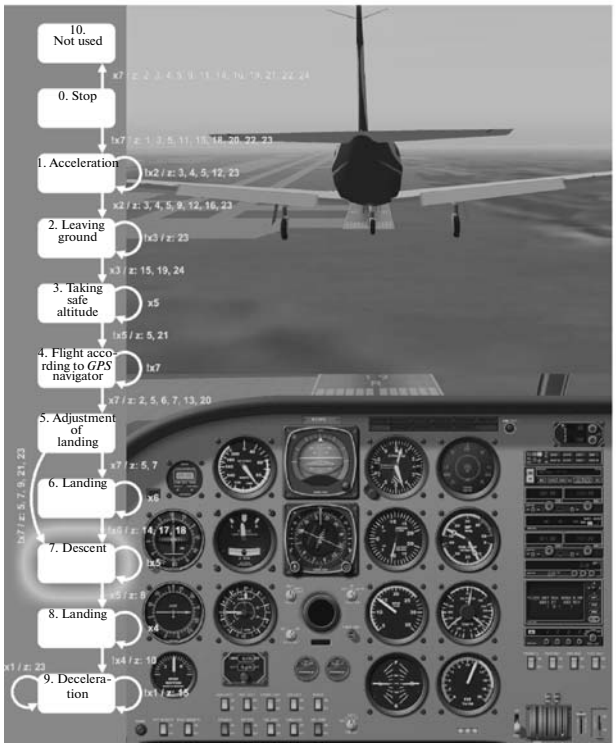


Fig. 15. Automaton and aircraft in different states: (a) acceleration; (b) leaving ground; (c) climb to safe altitude; (d) flight using GPS navigator; (e) adjustment of landing approach; (f) landing approach; (g) descent; (h) landing; (i) deceleration.

(e)

(f)

(g)



(h)

Fig. 15. (Contd.)

(i)

Fig. 15. (Contd.)

5.6. Analysis of Results of Experiment

When the optimization is completed, the test emulation of the flight controlled by the constructed optimal automaton was performed. Figure 15 shows the automaton, the aircraft position, and the device indications at different stages of test emulation. In the course of test emulation the deviation from the route and the recommended velocity, the flight altitude (which makes it possible to determine the time instants of leaving the ground and touching the ground), and the acting overloads were recorded. The recorded data are shown in Fig. 16.

It follows from the plot that the maximal deviation of the aircraft from the route during the whole flight time was 314 m. It did not exceed 60 m during the whole flight except for a 1 min segment in the middle of the plot when the deviation is the least critical. The time instants of leaving and touching the ground can be determined from the plot of the flight altitude: 35 and 271 s, respectively. It follows from the established time and the plot of deviation from the route that the maximal deviation during acceleration was 12 m, and during deceleration 10 m. The deviations which are significant but do not go beyond the runways provide the conclusion that the generated automaton guided the aircraft along the route with sufficient precision.

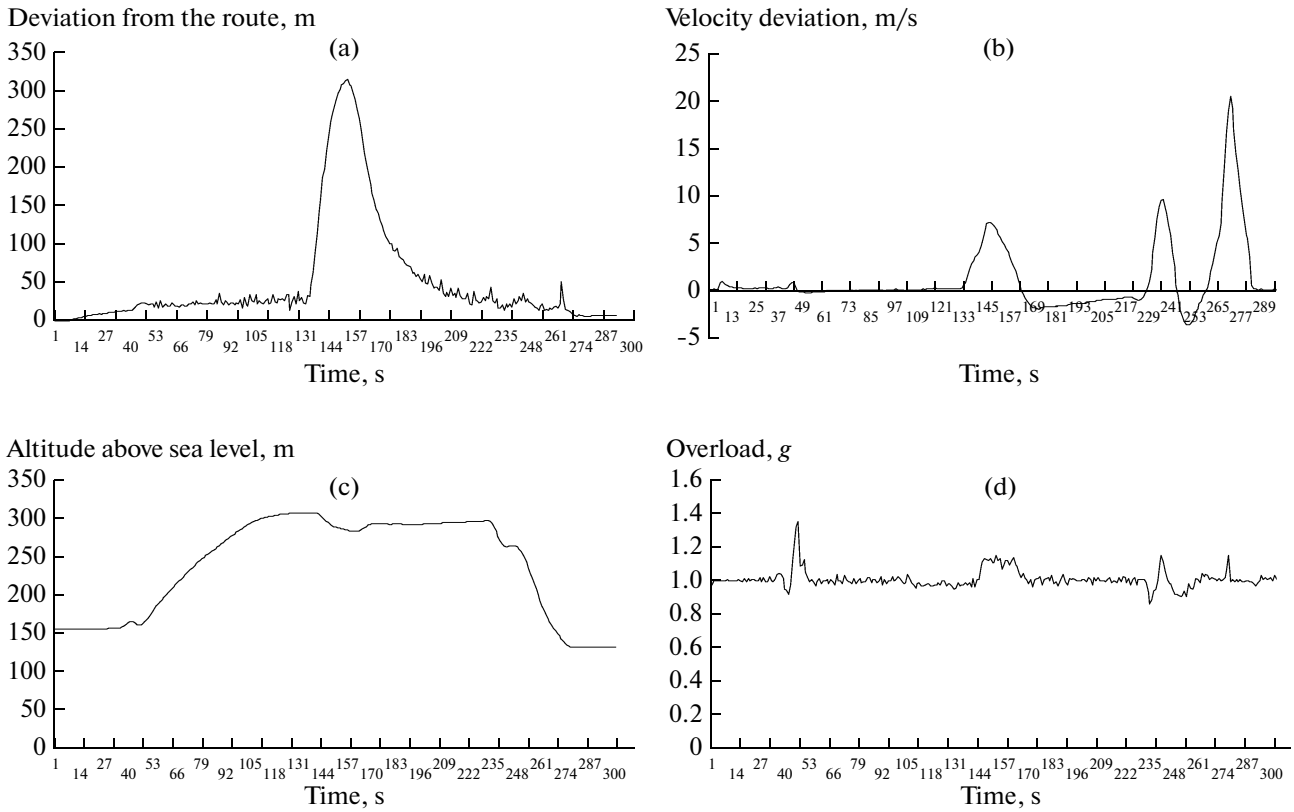


Fig. 16. Results of testing: (a) deviation from the route; (b) deviation from the recommended velocity; (c) flight altitude above the sea level; (d) overload.

It follows from the plot of recommended velocity that the velocity regime was observed, except for exceeding the velocity by 20.34 m/s at the time instant of touching the ground, which was compensated in the course of deceleration and did not result in a considerable excess in the stopping distance. The combined deviation of the aircraft position after stopping including the transverse deviation from the center of the runway and the excessive stopping distance is 6.5 m. It follows from the plot of overload that the enhanced deceleration did not result in a considerable discomfort of passengers. This analysis yields the conclusion on the satisfactory quality of the automatically constructed automaton.

CONCLUSIONS

The disadvantages of the existing methods of generation of finite automata using genetic algorithms were demonstrated and the method free from these disadvantages was proposed. The performance of the proposed method and its memory requirements were experimentally estimated. The method was successfully tested in the problem of automatic generation of the aircraft control system and proved its applicability for solution of practical tasks.

REFERENCES

1. V. M. Kureichik, "Genetic Algorithms: State of the Art, Problems, and Perspectives", Izv. Ross. Akad. Nauk, Izv. Ross. Akad. Nauk, Teor. Sist. Upr., No. 1, 144–160 (1999) [Comp. Syst. Sci. **38** (1), 137–152 (1999)].
2. V.V. Kureichik, V.M. Kureichik, and P.V. Sorokoletov, "Analysis and a Survey of Evolutionary Models", Izv. Ross. Akad. Nauk, Izv. Ross. Akad. Nauk, Teor. Sist. Upr., No. 5, 114–126 (2007) [Comp. Syst. Sci. **46** (5), 779–791 (2007)].
3. J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, 1992).
4. V. M. Kureichik and S. I. Rodzin, "Evolutionary Algorithms: Genetic Programming", Izv. Ross. Akad. Nauk, Izv. Ross. Akad. Nauk, Teor. Sist. Upr., No. 1, 127–137 (2002) [Comp. Syst. Sci. **41** (1), 123–132 (2002)].
5. E. M. Gold, "Language Identification in the Limit", Informat. Control, **10**, 447–474(1967).
6. A. Belz, "Computational Learning of Finite-State Models for Natural Language Processing", PhD thesis, University of Sussex, 2000.
7. C. H. Clelland and D. A. Newlands, "Pfsa Modelling of Behavioural Sequences by Evolutionary Programming", in *Proceedings of 2nd Australian Conf. on Complex Systems Complex'94, Rockhampton, Queensland, Australia, 1994*, 165–172.
8. S. Das and M.C. Mozer, "A Unified Gradient-Descent/Clustering Architecture for Finite State Machine Induction", Advances in Neural Information Processing Systems, **6** (1994).
9. M. M. Lankhorst, "A Genetic Algorithm for the Induction of Nondeterministic Pushdown Automata", Computing Science Report (University of Croningen Department of Computing Science, Groningen, 1995).
10. A. Belz and B. Eskikaya, "A Genetic Algorithm for Finite State Automata Induction with an Application to Phonotactics", in *Proceedings of ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing, Saarbruecken, Germany, 1998*, 9–17.
11. D. Ashlock, A. Wittrock, and T.-J. Wen, "Training Finite State Machines to Improve PCR Primer Design", in *Proceedings of Congress on Evolutionary Computation (CEC'02), Honolulu, Hawaii, USA, 2002*, 13–18.
12. D. A. Ashlock, S. J. Emrich, K. M. Bryden, et al., "A Comparison of Evolved Finite State Classifiers and Interpolated Markov Models for Improving PCR Primer Design", in *Proceedings of 2004 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB'04), Jolla, California, USA, 2004*, 190–197.
13. S. M. Lucas, "Evolving Finite State Transducers: Some Initial Explorations", in *Proceedings of Genetic Programming: 6th European Conference (EuroGPKh03)* (Springer, Berlin, 2003), 130–141.
14. P. G. Lobanov and A. A. Shalyto, "Application of Genetic Algorithms for Automatic Construction of Finite-State Automata in the Problem of Flibs", Izv. Ross. Akad. Nauk, Izv. Ross. Akad. Nauk, Teor. Sist. Upr., No. 5, 127–136 (2007) [Comp. Syst. Sci. **46** (5), 792–801 (2007)].
15. F. N. Tsarev and A. A. Shalyto, "Application of Genetic Algorithms for Construction of Automatons with Minimal Number of States for the "Smart Ant" Problem", in *Scientific Software in Education and Scientific Research* (SPbGU PU, St. Petersburg, 2008), 209–215.
16. A. Teller, and M. Veloso, *PADO: A New Learning Architecture for Object Recognition. Symbolic Visual Learning* (Oxford Univ. Press, New York, 1996).
17. W. Banzhaf, P. Nordin, R. E. Keller, et al., *Genetic Programming - An Introduction. On the Automatic Evolution of Computer Programs and its Application* (Morgan Kaufmann Publishers, San Francisco, 1998).
18. W. Kantschik, P. Dittrich, and M. Brameier, "Empirical Analysis of Different Levels of Meta-Evolution", in *Proceedings of Congress on Evolutionary Computation,, Washington DC, USA, 1999.*
19. W. Kantschik, P. Dittrich, M. Brameier, et al., "Meta-Evolution in Graph GP", in *Proceedings of Genetic Programming: Second European Workshop (EuroGP'99), Goeteborg, Sweden, 1999.*
20. A. Teller and M. Veloso, "Internal Reinforcement in a Connectionist Genetic Programming Approach", in *Artificial Intelligence* (North-Holland Pub. Co, 1970), 161.
21. J. H. Miller, "The Coevolution of Automata in the Repeated Prisoner's Dilemma. Working Paper" (Santa Fe Institute, 1989).
22. W. M. Spears and D. F. Gordon, "Evolving Finite-State Machine Strategies for Protecting Resources", in *Proceedings of International Syposium on Methodologies for*

Intelligent Systems, Charlotte, North Carolina, USA, 2000.

23. D. Ashlock, *Evolutionary Computation for Modeling and Optimization* (Springer, New York, 2006).
24. C. Frey and G. Leugering, "Evolving Strategies for Global Optimization. A Finite State Machine Approach", in *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, USA, 2001), 27–33.
25. P. Petrovic, "Simulated Evolution of Distributed FSA Behaviour-Based Arbitration", in *Proceedings of Eighth Scandinavian Conference on Artificial Intelligence (SCAI'03), Bergen, Norway, 2003.*
26. P. Petrovic, *Evolving Automatons for Distributed Behavior Arbitration. Technical Report* (Norwegian University of Science and Technology, Trondheim, Norway, 2005).
27. P. Petrovic, *Comparing Finite-State Automata Representation with GP-trees. Technical report* (Norwegian University of Science and Technology, Trondheim, Norway, 2006).
28. N. I. Polikarpova and A. A. Shalyto, *Automaton Programming* (Piter, St. Petersburg, 2009) [in Russian].
29. J.R. Koza, *Future Work and Practical Applications of Genetic Programming. Handbook of Evolutionary Computation* (IOP Publishing Ltd, Bristol, 1997).
30. N. I. Polikarpova, V. N. Tochilin, and A. A. Shalyto, "Application of Genetic Programming for Implementation of Systems with Complex Behavior", in *Proceedings of 4th International Scientific–Practical Conference on Integrated Models and Soft Calculations in Artificial Intelligence*, vol. 2 (Fizmatlit, Moscow, 2007), 598–604.
31. N. I. Polikarpova, V. N. Tochilin, and A. A. Shalyto, "Development of a Library for Generation of Control Automata Using Genetic Programming", in *Proceedings of 10th International Conference on Soft Calculations and Measurements*, vol. 2 (SPbGETU "LETI", St. Petersburg, 2007).
32. http://www.x-plane.com/.
33. E. Hull, K. Jackson, and J. Dick, *Requirements Engineering* (Springer, Berlin, 2002).