Application of Genetic Programming for Generation of Controllers represented by Automata

Andrey Davydov, Dmitry Sokolov, Fedor Tsarev and Anatoly Shalyto

Saint-Petersburg State University of Information Technologies, Mechanics and Optics (e-mail: andrey.a.davydov@gmail.com, dimoz_88@rambler.ru, fedor.tsarev@gmail.com, shalyto@mail.ifmo.ru).

Abstract: This paper proposes an application of genetic programming for construction of state machines controlling systems with complex behavior. Application of this method is illustrated on example of unmanned aerial vehicle (UAV) control. It helps to find control strategies of collaborative behavior of UAV teams. Multi-agent approach is used, where every agent that controls a UAV is presented by a deterministic finite state machine. Two representations of finite state machines are used: abridged transition tables and decision trees. Novel algorithms for fixing connections between states and for removing unachievable branches of trees are proposed.

1. INTRODUCTION

Automata based programming is an emerging programming paradigm based on representation of a program as a finite state machine (Shalyto 1991). Application of genetic programming for generating automata essentially reduces development time of automata based programs.

In this work the proposed method is illustrated on example of building a Mealy automata controller of unmanned aerial vehicles (UAV).

The paper is structured as follows.

The problem is described in Section 2. The idea of suggested solution is considered in Section 3 and next three sections explain it. The developed genetic algorithm is considered in Section 6. Methods of control system representation in terms of individual and genetic operators (crossover and mutation) are described in Sections 4 and 5.

2. PROBLEM DESCRIPTION

There is competition between two teams of UAVs (Parachsenko, Tsarev & Shalyto). Each team consists of the same number *N* of UAVs. The goal of the competition is reaching maximal distance by one UAV of a team. Further we will refer to the competition as “race”. The race takes place on the route which is a half-infinite stripe with width equal to 40 metres. Altitude change is forbidden so the route of competition can be considered two-dimensional.

At the start of the race UAVs of the first team are located in the air randomly on the same distance from the line of start in the left half of the route. The second team is located symmetrically in the right half of the route.

It is assumed that all UAVs start the race with the same initial speed, direction of movement and fuel supply.

UAVs can turn in process of racing and can change their speed controlled by fuel consumption.

UAVs leaving the route are defined as breaking the rules of the race. Leaving the route is defined as intersection of the centre of the UAV with the boundary of the route. Also UAV is said to crash if its speed becomes less than a pre-defined minimum value or in case of UAVs collision with speed higher than a pre-defined maximum value. Otherwise collisions are allowed.

The dynamics of UAV is defined by the Second Law of Newton and is described in the work (Parachsenko, Tsarev & Shalyto) in detail. It is important that location of other vehicles has influence on the vehicle’s slowing down (due to reactive force of engine) or speeding up (due to air resistance).

3. SUGGESTED SOLUTION

3.1 Common Approach

In the work (Parachsenko, Tsarev & Shalyto) a multi-agent approach (Rasse & Norwig 2006) to control UAVs was suggested. Every UAV is considered independently and its behaviour is described by a finite state machine, same for every UAV in the team.

The purpose of this work is building the controller state machine using genetic programming (Angeline & Pollack 1993, Chambers 1999, Jefferson et. al 1992, Koza 1992, Rassel & Norwig 2006, Tsarev & Shalyto 2007). We will use term “individual” when referring to a finite state machine in the context of genetic programming. In this work two representation methods for an individual are explored: by abridged transition tables (ATT) (Polykarpova, Tochilin & Shalyto) and decision trees (Danilov 2007). These methods are described in sections 4 and 5 respectively.

3.2 Structure of UAV Control System

Every UAV is under the influence of environment and other UAVs during the competition. For description of these forces the following Boolean input variables are used:

1 Route boundary is on the left.
2 Route boundary is on the right.
3 Other UAV is on the left.
4 Other UAV is on the right.
5 Other UAV is at the front.
6 Other UAV is behind.

These variables are input to the controller state machine which forms a sequence of output actions from the following list:

1 Set normal fuel consumption.
2 Increase fuel consumption by a fixed value.
3 Decrease fuel consumption by a fixed value.
4 Set fuel consumption to the maximal value.
5 Change the direction of aerodynamic rudder by a fixed value to the left.
6 Change the direction of aerodynamic rudder by a fixed value to the right.
7 Fly straight.

4. PRESENTATION OF AN INDIVIDUAL BY ABRIDGED TRANSITION TABLES

4.1 Structure of Individual

Every individual is defined by the following parameters:

- Array of states.
- Number of states.
- Number of processing input variables.
- Number of possible actions.

Every state stores ATT which contains:

- Array of meaningful input variables.
- Array of target states for all outgoing transitions.
- Array of actions for all outgoing transitions.

An example of the ATT for one state is presented in Fig.1. Column **S** is an array of target states, columns **Z1**—**Z4** are arrays of actions, **Zj**[**i**] is equal to one if **j**-th action is executed when **i**-th transition is selected. The Boolean **variables** array points which variables are meaningful: a variable is called *meaningful* in a current state of a state machine if it is included in any outgoing transition from this state.

4.2 Crossover

Let's designate parent individuals as *P1* and *P2*, their children as *S1* and *S2*, and the *k*-th state of individual *A* as *A.a*[*k*].

The algorithm of states crossover corresponds to one-point crossover of the corresponding columns in the transitions table. For every column the following actions are executed:

(1) random choice of a column's border of differentiation;

S	Z1	Z2	Z3	Z4
1	0	0	1	0
3	1	0	1	1
5	1	0	1	0
1	1	0	1	1

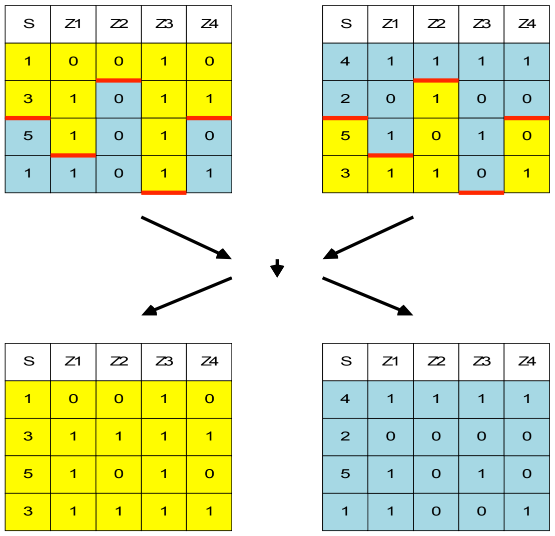
variables	A	B	C	D	E	F	G
	0	0	1	0	0	1	0

Fig. 1. Example of an abridged transition table.

(2) corresponding column of state *S1.a*[*k*] will be composed in the first part of column of state *P1.a*[*k*] and the second part of column of state *P2.a*[*k*];

(3) corresponding column of state *S2.a*[*k*] will be composed in the first part of column of state *P2.a*[*k*] and the second part of column of state *P1.a*[*k*].

This algorithm is illustrated in Fig. 2.



S	Z1	Z2	Z3	Z4
1	0	0	1	0
3	1	0	1	1
5	1	0	1	0
1	1	0	1	1

S	Z1	Z2	Z3	Z4
4	1	1	1	1
2	0	1	0	0
5	1	0	1	0
3	1	1	0	1

S	Z1	Z2	Z3	Z4
1	0	0	1	0
3	1	1	1	1
5	1	0	1	0
3	1	1	1	1

S	Z1	Z2	Z3	Z4
4	1	1	1	1
2	0	0	0	0
5	1	0	1	0
1	1	0	0	1

Fig. 2. Crossover of abridged transition tables.

For choice of meaningful variables *S1.a*[*k*] and *S2.a*[*k*] preview of lists of meaningful variables of parents is executed. Let's designate *v*[*k*] the *k*-th element of the list of meaningful variables.

One of the following statements is true:

- *P1.v*[*k*] = 0, *P2.v*[*k*] = 0 ⇒ *S1.v*[*k*] = 0, *S2.v*[*k*] = 0;
- *P1.v*[*k*] = 1, *P2.v*[*k*] = 1 ⇒ *S1.v*[*k*] = 1, *S2.v*[*k*] = 1;

• *P1.v*[*k*] = 0, *P2.v*[*k*] = 1 or *P1.v*[*k*] = 1, *P2.v*[*k*] = 0 ⇒ one of three following cases is possible:

(1) lists of meaningful variables of both children are not filled yet (number of meaningful variables in both lists is less than numbers of meaningful variables of parents) then with probability 1/2 one of following equations is satisfied *S1.v*[*k*] = 0, *S2.v*[*k*] = 1 or *S1.v*[*k*] = 1, *S2.v*[*k*] = 0;

(2) list of meaningful variables of the first child is filled, then *S1.v*[*k*] = 0, *S2.v*[*k*] = 1;

(3) list of meaningful variables of the second child is filled, then *S1.v*[*k*] = 1, *S2.v*[*k*] = 0.

4.3 Mutation

One of the following actions is executed:

• with probability 1/2 random change of the initial state;

• mutation of a random state;

State mutates as follows: for every row of the table of transitions the following actions are executed:

• with a fixed probability change transition from the current state to random value;

• with a fixed probability change all actions on the transition. Every action is executed on the transition with probability *n/k* where *n* is number of actions have been executed on the transition before mutation, *k* is number of possible actions.

The choice of meaningful variables of state is realised in the following way:

•a) randomly select two variables;

•b) if one of them is meaningful in the state and the other is not then they are swapped.

An example of mutation is shown in Fig. 3. The turning out parts of the state are marked by grey colour.

4.4 Fixing Connections between States.

After application of genetic operators, state machines may contain unachievable states. The following algorithm helps to solve this problem. It provides achievement of greater number of states from the initial state by changing some transitions. To search for achievable states, the Breadth First Search algorithm is used. It is executed once or more and if an unachievable state is found then random transition from a random achievable state is set to the current unachievable state.

Fig. 4 illustrates working of this algorithm. States achievable from the initial state are marked by grey colour.

4.5 Features of the ATT Method Application

The advantages of the ATT method in respect to the considered problem are as follows:

• incompatibility of some input variables (for example, "boundary is on the left" and "boundary is on the right") makes some transitions unusable. When we use ATTs the probability of such situation reduces, compared to full transition tables;

• reduction of the memory requirements and speeding-up the work of the genetic algorithm.

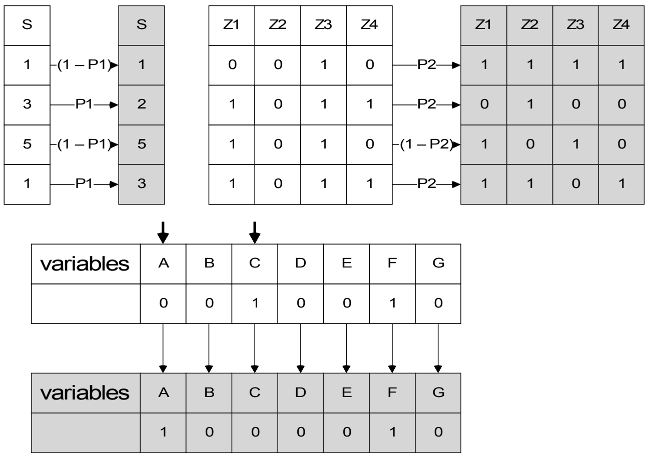


Fig. 3. Mutation of an abridged transition table.

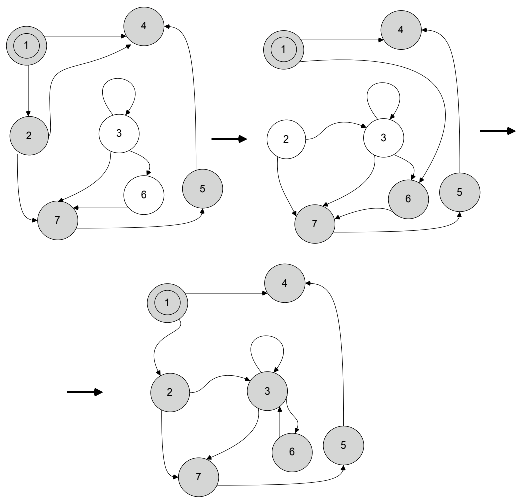


Fig. 4. Example of working of the algorithm of fixing connections between states.

When we use abridged transition tables all meaningful variables have the same priority because decision is made at once based on all the variables which are used in the state.

5. REPRESENTATION OF AN INDIVIDUAL BY DECISION TREE

5.1 Structure of an Individual

Every individual stores the following parameters:

• Array of states;

• Advisable height of the tree.

Every state is decision tree for a function of the following type: $f\colon \{0,1\}^n \to N \times \{0,1\}^k$, where *n* is number of entrance variables and *k* is number of possible actions of

UAV. This tree reads values of entrance variables and returns index of state which state machine must go to and vector of zeros and ones. If the *i*-th component of the vector equals to one then the UAV must execute the *i*-th action. We will call this vector action vector. An example of decision tree for a function of two Boolean variables (*A, B*) is shown in Fig. 5.

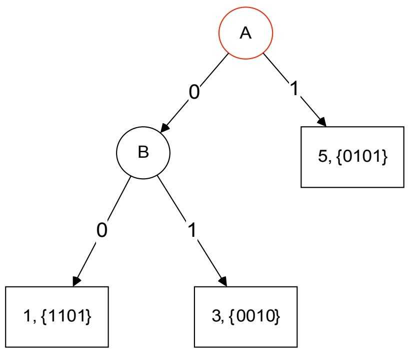


Fig. 5. Example of a decision tree.

Every node of a decision tree consists of the following parts:

• a pointer to the left child (this pointer is empty for leaves);

• a pointer to the right child (this pointer is empty for leaves);

• a variable which is used when the decomposition in the node is made;

• index of the state into which the transition from the current state follows;

• action vector.

The left child of the inside node corresponds to the case when the variable of decomposition equals zero and the right child corresponds to one.

5.2 Crossover

For every state the following algorithm will be executed. It chooses subtree of the state of the first parent and subtree of the state of the second parent and then swaps them. This algorithm is a modification of the Depth First Search and has recursive structure. When recursive call is invoked, the following action is executed:

• if current node is a leaf then return current subtree;

• else if current node is internal then with probability *P* return current subtree;

• else go to one of subtrees of the current node with equal probabilities.

Working of the algorithm is shown in Fig.6.

5.3 Mutation

The operator of mutation executes in the following way:

• with probability *1/2* random change of the initial state;

• mutation of a random state.

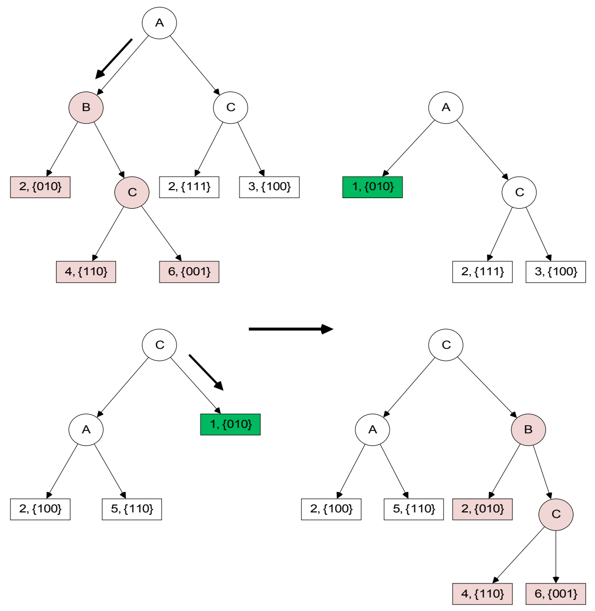


Fig. 6. Crossover of decision trees.

The algorithm of mutation of state is a modification of the Depth First Search. Following actions are executed:

• if current node is a leaf then return random generated decision tree;

• else with probability P return random generated decision tree;

• else go to one of the subtrees of current node with equal probabilities.

In fact, this algorithm randomly chooses a subtree and replaces it by a randomly generated tree. Choice of the subtree is not equiprobable: the higher located node has the higher probability of having its subtree chosen.

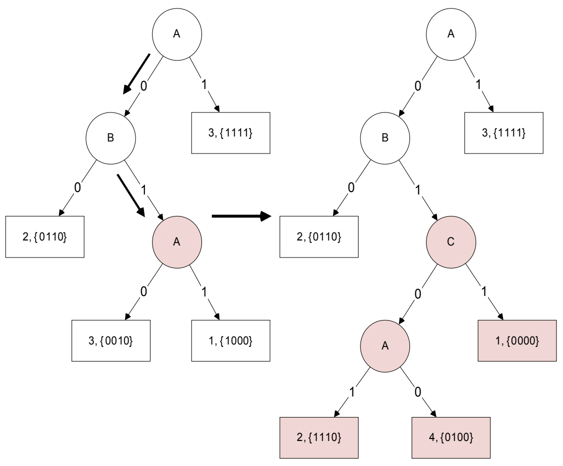


Fig. 7. Mutation of decision trees.

5.4 Removing of Unachievable Branches

If a decomposition variable is repeated twice on the path from the root to the node then the branch corresponding to the value which is opposite to the first occurrence of this variable will be unachievable. Fig. 8 shows such a situation. Vertices corresponding to the repeated decomposition variable are marked by red colour (both A nodes), the unachievable branch (C and two leafs) is marked by grey colour.

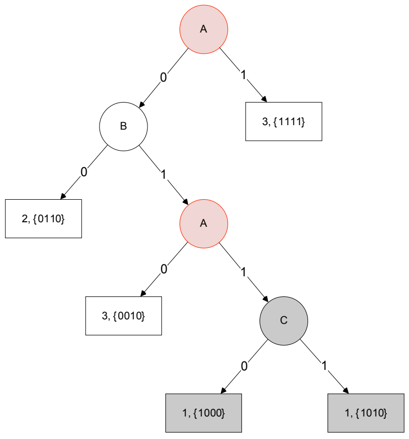


Fig. 8. Unachievable branches of decision tree.

To remove such unachievable branches, a modification of the Depth First Search is used. The decomposition variables on the path from root to nodes are searched recursively. If a variable is met twice then the current node is replaced by the root of an achievable subtree of current node. Decision about which subtree is achievable is made based on the stored values of variables. An example is shown in Fig.9.

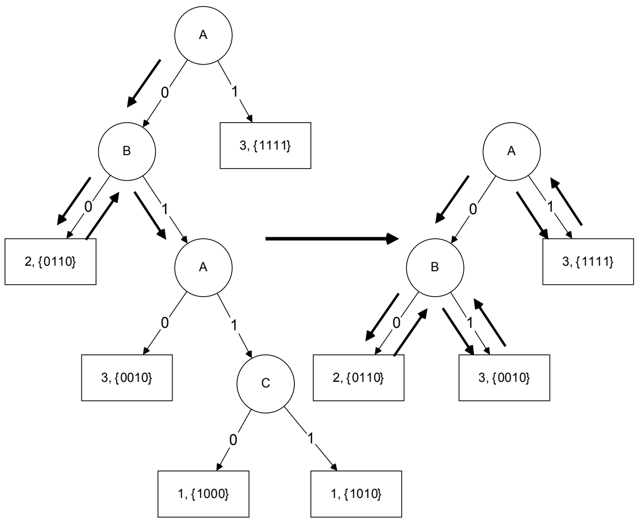


Fig. 9. Removing the unachievable branches of the decision tree.

5.5. Features of Application of Decision Trees

Decision tree has the following advantages:

• not all variables may be used. It allows to exclude the transitions with incompatible variables;

• reduction of using memory (compared to full trees) and even shorter tables of transitions.

A specific feature of decision trees is that in every state of state machine variables have different priorities. It's related to the fact that the higher node of tree is more possible to get into comparing to the other variables.

6. GENETIC ALGORITHM

In this work the *Island Genetic Algorithm* (Koza 1992) is used. The main schema of this algorithm is that there are few populations (*islands*). Most of the time, the progress of populations on every island occurs independently. After a fixed number of generations, the migration occurs. Migration is the transfer of a part of individuals from some island to another one.

6.1 Forming the Next Generation

Elitism is the main strategy of forming of the next generation. All individuals except the most fit ones are given up. The most fit individuals are called elite, they move up straight to the next generation.

Then the next generation is supplemented on a pro rata basis by random individuals, individuals which have mutated, and results of crossover of individuals from the current generation. Individuals that have right to give children are selected using a tournament: two pairs of individuals are chosen and the more fit individual in each pair becomes one of the parents.

6.2 Calculating the Fitness function

Fitness function of an individual must depend on the result of the team of UAVs using the strategy described by the individual. For the setting of dependence we must describe conditions of competition: initial coordinates of UAVs and the rival's team. It is a problem because if we chose initial coordinates randomly for every start of racing then fitness function will not be "objective" enough. Opposite way is to fix the initial coordinates and the rival.

We suggest to calculate fitness function as the mean value of a few competitions:

$$F=\frac{\sum r_i}{k}$$

We choose random initial coordinates but the same for all teams that realise the strategies described by the generated individuals. The number of competitions we choose equals 10, in order to reduce the time consumption.

For individuals presented as decision trees formula is changed a little: $F=\frac{\sum r_i}{k}-C*Z\left(h_{max},height\right)$ where *C* is some constant, *hmax* is the maximal height of decision trees corresponded to the states of the state machine, *height* is the “recommended” height of decision tree, and Z is the function described by the following way:

$$Z(a,b) = if\ (a>b)\ then\ (a-b)\ else\ 0$$

Thus this function guarantees that the decision tree doesn’t expand too much.

Two teams are chosen as rivals. The first team realises the aggressive strategy. Its strategy implicates that one UAV flies only forward with normal fuel consumption (i.e. such that provides the longest flight). Other UAVs of the team try to knock down the rival’s team members.

The second rival is a team with strategy described by finite state machine built using the described genetic algorithm at the first stage, i.e. when we use only one rival (aggressive).

This rival realises a friendly strategy. Its strategy implicates that UAVs help each other to fly and don't try to knock down the rival’s UAVs. It helps to train the new individuals on two different strategies at once: on the aggressive one and on the friendly one.

The problem is that it is impossible to grade a given individual on an absolute scale. It is caused by the non-determinism of initial parameters of the competition and of the rivals.

Number of generations of the genetic algorithm is limited, because the individuals generated in later generations don’t have a universal behaviour, but are rather adapted for specific rivals and initial conditions.

But if a specific rival is defined it will not be difficult to grow up a team which plays better against it.

6.3. Features of Applications of Island Genetic Algorithm

The Island Genetic Algorithm has the following advantages compared to the algorithms which generation evolves as a single entity:

• faster convergence to maximum, thanks to migrations;

• possibility of leaving the local maximum without losing the previous results, thanks to isolation of islands.

A disadvantage of the Island Genetic Algorithm is the frequent hitting of local maximums.

7. CONCLUSION

In this work we described an application of Island Genetic Algorithm and two methods of presentation of individuals (shorter table of transitions and decision trees) for building controller finite state machine for a model of UAV.

All automata, which are constructed by hand or by previous methods (Parachsenko, Tsarev & Shalyto) show results equal to 242 meters at best and less than 220 meters on the average. Methods which are described in this paper provide results equal to 310 meters at best and 270 on the average. It proves the efficiency of used methods of presentation of state machines and their optimisation in respect to the considered problem.

We can't state which method of the two suggested methods is better. The second method is a little more complex and when using the genetic algorithms, it works a little longer but it shows better results. It seems that these methods must be used for more complex problems in order to establish which one is better.

REFERENCES

Angeline P. J., Pollack J. (1993). Evolutionary Module Acquisition. Proceedings of the Second Annual Conference on Evolutionary Programming. La Jolla, California.

Chambers L. (1999). Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volume III. CRC Press.

Danilov V. R. (2007). Technology of genetic programming for generating state machines for controlling by system with complex behaviour. Bachelor diploma. SPbSU IFMO.

Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A. (1992). The Genesys System. Los Angeles, California.

Koza J. R. (1992). Genetic programming: on the programming of computers by means of natural selection. MIT Press.

Paracshenko D. A., Tsarev F.N., Shalyto A.A. (2006). The Technology of simultation of a class of multiagent system on basis of automata-based programming by the example if game Flying Plates Competition. Project documentation. SPbSU IFMO, Saint-Petersburg. http://is.ifmo.ru/unimod-projects/plates/

Polykarpova N.I., Tochilin V.N., Shalyto A.A. (2007). Using of genetic programming for implementation system with complex behaviour. *Scientific and technical bulletin. Effort over the range information technology*. SPbSU IFMO. Edt. 39, p. 276–293.

Rassel S., Norwig P. (2006). *Artificial intelligence. Modern Approach*. Williams, Moscow.

Shalyto A.A. (1998). Technolgy of automata-based programming. *Works of the first All-Russian scientist conference*. MSU, Moscow.

Tsarev F. N., Shalyto A. A. (2007). About constructing automata with minimal number of states for “Artificial Ant” problem. *Proceedings of X international conference on soft computing and measuring*. SPbSU “Eltech”. Volume II, 2007, p. 88–91.