Genetic Algorithm for Induction of Finite Automata with Continuous and Discrete Output Actions

Anton Alexandrov
St. Petersburg State University of IT, Mechanics and Optics
Russia, St. Petersburg, Kronverksky pr., 49
+7 (812) 232-43-18
alexandrov@rain.ifmo.ru

Alexey Sergushichev
St. Petersburg State University of IT, Mechanics and Optics
Russia, St. Petersburg, Kronverksky pr., 49
+7 (812) 232-43-18
alserg@rain.ifmo.ru

Sergey Kazakov, Fedor Tsarev
St. Petersburg State University of IT, Mechanics and Optics
Russia, St. Petersburg, Kronverksky pr., 49
+7 (812) 232-43-18
{svkazakov,tsarev}@rain.ifmo.ru

ABSTRACT
In this paper, we describe a genetic algorithm for induction of finite automata with continuous and discrete output actions. Input data for the algorithm is a set of tests. Each test consists of two sequences: input events and output actions. In previous works output actions were discrete, i.e. selected from the finite set, in this work output actions can also be continuous, i.e. represented by real numbers. Only the structure of automaton transitions graph is evolved by the genetic algorithm. Values of output actions are found using transition labeling algorithm, which aim is to maximize the value of fitness function. New transition labeling algorithm is proposed. It also works with continuous output actions and is based on equations system solving. In case of proper selection of fitness function, equations in this system are linear and it can be solved by the Gaussian elimination method. The unmanned airplane performing the loop is considered as an example of the controlled object.

Categories and Subject Descriptors
I.2.2 [**Artificial Intelligence**]: Automatic Programming – p*rogram synthesis.*

General Terms
Algorithms.

Keywords
Genetic Programming, Finite Automaton, Finite Automaton Induction, Continuous Output Actions.

1. INTRODUCTION
In the context of automata-based programming [1, 2] the behavior of software systems is described with so called automated controlled objects. Each automated controlled object consists of a controlled object and a finite automaton. The automaton takes events and variables as an input and outputs so called output actions for the controlled object. For many problems finite automata can be built manually, but there are problems for which manual construction is very difficult. Examples of such problems are "Artificial ant" [3, 4] and unmanned aircraft control [5].

GECCO'11, July 12–16, 2011, Dublin, Ireland.

There are several approaches to the latter problem. One of them is the induction of "ideal" trajectory from several flights performed by a human pilot [6]. Other approach is the usage of genetic algorithms for finite automata design [7–11]. In the paper [12] genetic algorithm is used for generation of top-level finite automaton for the unmanned aircraft control. The fitness function in [12] is calculated with behavior modeling of the unmanned aircraft in the environment. That modeling is a very time-consuming process and the fitness function evaluation for one automaton takes about 5 minutes. Therefore, all the process of evolving automaton may take several days or even months.

The goal of this paper is to describe the genetic algorithm for finite automaton induction which does not use modeling for the fitness function evaluation. It is proposed to use tests to describe the behavior of the controlled object. Tests are recorded during the control by a human. If enough tests are given as an input then it is expected that inaccuracies of the human control can be eliminated. This approach extends the approach described in paper [13] (only discrete output actions are considered there). In this paper output actions can also be continuous (represented by real numbers).

2. PROBLEM DEFINITION
The input data for the genetic algorithm is a set of tests which structure is described in the section 2.2. The goal of the genetic algorithm is to construct the finite automaton which behavior on these tests is as close as possible to the desired one.

2.1 Controlled Object
Controlled object is characterized by the set of its state parameters and the set of its controls. State parameters are called *input parameters* later on. E.g., if the controlled object is an airplane, one of the state parameters is its altitude. Parameters associated with controls are called *control parameters*. E.g., if the controlled object is also an airplane, one of the controls is the starter, another one is the control column. An example of the control parameter is the aileron angle, another one is the elevator angle. Some of the control parameters are discrete, i.e. their values are selected from some finite set (corresponding controls are called *discrete controls*), while other control parameters are continuous, i.e. their values are real numbers (corresponding controls are called *continuous controls*). Output actions changing discrete parameters are called *discrete actions*, and changing continuous parameters are called *continuous actions*. At each moment of time the value of the control parameter is the cumulative value of corresponding actions for previous moments of time.

A continuous action changes the control parameter by some value and a discrete action sets the parameter to some value. Note that consecutive continuous actions are equivalent to the sum of these actions and consecutive discrete actions are equivalent to the last one. E.g., one of the continuous control parameters for the unmanned airplane is the aileron angle. The aileron rotation by some angle is a continuous action for this control. So, consecutive rotations by angles x and y are equivalent to one rotation by the angle $x+y$. An example of the discrete control is the starter. There are two discrete actions for it: "turn on" and "turn off". A sequence of these actions is equivalent to the last of them.

2.2 Training data

Each test T consists of two parts: T.in and T.ans (see Figure 1).

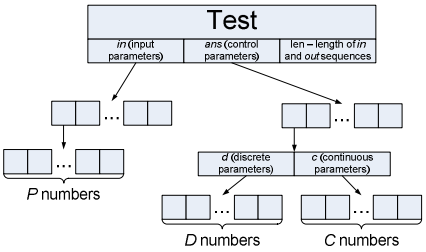

Figure 1. Test structure.

Each of these two parts is a sequence of length T.len: first of them contains values of input parameters and the second one contains reference values of control parameters recorded during human control. Each element T.in[t] of the input sequence contains P numbers: values of parameters at the moment of time t. Each element T.ans[t] contains two collections of numbers: T.ans[t].d and T.ans[t].c. Collection T.ans[t].d contains values of D discrete parameters and T.ans[t].c – values of C continuous parameters. A sequence of control parameters produced by a finite automaton on test T will be denoted as T.out. Its structure is the same as structure of T.ans. The test number i will be denoted by $T[i]$ later on. Total number of tests will be denoted by N.

2.3 Automaton-controlled object interaction

First of all, we define *a predicate*. A predicate is a statement about the controlled object state that can be written as mathematic formula. That formula can depend not only on input parameters at the current moment of time, but also on parameters of states in previous moments of time. E.g., the predicate can be "the airplane is descending", that is, current airplane altitude is strictly less than the previous one. The list of predicates that can be used is automaton is composed manually before running the genetic algorithm and is not changed during the run.

The main scheme of the interaction is shown on Figure 2.

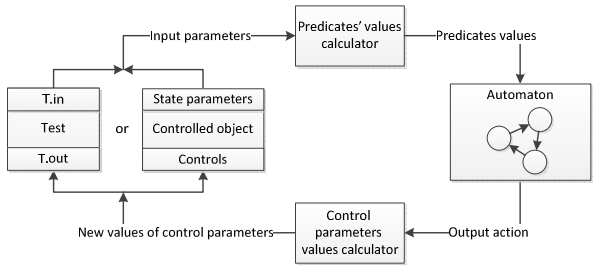

Figure 2. Automaton and controlled object communications.

In the beginning of each time step "Predicates' values calculator" receives values of each input parameter and calculates values of each predicate. Values of these predicates are processed by automaton in ascending order of their numbers. After getting a sequence of actions from the automaton, one resulting action is calculated. This action is applied to control parameters and their new values are sent to the controlled object.

3. Genetic algorithm

The proposed method differs from the classical genetic algorithm by an additional step which is done before the fitness function calculation – the transition labeling algorithm. This algorithm is similar to the one proposed in [14] and [15], but it works not only for discrete actions but for continuous as well.

3.1 Individual Representation

A finite automaton is represented as an object containing descriptions of transitions for each of the states and the initial state. The number of states is fixed and is the same for all finite automata generated during one run of the algorithm. The guard condition is specified for each transition. This condition has one of two forms: either "x_i" or "$\neg x_i$" where x_i is the i-th predicate. Output actions are not specified on transitions, that is, *an individual is just a "skeleton" of a finite automaton*. Concrete actions are to be determined by the transition labeling algorithm. An output action on one transition consists of a D–tuple of actions on all discrete controls and a C–tuple of actions on all continuous controls. Skeleton is represented with the full transition table: for each state and each value of each predicate (value can be "true" of "false") there can be a transition.

3.2 Fitness Function

A fitness function represents how close the behavior of the inducted automaton is to the desired one. It is calculated using the following formula:

$$Fitness = 1 - \sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{\rho(T[i].out, T[i].ans)^2}{\rho(T[i].ans, 0)^2}}$$

To calculate it, each of the N input sequences $T[i]$.in is given to the automaton as an input and corresponding output sequences $T[i]$.out are recorded. In the formula above 0 is a sequence of zero controls and ρ(out, ans) is a denotation of a distance between sequences of control parameters, which is calculated using the following formula:

$$\rho(out, ans) = \sqrt{\sum_{t=1}^{T.len}\left(\sum_{k=1}^{D}\left[out[t].d[k] \neq ans[t].d[k]\right] + \sum_{k=1}^{C}\left(out[t].c[k] - ans[t].c[k]\right)^2\right)}$$

Here D stands for the number of discrete controls, C – continuous.

As mentioned above, before calculating the fitness function the transition labeling algorithm is applied to an individual. Its goal is to find values of output actions leading to the maximal possible value of the fitness function for the "skeleton" represented by the given individual.

3.3 Transition Labeling Algorithm

First of all, for the purpose of the transition labeling all transitions of the automaton that is launched on all tests are recorded. To maximize the fitness function it is sufficient to minimize the following sum:

$$\sum_{i=1}^{N}\frac{\rho(T[i].out, T[i].ans)^2}{\rho(T[i].ans, 0)^2}$$

Sums corresponding to different control parameters can be minimized independently. Therefore, we need to minimize the following sum:

$$\sum_{i=1}^{N}\frac{\sum_{t=1}^{T[i].len}\left[T[i].out[t].d[k]\neq T[i].ans[t].d[k]\right]}{\rho(T[i].ans,0)^2}$$

for each k in the range [1 .. D] and the sum:

$$\sum_{i=1}^{N}\frac{\sum_{t=1}^{T[i].len}\left(T[i].out[t].c[m]-T[i].ans[t].c[m]\right)^2}{\rho(T[i].ans,0)^2}$$

for each m in the range [1 .. C].

To find the values of discrete parameters we can rewrite the first expression in the following way:

$$\sum_{i=1}^{N}\sum_{j=1}^{n}\sum_{t\in Time_{i,j}}\frac{\left[T[i].out[t].d[k]\neq T[i].ans[t].d[k]\right]}{\rho(T[i].ans,0)^2}=$$
$$=\sum_{j=1}^{n}\sum_{i=1}^{N}\sum_{t\in Time_{i,j}}\frac{\left[T[i].out[t].d[k]\neq T[i].ans[t].d[k]\right]}{\rho(T[i].ans,0)^2}$$

by grouping items corresponding to each transition. Here $Time_{i,j}$ consists of such values of t, that the last transition performed by the automaton on the test i equals to j, and n stands for the number of transitions in the automaton. This transformation implies that we can minimize sums $\sum_{i=1}^{N}\sum_{t\in Time_{i,j}}\frac{\left[T[i].out[t].d[k]\neq T[i].ans[t].d[k]\right]}{\rho(T[i].ans,0)^2}$ for each j in the range [1 .. n].

Now notice that $T[i]$.out[t].d[k] equals to the value of the k-th discrete parameter on the j-th transition when t belongs to $Time_{i,j}$.

Let's denote it by u and group items with the same value of $T[i]$.ans[t].d[k]. These transformations give us: $\sum_{h=1}^{G}\sum_{i=1}^{N}\sum_{t\in TimeV_{i,j,h}}\frac{\left[u\neq T[i].ans[t].d[k]\right]}{\rho(T[i].ans,0)^2}=\sum_{h=1}^{G}\sum_{i=1}^{N}\frac{\left[u\neq v_h\right]}{\rho(T[i].ans,0)^2}\sum_{t\in TimeV_{i,j,h}}1=$

$$=\sum_{h=1}^{G}\sum_{i=1}^{N}\frac{\left[u\neq v_h\right]}{\rho(T[i].ans,0)^2}\left|TimeV_{i,j,h}\right|=\sum_{h=1}^{G}\left[u\neq v_h\right]\sum_{i=1}^{N}\frac{\left|TimeV_{i,j,h}\right|}{\rho(T[i].ans,0)^2}.$$

Here v_h stands for the value of the h-th discrete parameter, G is the number of possible values of this parameter, $TimeV_{i,j,h}$ consists of such values of t from $Time_{i,j}$ that $T[i]$.ans[t].d[k] equals to v_h.

Let's denote the chosen value of the k-th discrete parameter on the j-th transition by v_m. This gives us:

$$\sum_{h=1}^{G}\left[v_m\neq v_h\right]\sum_{i=1}^{N}\frac{\left|TimeV_{i,j,h}\right|}{\rho(T[i].ans,0)^2}=$$
$$=\sum_{h=1}^{G}\sum_{i=1}^{N}\frac{\left|TimeV_{i,j,h}\right|}{\rho(T[i].ans,0)^2}-\sum_{i=1}^{N}\frac{\left|TimeV_{i,j,m}\right|}{\rho(T[i].ans,0)^2}=$$
$$=\sum_{i=1}^{N}\frac{\left|Time_{i,j}\right|}{\rho(T[i].ans,0)^2}-\sum_{i=1}^{N}\frac{\left|TimeV_{i,j,m}\right|}{\rho(T[i].ans,0)^2}.$$

Thus, to minimize this expression we should choose m equal to $\arg\max_h\sum_{i=1}^{N}\frac{\left|TimeV_{i,j,h}\right|}{\rho(T[i].ans,0)^2}$.

To find values of continuous parameters firstly notice that the value of the m-th continuous parameter at the moment t (which is $T[i]$.out[t].c[m]) equals to the sum of the parameter initial value (that is 0) and amounts of its changes on all transitions performed before this moment, that is, $T[i].out[t].c[m]=\sum_{j=1}^{n}\alpha_{i,j}[t]u_j$. Here u_j stands for the amount of change of the m-th continuous parameter on the j-th transition, and $\alpha_{i,j}[t]$ is the number of times the j-th transitions had been performed before the time t on the test i. So, we should minimize the following sum:

$$S=\sum_{i=1}^{N}\frac{\sum_{t=1}^{T[i].len}\left(\sum_{j=1}^{n}\alpha_{i,j}[t]u_j-T[i].ans[t].c[m]\right)^2}{\rho(T[i].ans,0)^2}$$

for each m in the range [1 .. C].

The partial derivative of S with respect to u_h looks as follows: $S'_{u_h}=\sum_{i=1}^{N}\frac{1}{\rho(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}2\alpha_{i,h}[t]\left(\sum_{j=1}^{n}\alpha_{i,j}[t]u_j-T[i].ans[t].c[m]\right)$.

After equating each of the derivatives to zero we get the following system of equations:

$$\begin{cases}\sum_{j=1}^{n}\left(\sum_{i=1}^{N}\frac{1}{\rho(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}\alpha_{i,1}[t]\alpha_{i,j}[t]\right)u_j=\sum_{i=1}^{N}\frac{1}{\rho(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}T[i].ans[t].c[m]\alpha_{i,1}[t];\\ \sum_{j=1}^{n}\left(\sum_{i=1}^{N}\frac{1}{\rho(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}\alpha_{i,2}[t]\alpha_{i,j}[t]\right)u_j=\sum_{i=1}^{N}\frac{1}{\rho(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}T[i].ans[t].c[m]\alpha_{i,2}[t];\\ \ldots\\ \sum_{j=1}^{n}\left(\sum_{i=1}^{N}\frac{1}{\rho(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}\alpha_{i,n}[t]\alpha_{i,j}[t]\right)u_j=\sum_{i=1}^{N}\frac{1}{\rho(T[i].ans,0)^2}\sum_{t=1}^{T[i].len}T[i].ans[t].c[m]\alpha_{i,n}[t].\end{cases}$$

It is necessary to notice that this system is linear which makes it easy to solve. This linearity is caused by the structure of the chosen fitness function.

4. Experiments

In order to check the efficiency of the method proposed the problem of generating an automaton to control the plane performing the loop trick is used. *FlightGear* airplane simulator (http://flightgear.org) is used to model the airplane being controlled. This software allows automated control of the airplane as well as manual and it can record control parameters as well as parameters of the flight. The problem is to generate an automaton that can be used to control the plane performing the loop and then continuing to fly ahead.

4.1 Usage of the method proposed

The following steps were performed:

- a set of predicates to describe the plane state was created;
- three independently used sets of tests were recorded: each set consisted of 10 tests; each test consisted of a few thousands input-output parameters sets; recording of parameters was performing 10 times per second;
- the algorithm was run for several tests sets and several algorithm parameters: population size – 100 individuals; the automaton contained not less than 2 and not more than 5 states; mutation probability was 0.5 and 0.1; selection strategy – tournament selection, crossover – generic automaton crossover, mutation – generic automaton mutation; elite size – 2 individuals;

All runs were performed on one core of *Intel Core 2 Duo T7250* processor of the computer with OS *Microsoft Windows XP*. The average running time of the algorithm was about 10 hours. This time was sufficient to generate about 2000 generations. That implies that the average individual processing time was about 0.2 sec. This is significally less than in paper [12].

The chosen predicates were: x_0 – the engine is turned on; x_1 – speeding-up of changing direction of plane moving is greater than zero; x_2 – speed of changing direction of plane moving is greater than zero; x_3 – the value of deviation from the initial direction is less than 1 degree; x_4 – the value of deviation from the initial direction is greater than zero;; x_5 – speeding-up of changing of plane heeling is greater than zero; x_6 – speed of the changing of plane heeling is greater than zero; x_7 – plane heeling is small (less than 1 degree); x_8 – plane heeling is positive; x_9 – speeding-up of changing the vertical speed of the plane is greater than zero; x_{10} – speed of changing the vertical speed of the plane is greater than zero; x_{11} – vertical speed is small (less than 0.1 m/sec) x_{12} – vertical speed is positive. The list of controls included magneto, starter, throttle, ailerons, elevator and rudder. While the first two controls were discrete, the others were continuous.

The results of running algorithms with different parameters showed that automata with 3-4 states were rather good and that more states automata had, the less understandable became its structure and also its behavior became worse.

4.2 Results

Genetic algorithm was run about 50 times and the best automaton was recorded for each run. These automata were analyzed by authors by watching its flight, and the best one was chosen. This automaton has 4 states and 68 transitions. During analyzing this automaton authors noticed that depending on some external conditions automaton performed the loop in three different ways. In the most common case the plane performed exactly one loop and then continue flying smoothly. Sometimes the plane performed two loops one by one. It was explained by the fact that the environment state after the first loop could be so close to the one in the beginning of the test that the automaton couldn't distinguish them. It also seemed possible for the automaton to perform more than two loops, but authors had never seen this case. Sometimes the plane failed to perform the loop but this behavior was very rare. Determing the exact conditions that imply the behavior of the plane is to be investigated.

5. CONCLUSION

A genetic algorithm for induction of finite automata with continuous and discrete output actions was proposed. This method was successfully tested on the real problem. Results showed that the generated automaton can provide better behavior than human. As compared with [12], this method provides much faster way to calculate the fitness function. Besides this advantage, the fitness function used in this method is much more general than the one used in [13]. Therefore, there is no need to modify when changing the controlled object.

The research was supported by Ministry of Education and Science of Russian Federation in the context of Federal Program "Scientific and pedagogical personnel of innovative Russia".

6. REFERENCES

[1] Polikarpova, N., Shalyto, A. *Automata-based programming*. Piter, 2009 (in Russian).

[2] Shalyto, A. *Logic Control and Reactive Systems: Algorithmization and Programming*. Automation and Remote Control, Vol. 62, No. 1, 2001, pp. 1–29.

[3] Angeline, P., Pollack, J. Evolutionary Module Acquisition. *Proceedings of the Second Annual Conference on Evolutionary Programming*. Cambridge: MIT Press. 1993. P.154-163.

[4] Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C., Wang, A. The Genesys System: Evolution as a Theme in Artificial Life. *Proceedings of Second Conference on Artificial Life*. MA: Addison-Wesley. 1992. P. 549–578.

[5] Paraschenko, D., Shalyto, A., Tsarev, F. Modeling Technology for One Class of Multi-Agent Systems with Automata Based Programming. *Proceedings of 2006 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA-2006)*. La Coruna. Spain. 2006, pp.15– 20.

[6] Coates, A., Abbeel, P., Ng, A. Y. Learning for Control from Multiple Demonstrations. *Proceedings of the 25th International Conference on Machine Learning*. Helsinki: 2008. P.144 – 151.

[7] Gladkov, L. A., Kureichik, V. V., Kureichik, V. M. *Genetic Algorithms*. Moscow. Fizmatlit, 2006.

[8] Russel, S., Norvig, P. *Artificial Intellingence: A Modern Approach*. Prentice Hall, 2009.

[9] Koza, J. R. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, 1992.

[10] Kureichik, V. M. *Genetic Algorithms: State of the Art, Problems, and Perspectives*. Journal of Computer and Systems Sciences International, 1999. Vol. 38, № 1, pp. 137–157.

[11] Kureichik, V. M., Rodzin S. I. *Evolutionary Algorithms: Genetic Programming*. Journal of Computer and Systems Sciences International, 2002. Vol. 41, № 1, pp. 123–132.

[12] Polikarpova, N., Tochilin, V., Shalyto A. *Method of Reduced Tables for Generation of Automata with a Large Number of Input Variables Based on Genetic Programming*. Journal of Computer and Systems Sciences International, 2010. Vol. 49, № 2, pp. 265–282.

[13] Tsarev, F. *Method of finite state machine induction from tests with genetic programming*. Information and Control Systems (Informatsionno-upravljajuschie sistemy, in Russian). 2010. № 5, c. 31–36.

[14] Lucas, S., Reynolds, J. Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm // *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 27, №7, 2005, pp. 1063–1074.

[15] Lucas, S. Evolving Finite-State Transducers: Some Initial Explorations. *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. Volume 2610/2003, pp. 241–257