Improving the Quality of Supervised Finite-State Machine Construction Using Real-Valued Variables

Igor Buzhinsky, Daniil Chivilikhin, Vladimir Ulyantsev, Fedor Tsarev
ITMO University
Computer Technologies Laboratory
Saint Petersburg, Russia
{buzhinsky, chivdan, ulyantsev, tsarev}@rain.ifmo.ru

ABSTRACT

The use of finite-state machines (FSMs) is a reliable choice for control system design since they can be formally verified. In this paper a problem of constructing FSMs with real-valued input and control parameters is considered. It is supposed that a set of human-created behavior examples, or tests, is available. One of the earlier approaches for solving the problem suggested using genetic algorithms together with a transition labeling algorithm. This paper improves this approach via the use of real-valued variables which are calculated using the FSM's input data. FSMs with real-valued variables are represented as systems of linear controllers. The new approach allows to synthesize FSMs of better quality than it was possible earlier.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program synthesis*

Keywords

Finite-State Machine; Finite-State Machine Construction; Ant Colony Optimization

1. INTRODUCTION

In this paper we deal with the problem of test-based finite-state machine (FSM) [10] construction for controlling objects in complex environments. The use of behavior examples, or tests, is reasonable when it is difficult to formalize the desired system behavior. Finite-state machines are widely used in the development of reactive systems [8,9] and can be easily verified using the *Model Checking* [6] approach, which allows to check whether some temporal properties are satisfied for them. The ease of FSM verification, which is the main advantage of their application, contributes to their ability to be the components of reliable software. FSMs can also be used as models of existing software systems [16].

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.

http://dx.doi.org/10.1145/2598394.2605679.

Manual FSM construction is usually hard. For example, the optimal FSM for the Artificial Ant problem [11] was found only using automated FSM synthesis [15] with genetic algorithms [11]. Genetic algorithms and other search optimization techniques require a fitness function, a measure of the solution quality, to be defined. There are two approaches for FSM fitness function definition. One approach, which often leads to huge time requirements, is to model FSM execution in a simulated environment [14]. In [2], another approach, which suggested using a computationally cheap test-based fitness function, was applied for searching FSMs capable of controlling an aircraft model. FSMs in [2] were equipped with real-valued output actions which were assigned automatically given the transition function of an FSM. This approach is model-free and thus is applicable for various controlled objects.

Model-based approaches are also known for similar problems. One of such approaches [1], which does not use FSMs and is mostly based on machine learning techniques, was developed for the autonomous helicopter control and suggested finding a hidden trajectory which is implicitly encoded in tests. Another approach [13] allows to create correct-by-design finite-state controllers.

In this paper the approach suggested in [2] is improved. The same controlled object as in [2], an aircraft model, is used. This paper suggests a way to construct FSMs with better correspondence to tests, which remains computationally inexpensive and model-free. A key feature of the proposed method is the use of so-called real-valued variables as inputs for FSMs.

In section 2 we formally describe the problem being solved. Section 3 describes the FSM construction method used in this paper. Finally, in section 4 the experimental evaluation of the method is presented.

2. PROBLEM STATEMENT

An FSM is a sextuple $(S, s_0, E, A, \delta, \lambda)$ where S is a finite set of states, $s_0 \in S$ is a start state, E is a set of input events, A is a set of output actions, $\delta : S \times E \to S$ is a transition function, and λ is an output function ($\lambda : S \times E \to A$ for Mealy machines and $\lambda : S \to A$ for Moore machines). An FSM execution is a sequence of cycles: on each cycle the FSM receives an input event, generates an output action according to λ and changes its active state according to δ. In this paper the cycles are equally spaced in time with an interval of 0.1 seconds. The way how FSMs deal with real-valued inputs and outputs will be explained later.

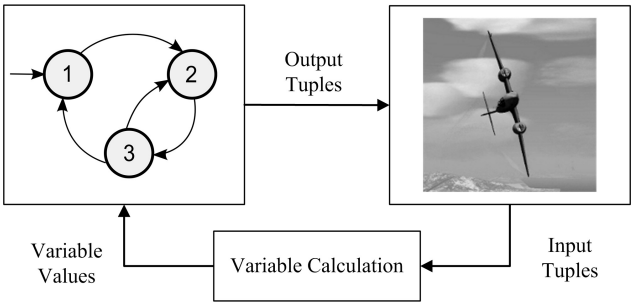

Figure 1: Aircraft control procedure

Consider a finite set of tests which describe the proper behavior of the controlled object. Informally, a test consists of the input data describing the object's state at different timestamps and the output data which shows how the object should be controlled. Tests are obtained from a human expert. The problem is to construct an FSM with a behavior close to the one recorded in the tests.

From now on, we will focus on the unmanned aircraft control problem. The *FlightGear* flight simulator (`http://www.flightgear.org/`) is used for test recording and FSM execution. The input part of the tests consists of flight parameters: altitude, airspeed, etc. The output part is formed by aircraft control device (elevator, ailerons, etc.) positions characterized by real values, which will be referred to as control parameters.

2.1 Control Procedure

An *input tuple* is a sequence of P real numbers $(i_1, ..., i_P)$. Applied to the problem under consideration, each number corresponds to one of the flight parameters of the unmanned aircraft. The controlled object also has C control parameters. They are of two types: discrete (e.g. starter position), which have finite domains, and continuous, or real-valued ones (e.g. rudder position). For instance, the rudder position from the leftmost to the rightmost can be described by values from $[-1, 1]$. From now on, we will assume that all control parameters are continuous.

An *output tuple* $(o_1, ..., o_C)$ is a snapshot of all C control parameters at some timestamp. As the control parameters are bounded, for each parameter i and for each output tuple o there exist $c_i^{\min}$ and $c_i^{\max}$ such that $c_i^{\min} \le o_i \le c_i^{\max}$.

At the beginning of each cycle the FSM receives an input tuple. After that, *variable* values are calculated. Variables are functions of input tuples and their history. For instance, a variable can transform the aircraft's altitude to its derivative or to its square, or it can just return the altitude untransformed. A specific class of variables, predicates, have Boolean domain. The statement "altitude is decreasing" is an example of a predicate. Variable values determine the FSM's output actions on the current cycle and its new state. The scheme of the control procedure is shown in Fig. 1.

2.2 Tests

Tests describe tasks (e.g. aerobatic figures) an unmanned aircraft is desired to perform. Denote the length of the i-th test in a test set ($1 \le i \le N$, where N is the number of tests) as len[i]. It is equal to the number of timestamps recorded in a test. The difference between the recorded timestamps is equal to the cycle length (0.1 s). Test i is formed by two sequences. The input sequence, in[i], consists of input tuples in$[i, t]$, where t ($1 \le t \le$ len[i]) is the time in cycles. The output sequence, out[i], consists of output tuples out$[i, t]$.

3. PROPOSED APPROACH

In this section the proposed improvement of the method [2] is described. The general scheme of the used approach does not differ significantly from the one from [2]: FSM search is based on metaheuristics, and the individuals are FSMs with undefined output functions, or *skeletons*. The output function is derived automatically for each individual.

The main novelty of the proposed approach involves representing FSMs as systems of linear controllers. The fitness function was also modified to simplify the search of visually simpler FSMs with clearly distinct states. Instead of a genetic algorithm applied in [2], we use an ant colony optimization (ACO) [7] algorithm suggested in [5]. It has some benefits in computational effort on the considered problem [3] and does not require to define a crossover operator.

3.1 FSM Representation

The previous approach [2] for the problem considered in this paper only allowed to construct FSMs which used predicates, or Boolean variables, for output action generation. This led to the impossibility of inferring smooth control laws. The presence of the issue has been tested experimentally: the aircraft model under control of such FSMs was unable to perform a 180° turn in the horizontal plane.

We use a new method of FSM representation which involves both predicates and real-valued variables. Predicates are only used as transition conditions. Consider a set of predicates $p_1, ..., p_m$. In each state only several of them are *significant*: for each state $s \in S$ and a combination of significant predicate values in s, a transition is defined. To encode such data in individuals, a Boolean mask defining which predicates are significant, and a transition table are stored for each state (assume that $|S|$ is fixed during optimization). This *reduced table* approach was suggested in [14].

Oppositely, real-valued variables are only used for output generation. For each control parameter j consider a set of real-valued variables $v_{j,1}, ..., v_{j,n_j}$. Each state $s \in S$ defines a controller for each control parameter, which uses the corresponding variable values as inputs. The controllers were chosen to be linear to make the automatic derivation of the output function simple. Assume u_j is the j-th control parameter value and s is the current state, then on each cycle the change of u_j is calculated according to the formula:

$$\Delta u_j = \sum_{i=1}^{n_j} r_{s,i,j} v_{j,i}. \quad (1)$$

Real-valued variables, as well as predicates, are to be defined before the optimization, but the numbers $r_{s,i,j}$ are chosen automatically. Firstly, some of the variables are actually not used in some states (for each control parameter a significance mask for real-valued variables is also stored), so the corresponding $r_{s,i,j}$ are equal to zero. The use of significance masks for both types of variables makes the proposed FSM representation scalable for large values of m and $n_1, ..., n_C$. The remaining coefficients are set to maximize the fitness function for a given skeleton.

An example of a transition diagram of a three-state FSM (in case of a single control parameter u) is shown in Fig. 2.

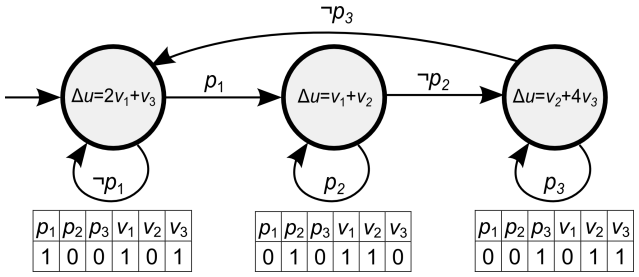

Figure 2: A transition diagram of an FSM

3.2 Fitness Function

Assume ans[i] is the output tuple sequence generated by a fixed FSM in response to in[i]. To measure FSM behavior quality on a single test, the following distance between the output sequences is considered:

$$\rho\left(\text{ans}[i], \text{out}[i]\right) = \sqrt{\frac{1}{\text{len}[i]} \sum_{t=1}^{\text{len}[i]} \frac{1}{C} \sum_{j=1}^{C} \left(\frac{\text{ans}[i,t]_j - \text{out}[i,t]_j}{c_j^{\max} - c_j^{\min}} \right)^2}$$

Another measure we found to be essential is the number of times an FSM changes its state during the execution on tests. When this number is large, it is usually difficult to assign intuitive meanings to different states. The values $\max(\tau_i - |S| + 1, 0)$, where τ_i is the number of state changes on test i, can serve as additional penalties. The described distances and penalties for different tests can be combined:

$$P_\rho = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \rho^2(\text{ans}[i], \text{out}[i])},$$

$$P_\tau = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(\max(\tau_i - |S| + 1, 0)\right)^2}.$$

Finally, the fitness function f is defined in the following way:

$$f = 1 - P_\rho - K \cdot P_\tau,$$

where K is a small number (we used $K = 0.00015$). A fitness function similar to f, but which did not include the transition penalties P_τ, was used in [2].

A so-called transition labeling algorithm was used in [2] to automatically derive the output functions of FSMs to maximize the fitness function. A similar approach was developed for the presented FSM representation. We omit the math, which partly repeats the one from [2], and only present the result: the problem of maximizing f is reduced to solving C systems of linear equations. To compute a single system's matrix, $O\left(M_j^2 \sum_{i=1}^{N} \text{len}[i]\right)$ time is required, where M_j is the number of non-zero values $r_{s,i,j}$ (for a fixed j). Then each system can be solved in $O(M_j^3)$ time. The linearity of the resulting systems comes from the linearity of (1).

4. EXPERIMENTAL EVALUATION

The experiments, which were performed to evaluate the proposed FSM representation, included running FSM construction with two FSM representations (the one from [2] and the proposed one) and measuring the solution quality in *FlightGear*. This evaluation is an extended version of the one presented in [4]. To perform the evaluation, three test sets were recorded in *FlightGear*. They included a loop (33 tests), a barrel roll (28 tests) and a banked 180° turn in the horizontal plane (19 tests). Fig. 3 shows several examples of paths performed by the aircraft during test recording.

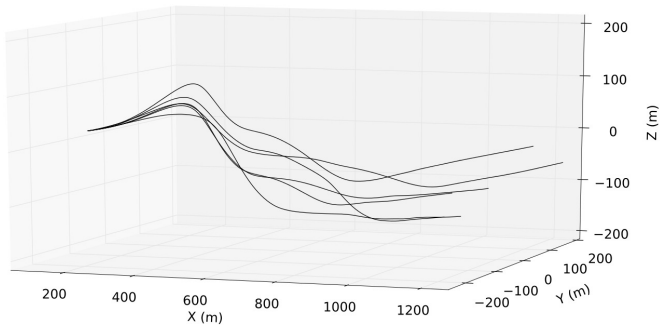

Figure 3: Several aircraft paths from the barrel roll test set

Table 1: Parameter values tuned with *irace*

FSM representation	N_{ants}	N_{mut}	n_{stag}	p_{new}
Proposed in this paper	5	13	55	0.1428
Previous one	2	29	8	0.2471

The penalties P_τ were disabled ($K = 0$) while constructing FSMs with the previous representation to make f similar to the one from [2]. For each test set a predicate set was chosen for the previous FSM representation, and both predicate and variable sets were chosen for the proposed representation. The variable choice process was iterative: if the method did not yield an FSM with proper quality (measured in simulation), the variable set was adjusted. We failed to come up with a good predicate set to construct an FSM performing the 180° turn with the previous representation, which led to poor simulation results for this case.

For each of three test sets, for $|S| = 3, 4, 5$ and for each FSM representation 50 ACO executions were performed. Pheromone-based edge selection was changed to uniformly random, as pheromone did not influence the algorithm's performance in a measurable way. The number of ants N_{ants} and other pheromone non-related parameters N_{mut}, n_{stag}, p_{new} (see [5]) were tuned with the *irace* [12] package (1000 ACO runs were performed for each representation, see Table 1 for the tuned values). The termination criterion was stagnation during 5000 fitness evaluations. In Table 2 (columns 3–5) the median fitness values reached on different problem instances are shown. The table shows that the use of the proposed FSM representation slightly improves the fitness value on the loop and barrel roll test sets, and retains the fitness value nearly the same on the turn test set. The time required for a single ACO run often did not exceed 10 minutes on a personal computer with a quad-core *Intel Core i7–2670QM* 2.2GHz processor. These performance results are close to the ones from [3].

The constructed FSMs were also examined in computer simulation. Two quality criteria measured the average pitch and roll deviations from the values recorded in the tests. The pitch criterion value was equal to the average distance $|\alpha_{i,t}^{\text{test}} - \alpha_{j,t}^{\text{run}}|$ between the pitch angles at timestamps t of the i-th test ($\alpha_{i,t}^{\text{test}}$) and of the j-th FSM execution in simulation

Table 2: FSM quality metric values on different problem instances

$\|S\|$	FSM Representation	Median fitness values			Median roll errors (°)			Median pitch errors (°)		
		Loop	Barrel roll	Turn	Loop	Barrel roll	Turn	Loop	Barrel roll	Turn
3	Proposed in this paper	0.9856	0.9854	0.9892	1.71	16.52	4.80	17.21	3.20	1.95
	Previous one	0.9812	0.9832	0.9894	6.37	18.56	50.29	20.54	4.44	7.58
4	Proposed in this paper	0.9866	0.9863	0.9898	2.41	15.35	4.10	23.04	2.51	1.42
	Previous one	0.9836	0.9856	0.9901	6.32	21.86	57.04	22.11	4.08	6.79
5	Proposed in this paper	0.9873	0.9868	0.9901	3.21	14.74	4.07	25.27	2.43	1.36
	Previous one	0.9842	0.9858	0.9902	9.54	22.99	45.83	24.44	4.68	7.83

($\alpha^{\text{run}}_{j,t}$), where $i = 1..N, t = 1..\text{len}[i]$ and $j = 1..10$. A similar criterion was used for the roll angle. For each problem instance, 50 FSMs with the best fitness values (each FSM from a separate run) were examined in simulation. Table 2 (columns 6–11) shows the median quality criteria values for the described FSM groups. It implies that the quality values are generally better for the suggested FSM representation.

Rather high pitch and roll error values for the 180° turn performed by FSMs with the previous representation signify their poor quality. None of such FSMs were able to perform the turn from the beginning to the end. The situation is different for the suggested FSM representation.

A video record of the Gloster Meteor jet fighter performing the turn is available at `http://youtu.be/n9q5FmCYs6M`.

5. CONCLUSION

An earlier developed method [2] of supervised finite-state machine construction has been improved. The improvement allows to synthesize FSMs which use real-valued variables to form output actions. The benefits of the suggested approach were shown on three test sets for the unmanned aircraft control problem. The use of the modified method improves the quality of generated solutions. The failure of the original method on one of the test sets is no longer the case for the new FSM representation.

6. ACKNOWLEDGEMENTS

This work was financially supported by the Government of Russian Federation, Grant 074-U01, and by RFBR, research project No. 14-07-31244 mol_a.

7. REFERENCES

[1] P. Abbeel, A. Coates, and A. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[2] A. Alexandrov, A. Sergushichev, S. Kazakov, and F. Tsarev. Genetic algorithm for induction of finite automata with continuous and discrete output actions. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation (GECCO '11)*, pages 775–778. ACM, 2011.

[3] I. Buzhinsky, V. Ulyantsev, and A. Shalyto. Test-based induction of finite-state machines with continuous output actions. In *Proceedings of the 7th IFAC Conference on Manufacturing Modelling, Management, and Control (MIM '13)*, pages 1049–1054. IFAC, 2013.

[4] I. Buzhinsky, V. Ulyantsev, F. Tsarev, and A. Shalyto. Search-based construction of finite-state machines with real-valued actions: New representation model. In *Genetic and Evolutionary Computation Conference (GECCO '13) Companion*, pages 199–200. ACM, 2013.

[5] D. Chivilikhin and V. Ulyantsev. Muacosm – a new mutation-based ant colony optimization algorithm for learning finite-state machines. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '13)*, pages 511–518. ACM, 2013.

[6] E. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT press, 1999.

[7] M. Dorigo and T. Stutzle. *Ant Colony Optimization*. MIT Press, US, 2004.

[8] D. Harel and A. Pnueli. On the development of reactive systems. *Logic and Models of Concurrent Systems*, pages 477–498, 1985.

[9] D. Harel and M. Politi. *Modeling Reactive Systems with Statechart. The Statemate Approach*. McGraw-Hill, NY, 1998.

[10] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction To Automata Theory, Languages, and Computation (3rd Edition)*. Prentice Hall, 2006.

[11] J. Koza. *Genetic programming: on the programming of computers by natural selection*. MIT Press, Cambridge, MA, USA, 1992.

[12] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université libre de Bruxelles, Belgium, 2011.

[13] M. Mazo, A. Davitian, and P. Tabuada. Pessoa: A tool for embedded control software synthesis. *Lecture Notes in Computer Science*, 6174:566–569, 2010.

[14] N. Polikarpova, V. Tochilin, and A. Shalyto. 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*, 49(2):265–282, 2010.

[15] F. Tsarev and A. Shalyto. Use of genetic programming for finite-state machine generation in the smart ant problem. In *Proceedings of the IV International Scientic-Practical Conference "Integrated Models and Soft Computing in Artificial Intelligence"*, pages 590–597, 2007.

[16] N. Walkinshaw, R. Taylor, and J. Derrick. Inferring extended finite state machine models from software executions. In *Proceedings of the 20th Working Conference on Reverse Engineering (WCRE '13)*, pages 301–310. IEEE Computer Society Press, 2013.