Automata Construct with Genetic Algorithm

Vít Fábera

Department of Informatics and Telecommunication, Faculty of Transportation Sciences, Czech Technical University, Konviktská 20, 110 00 Praha 1, Czech Republic, e-mail: fabera@fd.cvut.cz

Vlastimil Jáneš

Department of Control and Telematics, Faculty of Transportation Sciences, Czech Technical University, Konviktská 20, 110 00 Praha 1, Czech Republic, e-mail: janes@fd.cvut.cz

Abstract: The first step is to create an automaton model in a process of digital hardware design. The automaton model is an input to software synthesis tools and is created by an engineer. The genetic programming as a part of artificial intelligence tries to create programs using evolutionary principles. Similar approach can be used in a hardware design – to create automatically an automaton model by this technique. The algorithm called "evolutionary programming" was created in 60's by Fogel and it was the first attempt to construct an automaton by evolutionary principle, but only with mutation operator. Some recently works tries to construct automata without output function using genetic algorithm (for purpose of lexical analysers). This contribution describes experiments with genetic algorithms (GA) on more general automata (Moore and Meally), compares "classical" GA with genetic operators affected by probability. The minimisation is built-in genetic algorithm. The algorithm is tested on simple examples.

1 Introduction

The first step is to create an automaton model in a process of digital hardware design. Further, the automaton model is an input of software synthesis tools. The model has to be created by engineer. The genetic programming as a part of artificial intelligence tries to create programs using evolutionary principles (a program is created by another program instead of programmers by genetic algorithm). Similar approach can be used in a hardware design, i.e. to create automatically an automaton using evolutionary techniques. This idea isn't a new one. Automata have already been constructed by evolutional techniques in 60's. Fogel [1] created an "evolutional programming" that was the first attempt to

construct an automaton by evolutionary approach. The algorithm only used a mutation operator in contrast to present genetic operators (crossover and mutation). Some recently works [2] tries to construct automata without output function using genetic algorithm (for purpose of lexical analysers).

This contribution describes experiments with genetic algorithms (GA) on more general automata (Moore and Meally), compares "classical" GA with genetic algorithm, where genetic operators are affected by probability. The minimisation is built in genetic algorithm. The algorithm is tested on simple examples.

1.1 Finite Automata

A finite automaton is a mathematical behavioral model of a sequential logical system and it is defined as

$$A = (\mathbf{X}, \mathbf{Y}, \mathbf{Q}, \delta, \lambda, Q_0),$$

where

$\mathbf{X}$ .. finite set of input symbols

$\mathbf{Y}$ .. finite set of output symbols

$\mathbf{Q}$ .. finite set of internal states

δ .. transition function (δ: $\mathbf{Q} \times \mathbf{X} \rightarrow \mathbf{Q}$)

λ .. output function (λ: $\mathbf{Q} \times \mathbf{X} \rightarrow \mathbf{Y}$, ev. λ: $\mathbf{Q} \rightarrow \mathbf{Y}$)

Q_0 .. initial internal state ($Q_0 \in \mathbf{Q}$)

Transition and output functions are realised with combinational logic, the internal state is stored in flip-flops. If an output function has a form λ: $\mathbf{Q} \times \mathbf{X} \rightarrow \mathbf{Y}$, the automaton is a Meally one. If an output function has a simpler form λ: $\mathbf{Q} \times \mathbf{X} \rightarrow \mathbf{Y}$, the automaton is a Moore one.

1.2 Genetic Algorithm Overview

Genetic algorithms (GAs) are evolutionary techniques to search an optimal problem solution. The principle is derived from Darwin's theory of evolution and was introduced by Holland in 70's from biological point of view [3] and by Goldberg from technical point of view [4]. In nature, individuals reproduce and children have different characteristics because crossover of chromosomes happen (offspring inherit characteristics from mother and father). Sometimes, genes change due to environment. This is a mutation process. Good individuals survive during centuries and the percentage of individuals with bad characteristics degreases.

GAs copy this principle. Theory of GAs is based on schemata theory and probability. The first step of the algorithm is to generate set of N initial solutions randomly. The set is called generation, one solution is an individual. The quality of the each individual is measured by the fitness function. In the second step, pairs of individuals (parents) from generation are selected randomly but probability of selection is proportional to their fitness. It means better individuals have greater chance to be propagated into next generation. Roulette-wheel selection, Baker's Stochastic Universal Sampling or Tournament selection are the most used methods [5]. Then, crossover operation is applied to the selected pair of parents with probability p_c and newly created individuals (children) are copied into the next generation. Parents are copied into the next generation with probability (1-p_c) without changing. The parents selection and the crossover operation repeats until new generation is created. Finally, each individual in the new generation is mutated with a small probability p_m or left not changed. Individuals are evaluated by fitness function. New generations are created repeatedly applying selection, crossover and mutation processes. The algorithm terminates when a criterion is achieved. The criterion is usually formulated as follows: terminate when a solution of given quality is found (the given value of fitness function is reached) or a given number of generation was created. The flowchart graph of GA is in Figure 1.

The individual is often represented as binary, integer or real string. The crossover between two parents has the form: crossover point is generated randomly, the part situated on the left side from crossover point of parent no. 1 and the right part of parent no. 2 are copied into the first child. The second child is generated from left part of the second parent and the right part of the first parent. An allele (index in string) is selected randomly and the gene (value) is changed (inverted) in mutation process. It must be checked in both cases if offspring are valid solution.

Parameters of GA are set empirically. The values about p_c = 0,7-0,9 and p_m = 0,05-0,2 are recommended. A crossover operation helps to tend to better solution, a mutation process prevents to lock in a local optima. It's said by researchers that higher value of p_m lead to worse convergence. But GA with higher p_m are successful in some application (for example, Fogel [1] only used a mutation, searching a solution in [6] was successfully verified with $p_m = 0.6$).

Variations of standard GA were established. Newly created offspring (by crossover or mutation) replace their parents if only they are better than ones. Two-point crossover is used instead of classical one-point crossover. Genetic programming operates very often over tree-representation of individuals [7]. Other derivation is parents are selected with the same probability for crossover, "intergeneration" is created and selection is applied during a copying individuals into new generation. Nowadays, all possible approaches are mixed and it's mentioned generally about evolution techniques.

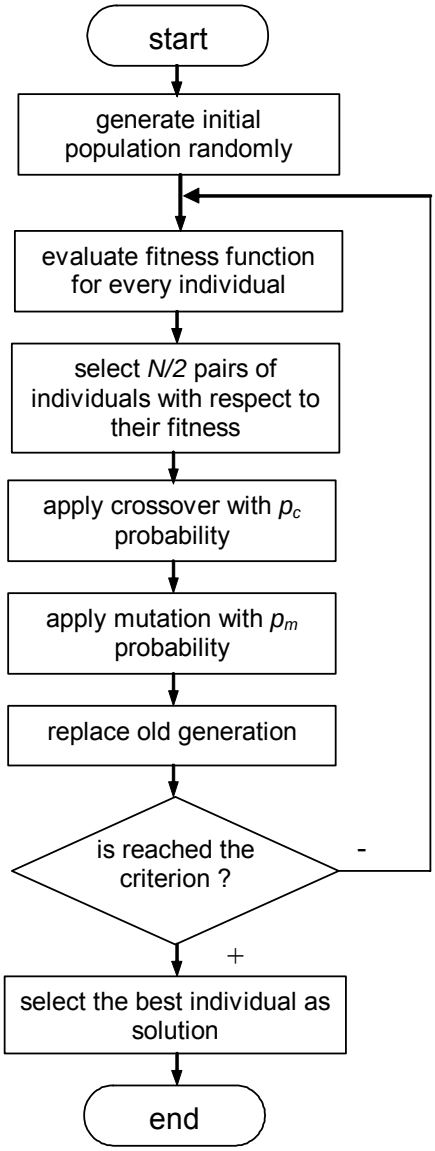


Figure 1
Genetic algorithm

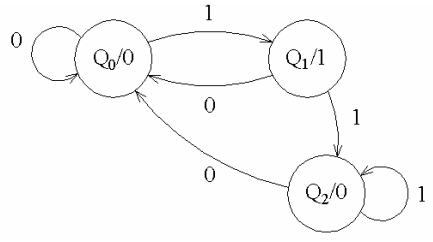
2 Genetic Algorithm and Automata

As mentioned, Fogel [1] created automata with mutation operator, Lucas [2] designed automata without output function using GA. Collins and Jefferson solved "An Artificial Ant" problem by searching a Meally automaton with constant number of internal states, *n=4.* They used classical GA with stationary length of linear chromosome [8]. The aim was to create more general automaton (Meally or Moore) with classical genetic algorithm. The algorithm was tested on simple examples for the present. Four examples are presented in this paper.

3 Individuals Representation

The individual is a final automaton. The automaton is determined by transition and output tables, both represented as matrices - transition matrix and output matrix. The output table is degraded onto vector in the case of Moore automaton. Input parameters of algorithm for creating an individual are as follows: the number of input symbols *x*, the number of output symbols *y*, and a upper bound of internal states *q*. Input symbol are automatically coded with numbers 0, ..., *x*-1, output symbols are automatically 0, ..., *y*-1. Internal states are marked 0, ..., *q*-1, initial state is always Q_0, i.e. 0 value. Hence, the transition and output matrices have *q* rows and *x* columns. The output matrix of Moore automaton is a *q* dimensional vector. The value of transition matrix in i^{th} row and j^{th} column is the next state for present state *i* and input *j*. If the value is -1, the transition or output isn't defined.

Example: we have an Moore automaton model of pulse generator, realized as synchronous sequential system, see Figure 2. The sets of input and output symbols are both {0,1}, the automaton has three internal states Q_0, Q_1, Q_2, represented by numbers 0, 1, 2. A matrix representation is also in Figure 2. The value 1 located in the row 0 and column 1 determines that the automaton changes its internal states from state 0 to state 1 applying input symbol 1.



transition matrix

	col 0	col 1
row 0	0	1
row 1	0	2
row 2	0	2

output matrix

0
1
0

Figure 2
An automaton example and its representation

This matrix representation of an individual (called 2D chromosome) is simple and it's suitable for easy implementing of genetic operator. A problem can be with upper bound of internal states, the user of algorithm has to estimate minimal number of internal states needed to solve given problem. It can be eliminated by modification of algorithm. The maximal size of automaton can be increased during running of algorithm, if it is founded that all or most of states are used and problem is not solved (not implemented yet).

4 Genetic Operators

Standard genetic operators *one-point crossover* and *mutation* are used.

Crossover

Two parents are selected randomly from generation with the same probability. The crossover point is selected randomly with uniform probability from transition or output matrix and two offspring are created, Figure 3.

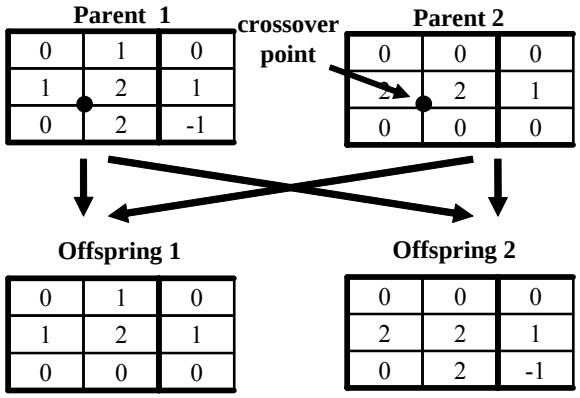


Figure 3
Crossover operator

Mutation

It is selected with probability p=0.5 if the transition table or output table is mutated. Then the transition or output is selected randomly and it is changed, Figure 4.

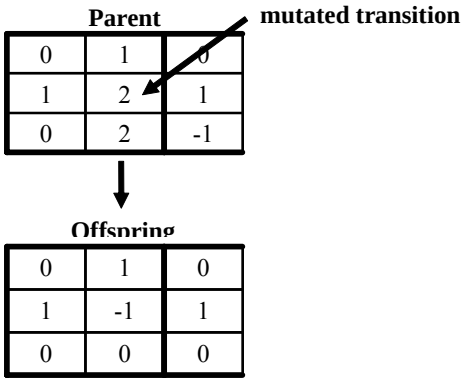


Figure 4
Mutation operator

5 Fitness Function and Selection

Fitness function should verified that the automaton corresponds to specification. That is a general problem to verify automaton, especially if the specification isn't formalized. One of behavioural model of sequential systems is sequential mapping that maps input sequence of final length to output sequence. An idea is to create training set of pairs of input and matching output sequences, similar as in neural networks. Each individual is tested on this training set. The fitness function measures successful rate how the individual passed the test. It's clear that automaton which pass all pairs needn't be general one to solve the problem if a training set is not designed good (it happened within water pumps example). If complex combinational circuits are designed by GA, neither all pairs are tested due to combinatorial catastrophe, despite of these algorithms are used. This problem will not be probably solved in the future.

Nevertheless, the training set consists of sequence pairs is a base of testing individual. An example of training set related to pulse generator is in Figure 5. Because it's a Moore automaton, the valid output symbol is located in the output sequence one clock cycle late.

The output sequence is calculated for each automaton (individual) and each input sequence in the set. The calculated sequence is compared with the right output sequence in the set. The ratio $P = \frac{l}{n}$ is calculated for each pair, where n is the length of sequence and l is the length of the beginning successful part of the

output sequence. The ratio $ratio = \min P_i$ over all pairs is found and it's a base value of fitness function (very strict approach). The function (defined after this manner) gives a chance to algorithm to construct an automaton as by a man, from the beginning state step by step.

00001000111100
00000100010000

01000111111100
00100010000000

01100011100000
00100001000000

11110011111000
01000001000000

Figure 5
Example of a training set

Two automata with different number of internal states can be equivalent. After automaton is constructed, designers try to do minimisation of the automaton. This aspect is taken into account. So, the fitness function is defined as two dimensional:

$$F = [ratio,k],$$

where *ratio* is a ration defined above and k is the size of the automaton (the size of connected component calculated from the initial state 0). Let F_1 be $F_1 = [ratio_1, k_1]$ and F_2 be $F_2 = [ratio_2, k_2]$. Automaton A_1 has value of fitness equal to F_1, automaton A_2 has value of fitness equal to F_2. Order is defined: $A_1 < A_2$ (A_1 is a worse automaton than A_2), iff

$$ratio_1 < ratio_2 \text{ or } (ratio_1 = ratio_2 \text{ and } k_1 > k_2)$$

Selection is a tournament method, because it is good realized with two-dimensional function.

6 Mutation Affected by Probability

Mutated gene (transition or output in tables) is selected randomly with uniform distribution. It may result that already good created transitions and outputs are uselessly corrupted. Transitions and outputs which seem to be good should have lower probability of change than others. That's way probability matrix for each

automaton is calculated during fitness function estimation of each individual. Lucas and Reynolds [2] label states during estimation of individual. Because their automata have no output function the labeling is only determined by correspondence of expected and reached final state. If correspondence is reached all state in trajectory are positive labeled.

Moore and Meally automata have output functions, so we can compare output symbol of the tested automaton with symbol in training sequence at each step. *Score* of each transition and output is calculated by the rule described bellow. First, we suppose that the automaton made a transition from state i to state j applying input symbol x. Current input symbol is y. If the output symbol of the tested automaton is equal to output symbol in the sequence at current step, the value -1 is added to score of output which is determined with the state j in the case of Moore automaton or with the state j and input y in the case of Meally automaton respectively. Transition from the state i and input symbol x is affected in the same way. If the output symbol of the tested automaton is not equal to output symbol in the sequence at current step, the output and transition are penalized – the value 1 is added. It is summarized concurrently how many times the transition (or output) was tested (value marked by n). After all automata are estimated, the $ratio = \frac{score}{n}$ is calculated for each transition and output. If n=0 the let the ratio be 0. The *ratio* values are in <-1,1> range. The -1 value means that the transition (output) are absolutely successful. The *ratio* is transformed with linear formula:

$$p = a(ratio + 1) + d$$

Values p are normalized so that the sum across all matrix is equal to 1.0. The values are probabilities of mutation transition (or output) in tables. The probability of change of gene is proportional to its quality.

7 Experiments

The algorithm is implemented in Borland C++ Builder. Results are shown on four examples: pulse generator (Figure 2), autonomous counter mod 8, counter mod 8 with enable, automaton for control water pumps (Figure 6).

The function of counters is clear. The automaton for control water pumps has three bit input symbols, each bit is a state of a one water level sensor. The automaton controls two pumps.

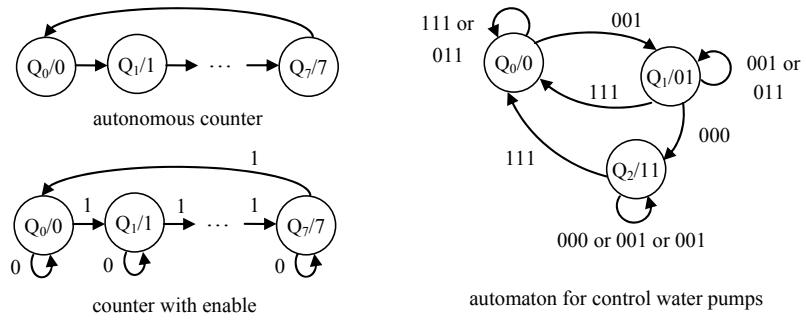


Figure 6
Example of automata

Results are shown in tables below. The algorithm was run ten times with the same parameters. Training sets of sequences were created by author. It's shown the smallest, average and the greatest number of generation, when the right solution was founded for the first time. Normal algorithm and "probability" algorithm are compared. Other parameters are: N is the size of generation, q is maximum number of states (upper limit), I is the number of iteration.

The symbol NF means that the right solution wasn't found in all 10 runs. Number of examples and runs is small for the present to do statistical conclusion for the present. Nevertheless, the algorithm seems to be relatively robust to parameters, if it found a solution. The mutation operator is meaningful. Probability version of algorithm increased its ability rapidly in the case of counters.

It was observed, when the probability version was tested with fourth example, if many non-minimal (but good) solutions dominate in population, minimal solution is searched hardly. Naturally, if the additive parameter equals to 0, absolutely successful transitions or output have zero probability of change. Small value of d parameter about 0.2 would be more suitable.

Algorithm was tested on two simple Meally automatons besides – automatons which receive serial input of 1 bit width and recognized subsequence of odd (even respectively) numbers 1. The solution was found in 78 generation for the first time.

Automaton which receives three bit serial numbers and recognizes them wasn't found with parameters N=1000, I=1000 not even with probability version. Its graph is tree based. A part of tree appeared when Moore automaton was being created, but not with Meally.

Pulse generator			
Normal algorithm			
N = 100, q = 5, I = 100			
	Smallest	**Average**	**Greatest**
Pc = 0.8, Pm = 0.3	53	77	84
Pc = 0.5, Pm = 0.3	29	86	57
Pc = 0.2, Pm = 0.6	11	22	47
Pc = 0.2, Pm = 0.8	21	64	75
Probability algorithm			
N = 100, q = 5, I = 100, Pc = 0.2, Pm = 0.6			
a = 1, d = 0.0	NF	NF	NF
a = 1, d = 0.2	19		
a = 1, d = 0.4	7	44	73

Autonomous counter			
Normal algorithm			
N = 100, q = 10, I = 500			
	Smallest	**Average**	**Greatest**
Pc = 0.8, Pm = 0.3	168	228	378
Pc = 0.8, Pm = 0.5	98	-	396/NF
Pc = 0.2, Pm = 0.6	97	278	457
N = 500, q = 10, I = 100			
Pc = 0.8, Pm = 0.3	47	72	86
Probability algorithm			
N = 100, q = 10, I = 100, Pc = 0.2, Pm = 0.6			
a = 1, d = 0.0	54	57	58

Experiments showed that this “strict fitness” is suitable for transition graph with cycles, but not for automata with tree-based transition graph. It also degrades crossover operator. Next work will focus on “classical fitness” that passes all test sequence till the end.

Counter with enable			
Normal algorithm			
N = 100, q = 10, I = 500			
	Smallest	**Average**	**Greatest**
Pc = 0.8, Pm = 0.3	462	-	474/NF
Pc = 0.8, Pm = 0.5	98	-	396/NF
Pc = 0.2, Pm = 0.6	414	-	491/NF
N = 500, q = 10, I = 100			
Pc = 0.8, Pm = 0.3	154	247	336
Probability algorithm			
N = 100, q = 10, I = 100, Pc = 0.2, Pm = 0.6			
a = 1, d = 0.0	79	132	159
a = 1, d = 0.2	86	112	133

Controlling water pumps			
Normal algorithm			
N = 100, q = 5, I = 100			
	Smallest	**Average**	**Greatest**
Pc = 0.8, Pm = 0.3	71*	-	87/NF
Pc = 0.8, Pm = 0.5	78	-	80/NF
Pc = 0.2, Pm = 0.6	NF	NF	NF
*automaton wasn't minimal, but was OK			
N = 500, q = 5, I = 100			
Pc = 0.8, Pm = 0.3	34	59	60
Probability algorithm			
N = 100, q = 10, I = 100, Pc = 0.2, Pm = 0.6			
a = 1, d = 0.0	46**	132	65
a = 1, d = 0.2	50***	-	82/NF
** four of solutions were not minimal			
*** two of solutions were not minimal			

Conclusion

The article summarizes the first experience with automata constructed with classical genetic algorithm. It compares classical algorithm with mutation affected by probability matrix that seems to be more successful especially for “chain-like” graphs. Very difficult is to create automaton with tree-based transition graph than a cyclic graph. Next work will focus on new crossover operator affected by probability as well as mutation and other variants of fitness functions.

References

[1] Fogel L. J., Owens A. J., Walsh M. J.: Artificial Intelligence Through Simulated Evolution, Wiley, New York, 1966

[2] Lucas S. M., Reynolds T. J.: Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm, IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 27, No. 7, July 2005

[3] Holland J. H.: Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975

[4] Goldberg D. E.: Genetic Algorithm in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989

[5] Štěpánková a kol.: Umělá inteligence 3, 4, Academia, 2002, in Czech

[6] Fábera V.: The solving of „criss-cross“ crossword using genetic algorithm, in Proceedings of “Informatika a Informačné technologie 2004” Conference, Banská Bystrica, Slovakia, September 2004, in Czech

[7] Koza J. R.: Genetic Programming, MIT Press, Cambridge, MA, 1994

[8] Collins R., Jefferson D.: Representations for artificial organisms, in Proceedings of the First International Conference on Simulation of Adaptive Behavior, MIT Press, 1991