An Incremental Learning Algorithm for Deterministic Finite Automata using Evolutionary Algorithms

Jonatan Gomez

Universidad Nacional de Colombia

1 Introduction

This work proposes an approach for learning Deterministic Finite Automata (DFA) that combines Incremental Learning and Evolutionary Algorithms. First, the training sequences are sorted according to its length from shortest to longest. Then, the training sequences are divided in a suitable number of groups (M) of the same size. Second, the Hybrid Adaptive Evolutionary Algorithm (HaEa) proposed by Gomez [2] is used to evolve a DFA population that is able to classify correctly (with 80% precision) the first group (the shortest sequences). Third, the population is evolved again with HaEa but searching for DFA that are able to classify correctly the first and second groups. This process is repeated until each automaton in the population tries to classify correctly (80% precision) all the training samples. Finally, another HaEa is used to refine the automata in the population (100% precision).

Algorithm 1 Incremental DFA Learning algorithm using HaEa

```
DFA_Learning( Training_set x )
1. POP_SIZE = 20 // population size
2. MAX_GEN = 500 // maximum number of generations
3. M = 40 // number of groups
5. Sort(x) // sorting the data set according to sequence length
6. P = InitPopulation( POP_SIZE )
7. for i=1 to M do // incremental learning
8.    P = Evolve_HaEa( P, MAX_GEN, x, i, M, 0.8 )
9. P = Evolve_HaEa( P, MAX_GEN, x, i, M, 1.0 ) // refinement
```

2 Encoding

A deterministic and finite automaton of n states is encoded using a sequence of $32 * 2 * n$ bits as shown in figure 1. Here, $< s_1, x, s_2 >$ represents the edge starting at state s_1, ending at state s_2 with transition symbol x.

Fig. 1. Encoding of an automaton of n states.

State 0		State 1		...	State $n-1$	
$< 0, 0, x_{0,0} >$	$< 0, 1, x_{0,1} >$	$< 1, 0, x_{1,0} >$	$< 1, 1, x_{1,1} >$	...	$< n-1, 0, x_{n-1,0} >$	$< n-1, 1, x_{n-1,1} >$

As shown, only the topology of the automaton is encoded. It is possible to determine the label of each state as proposed by Lucas and Raymond [4]. Moreover, only the ending state of an edge is stored. We use the java representation of an integer with this purpose.The initial population is randomly generated with variable length varying between 10 and 50 states. Since the ending state encoded can be any integer value, possibly higher than the number of states in the automaton, the absolute value and modulus functions are used to restrict their values to the appropriated values. It is only done in the DFA decoding process, see equation 1.

$$x_{i,j} = |x_{i,j}| \ mod \, n \tag{1}$$

3 Hybrid Adaptive Evolutionary Algorithm

Algorithm 2 presents the proposed Hybrid Adaptive Evolutionary Algorithm (**HaEa**). This algorithm is a mixture of ideas borrowed from Evolutionary Strategies (**ES**), decentralized control adaptation, and central control adaptation.

3.1 Selection Mechanism

In HaEa, each individual is "independently" evolved from the other individuals of the population, as in evolutionary strategies [1]. In each generation, every individual selects only one operator from the set of possible operators (line 8). Such operator is selected according to the operator rates encoded into the individual. When a non-unary operator is applied, additional parents (the individual being evolved is considered a parent) are chosen according to any selection strategy, see line 9. As can be noticed, HaEa does not generate a parent population from which the next generation is totally produced. Among the offspring produced by the genetic operator, only one individual is chosen as child (line 11), and will take the place of its parent in the next population (line 17). In order to be able to preserve good individuals through evolution, HaEa compares the parent individual against the offspring generated by the operator. The Best selection mechanism will determine the individual (parent or offspring) that has the highest fitness (line 11). Therefore, an individual is preserved through evolution if it is better than all the possible individuals generated by applying the genetic operator.

Algorithm 2 Hybrid Adaptive Evolutionary Algorithm (HAEA)

```
HaEa( λ, terminationCondition )
1.  t_0 = 0
2.  P_0 = initPopulation( λ ) ,
3.  while( terminationCondition( t, P_t ) is false ) do
4.    P_{t+1} ={}
5.    for each ind ∈ P_t do
6.      rates = extract_rates( ind )
7.      δ = random(0,1)      // learning rate
8.      oper = Op_Select( operators, rates )
9.      parents = ParentSelection(P_t, ind )
10.     offspring = apply( oper, parents )
11.     child = Best( offspring, ind )
12.       if( fitness( child ) > fitness( ind ) ) then
13.         rates[oper] = (1.0 + δ)*rates[oper]    //reward
14.     else
15.       rates[oper] = (1.0 - δ)*rates[oper]    //punish
16.     normalize_rates( rates )
17.     set_rates( child, rates )
18.     P_{t+1} = P_{t+1}∪ {child}
19.   t = t + 1
```

3.2 Encoding of Genetic Operator Rates

The genetic operator rates are encoded into the individual in the same way as decentralized control adaptation techniques, see figure 2. These probabilities are initialized (into the *initPopulation* method) with values following a uniform distribution $U[0, 1]$. A roullete selection scheme is used to select the operator to be applied (line 8). To do this, the operator rates arenormalized in such a way that their summation is equal to one (line 16).

SOLUTION	**OPER**$_1$	**...**	**OPER**$_n$
100101011..01	0.3	...	0.1

Fig. 2. Encoding of the operator probabilities in the chromosome

3.3 Adapting the Probabilities

The performance of the child is compared against its parent performance in order to determine the productivity of the operator (lines 12-15). The operator is rewarded if the child is better than the parent and punished if it is worst. The magnitude of reward/punishment is defined by a learning rate that is randomly generated (line 7). Finally, operator rates are recomputed, normalized,

and assigned to the individual that will be copied to the next population (lines 16-17). The learning rate is generated in a random fashion instead of setting it to a specific value for two main reasons. First, there is not a clear indication of the correct value that should be given for the learning rate; it can depend on the problem being solved. Second, several experiments encoding the learning rate into the chromosome [3] show that the behavior of the learning rate can be simulated with a random variable with uniform distribution.

3.4 Properties

Contrary to other adaptation techniques, HAEA does not try to determine and maintain an optimal rate for each genetic operator. Instead, HAEA tries to determine the appropiate operator rate at each instance in time according to the concrete conditions of the individuals. If the optimal solution is reached by an individual in some generation, then the rates of the individual will converge to the same value in subsequent generations. This is true because no genetic operator is able to improve the optimal solution, therefore any operator will be punished when applied and the other operators will be rewarded.

HAEA uses the same amount of extra information as a decentralized adaptive control; HAEA requires a matrix of $n * M$ doubles, where n is the number of different genetic operators and M is the population size. Thus, the space complexity of HAEA is linear with respect to the number of operators (the population size is considered a constant). Also, the time expended in calculating and normalizing the operator rates is linear with respect to the number of operators $n * M$ (lines 8 and 12-16). HAEA does not require special operators or additional parameter settings. Well known genetic operators can be used without any modification. Different schemes can be used in encoding the solution: binary, real, trees, programs, etc. The average fitness of the population grows monotonically iteration by iteration. One individual is always replaced by an individual with equal or higher fitness.

4 Genetic Operators

Three well known genetic operators were used for evolving the DFA: single point mutation, single point crossover, and a simple transposition.

- In the single bit mutation, one bit of the solution part is randomly selected (with uniform distribution) and flipped, see figure 3. Notice that this genetic operator always modifies the genome by changing only one single bit.

- In the simple transposition operator, two points in the solution part are randomly selected and the genes between such points are transposed [5], see figure 4.

SOLUTION	$OPER_1$	...	$OPER_n$
100 **1** 1100101	0.3	...	0.1

(a) Parent

SOLUTION	$OPER_1$	...	$OPER_n$
100 **0** 1100101	0.3	...	0.1

(b) Offspring

Fig. 3. Single bit mutation

SOLUTION	$OPER_1$	...	$OPER_n$
100 **11100** 101	0.3	...	0.1

(a) Parent

SOLUTION	$OPER_1$	...	$OPER_n$
100 **00111** 101	0.3	...	0.1

(b) Offspring

Fig. 4. Simple transposition operator

– In single point crossover, a cutting point in the solution part is randomly selected. Parents are divided in two parts (left and right) using such cutting point. The left part of one parent is combined with the right part of the other, see figure 5.

SOLUTION	$OPER_1$	...	$OPER_n$
100 * 11100101	0.3	...	01

SOLUTION	$OPER_1$	...	$OPER_n$
111 * 00011000	0.2	...	0.4

(a) Parents

SOLUTION	$OPER_1$	...	$OPER_n$
100 * 00011000	0.3	...	0.1

SOLUTION	$OPER_1$	...	$OPER_n$
111 * 11100101	0.2	...	0.4

(b) Offspring

Fig. 5. Single bit crossover

5 Additional Considerations

In order to aliviate the take over effect on small populations and maintain diversity, we replace the full population (exception done with the best individual) with a new one randomly generated when there population is totally take over by the best individual, see Algorithm 3.

6 Running the Program

The command for executing the associated program is:

Algorithm 3 Maintaining Diversity on HaEA for DFA Learning

```
EVOLVE_HAEA( P, MAX_GEN, x, i, M, 1.0 ) // refinement
1. k = 1
2. while( k<MAX_GEN and fitness_best(P) < 0.8 ) do
3.   P = HAEA( P, x, i, M )
4.   if( fitness_best(P) = fitness_worst(P) ) then
5.     b = best(P)
6.     P = INITPOPULATION(POP_SIZE-1)
7.     P = P + b
8. return P
```

java -classpath "gecco04.jar" gecco04.DFATest train.txt test.txt out.txt [time]
where,
train.txt is the training set,
test.txt is the testing set,
out.txt is the output file, and
time is an optional parameter for ending the algorithm (by default is 10 minutes).

References

1. T. Back. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.
2. J. Gomez. Self adaptation of operator rates in evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, June 2004.
3. J. Gomez and D. Dasgupta. Using competitive operators and a local selection scheme in genetic search. In *Late-breaking papers GECCO 2002*, 2002.
4. S. M. Lucas and T. J. Reynolds. Learning dfa: Evolution versus evidence driven state merging. In *Proceedings of the Congress on Evolutionary Computation (2003)*, 2003.
5. A. Simoes and E. Costa. Transposition: a biologically inspired mechanism to use with genetic algorithms. In *Fourth International Conference on Neural Networks and Genetic Algorithms*, pages 612–619, 1999.