Reports

Machine Learning and Inference Laboratory

Initial Experiments with the LEM1 Learnable Evolution Model:
An Application to Function Optimization and Evolvable Hardware

Ryszard S. Michalski*
Qi Zhang

Machine Learning and Inference Laboratory
George Mason University

P 99-4
MLI 99-4

May 1999

* Also with GMU Department of Systems Engineering and Operations Research, Department of Computer Science, and Institute of Computer Science at the Polish Academy of Sciences

Initial Experiments with the LEM1 Learnable Evolution Model: An Application to Function Optimization and Evolvable Hardware

Ryszard S. Michalski and Qi Zhang

ABSTRACT

This report presents results of a series of experiments on applying LEM1, a preliminary implementation of Learnable Evolution Model, to a sample of problems in function optimization and evolvable hardware. The Learnable Evolution Model, introduced in [Michalski, 1998, 1999], employs machine learning to guide evolutionary computation. Specifically, in Machine Learning mode, it computes *qualitative differentials* that hypothesize the desirable direction of evolution. A new population is created by instantiating generated hypotheses. In LEM1, Machine Learning mode employs AQ15c learning program and Darwinian Evolution mode employs genetic algorithm GA2. In the experiments, LEM1 significantly outperformed two canonical genetic algorithms, GA1 and GA2, used in the study.

Keywords: Evolutionary Computation, Learnable Evolution Model, Genetic Algorithms, Function Optimization,

Acknowledgements

.

This research was conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's research activities have been supported in part by the National Science Foundation under Grants No. IIS-9904078 and IRI-9510644, in part by the Defense Advanced Research Projects Agency under Grant No. F49620-95-1-0462 administered by the Air Force Office of Scientific Research, and inpart by the Office of Naval Research under Grant No. N00014-91-J-1351.

1 INTRODUCTION

Research on evolutionary computation goes back the 1960s, when researchers started to use computers for modeling processes of natural evolution. The basic approach to evolutionary computation is based on the Darwinian model of natural evolution. According to a simplified version of this model, an evolutionary process proceeds through the following steps:

> (1) individuals in a population undergo changes due to mutation and recombination operations,
>
> (2) individuals that are most fit according to some measure, have the greatest probability to be selected for the new population ("survival of the fittest"). The process repeats from step (1), and continues until a termination condition is met.

One of the most popular approaches to evolutionary computation are genetic algorithms (GAs). Starting with an initial population of individuals represented by vectors of attribute values (in the original version, binary strings), a genetic algorithm in consecutive steps of evolution generates new populations by applying operators of random mutation, crossover, and selection. The selection operator may implement the survival of the fittest principle or some other method (e.g., Goldberg, 1989; Michalewicz, 1992). When applying a genetic algorithm to problem solving, individuals are candidate solutions.

There are two major problems with this form of computation. First, the search process is semi-blind: the mutation is a random change of an individual (e.g., problem solution); crossover is a semi-random combination of two candidate solutions; and selection the survival of the fittest is a form of parallel hill-climbing. Consequently, computational processes based on Darwinian evolution are not very efficient. A low efficiency is the major obstacle in the applications of genetic algorithms to very complex problems.

Second, an evolution process is prone to finding a locally optimal solution. In order to escape from a local optimum the mutation operator and/or crossover operator may need to be properly adjusted. For example, the mutation operator may have to be set to make larger steps. What mutation step is needed for a given problem cannot be, however, predicted in advance. In general, for each type of landscape a different mutation rate and the size of mutation step may be most effective. Thus, the performance of a genetic algorithm on a given problem may strongly depend on the settings of its parameters. Determining appropriate parameter settings for different algorithms and different problems is one of the central topics in the field of evolutionary computation.

A new type of evolutionary computation, called *Learnable Evolution Model* (henceforth called LEM) has been recently introduced (Michalski, 1998a, 1998b). In LEM, an evolutionary process, specifically, the creation of new populations is guided by machine learning. The basic idea is that a machine learning system is employed to determine hypothesis why certain individuals in a given population (or populations) are better than others in performing a given class of tasks.

These reasons, expressed as inductive hypotheses, are then used for creating a new generation of individuals.

This report describes the application of LEM1, a preliminary implementation LEM to a class of problems in function optimization and evolvable hardware. A series of experiments were conducted on the set of five benchmark functions (De Jong 1975; Goldberg 1989). In the experiments, LEM1 outperformed genetic algorithms used in the study, sometimes achieving a two or more orders of magnitude speed up over in terms of the number of steps of an evolution process to reach solution. For some problems, LEM achieved the optimal solution while genetic algorithms employed in the study could not come even close to it.

The following material is organized as follows: Section 2 describes briefly the general algorithm of LEM; Section 3 details LEM in case of function optimization and other genetic algorithms used in the experiments; Section 4 reports results plus a discussion of them; Section 5 gives a brief description of related work; Section 6 concludes the report and briefs future work.

2 BRIEF REVIEW OF THE LEM METHODOLOGY

This section briefly reviews of the LEM methodology as described in (Michalski, 1998a; 1998b). LEM can work as a combination of Machine Learning mode and Darwinian Evolution mode, or in Machine Learning mode alone. In Machine Learning mode, LEM applies a machine learning system to create descriptions differentiating high performing individuals (the HIGH group or H-group) from low performing (the LOW group or L-group). Such descriptions are indicative of the direction of slope of the fitness landscape, and have been termed *qualitative differentials* (Michalski, 1998). Qualitative differentials are instantiated to generate individuals for a new population.

Machine Learning mode can potentially employ any machine learning method that is able to create discriminant descriptions of individuals in a population. Symbolic learners such as AQ-type rule learning program (Michalski et al. 1986; Kaufman and Michalski, 1999), CN2 (Clark and Niblett, 1989) or C4.5 (Quinlan, 1993) produce descriptions that are relatively easy to interpret and this is an advantage for many applications. An AQ-type learner has an additional advantage that it uses a description language that includes internal disjunction operator. This operator is particularly useful for LEM as descriptions with this operator facilitate generation of events that differ slightly (Michalski, 1998a,b). Darwinian evolution mode can employ any existing method of evolutionary computation that uses some form of mutation, crossover (optionally) and selection operators, for example, a genetic algorithm, an evolution strategy or a method of evolution programming.

There can be different variants of LEM. One variant is to alternate between Machine Learning and Darwinian evolution mode. A switch between the modes occurs when there is no sufficient progress of evolution. Another, simpler variant, is to repeatedly apply Machine Learning mode (possibly with different starting populations) until a termination criterion is satisfied. By differently changing parameters of the learning program or the learning program itself during the evolutionary process, one can obtain many different variants of LEM.

A general description of LEM, based on the ideas in (Michalski, 1998b), is presented in Table 1.

(1) Randomly or according to some method generate a population of individuals.

(2) **While** *LEM termination condition* is not met

(2.1) **While** the best individual in a sequence of *learn-probe* generations is better by the *learn-threshold* than the best one found in previous generations execute *Machine Learning mode* :

(2.1.1) Determine H-groups and L-groups in the population based on the fitness function applied to the individuals in the population.

(2.1.2) Apply a machine learning method to create a discriminant description that distinguishes H-group from L-group.

(2.1.3) Generate a new population of individuals by replacing L-group individuals by those satisfying the description of H-group (in general, this step may involve generating a larger set of new individuals and then applying some form of selection)

(2. 2) If variant is uniLEM, go to step (1), otherwise go to (2.3)

(2. 3) **While** the best individual in a sequence of *dar-probe* generations is better by the *gen-threshold* than the best one found in previous generations execute the *Darwinian Evolution mode* :

Apply an evolutionary computation algorithm that employs some form of mutation, crossover (optionally) and selection operators to generate consecutive populations.

Table 1. A general description of LEM.

The H-group and L-group in a given population can be determined in different ways. One way is the *population-based method* in which the H-group is determined by selecting HPT% best individuals from the population, and the L-group is determined by selecting LPT% worst individuals. HPT (High Population Threshold) and LPT (Low Population Threshold) are parameters of the method, which in our experiments were 30% .

A precondition for applying LEM (as for applying any evolutionary computation method) is the ability to evaluate individuals according to some fitness function that reflects the individual's quality from the viewpoint of the given goal or class of tasks). In the case of applying evolutionary computation to function optimization, a fitness value of an individual reflects the closeness of the solution (represented by the individual) to the optimum value. When applying LEM to problem solving, an individual is a candidate problem solution (e.g., in the form of a

vector of attribute values). A LEM termination condition can be, for example, that the best individual generated so far meets a specified criterion, or that the allocated computational resources are exhausted.

3 PRELIMINARY IMPLEMENTATION: LEM1

To test the LEM methodology, we have implemented in a preliminary fashion in program LEM1. In Machine mode, LEM1 employs AQ15c rule learning program (Michalski at al., 1986 et. al., Wnek et al. 1995), and in Darwinian Evolution mode, genetic algorithm GA2. In testing experiments, LEM1, GA2 run independently, and another genetic algorithm GA1 were applied to a series of function optimization problems and a problem in evolvable hardware. GA1 and GA2 were provided by Professor Kenneth De Jong.

GA1 is a simple genetic algorithm using a real-valued representation, mutation, and crossover. In mutation, each gene (i.e., variable) has 1/L chance of being mutated, where L is the number of genes. The selected genes are mutated by incrementing or decrementing their values by a fixed small amount (0.1 in our experiments). Give two parents during crossover, the child is produced via using a simple uniform crossover, each gene being inherited with equal probability from each parent. s GA2 is a simple genetic algorithm using a binary representation with mutation and

GA2 is a simple genetic algorithm using a binary representation with mutation and crossover. In mutation, each gene (i.e., bit) has 1/L chance of being mutated, where L is the sum of the numbers of genes. The selected genes are mutated by changing their values from 1 to 0 or vice versa. When selecting two parents for producing a child, a simple uniform crossover is adopted, each child gene being inherited with a given probability from each parent (0.95 for the firstly selected parent and 0.05 for the later selected parent).

The preliminary version of LEM has been implemented in program LEM1/AQ15c-GA2 according to the algorithm:

(1) Randomly generate the first generation of individuals for evolution.

(2) Apply the Darwinian Evolution mode by running GA2 program.

(3) If the best-so-far is not improved or increased by a certain threshold α in consecutive N generations, change the work mode into AQ; otherwise, the program continues to work in GA mode.

(4) In AQ mode, the best P % and the worst P% individuals in each generation are used as training examples. Each variable in a fitness function is taken as an attribute for the learning algorithm and uniformly discretized into M levels.

(5) Take the strongest rule of describing the best P % individuals as guidance, randomly generate individuals to replace the worst P % individuals, i.e., the generated individuals are covered logically by this rule. If an attribute is used in the rule, then its variable value is generated by randomly selecting one valid value and converting it back to a corresponding real value. If an

attribute is not used in the rule, then its variable value is generated by randomly selecting one value from the values of the best *P* % individuals.

(6) If the best-so-far is not improved or increased by a certain threshold α in consecutive *N* generations, change the work mode into GA2; otherwise, the program continues to work in AQ learning mode.

(7) Repeat (2) to (6) until a certain number of individuals are generated or a condition is satisfied.

In the experiments, the population of each is generation is 20; α is 0.01; *N* is 3; *M* is 200; *P* is 30. When generating individuals according to a rule, a variable value is generated based on a heuristic called MDN rule (*maximum distance to negative*.), i.e., the generated value is as far away as possible from the area defined by the rules of describing the worst 30%. For example, if all the values of the worst 30% individuals are less than all the values of the best 30%, then the generated value is more likely bigger than smaller within its valid range (defined in the learned rule). Five situations are identified in this heuristic and are illustrated in the following in terms of variable value distribution on an axis:

```
- - - - - - + + + + +
+ + + + + - - - - - -
- - - - + + + + - - -
+ + + - - - - - + + +
- - + - + - - + + - -
```

In the above, "+" represents a value taken by one of the best 30% individuals and "-" represents a value taken by one of the best 30% individuals. The results presented are the average over ten runs of each algorithm. Each variable in a function is taken as an attribute. The population size is 20. Five of the five functions listed in "Genetic Algorithm" by David Goldberg, page 108, were used.

Note: these functions are known as workbench problems, and frequently used for testing performance of different genetic algorithms.

4 EXPERIMENTAL STUDY 1: FUNCTION OPTIMIZATION

4.1 Problem Description

Experimental Study 1 used five problems described in (DeJong, 1975; Goldberg, 1989). These problems have been specifically designed to test performance of genetic algorithms. They test the ability of an algorithm to find a solution for different classes of function optimization tasks: simple and complex, continuous and discrete, with few and many variables, with noise and without noise. To see how LEM's performance compares with the performance of some other evolutionary computation algorithms, we also applied two standard genetic algorithms to the above problems. All algorithms were applied to the problem of finding the function maximum, as well as the function minimum. This paper presents only results from seeking the function maximum. The results from seeking the function minimum were similar.

Genetic algorithms used in this study are called GA1 and GA2 (De Jong, 1997). GA1 uses a real-valued representation of individuals in a population, and employs the standard selection, mutation and crossover genetic operators. The mutation operator changes each gene (i.e., a variable value) with the probability of 1/L, where L is the number of genes. Selected genes are mutated by incrementing or decrementing their values by a fixed, small amount (0.1, in our experiments). The crossover operator takes two "parent" individuals, and produces a child, using a uniform crossover, that is, each gene is inherited with equal probability from each parent.

The second genetic algorithm, GA2, uses a binary representation of individuals, and employs the same selection, mutation and crossover operators. The mutation operator changes each gene (in this case, 1 bit) with the same probability as above, that is, 1/L, where L is the numbers of binary genes. The selected genes are mutated by changing their values from 1 to 0 or vice versa. The crossover operator selects two parents and produces a child whose genes inherit a parent gene with a given probability of 0.95 for the first selected parent, and 0.05 for the second parent. The purpose of using two different genetic algorithms was to improve chances for obtaining good results from genetic algorithms in the comparative study with LEM.

LEM1, employed in this study, is an initial implementation of the LEM method described above. It employs AQ15c rule learning algorithm in the symbolic learning mode, and GA1 for the genetic algorithm mode. In the experiments, the population of each generation was 20; the *learn-length was* 3, the *learn-threshold* was 0.01; *gen-length was 3, gen-threshold* was 0.01, HT and LT were 30%, and Ki, I=1,2,...., was 200 for all attributes used in the experiment. We have executed 10 runs of each algorithm. Presented results are averages of these runs. Runs differed in the seeds used for starting a random number generator. LEM and genetic algorithm GA1 used the same seeds.

4.2 Experiments

Problem 1: Find the maximum of the following function (Figure 1).

$$f_1(x_i) = \sum_{1}^{3} x_i^2, \qquad -5.12 \le x_i \le 5.12$$

Maximum: 78.643; Minimum: 0

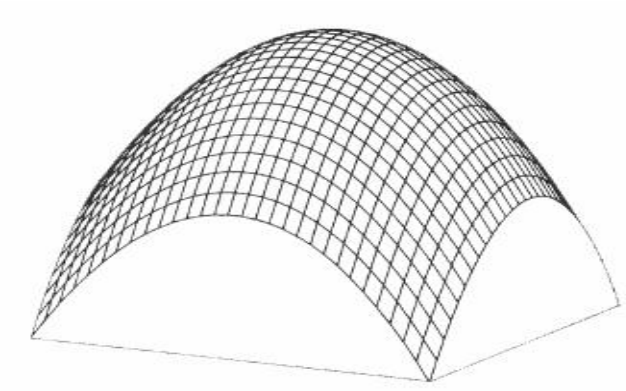

Figure 1. Inverted graph of function f_1 projected into two-dimensional space
(Reprinted with the permission of Kenneth DeJong)

Figure 2 presents results from applying GA1, GA2 and LEM to finding the maximum of f_1. The results show that the LEM's solution approached the maximum significantly quicker than GA1 and GA2: LEM found the near-maximum within about 30 generations, while GA1 and GA2 were at some distance from the maximum even at the 500th generation.

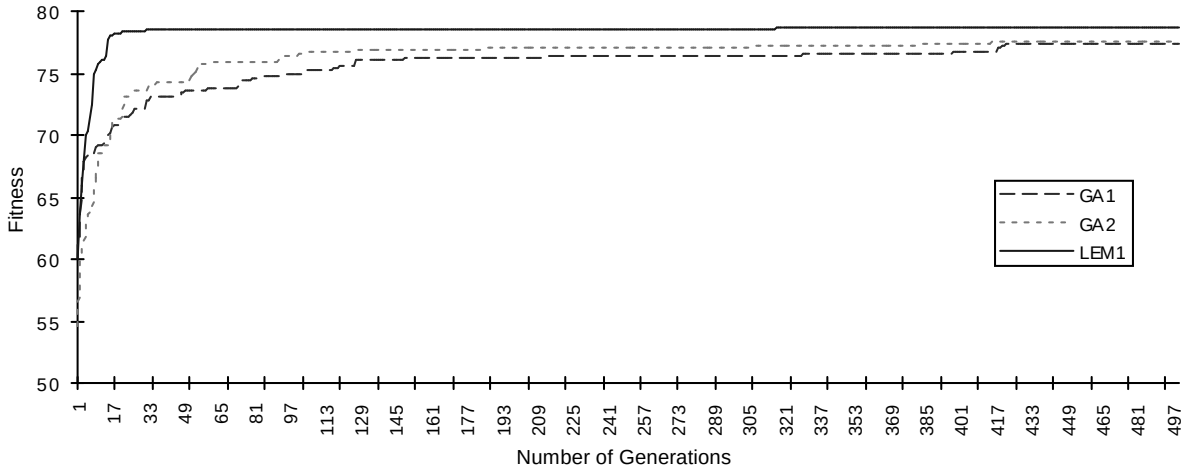

Figure 2. The evolution process through 500 generations (10000 births) for function f_1.

In order to characterize the performance of the tested algorithms, we use a measure, called the "Relative-Distance-to-Target," defined as the ratio of the difference between the target (here, the function maximum), and the result obtained by an algorithm, over the target value, expressed in percentage, at a specific number of generations (here, 500). Table 1 presents the Relative-Distance-to-Target for all three algorithms.

	Relative-Distance-to-Target	**Best-So-Far Fitness Value**
GA1	1.596%	77.4
GA2	1.456%	77.5
LEM	0.0381%	78.6

The function maximum: 78.643

Table 1. The Relative-Distance-to-Target for each algorithm at the 500th generation (at 10,000th birth) in seeking the maximum of function f_1.

To evaluate the performance of algorithms in another way, we determined the *δ-close number*, that is, the number of generations in the evolution process after which the Relative-Distance-to-Target of the solution produced by an algorithm reaches a given value, δ. Table 2 presents δ-close numbers for different values of δ and different algorithms. To achieve δ=0.038, LEM needed 327 generations, while GA1 and GA2 were not δ-close at 10,000th generation.

δ	0.038%	0.1%	1.0%	2.0%	3.0%	4.0%	5.0%	6.0%
GA1	UNS	UNS	UNS	420	212	120	96	74
GA2	UNS	UNS	1386	186	95	55	53	33
LEM	327	125	16	15	14	10	8	8

"UNS" means unsuccessful within 10,000 generations (200,000 births)

Table 2. The number of generations for the fitness value to become δ-close to the target.

By dividing the δ-close number for GA1 and GA2 by the δ-close number for LEM, we estimated the speed-up of LEM over GA1 and GA2, respectively. The speed-up of 10 means that LEM reaches the given δ-close fitness function value using 10 times fewer evolution generations than the algorithm with which it is being compared . Table 3 presents LEM speed-ups over GA1 and GA2 for different values of δ. One can see, for example, that for δ equal 0.1%, the speed-up of LEM over GA1 and GA2 was more than 80 (i.e., GA1 and GA2 did not produce a δ-close solution after 80 times more generations than LEM). For δ equal 1%, GA1 solution was not δ-close even after 625 times more generations than LEM1.

	LEM's speed-up for different δ							
δ	**0.038%**	**0.1%**	**1.0%**	**2.0%**	**3.0%**	**4.0%**	**5.0%**	**6.0%**
LEM/GA1	>>30.6*	>> 80*	>> 625*	28	15.14	12	12	9.25
LEM/GA2	>>30.6	>> 80	86	12.4	6.79	5.5	6.6	4

* GA1 and GA2 solutions have not become δ-close to the maximum within 10 000 generations;
">> N" means that if the solution was δ-close at 10 000th generation, the speedup would be N.

Table 3. LEM's speed-up over GA1 and GA2 for different values of δ in seeking the maximum of function f_1.

Problem 2. Testing the ability of an algorithm to find a solution (maximum) of a complex function.

$$f_2(x_i) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2, \qquad -2.048 \le x_i \le 2.048$$

Maximum: 3905.926. Minimum: 0

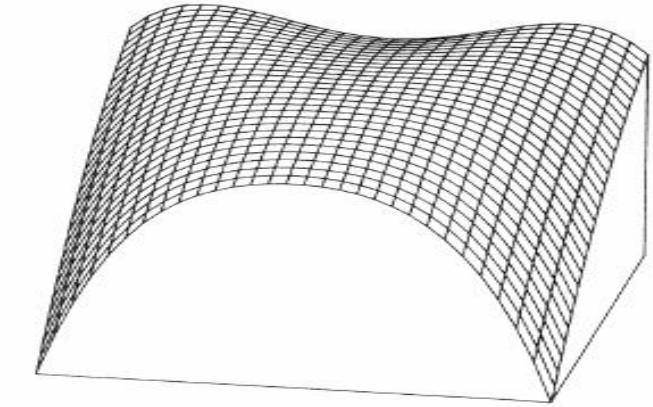

Figure 3. Inverted two-dimensional graph of f_2 (reprinted with permission of Kenneth DeJong)

Results from seeking the function maximum are presented in Figure 4, and Tables 4, 5 and 6.

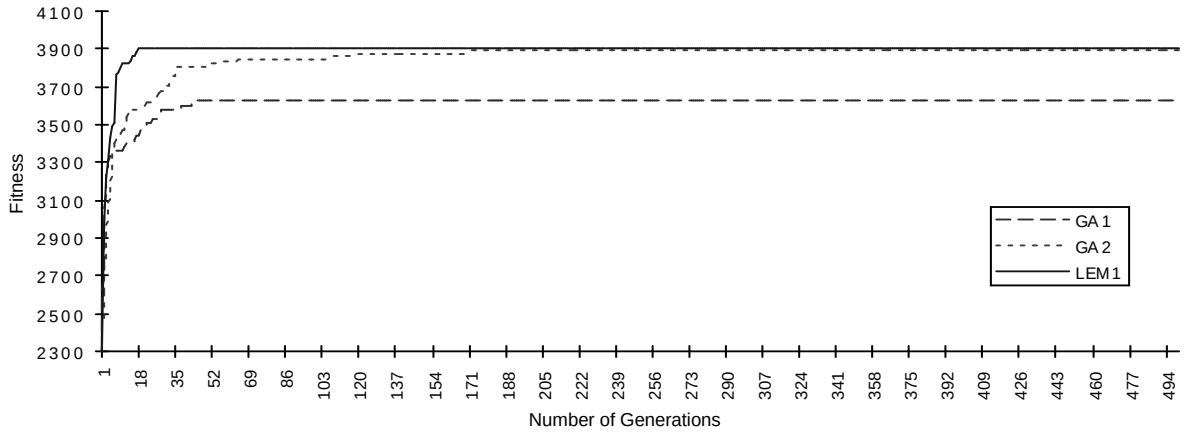

Figure 4. Evolution process within 500 generations (10000 births) for function f_2.

As shown in Figure 4, LEM's solution reached the absolute maximum much faster than GA1 and GA2. GA1 did not reach function maximum even within 500 generations.

	RelativeDistance-to-Target	**Best-So-Far Fitness Value**
GA1	7.2%	3625.777
GA2	0.380%	3891.084
LEM	0.096%	3902.177

Table 4. The Relative-Distance-to-Target at 500th generation (10000 births) for function f_2.

δ	**0.096%**	**0.1%**	**1.0%**	**2.0%**	**3.0%**	**4.0%**	**5.0%**	**6.0%**
GA1	UNS	UNS	UNS	UNS	UNS	UNS	UNS	UNS
GA2	UNS	UNS	120	56	56	34	33	28
LEM	452	152	17	14	10	8	8	8

"UNS" means unsuccessful within 10000 generations (200000 births)

Table 5. The δ-close numbers for different δ when maximizing function f_2.

	LEM Speed-up for different δ							
δ	**0.096%**	**0.1%**	**1.0%**	**2.0%**	**3.0%**	**4.0%**	**5.0%**	**6.0%**
LEM/GA1	>>22.1	>>66	>>588	>>714	>>1000	>>1250	>>1250	>>1250
LEM/GA2	>>22.1	>>66	7	4	5.6	4.25	4.125	3.5

* GA1 and GA2 solutions have not become δ-close to the maximum within 10 000 generations;

">> N" means that if they were δ-close at 10 000th generation, the speedup would be N.

Table 6. LEM's speed-up over GA1 and GA2 for different δ.

Problem 3: Testing the ability of an algorithm to find a solution (maximum) for a non-differentiable function f_3:

$$f_3(x_i) = \sum_{1}^{5} \text{integer}(x_i), \qquad -5.12 \le x_i \le 5.12$$

Maximum: 25; Minimum: -30

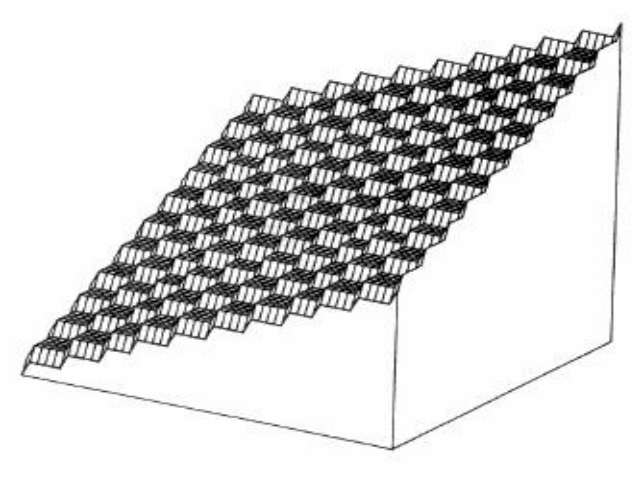

Figure 5. Inverted two-dimensional graph of f_3
(Reprinted with permission of Kenneth DeJong)

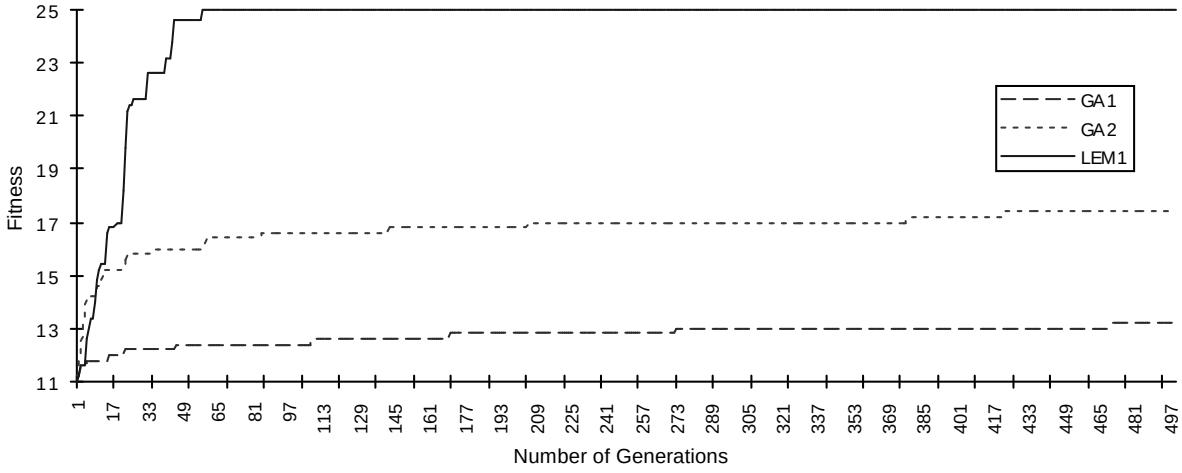

Figure 6. The evolution process within 500 generations (10000 births) for function f_3.

As Figure 6 shows, for Problem 3 (a non-differentiable function), LEM has very significantly outperformed GA1 and GA2. Its solution reached maximum after about 50 generations, while GA1's and GA2's solutions were far from the maximum even after 500 generations (Table 7).

	Relative Distance-to-Target	Best-So-Far Fitness Value
GA1	47.2%	13.2
GA2	30.4%	17.4
LEM	0.0%	25.0 (max)

Table 7. The Relative-Distance-to-Target within 500 generations (10000 births).

δ	.000%	1.0%	2.0%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
GA1	UNS	UNS	UNS	UNS	UNS	UNS	UNS	UNS	UNS
GA2	UNS	UNS	UNS	UNS	UNS	UNS	UNS	UNS	UNS
LEM	58	58	45	45	45	44	44	44	42

"UNS" means unsuccessful within 10,000 generations (200000 births)

Table 8. The δ-close number for different δ.

	LEM Speed-up for Different δ								
δ	0.0%	1.0%	2.0%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
GA1/LEM	>>172	>>172	>>222	>>222	>>222	>>227	>>227	>>227	>>238
GA2/LEM	>>172	>>172	>>222	>>222	>>222	>>227	>>227	>>227	>>238

*GA1 and GA2 solutions have not become δ-close to the maximum within 10,000 generations

">> N" means that if the solution was δ-close at 10 000th generation, the speedup would be N.

Table 9. LEM's speed-up over GA1 and GA2 for different δ.

Problem 4: Testing the ability of an algorithm to find a solution (maximum) of a function of a large number of continuous variables (30) and with added Gaussian noise (Figure 7).

$$f_4(x_i) = \sum_{1}^{30} i x_i^4 + \text{Gauss}(0,1), \qquad -1.28 \le x_i \le 1.28$$

Maximum: approximately 1248.225. Minimum: 0

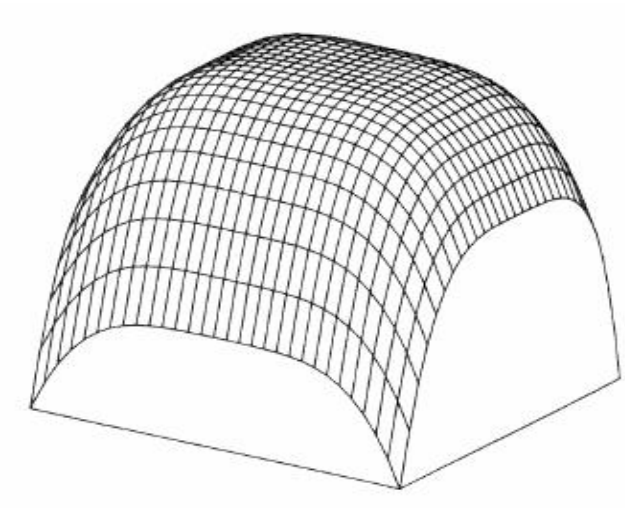

Figure 7. Inverted, two-dimensional graph of f_4
(Reprinted with permission of Kenneth DeJong)

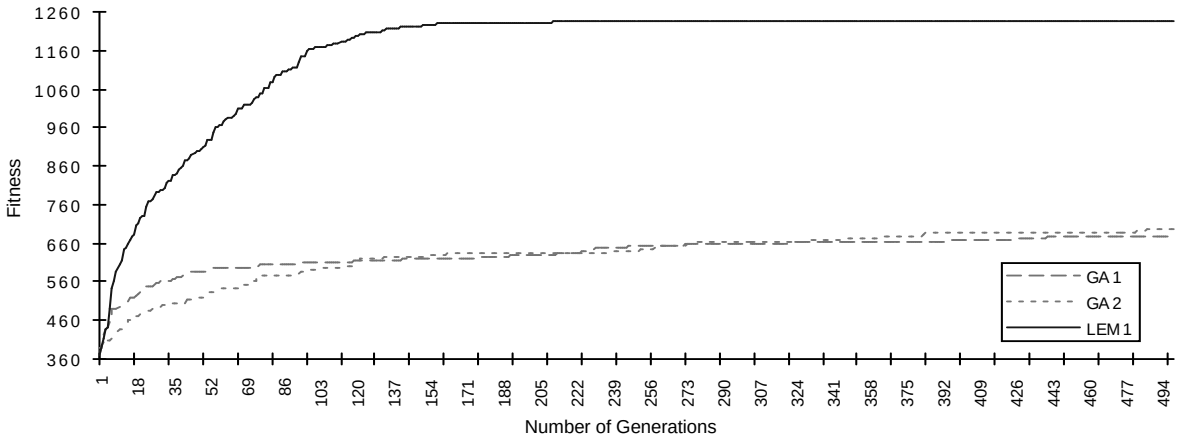

Figure 8. The evolution process within 500 generations (10000 births) for function f_4.

As one can see in Figure 8, for a problem with many variables plus noise, LEM very significantly outperformed GA1 and GA2. It came very close to the maximum already at 150th generation, while GA1 and GA2 were still over 44% away from the maximum after 500th generation.

	Relative-Distance-to-Target	**Best-So-Far Fitness Value**
GA1	45.77%	677.450
GA2	44.26%	695.710
LEM	1.006%	1235.702

Table 10. The Relative-Distance-to-Target during 500 generations (10000 births).

δ	**1.0%**	**2.0%**	**3.0%**	**4.0%**	**5.0%**	**6.0%**	**7.0%**	**8.0**
GA1	UNS	UNS	UNS	UNS	UNS	UNS	UNS	UNS
GA2	UNS	UNS	UNS	UNS	UNS	UNS	UNS	UNS
LEM	491	151	132	120	116	110	98	97

UNS means unsuccessful within 10000 generations (200000 births).

Table 11. The number of generations after which the fitness value becomes δ-close to the maximum.

	LEM Speed-up Ratio for Different δ							
δ	1.006%	2.0%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
GA1/LEM	>>20.37	>>66.23	>>75.76	>>83.33	>>86.21	>>90.91	>>102.0	>>103.1
GA2/LEM	>>20.37	>>66.23	>>75.76	>>83.33	>>86.21	>>90.91	>>102.0	>>103.1

* GA1 and GA2 solutions have not become δ-close to the maximum within 10 000 generations; ">> N" means that if they were δ-close at 10 000th generation, the speedup would be N.

Table 12. LEM speed-up over GA1 and GA2 for different values of δ.

Problem 5: Testing the ability of an algorithm to find a solution (maximum) of a multi-mode function.

$$f_5(x_i) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}, \qquad -65.536 \le x_i \le 65.536$$

Maximum: approximately 1.0. Minimum: approximately 0.002

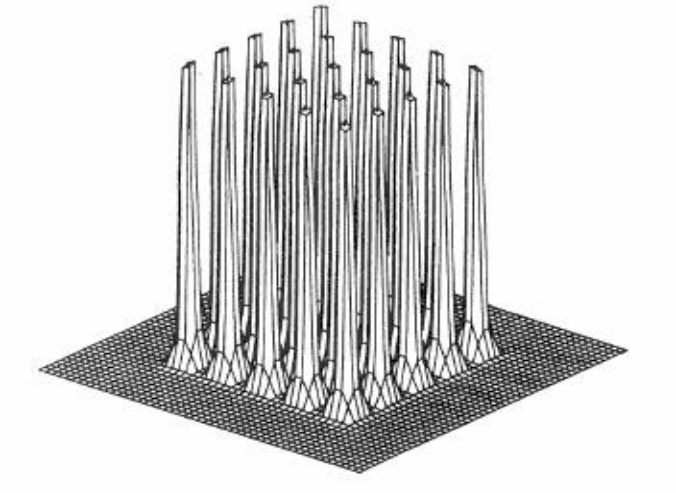

Figure 9. Inverted two-dimensional graph of f_5

(Reprinted with permission of Kenneth DeJong)

As shown in Figure 10, for problem 5, GA1 (which uses a multi-valued representation) performed very poorly. At an early stage of evolution, LEM (which employs GA1 for the genetic algorithm phase) did worse than GA2 (which uses a binary representation and can make small adjustments). After 500 generations, however, LEM reached the maximum, while GA2 was still 6.7% from maximum and remained so within 10,000 generations (Table 13).

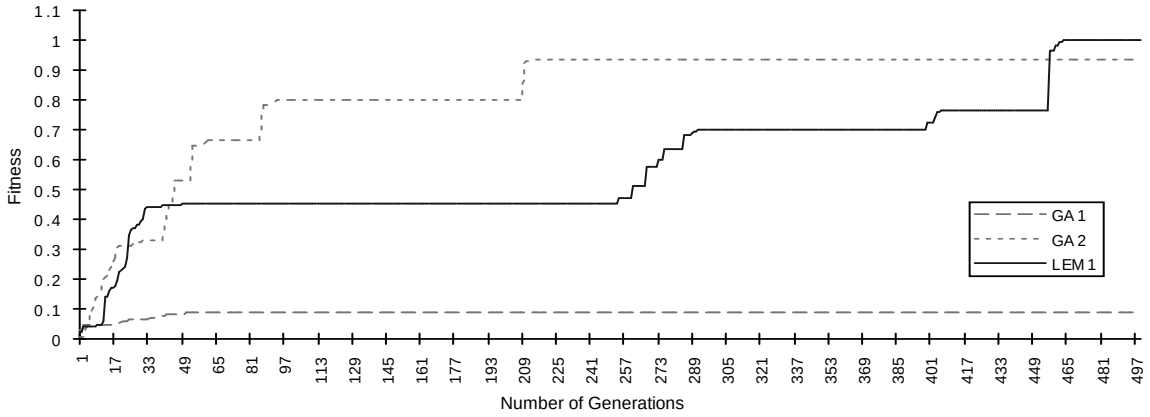

Figure 10: The evolution process within 500 generations (10000 births) for function f_5.

	Relative-Distance-to-Target	**Best-So-Far Fitness Value**
GA1	91.4%	0.086
GA2	6.8%	0.934
LEM	0.0%	1.002

Table 13. Relative-Distance-to Target at 500 generation (10000 births) for function f_5.

δ	**0.0%**	**1.0%**	**2.0%**	**3.0%**	**4.0%**	**5.0%**	**6.0%**	**7.0%**
GA1	UNS	UNS	UNS	UNS	UNS	UNS	UNS	UNS
GA2	UNS	UNS	UNS	UNS	UNS	UNS	UNS	214
LEM	465	462	460	460	457	457	457	457

"UNS" means unsuccessful within 10000 generations (200000 births)

Table 14. The number of generations at which fitness value became δ-close to the maximum.

	LEM's speed-up ratio for different δ							
δ	**0.0%**	**1.0%**	**2.0%**	**3.0%**	**4.0%**	**5.0%**	**6.0%**	**7.0%**
GA1/LEM	>>21.51	>>21.66	>>21.7	>>21.7	>>22	>>22	>>22	>>22
GA2/LEM	>>21.51	>>21.66	>>21.74	>>21.74	>>21.88	>>21.88	>>21.88	>>0.47

* GA1 and GA2 solutions have not become δ-close within 10,000 generations (when we stopped the experiment)

">> N" means that if the solution was δ-close at 10,000th generation, the speedup would be N.

Table 15. LEM's speed-up over GA1 and GA2 for different δ for problem 5.

4.3 Discussion of Results

In all experiments, LEM1 clearly outperformed GA1 and GA2. In some cases, in particular for Problem 3 (a non-differentiable function) , and Problem 4 (thirty variables plus noise), LEM's performance was by far better than GA1 and GA2. These algorithms could not reach the solution even after 10,000 generations, while LEM reached it in about 50 generations for f3, and 200

generations, for f4. GA2, which uses binary representation, as compared to GA1, which uses multivalued representation, is capable of making finer mutations and cross-overs, and often performes better.

Therefore, it can more quickly reach the global optimum if the function surface is relatively smooth. On the other hand, GA1 is more likely to generate new individuals which are very different from their parents. The seems to explain why, in the case of function f5 (that require small departures from parents), GA2 did significantly better than GA1, and, at the early stages of evolution, better than LEM1 (which employs GA1 for the genetic algorithm phase). At the later evolution stages, LEM was, however, able to reach the maximum, while GA2 could not.

5 EXPERIMENTAL STUDY 2: MORPHOGENETIC SYSTEMS

5.1 Problem description

The objective of this study was to apply LEM as a tool in the development of morphogenetic systems. By a morphogenic system we mean a system that is "grown" from "chromosomes," in analogy to biological systems, such as plants and animals.

In this study, LEM was applied to "grow" a task-oriented neural net in a cellular automata space. To this end, we employed a program that allows a user to grow neural nets in a simulated cellular automata machine. The program, which we call here for short, NC (a neural net in the cellular automata simulator), was developed by Dr. Hugo de Garis and his collaborators at the ATR Brain Building Laboratory in Japan (the BRAIN-CAM method). NC uses a standard genetic algorithm for implementing a morphogenetic process within a cellular automata system. To provide a context for this study, let us briefly describe major steps in the NC method :

(1) Design a fitness function that characterizes the task system (that is, a neural net in the cellular automate space that performs a desirable function or class of functions)

(2) Randomly generate a population of individuals, called chromosomes (in the first implementation of the system, chromosomes were integer strings, in the next implementation, they are arrays. In our experiments, we used the second implementation) .

(3) Use chromosomes to grow a neural net in the cellular automate space , using hand-crafted rules that govern the growth of a neural net based on the "genes" in chromosomes.

(4) Calculate the fitness value of each neural net in a population, and assign this fitness value to the chromosome that led to the generation of the corresponding net.

(5) Sort in ascending order the chromosomes that generated neural nets, according to the fitness value of these nets. Replace the half of the chromosomes with the lowest fitness value by the half with highest fitness value.

(6) Randomly select from the generated population pairs of individuals and perform the crossover operation. Randomly determine one cutting point, and exchange portions between two individuals around this point.

(7) Mutate each chromosome so obtained.

(8) Repeat the sequence of steps 3-7, until the best fitness value of a chromosome is above a given threshold, or the number of iterations (generations) reaches a predefined threshold.

In the current implementation of NC, 30% individuals in the new generation were generated by crossover, and the rest were copied from the parent generation. The probability of mutating a gene in a chromosome was set to 0.02.

5.2 The NC-LEM system: Combining NC with LEM

In this study we incorporated LEM in the NC system in order to speed-up the process of evolving task-oriented systems. The general idea was to use symbolic learning to determine the "reasons" why some chromosomes produce systems performing a given task better, and other chromosomes produce systems performing it worse. These reasons are expresed in the decision rules generated by a symbolic learning system (in the study AQ15), using data from experiments performed. The rules are then used to generate a "better" population of chromosomes in the next generation of the evolutionary process. Below is a description of the first version of the system, which we call for short NC-LEM.

(1) Randomly generate the first generation of chromosomes for evolution.

(2) Execute the genetic-based system growing process (NC): if the fitness of the best individual in the population is not improved by the given *gen-threshold*, in consecutive *gen-length* generations, transfer control to the symbolic learning mode (AQ)

(3) Execute the *symbolic learning mode*:

--- Determine HIGH and LOW solutions in the current population. The HIGH solutions are HT% (HT is a threshold < 50%) best solutions in the population, and the LOW solutions are LT% (LT is a threshold < 50%) worst solutions. The "best" and "worst" are determined on the basis of a given fitness function.

--- Apply a symbolic learning method (e.g., the AQ-type) for determining rules distinguishing HIGH and LOW solutions. Each gene (a cell in the CA matrix--see below) in a chromosome is considered to be an attribute with K values (there are K types of genes).

--- Generate a new population of solutions by replacing not-HIGH individuals by those satisfying the learned rules (the selection of such individuals is random or done according to some selection rules).

--- Continue the process as long as the best solution within a sequence of *learn-length* iterations is better by the *learn-threshold* than the previously found best solution

(4) Repeat the process from step 2. Continue switching between (2) and (3) until the termination condition is met (e.g., the solution is satisfactory, or the allocated computational resources are exhausted).

The CA matrix is generally a 3D cube (a 3D chromosome). In the pilot experiments, CA was a 12x12x1 cube, that is, a two-dimensional matrix. Thus, the number of genes was 144, and the number of gene types was K= 128 (a nominal attribute). The gene types represent different types of actions in the cellular automata space. Thus, the total representation space was 128^{144}.

5.3 Results from Study 2

In this study, NC-LEM was applied to the problem of evolving a "timer" circuit. The desired output from the circuit during 100 clock cycles is to be 0 for the first 40 clocks, 1 for the next 30 clocks, and 0 again for the remaining 30 clocks. Two populations were used in the experiments, one of size 10, and second of size 20. For each population size, NC system (marked as CA) and NC-LEM (marked as LEM) were executed. Figure 11 depicts the evolution process with population size of 10.

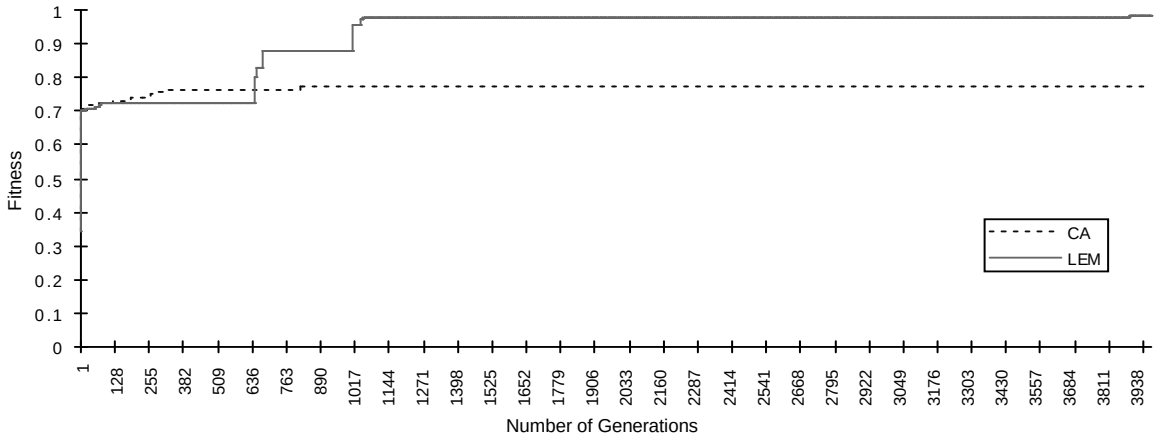

Figure 11. Evolving a timer circuit within 4000 generations with the population size of 10.

As shown in Figure 11, NC-LEM was initially “stuck” for many generations (about 640) at a local optimum, but then relatively quickly moved toward the correct design. The NC was at first gradually improving on the initial solution, but then got “stuck” toward the end of the experiment (at 4000 generations) at the local optimum, and did not reach the correct design.

Figure 12 shows the result for the population size of 20. In this case, NC-LEM reached a near-perfect solution after about 300 generations, while NC reached a similar solution after about 1700 generations.

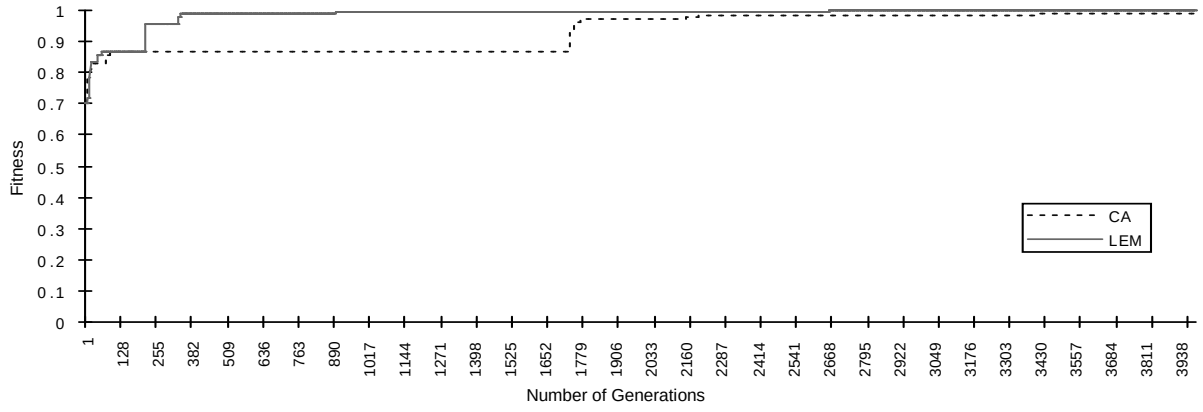

Figure 12. Evolving a timer circuit within 4000 generations with a population size of 20.

δ	0%	0%	0%	0%	0%
NC	UNS	7371	7371	7371	7371
NC-LEM	UNS	3916	1018	1018	679

"UNS" means unsuccessful within 10,000 generations.

Table 16. The number of generations for the fitness function become δ-close to maximum with a population size of 10.

NC-LEM Speed-up for Different δ							
δ	0.0%	1.0%	2.0%	3.0%	5.0%	8.0%	11.0%
NC-LEM/NC Speed-up	UNS	UNS	2	7	7	7	11

"UNS" means unsuccessful within 10,000 generations.

Table 17. NC-LEM/NC speed-up when the fitness value becomes δ-close to the maximum with a population size of 10.

δ	0.0%	1.0%	2.0%	3.0%	4.0%	5.0%	6.0%	7.0%
NC	966	443	209	790	765	765	749	747
NC-LEM	678	346	346	339	339	220	220	220

Table 18. The number of generations needed for the fitness to become δ-close to maximum with a population size of 20.

LEM Speed-up Ratio for Different δ								
δ	0.0%	1.0%	2.0%	3.0%	4.0%	5.0%	6.0%	7.0%
NC-LEM/NC Speed-up Ratio	1.8	10	6	5.3	5.2	8	8	8

Table 19. NC-LEM/NC speed-up when the fitness value becomes δ-close to maximum with a population size of 20.

The above results indicate that the NC-LEM system has significantly outperformed the NC system. In many cases, the speedup ratio was by nearly one order of magnitude. In these experiments we used, however, a very rudimentary implementation of NC-LEM. This initial NC-LEM could be substantially improved be developing a better method for representing chromosomes, and applying a better way of generating new individuals using the learned rules, as well as improving the rule learning process. This study suggests that LEM could be useful for the development of morphogenetic systems.

6 RELATION TO OTHER WORK

The described LEM methodology is an original development, and, to the authors' knowledge, has not been described elsewhere. There has been, however, somewhat related work, for example,

Grefenstette (1991) described a genetic learning system, SAMUEL, designed for sequential decision problems.

The SAMUEL implements a form of "Lamarckian feature", a localized operation, concerned with modifying an individual in a population. In contrast, LEM uses a symbolic learning system to determine general patterns characterizing a *set* of well-performing individuals, and then employs these patterns to improve the new generation. Thus, LEM performs a global operation, which utilizes lessons from the experience of a population or populations of individuals.

Another somewhat related work was done by Vafaie and De Jong (1991), who used a standard genetic algorithm to improve rules produced by an inductive learning system (AQ).

7 CONCLUSION

The experiments presented here have demonstrated a significant promise of the LEM methodology as a basis for developing a new type of evolutionary learning systems. There are, however, many unanswered questions regarding the LEM evolutionary learning methodology. These include a need for a systematic theoretical and practical investigation of this methodology, working with different parameters, understanding of the trade-offs it represents, studying its performance in different modes of operation (learning mode only vs. genetic algorithm mode only, vs. various combinations of these modes), and determining the type of tasks for which it will likely be successful.

The initial system, LEM1, could be improved by employing a more advanced symbolic learning method (e.g., AQ18 rather than AQ15; Michalski, 1998; Bloedorn et al. 1998; Bloedorn and Michalski, 1998), and applying a more advanced method for generating populations individuals using the learned rules.

One of the major aspects of the LEM methodology is that it requires a symbolic learning system that can learn discriminant descriptions of groups of individuals in a population. If individuals are represented by attribute value vectors or Horn clauses, then existing attributional learning or inductive logic programming methods can be applied. If descriptions are more complex, e.g., are hierarchies of operators representing a computer program, as in evolutionary programming (Koza, 1994), then one needs a learning program able to work with such representations. The AQ-type algorithms could work with such problems, if an appropriate representation of the input data, generated descriptions, and generalization operators were developed.

Another interesting research task is to investigate the applicability and performance other learning approaches in the LEM's inductive learning step, for example, decision tree learning or neural nets.

Summarizing, the LEM methodology and its LEM1 implementation represent a novel way of integrating symbolic learning with evolutionary computation. The preliminary results provide a strong justification for conducting further investigations in this direction.

REFERENCES

Ackley, D. and Littman, M. “Interactions between learning and evolution,” In C.G. Langton, C. Taylor, J.D. Farmer, and S. Rasmussen (eds), *Artificial Life II*, Addison-Wesley, 1992.

Baldwin, J.M. “A New Factor in Evolution,” *American Naturalist* , vol 30, pp.441-451, 536-553, 1896.

Blickle, T. Teich, J., and Thiele L., “System-Level Synthesis using Evolutionary Algorithms”, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), TIK-Report, Nr. 16, April 1996.

Clark, P. and Niblett, T. The CN2 induction algorithm. Machine Learning Journal, 3(4):261--283, 1989.

de Garis, H. “CAM-BRAIN: The Evolutionary Engineering of a Billion Neuron Artificial Brain by 2001 Which Grows/Evolve at Electronic Speeds Inside a Cellular Automata Machine (CAM)”, Lecture Notes in Computer Science - Towards Evolvable Hardware, Vol. 1062, pp 76-98, Springer-Verlag, 1996.

de Garis, Hugo “Evolvable Hardware Workshop Resort”, Technical Report, ATR Human Information Processing Research Laboratories, Evolutionary Systems Department, Kyoto, Japan, 1996.

de Garis, Hugo, “CAM-BRAIN: The Evolutionary Engineering of a Billion Neuron Artificial Brain by 2001 Which Grows/Evolve at Electronic Speeds Inside a Cellular Automata Machine (CAM)”, *Lecture Notes in Computer Science -Towards Evolvable Hardware*, Vol. 1062, pp. 76-98, Springer-Verlag, 1996.

Goldberg, D.E., “Genetic Algorithms in Search, Optimization & Machine Learning”, Addison-Wesley Publishing Company, 1989.

Grefenstette, J. “Lamarckian Learning in Multi-agent Environment,” *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. Belew and L. Booker (Eds.), San Mateo, GA: Morgan Kaufmann, pp. 303-310, 1991.

Grefenstette, J. “Lamarckian Learning in Multi-agent Environment”, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, GA: Morgan Kaufmann, p303-310, 1991.

Higuchi, T., Iwata, M., Kajitani, I., Iba, H., Hirao, Y. Furuya, T., Manderick, B. “Evolvable Hardware and Its Applications to Pattern Recognition and Fault-Tolerant Systems”, Lecture Notes in Computer Science - Towards Evolvable Hardware, Vol. 1062, pp 118-135, Springer-Verlag, 1996.

Hinton. G.E., and Nowlan, S.J. “How learning can fuide evolution,” *Complex Systems* 1: 495-502, 1987.

Holland, John H., “Adaptation in Natural and Artificial Systems”, Ann Arbor: The Univesity of Michigan press, 1975.

John H. Holland, “Adaptation in Natural and Artificial Systems”, Ann Arbor: The Univesity of Michigan press, 1975.

Jun’ichi Mizoguchi, Hitoshi Hemmo and Katsunori Shimohara, “Production Genetic Algorithms for Automated Hardware Design through an Evolutionary Process”, IEEE Conference n Evolutionary Computation, 1994.

Kitano, H. “Morphogenesis of Evolvable Systems”, Lecture Notes in Computer Science - Towards Evolvable Hardware, Vol. 1062, pp 99-117, Springer-Verlag, 1996.

Kitano, H. “Morphogenesis of Evolvable Systems”, Lecture Notes in Computer Science - Towards Evolvable Hardware, Vol. 1062, pp 99-117, Springer-Verlag, 1996.

Koza, John R., Bennet, Forrest H. III, Andre, David, Keane, Martin A., “Towards Evolution of Electronic Animals Using Genet.ic Programming”, Artificial Life V Conference in Nara, Japan, May 16-18, 1996.

M. Tomasini, “Evolutionary Algorithms”, Lecture Notes in Computer Science - Towards Evolvable Hardware, Vol. 1062, pp 19-47, Springer-Verlag, 1996.

Michalewicz, Z. “Genetic Algorithms + Data Structures = Evolution Programs,” Springer -Verlag, 1994.

Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1994.

Michalski, R. S., "On the Quasi-Minimal Solution of the General Covering Problem," *Proceedings of the V International Symposium on Information Processing* (FCIP 69), Vol. A3 (Switching Circuits), pp. 125-128., Yugoslavia, Bled, October 8-11, 1969.
Michalski, R.S., LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning, *Machine Learning*, December, 1998b (submitted).
Michalski, R.S., LEARNABLE EVOLUTION: Combining symbolic and Evolutionary Learning. Initial Results, Proceedings of the Fourth International Workshop on Multistrategy Learning (MSL 98), Floriana Esposito, Ryszard S. Michalski, and Lorenza Saitta (eds.), pp. 14-20, Desenzano del Garda, Italy, June 11-13, 1998a (accepted for publication).

Michalski, R.S., Mozetic, I., Hong, J., and Lavrac, N., “The Multipurpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains,” *Proceedings of the Fifth National Conference on Artificial Intelligence*, August, Philadelphia, PA., pp. 1041-1045, 1986.

Mizoguchi, J. Hemmo, H. and Shimohara, K., “Production Genetic Algorithms for Automated Hardware Design through an Evolutionary Process”, *IEEE Conference n Evolutionary Computation*, 1994.

Mondada, F., Floreano, D., “Evolution and Mobile Autonomous Robotics”, *Lecture Notes in Computer Science - Towards Evolvable Hardware*, Vol. 1062, pp 221-249, Springer-Verlag, 1996.

Mondada, F.,Floreano, D., "Evolution and Mobile Autonomous Robotics", Lecture Notes in Computer Science - Towards Evolvable Hardware, Vol. 1062, pp 221-249, Springer-Verlag, 1996.

Quinlan, J.R., "C4.5: Programs for Machine Learning", Morgan Kaufmann, Los Altos, California, 1993.

Sanches, E. "Field Programmable Gate Array (FPGA) Circuits", *Lecture Notes in Computer Science-Towards Evolvable Hardware*, Vol. 1062, pp 1-18, Springer-Verlag, 1996.

Sanches, E., "Field Programmable Gate Array (FPGA) Circuits", Lecture Notes in Computer Science - Towards Evolvable Hardware, Vol. 1062, pp 1-18, Springer-Verlag, 1996.

T. Higuchi, M. Iwata, I. Kajitani, H. Iba, Y. Hirao, T. Furuya, B. Manderick, "Evolvable Hardware and Its Applications to Pattern Recognition and Fault-Tolerant Systems", Lecture Notes in Computer Science - Towards Evolvable Hardware, Vol. 1062, pp 118-135, Springer-Verlag, 1996.

Thompson, A., "Evolving Electronic Robot Controllers That Exploit Hardware Resources", *Proceedings of the 3rd European Conference on Artificial Life*, Springer Verlag, 1995.

Thompson, Adrian "Evolving Electronic Robot Controllers That Exploit Hardware Resources", Proceedings of the 3rd European Conference on Artificial Life, Springer Verlag, 1995.

Tobias Blickle, Jurgen Teich, Lothar Thiele, "System-Level Synthesis using Evolutionary Algorithms", Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), TIK-Report, Nr. 16, April 1996

Tomasini, M., "Evolutionary Algorithms", *Lecture Notes in Computer Science-Towards Evolvable Hardware*, Vol. 1062, pp 19-47, Springer-Verlag, 1996.

Wnek, J., Kaufman K., Bloedorn, E. and Michalski R.S., "Inductive Learning System AQ15c: The Method and User's Guide," *Reports of the Machine Learning and Inference Laboratory*, MLI 95-4, George Mason University, Fairfax, VA, March 1995.