An Application of Lamarckian Evolution Model to Function Optimization

R. S. Michalski Q. Zhang

P 98-4
MLI 98-3

An Application of Lamarckian Evolution Model to Function Optimization

Ryszard S. Michalski* Qi Zhang

Machine Learning and Inference Laboratory
School of Information Technology and Engineering
George Mason University
Fairfax, Virginia 22030-4444

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

Publication No
P 98-4

Reports of the Machine Learning and Inference Laboratory
MLI 98-3

January 1998

ABSTRACT

This paper describes an application of Lamarckian Evolution Model (LEM) to function optimization. This model, proposed by Michalski, is inspired by a simple observation that acquisition of patterns or descriptions of the elites in populations and hence guiding the generation of new populations according to learned patterns may lead to a much more effective evolution, in terms of the evolution speed and the ability of reaching a goal, than a natural evolution. As a validation of LEM, a series of experiments were conducted to compare LEM and GAs in their performance to function optimization. Results have strongly indicated the effectiveness of LEM.

Keyword: Lamarckian Evolution, Genetic Algorithms, Function Optimization.

ACKNOWLEDGMENTS

This research was conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's activities are supported in part by the Defense Advanced Research Projects Agency under grant F49620-95-1-0462, administered by the Air Force Office of Scientific Research, in part by the National Science Foundation under grants DMI-9496192 and IRI-9020266, and in part by the Office of Naval Research under grant N00014-91-J-1351

1 INTRODUCTION

Natural evolution is a fascinating phenomenon studied over centuries by scientists. Some of them were trying to describe it in an algorithmic view. The conception of computers provided a powerful tool for implementing and testing this view. In 1960s, computer scientists in several institutions began their independent studies, in different perspectives, of this phenomenon by using computers for modeling. This marked the conception of a new computational model, now widely called evolutionary computation. Genetic algorithms (GAs), one of the approaches, are viewed by many a key element in designing and implementing robust adaptive systems, GAs have been applied to many practical problems and promising results have been obtained.

Starting with an initial population of individuals, a genetic algorithm generates new populations algorithms by applying operators such as selection, crossover, and mutation to the current population. Traditional genetic algorithms are basically Darwinian-type. There are two potential problems with it. Firstly, the search process is semi-blind: the crossover is a semi-random combination of two candidate solutions; the mutation is a random change of the newly-generated solution; the survival of the fittest is a form of parallel hill-climbing. In this form of evolution, individuals cannot pass to the next generation the lessons learned from their experience. Consequently, computational processes based on Darwinian evolution are not efficient. A low efficiency is the major obstacle in the applications of genetic algorithms to very large, or ultra-complex problems. Secondly, because of its semi-blindness, an evolution process is easy to get stuck around a locally optimal solution and quite likely unable to escape such a area in a search space. This situation in turn worsens the low efficiency problem.

To attack such problems, Michalski proposed a novel evolutionary computation model, called *Lamarckian evolution model* (henceforth, LEM). A LEM employs along the standard genetic operators also machine learning operators. The basic idea is that a machine learning system is used to determine "reasons" why certain individuals in a population are better than others in performing a specific task. These reasons, expressed as rules, are then used for creating a new generation of individuals probabilistically.

This report describes the LEM algorithm and its application to function optimization. A series of experiments were conducted on the set of five benchmark functions (De Jong 1975; Goldberg 1989). The results obtained are very promising. In our experiments, LEM has shown a dramatically improved performance over the conventional genetic algorithms, frequently a two or

three orders of magnitude speed up of an evolution process. In some cases, LEM achieved the optimal solution while genetic algorithms employed in the study could not come even close to it.

The following is organized as follows: section 2 describes 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 an overview of Lamarckian-related genetic evolution; section 6 concludes this report and briefs future work.

2 LEM Algorithm

To attack the semi-blindness problem of traditional genetic algorithms, LEM introduces symbolic learning into an evolution process. In particular, individuals of new generations are created in some generations according to hypotheses produced by symbolic learning. When LEM is in the symbolic learning mode (actually traditional genetic algorithms are also adopted in LEM), the "reasons" why some individuals are better than others in performing a given task are created by a symbolic learning method. These reasons are expressed as a set of rules. The ruleset is then used to create a new generation of individuals that are likely to have a higher "fitness", i.e., represent a better problem solution. A general description of LEM is given in Table 1.

In the algorithm, an individual is a problem solution (e.g., a vector of real numbers for a function). A 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. In the genetic algorithm mode, any genetic algorithm (standard or improved, simple or complex) can be adopted; the better performance it has, the better performance LEM should have. In the symbolic learning mode, any symbolic learning algorithm such as AQ (Michalski 1969, Michalski et al. 1986), CN2 (Clark and Niblett, 1989) and C4.5 (Quinlan, 1993) is applicable here. For AQ-type symbolic learning methods discretization of continuous attributes is needed before learning. When generating individuals according to learned rules, a selection mechanism is needed because usually a large number of candidates are available. New individuals can be selected from among those satisfying the ruleset either *randomly* or according to *heuristics*.. When designing (also confirmed in our experiments), we realize that adoption of a symbolic learning method without incorporation of a genetic algorithm would quite possibly make evolution get stuck around a locally optimal solution. With the introduction of a genetic algorithm, LEM is equipped with a more powerful search capability because of the semi-blind search of a genetic algorithm. This is very important especially when only a few candidates satisfy the ruleset and all of them fall within a locally optimal area.

(1) Randomly or according to prior knowledge, generate an initial population of individuals.

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

(2.1) **While** (the best individual in a sequence of *gen-length* generations is better by the *gen-threshold* than the best one found in previous generations)

Execute the *genetic algorithm mode* by :

(2.1.1) Conducting evolution in the same way as a genetic algorithm by using selection, crossover and mutation and other operators.

(2.2) **While** (the best individual in a sequence of *learn-length* generations is better by the *learn-threshold* than the best one found in previous generations)

Execute the *symbolic learning mode* by :

(2.2.1) Collecting HIGH and LOW individuals in the population. The HIGH individuals are HT% (HT < 50) best individuals in the population, and the LOW individuals are LT% (LT < 50) worst individuals. The best and worst individuals are determined according to their fitness values for a given task or problem.

(2.2.2) Applying a symbolic learning method for determining rules distinguishing HIGH and LOW individuals.

(2.2.3) Generating a new population of individuals by replacing LOW individuals by those satisfying the learned rules for HIGH individuals.

(2.2.4) Evaluating the individuals in the new generation.

Table 1. A general version of LEM algorithm.

Also in the algorithm, there are several parameters such as *gen-length, gen-threshold, learn-length, learn-threshold,* HT, LT. They should be determined experimentally for a given class of problems, or by a prior analysis.

3 ALGORITHMS IN EXPERIMENTS

3.1 LEM

In our function optimization experiments, an individual is a vector of real numbers which leads to a function value which is in turn used in the definition of a fitness function. The termination condition is defined as an evolution up to 10,000 generations. The population size is 20.

The genetic algorithm used in LEM in the experiments is a simple one which employs a real-valued representation, a selection operator based on a roulette wheel, a uniform crossover operator in which a gene with 0.5 probability inherits a gene (i.e., a variable value) from either parents, a mutation operator in which each gene is mutated with the probability of 1/L, where L is the number of genes. When mutated, a gene is either incremented or decreased by a small amount 0.1. All the settings in this genetic algorithm is pretty common for such a algorithm. In fact, we imported this algorithm and the other one (see the next subsection) from De Jong (De Jong 1998) without any change.

Regarding the symbolic learning method, we used the AQ rule learning method (Michalski 1969, Michalski et al. 1986) due to its powerful representation capabilities, and high learning capabilities. Since discretization is needed for continuous attributes in this method, we discretized each variable (i.e., attribute) uniformly into 200 intervals. We also adopted a heuristics called *Maximum Distance to Negatives (MDN)*. This heuristic suggests that before selecting a variable value according to a learned ruleset, the distribution pattern of HIGH and LOW individuals on this variable be identified and selection of a value far away from LOW individuals on this variable be given more probability. We identified 5 patterns in our experiments. Suppose "+" represents a value from a HIGH individual on a variable and "-" a value from a LOW individual on the same variable. The patterns are given in Table 2.

(1): - - - - - - + + + + +
(2): + + + + + - - - - - -
(3): - - - + + + + + - - -
(4): + + + - - - - - - + +
(5): - - + - + - + + + - -

Table 2. Distribution patterns of HIGH and LOW individuals on a variable.

For pattern (5) in Table 2, we adopted a uniform probability of selecting a variable value from the 5 available "+" values. In other cases, a polynomial curve-liked probability distribution (e.g., x^2 or

x^3) is given to available values. For pattern (1), for example, the "+" value on the rightmost has the highest probability of being selected.

In the experiments, both *gen-length* and *learn-length* are set to 3 and both *learn-threshold* and *gen-threshold* are set to 0.01. HT and LT are set to 30 (i.e., 6 best individual and 6 worst individuals are used for learning).

3.2 GA1

This is the same genetic algorithm as used in LEM described in the above subsection.

3.3 GA2

We tried to obtain good results from genetic algorithms so that we could have a fair comparison. Hence, we tested another genetic algorithm on all the problems. This genetic algorithm, GA2, uses a binary representation, and employs the same selection scheme as in GA1, a crossover operator in which a new gene inherits from the first-selected parent with the probability of 0.95 and from the other 0.05, and the same mutation scheme as in GA1. When mutation, the value of a gene is changed from 1 to 0 or vice verse. The resolution of each variable is set to .0001. Same as GA1, GA2 was also imported from De Jong and we kept all the original parameter settings.

4 RESULTS AND DISCUSSION

4.1 Problems

In the experiments we used five problems presented in (De Jong, 1975; Goldberg, 1989) that have been developed to test the performance of genetic algorithms. These problems test the ability of an algorithm to find a solution for different classes of optimization tasks (simple and complex, continuos and discrete, with few and many variables, with noise and without noise).

All three algorithms mentioned above were applied to seeking the function maximum, as well as the function minimum. In the case of the function maximum, a fitness of an individual (i.e., a vector of variable values) is defined as its corresponding function value. In the case of the function minimum, the fitness of an individual is also defined as its corresponding function value. However, for functions which can take on negative values, we added a fixed positive number to an

individual's function value to make sure the positiveness of its fitness value. This is because we used a roulette-wheel based selection scheme.

In this report, we present the results only from seeking the function maximum (the results from seeking the function minimum are very similar). Each algorithm was run 10 times on each problem and the average is reported in the following tables and figures.

4.2 Results

To evaluate performance, we defined the concept of *δ-close number,* that is, the number of generations in the evolution process at which the best individual obtained so far is δ distant to the goal (function maximum or function maximum). Here we used as δ the percentage of this distance over the whole range of function values. Another comparison is *speed-up ratio* which is defined as the ratio of two generation numbers when two algorithms are equally distant to the goal.

Problem 1: Testing the ability to find a solution for a simple function of three continuos arguments.

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

Maximum: 78.643; Minimum: 0

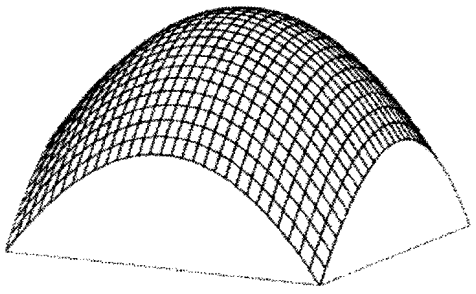

Figure 1. Inverted two-dimensional graph of function f_1
(Reprinted with the permission of Kenneth De Jong)

Figure 2 presents results from applying GA1, GA2 and LEM to function f_1 (the case of seeking function maximum). The results show that LEM's solution approached the maximum much faster than GA1 and GA2, specifically, within about 20 generations; while solutions from GA1 and GA2 were not close to the maximum even at 500th generation.

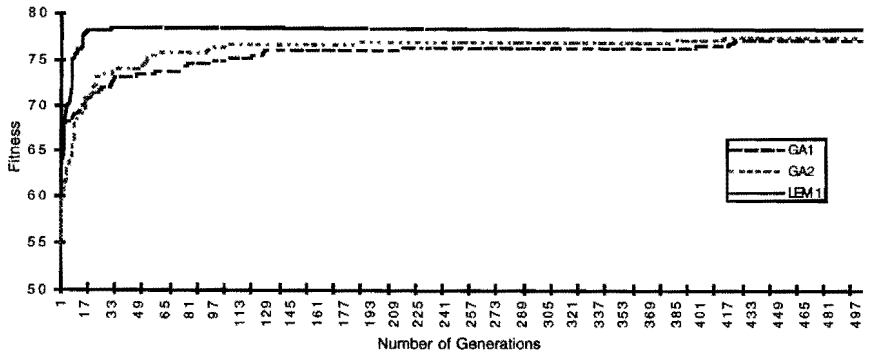

Figure 2. The evolution process within 500 generations for function f_1.

Table 3 presents δ-close numbers for different values of δ and algorithms. For example, 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 3. The number of generations for the fitness value become δ-close to the goal.

Table 4 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 LEM.

	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 4. LEM's speed-up over GA1 and GA2 for different values of δ.

Problem 2: Testing the ability to find a solution 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

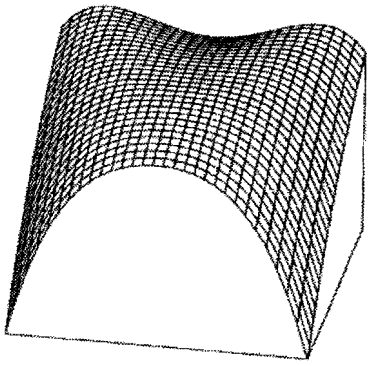

Figure 3. Inverted two-dimensional graph of f_2
(Reprinted with permission of Kenneth De Jong)

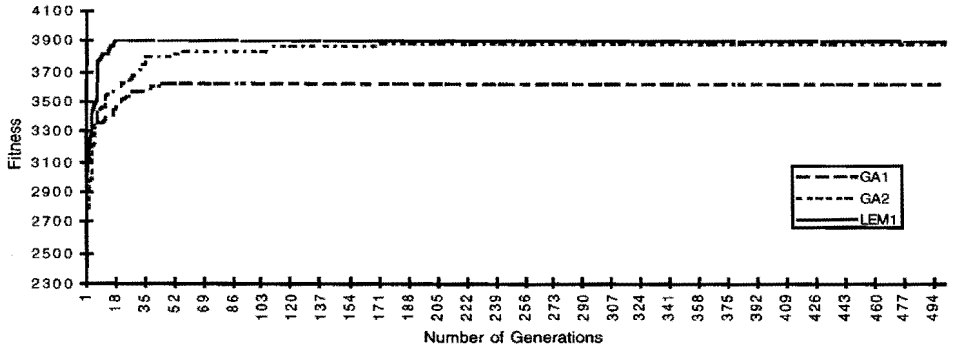

Figure 4. The evolution process within 500 generations for function f_2.

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 10,000 generations (200,000 births)

Table 5. The number of generations for the fitness value become δ-close to the goal.

	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 the solution was δ-close at 10 000th generation, the speedup would be N.

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

Problem 3: Testing the ability to find a solution 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

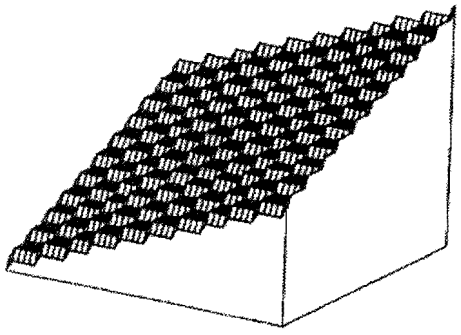

Figure 5. Inverted two-dimensional graph of f_3
(Reprinted with permission of Kenneth De Jong)

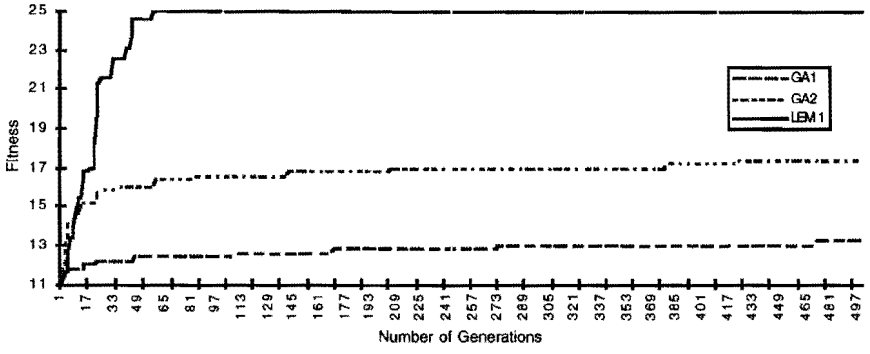

Figure 6. The evolution process within 500 generations for function f_3.

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).

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 (200,000 births)

Table 7. The number of generations for the fitness value become δ-close to the goal.

	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 8. LEM's speed-up over GA1 and GA2 for different values of δ

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

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

Maximum: approximately 1248.225. Minimum: 0

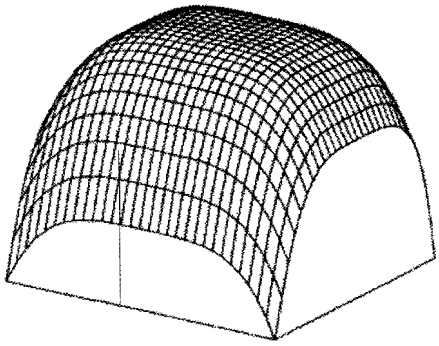

Figure 7. Inverted, two-dimensional graph of f_4
(Reprinted with permission of Kenneth De Jong)

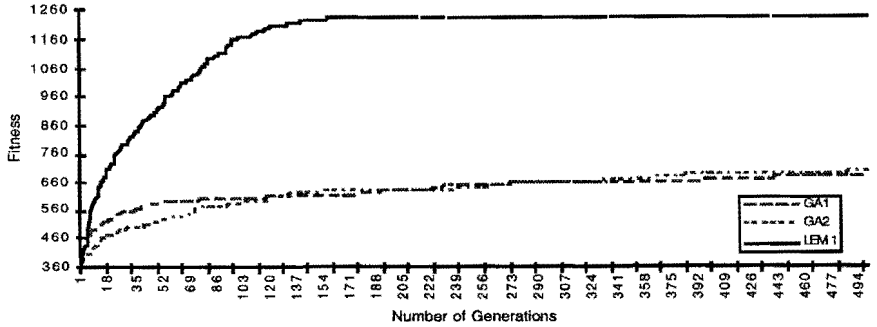

Figure 8. The evolution process within 500 generations for function f_4.

As one can see in Figure 8, for a problem with many variables plus noise, LEM very significantly outperformed GA1 and GA2.

δ	1.006	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 10,000 generations (200,000 births)

Table 9. The number of generations for the fitness value become δ-close to the goal.

	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 the solution was δ-close at 10 000th generation, the speedup would be N.

Table 10. LEM's speed-up over GA1 and GA2 for different values of δ

Problem 5: Testing the ability to find a solution 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

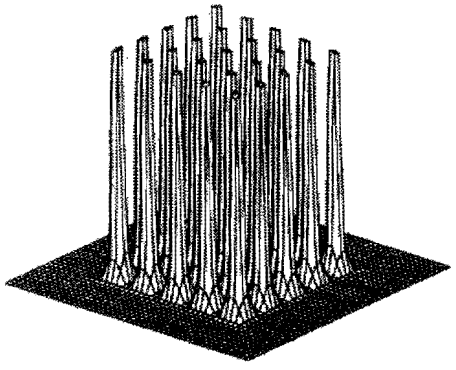

Figure 9. Inverted two-dimensional graph of f_5
(Reprinted with permission of Kenneth De Jong)

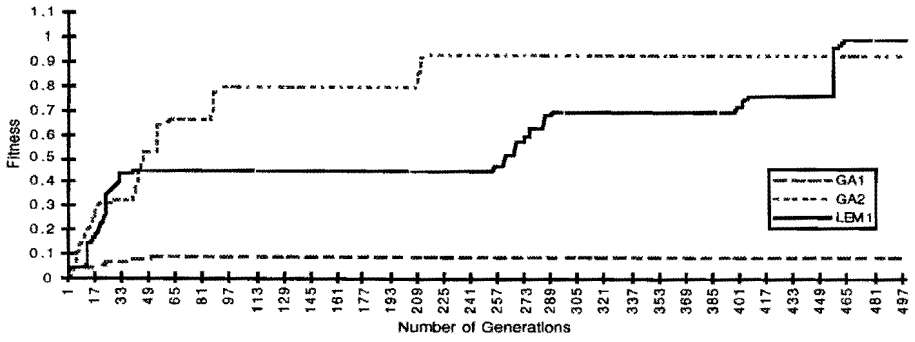

Figure 10. The evolution process within 500 generations for function f_5.

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 finer adjustments). After 500 generations, however, LEM reached the maximum, while GA2 was still 6.7% from maximum and remained so within 10,000 generations.

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 10,000 generations (200,000 births)

Table 11. The number of generations for the fitness value become δ-close to the goal.

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

4.3 Discussion

In all experiments, LEM clearly outperformed GA1 and GA2. In some cases, in particular for Problem 3 (a non-differentiable function) , and Problem 4 (many 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 a binary representation, as compared to GA1, which uses a multi-valued representation, is capable of making finer adjustments when doing crossover and mutation, and often performs 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. This seems, for example, 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 the current LEM (which employs only GA1 for the genetic algorithm mode). At the later evolution stages, LEM was, however, able to reach the maximum, while GA2 could not. It is very likely that LEM built on GA2 would have better performance than the current version (built on GA1) for the above testing problems.

Experiments seeking the function minimum were also conducted and the same patterns of results were acquired.

We noticed that many parameters are adjustable in LEM. Clearly, a systematic experiments for examining their effects on evolution is very expensive. Indeed, we did try changes to some parameters such as *gen-length*, *learn-length*, mutation rate in GA1, discretization intervals and so on and obtained similar results. We did not attempted to find the best combination of these parameters. Actually such a combination is another optimization problem.

5 RELATED WORK

The relationship between learning and evolution is a long-studied problem and is still very intriguing. The view of the well-known Lamarckian hypothesis on evolution is that traits acquired during the lifetime of an organism can be transmitted genetically to the organism's offspring. These traits are typically physical. What is learned during an organism's lifetime also can be thought of as a type of acquired trait and so can, in a Lamarckian view, be passed genetically and directly to its offspring. Unfortunately, this is not the case in reality. A better explanation of the effect of learning on evolution is the Baldwin effect which views the effect of learning of evolution as being indirect, during which individuals best able to learn to survive have the most offspring thus increasing the frequency of the genes responsible for learning, and further leading to, via selection, a genetic coding of a learned trait (Baldwin, 1896). No matter whether these two hypotheses are true in nature, they provide some insights into this learning and evolution issue and we can test them on computers.

Hinton and Nowlan (1987) thought their work confirmed the Baldwin effect. In their experiments, an individual (an neural network) with high learning ability is more likely to pass indirectly their genetic information to the next generation. This passing is indirect, which is unlike LEM which extracts "reasons" from a pool of HIGH and LOW individuals and directly pass extracted information to offspring. Thus, LEM is more Lamarckian in flavor. However, it should be noted that LEM pass extracted patterns in a probabilistic way and offspring do not copy their parents and may be quite different. Unlike the Baldwin effect, LEM does not focus on the learning ability of an individual, but on eliciting the reasons why a set of good individuals are good and hence it seems like LEM uses a meta-level mechanism to guide evolution.

Similarly, using their Evolutionary Reinforcement Learning model to study the Baldwin effect, Ackley and Littman (1992) incorporated reinforcement learning into an evolutionary framework and obtained many interesting results. One of them is that learning of an agent seemed to be important for keeping individuals (agents) alive.

SAMUEL (Grefenstette 1991) is a genetic learning system designed for sequential decision problems in a multi-agent environment. A strategy (i.e., a set of if-then control rules) is applied to a given world state and then some actions are performed. The modification to the rules in a strategy is triggered either directly by the strategy's interaction with the environment or indirectly by the strength of the rules within the strategy. The changes of a strategy are later passed to its offspring. Hence, these rule modification operators are more Lamarckian in flavor than Hinton, Ackley and others' work in terms of its direct passing changes to offspring. Compared to LEM, SAMUEL's Lamarckian feature is a localized operation because the modification is based on a single strategy's previous behavior while LEM extracts patterns from a set of individuals using an advanced symbolic learning system, and then directly employs the learned patterns to generate some individuals for the new generation. Thus, the Lamarckian operation in LEM is global. It is speculated that LEM should have better performance than SAMUEL because acquired traits based on many individuals are more reliable . Nevertheless, SAMUEL may have more explorative power. To avoid possible low explorative power, LEM include a genetic algorithm.

6 CONCLUSION

In order to avoid the potential problems of low efficiency and locally optimal solution, an idea of a Lamarckian evolution model is applied to genetic evolution. It incorporates a symbolic learning method used for discovering reasons why some individuals in an evolution process are better than others. The learned knowledge is used for guiding generation of individuals for new generations.

Expecting the possible limitations of the goodness of those individuals used for learning and hence the low quality of learned knowledge, this model stills adopts the genetic algorithm for exploring diversified individuals in case of little improvement due to Lamarckian evolution. Application of this model to function optimization produced significantly better results than traditional genetic algorithms in terms of evolution speed and reaching a goal.

There are many interesting issues that are worth further study. For instance, we notice that there are many parameters such as learn-length, gen-length, HIGH, LOW, thresholds and even the included learning method in the LEM algorithm need to be set before an evolution. Finding a good combination of them is problem specific and difficult. One solution is automating this by letting them be self-adaptable, i.e., putting an internal mechanism inside LEM which adjusts parameters during an evolution process. Testing on these problems other more powerful genetic algorithms and inclusion of them in LEM will give us a better insight into LEM's performance. It is highly likely that LEM built on more powerful GAs such as GA2 would have better performance than the current version (LEM plus GA1) for the above testing problems. Application of LEM, in addition to function optimization, to different types problems will lead to a better understanding to what problems LEM is effective and how. A possible work is to apply LEM to evolution of "artificial brain" (de Garis 1996) in which evolution speed and the ability of reaching of a goal are also very important (actually we already got initial promising results). One theoretical issue is to investigate the LEM methodology further in order to understand its theoretical aspects, limits, and to develop more advanced versions.

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.

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

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.

De Jong, K. "Genetic Algorithms: Theory and Practice," To be published, 1998.

Goldberg, D., "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.

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

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., 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.

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