Combining Machine Learning with Evolutionary Computation: Recent Results on LEM

Guido Cervone
Ryszard S. Michalski*
Kenneth K. Kaufman
Liviu A. Panait

Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA, 22030

*Also with the Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Abstract

The Learnable Evolution Model (LEM), first presented at the Fourth International Workshop on Multistrategy Learning, employs machine learing to guide evolutionary computation . Specifically, LEM integrates two modes of operation: Machine Learning mode, which employs a machine learning algorithm, and Darwinian Evolution mode, which employs a conventional evolutionary algorithm. The central new idea of LEM is that in machine lear ning mode, new individuals are "genetically engineered" by a repeated process of hypothesis formation and instantiation, rather than created by random operators of mutation and/or recombination, as in Darwinian -type evolutionary algorithms. At each stage o f evoluation, hypotheses are induced by a machine learning system from examples of high and low performance individuals. New individuals are created by instantiating the hypotheses in different ways. In recent experiments concerned with complex function op timization problems, LEM has signif icantly outperformed selected evolutionary computation a lgorithms, sometimes achieving speed -ups of the evolutionary process by two or more orders of magnitude (in terms of the number of generations). In another recent application involving a problem of optimizing heat exchangers, LEM produced designs equal or superior to best expert designs. The recent results have confirmed earlier findings that LEM is able to significantly speed-up evolutionary processes (in terms of the number of generations) for certain problems. Further research is needed to determine classes of problems for which LEM is most advantagious.

.

1 Introduction

The idea that machine learning can be used to directly guide evolutionary computation was first presented at the Fourth International Workshop on Multistrategy Learning (Michalski, 1998). This presentation described the Learnable Evolution Model (LEM), which integrates a machine learning algorithm with a conventional evolutionary algorithm, and repo rted initial results from LEM's application to selected function optimization problems. Presented results were very promising but tentative. They were obtained using LEM1, a rudimentary implementation of the proposed method, and the experiments were perfor med only on a few problems.

Subsequently, a more advanced implementation, LEM2, was developed, and many more experiments were performed with it (Cervone, 1999). The original methodology was also substantially extended and improved (Michalski, 2000). One of the important improvements is the development of the adaptive anchoring discretization method, ANCHOR, for handling continuous variables (Michalski and Cervone, 2000). This paper presents recent results from the application of LEM2 to a range of functio n optimization problems and to a problem of designing optimal heat exchangers. To provide the reader with a sufficient background information, the next section briefly reviews the current version of the Learnable Evolution Model.

2 A Brief Overview of the Learnable Evolution Model

The Learnable Evolution Model (LEM) represents a fundamentally different approach to evolutionary processes than Darwinian -type evolutionary algorithms. In Darwinian -type evolutionary algorithms, new individuals are generated b y processes of mutation and/or recombination. These are semi -blind operators that take into consideration neither the experience of individuals in a given population (like in Lamarckian type of evolution), nor the past history of evolution. In LEM, the evo lution is guided by hypotheses derived from the current and, optionally also past generations of individuals. These hypotheses identify the areas of the search space (landscape) that most likely contain the global optimum (or optima). The machine learning program is used in LEM either as the sole engine of evolutionary change (the uniLEM version), or in combination with the Darwinian-type of evolution process (the duoLEM version).

The duoLEM version integrates two modes of operation: Machine Learning mode and Darwinian Evolution mode. The Darwinian Evolution mode implements a conventional evolutionary algorithm, which employs mutation and/or recombination operators to generate new individuals. The Machine Learning mode generates new individuals by a process of hypothesis generation and instantiation. Specifically, at each step of evolution, it selects two groups of individuals from the current population: High -performing individuals (H-group), which score high on the fitness function, and Low -performance individuals (L-group), which score low on the fitness function. These groups are selected from the current population or from some combination of the currrent and past populations. These two groups are then supplied to a learning program that generates hypot heses distinguishing between the H -group and the L -group. New individuals are generated by instantiating the hypotheses in various ways. These new individuals compete with the existing individuals for the inclusion in the new population.

In the duoLEM vers ion, LEM alternates between the two modes of operation, switching to another mode when *a mode termination condition* is met (e.g., when there is an insufficient improvement of the fitness function after a certain number of populations). In the uniLEM versio n, the evolution process is guided solely by the machine learning program. When the mode termination condition is met, a *StartOver* operation is performed. In such an operation, system generates a new population randomly, or according to certain rules (Michalski, 2000).

Figure 1 presents a flowchart of uniLEM and duoLEM version of LEM. For a comprehensive description of the LEM methodology refer to (Michalski, 1998, Cervone, 1999, Michalski, 2000).

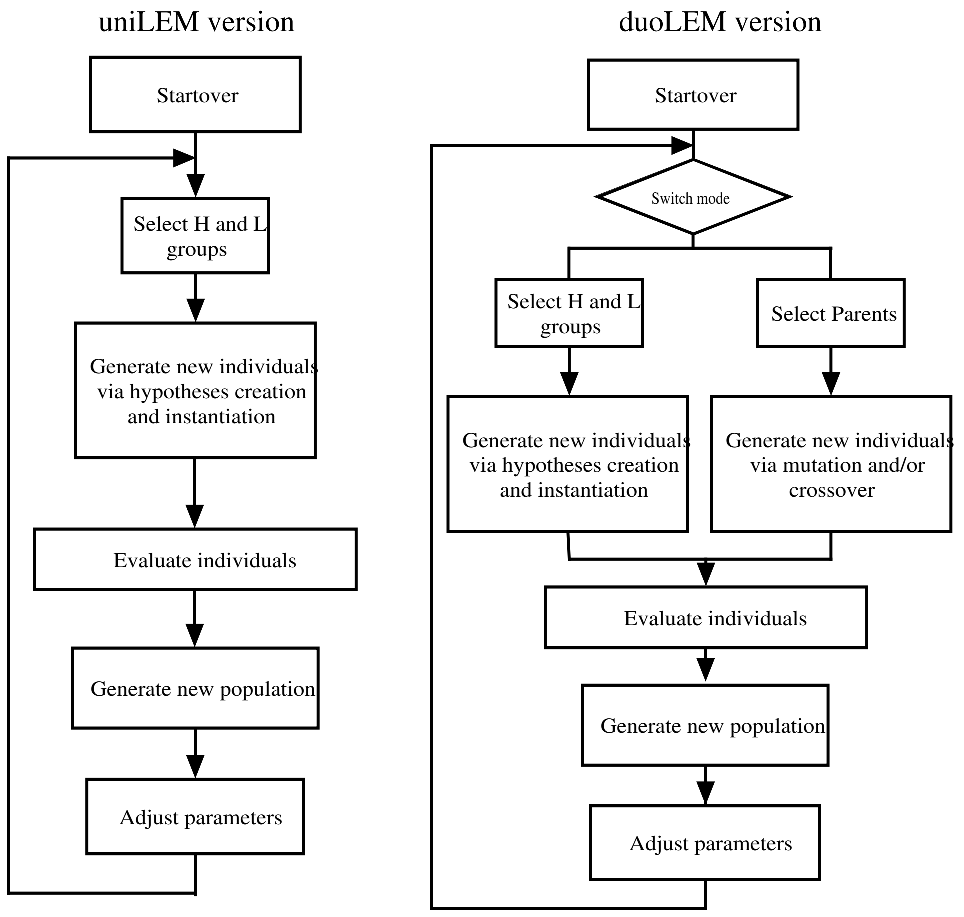

Figure 1. A flowchart of the uniLEM and duoLEM versions.

Below is a brief description of the individual steps, with an indication of how they are implemented in the LEM2 system.

StartOver: This operator generates a new population randomly or according to cetain rules. In LEM2, a new population is generated randomly, with a proviso that a number of the best performing individuals from the past populations are added to the newly generated population (elitism).

Select H -group and L -group: This selection can be done in LEM2 using one of tw o methods: Fitness - Based Selection (FBS), or Population -Based Selection (PBS). In FBS, the H -group (L-group) consists of individuals whose fitness is above the HFT% from the top value (below the LFT% from the lowest value). In PBS, the H -group (L-group) consists of HPT% highest-fitness (LPT% lowest-fitness) individuals in the population. Figure 2 illustrates these two selection methods and the parameters HFT (high fitness threshold), LFT (low fitness threshold), HPT (high population threshold), LPT (low population threshold).

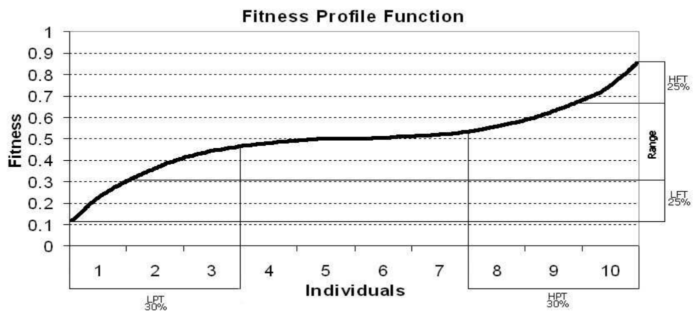

Figure 2. An example of the fitness profile function, and an illustration of parameters HFT, LFT, HPT, LPT would select the H and L groups.

Select parents: The selection of the parents is related to the Darwinian mode. It sele cts representative individuals (parents) from the current population that will be mutated and/or recombined. LEM2 implements two types of mutation: deterministic and uniform. In the first every individual in the population is selected, while in the latte r, every individual has the same chance of being selected, independently from its fitness.

Generate new individuals via hypothesis creation and instantiation: The LEM methodology is not constrained to any particular learning algorithm, but can be used, in principle, with any concept learning method. LEM2 employs AQ18 rule learning program that is highly suitable for LEM due to its various characteristics, such as the ability to learn rules with different levels of generality, the use of internal disjunction operator, and a powerful knowledge representation.

Figures 3 and 4 show an example of the input and output from AQ18, respectively (after small editing).

```
 Parameters
 run  ambig  trim  mode  maxstar   noise
  1   empty spec  ic         1        no

 Variables
   #  type   size   cost   name
   1  lin     15    1.00   x1.x1
   2  lin     15    1.00   x2.x2
   3  lin     15    1.00   x3.x3
   4  lin     15    1.00   x4.x4

H-group events
   #  x1  x2  x3  x4  Weight
   1  7   7   7   8   12
   2  7   6   6   7   9
   3  7   6   7   8   7
   4  6   6   7   8   5
   5  6   7   6   8   5

 L-group events
   #  x1  x2  x3   x4 Weight
   1  6   6   14   8   5
   2  6   7   14   8   3
   3  7   7   14   7   1
```

```
A rule generalizing the H-group:

[x1=6..7] & [x2=6..7] & [x3=6..8] & [x4=7..8]

This learning process used:
  System time:    0.0 seconds
  User time:      0.0 seconds
```

Note: The values in the conditions of the rule above are symbols representing ranges of original values of these variables, not the original values. These ranges has been detemined in the process of *adaptive anchoring quantization* (Michalski and Cervone, 2000).

←*Figure 4.* AQ18 input. ↑*Figure 5.* AQ18 output.

AQ18 takes as intput the H -group and L -group, a specification of the types and domains of the variables, plus, optionally, control parameters [see (Kaufman and Michalski, 2000) for a detailed explanation], and produces a set of attributional rules with annotations characterizing the rules. Each learned rule is a conjunction of conditions that specify ranges of values an attribute may take (in the case when AQ18 runs without invoking constructive induction). A rule is instantiated by selecting values satisfying rule conditions. The learned rules are used to generate new individuals by randomizing variables within the ranges of values defined by the rule conditions. If a rule does not include some variable, it means that this variable was found irrelevant for distinguishing between the H -group and the L -group. Variables that are not included in the rule are instantiated by randomly choosing a value from their domain, or choosing a value that is present in a randomly selected individual from the current population.

Generate new individuals via mutation and/or crossover: Individuals in the parent population are mutated and/or recombined. Research on Darwinian -type evolutionary algorithms has investigated many different forms of mutation and recombination.

Evaluate individuals: For each new individual, its fitness is evaluated according to a given fitness function or by some process, e.g., simulation. In the latter case, this operation may be costly and time-consuming.

Generate new population: This step involves creating a new population that combines individuals from the previous population with new individuals generated according to the rules learned. Different methods can be used for this purpose. These methods can be divided into *intergenerational* and *generational*. In the methods of the first group, both newly generated and previous individuals compete for inclusion in the new population. In the methods of the second group, only newly generated individuals compete for the inclusion.

Adjust Parameters: LEM keeps statistics regarding the number of successful births, the change in the highest-so-far fitness score, and others. Using these statistics, it can adjust its behavior in the evolutionary process. For example, it may find that at a given step generating more general or more specific rules may be more desirable, that parameters controlling the selection of H-group and L-group need to be changed, or that the mutation rate for the Darwinian evolutionary mode need to be adjusted.

3 LEM Implementations: LEM2, LEM1, and ISHED1

LEM2 is the newest general -purpose implementation of LEM, and represents a significant improvement over LEM1, the first, rudimentary implementation (Michalski and Zhang, 1999). LEM1, presented at MSL98, employs the AQ15c machine learning program in Machine Learning mode and GA1 and GA2, two simple evolutionary algorithms, in Darwinian evolution mode. GA1 and GA2 use a deterministic selection mechanism and a real-value representation of the variables. The main differences between the two are that GA1 generates new individuals only through a uniform Gaussian mutation operator, while GA2 uses also a uniform crossover operator. Continuous variables are discretized into a fixed number of values. LEM1 was applied to function optimization (Michalski and Zhang, 2000), and a problem in designing non-linear digital filters (Coletti et al. 1999).

LEM2 was programmed using EC++, a generic Evolutionary Computation Library (Cervone and Coletti, 2000). In Machine Learning mode, it employs the AQ18 rule learning program (Kaufman and Michalski, 2000a). The main features or improvements introduced to LEM2 in relation to LEM1 include:

A. A new method for discretizing continuous variables has been developed and implemented. The method, called *Adaptive Anchoring Discretization*, briefly ANCHOR (Michalski and Cervone, 2000) gradually and adaptively increases the resolution of continuous variables in the process of evolution. The method has drastically improved the efficiency of LEM in the case when individuals are described by continuous variables.

B. New individuals are generated by instantiating multiple rules rather than only the strongest rule in the ruleset generated by the learning program. This allows the system to explore in parallel several subareas of the search space, which is important in the case of multi-modal landscapes.

C. The number of new individuals generated from a single rule is not fixed, but is proportional to the *rule fitness*, defined as the sum of fitnesses of examples covered by the rule.

D. In addition to the population -based method for selecting the H -group and L -group, LEM2 can also uses the fitness-based method.

E. The cost of variables in adjusted dynamically in the evolution process. Each time a variable is included in a ruleset generated by the learning program, its cost is increased. This way, the system gives preference to variables that were not included in the previously learned ruleset. This feature has proven to be useful in optimizing functions with very large numbers of variables.

F. The uniLEM version has been implemented, that is, the evolution process repet itively executes only Machine Learning mode. There is no separate Darwinian Evolution mode.

G. A simple version of the StartOver operation has been implemented for the uniLEM version. Specifically, when the fitness profile function is flat for a controlled n umber of generations, new individuals are created randomly and inserted into the current population.

H. Parameters controlling the creation of H -group and L-group in each step of evolution, *the population lookback* and the *description lookback*, have been implemented in LEM2 (Michalski, 2000).

LEM2 was applied to a range of optimization problems, and its performance was compared to that of conventional Darwinian-type evolutionary algorithms (Cervone, 1999). ISHED1 is an implementation of the LEM methodology t ailored toward a specific application domain, namely, to the design of heat exchanger systems. Specifically, it conducts an evolutionary optimization process to determine the best arrangement of the evaporator tubes in the heat exchanger of an air conditio ning system under given technical and environmental constraints (see Section 4.2). Special *structure modifying operators* have been implemented that modify structures according to the expert domain knowledge. A detailed description of ISHED1 is in (Kaufman and Michalski, 2000).

4 Experiments

This section presents selected results from testing and validating the LEM methodology using LEM2. To maximize the objectivity of LEM2 testing, the results from conventional evolutionary algorithms on the same probl em, which were found in the literature or on the web, were compared with the corresponding results from LEM2.

For problems for which we were unable to find such results in the literature or on the web, we applied a conventional evolutionary algorithm, ES, that we re-implemented in C++ from an existing version in C (that was obtained from Ken De Jong). ES uses a real-valued representation and deterministic selection (i.e., each parent is selected and then mutated a fixed number of times, defined by the *brood parameter*). The mutation is done according to the Gaussian distribution, in which the mean is the value being mutated and the standard deviation is a controllable parameter. Each variable has $1/L$ probability of being mutated, where L is the total number of variables defining an individual. The new individuals and their parents are sorted according to their fitness, and the *popsize* highest-fitness individuals are included in the next generation, where *popsize* is a fixed population size.

For some of the problems, we found on the web results from the application of Parallel GA (PGA). PGA is a standard genetic algorithm (that uses a binary -string representation, mutation and crossover operators, and fitness-proportional selection) that simultaneously maintains separate subpopulations of individuals (the number of subpopulations and their sizes are specified by user-provided parameters).

4.1 Application to Function Optimization

This section presents a selection of results from the application of LEM2, ES, and PGA (when a to three well-known function optimization problems, namely, the Rosenbrock function, the Rastrigin function and the Gaussian Quartic function.

<u>*Problem 1.*</u> Find the minimum of R osenbrock function (Rosenbrock, 1960) in which the nu mber of arguments, *n*, is raised to 100, and each argument ranges between –5.12 and 5.12:

$$Ros(x_1, x_2, .. x_n) = \sum_{i=1}^{n-1} (100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$$

This is a rather complex optimization problem because the function has a very narrow and sharp ridge and runs around a parabola, so the variables are interrelated (Figure 5).

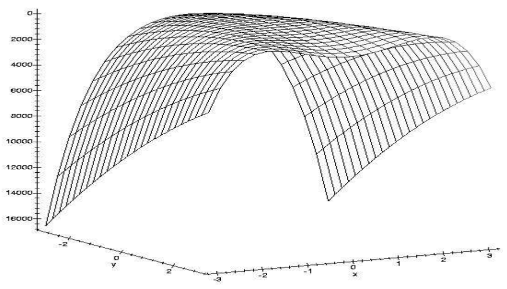

Figure 5. An inverted graph of a two-dimensional projection of the Rosenbrock function.

For comparison, the ES algorithm was also applied to the same problem. The results of this experiment are graphically presented in Figure 6. Two different population sizes were used, 100 and 150, for both LEM2 and ES. Each experiment was repeated 10 times and the results averaged.

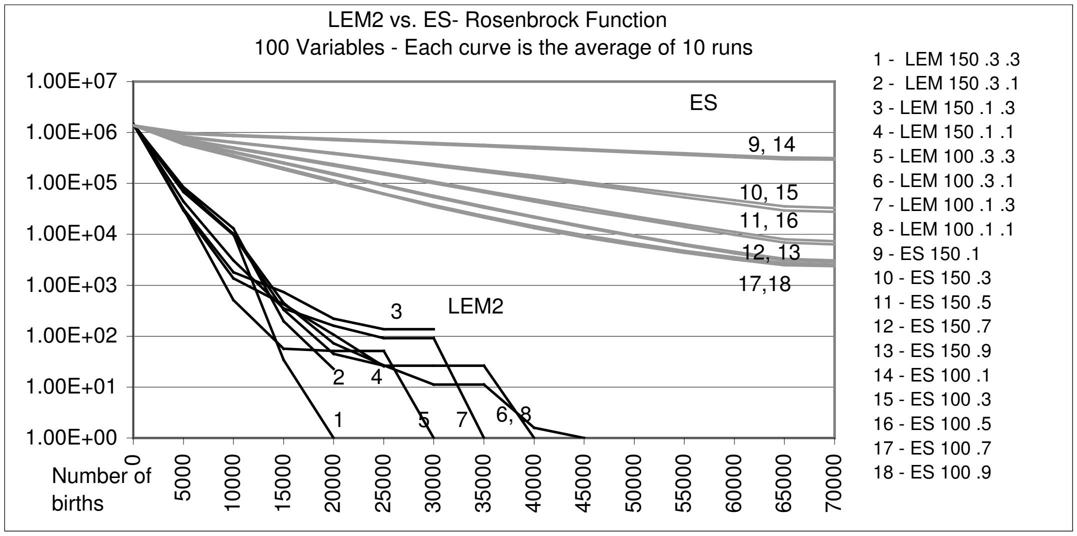

Figure 6. Results from LEM2 and ES for the Rosenbrock function optimization.

In Figure 6, LEM *a,b,c* means that the method was LEM2, the population size was *a*, and HPT and LPT parameters were *b* and *c*, respectively. ES *a,b* means that population size was *a* and mutation rate b. As shown in Figure 6, LEM2 was significantly less d ependent on the input parameters than ES, and also converged to the function minimum much faster. It is possible to notice that some of the LEM curves (e.g. 5,6,7,8) show a long horizontal line, meaning that for several births the algorithm did not improve the global optimum, a nd then a steep vertical line. This behavior is the result of the startover operator, which introduced new individuals in the population, therefore allowed LEM to discover those areas of the space most favorable to direct the evolution.

LEM2's results were also compared with the best available results previously published for this function (CHC). These results concern the Rosenbrock function with a much smaller number of variables (only 2 and

4). They are summarized in Table 1, which shows the number of evaluations needed to come *δ-close* to the global optimum, and the relative speedups.

The value of δ-close specifies the number of generations after which the relative distance from the solution to the target (global optimum) produced by an algori thm becomes smaller than δ. The *speedup* of algorithm A over B for a given δ, is defined as the ratio, expressed in percentage, of the number of births required by B to the number of births required by A to achieve the δ-close result.

In the case of t wo variables, the best result was achieved using the CHC+BLX algorithm (briefly, CHC) that required 4893 evaluations (Eschelman and Shaffer, 1993). In contrast, LEM2 found the global minimum using only 101 evaluations (a speedup of nearly 5000%).

Rosenbrock function minimization 2 vars	δ=0
LEM2 (uniLEM)	101
CHC	4893
Speedup LEM2/CHC	4800%

Table 1. Results for the Rosenbrock function of 2 variables.

In the case of four variables, the best published resu lt was achieved by a breeder GA, that required about 250,000 evaluations (births) to achieve a result with δ=0.1 (Schlierkamp-Voosen and Muhlenbein 1994). LEM2 found the global optimum (δ=0) with only 281 evaluations, that is, the speedup of LEM2 over GA was *at least* 75,000% (since the result published for GA referred to δ=0.1 rather than δ=0.1). Table 2 summarizes the results. These results indicate that LEM2 was able to rapidly locate the portion of the landscape containing the global optimum.

Rosenbrock function minimization (4 variables)	
LEM2 (uniLEM)	*δ=0:* 281
GA	*δ=0.1:* 77,000
Speedup LEM2/GA	≥ 27,500%

Table 2. Results for the Rosenbrock function of 4 variables.

Figure 7 illustrates sample rules that AQ ge nerated when LEM was applied to find the minimum of the Rosenbrock function with four variables, and also how they match the H -group individuals. The variables are discretized using the values shown in the Table 3.

$$$$Update this table

Value 0	–2 .. -1.2
Value 1	–1.2 .. -.4
Value 2	–.4 .. -.4
Value 3	.4 .. 1.2
Value 4	1.2 .. 2

Table 3. A correspondence of the symbolic values to real values of variables in Figure 7.

The minimum is found when all the Xs are equal to 1, and this will be repres ented in the diagram by value 3, since 3 describes the range between -.4 and 1.2, which includes 1. The global solution is indicated in Figure 7 by a circle.

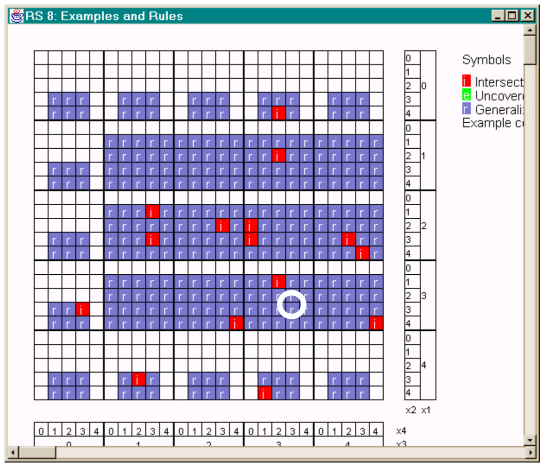

Figure 7 .Learned Hypotheses and H-group individuals.

The learned hypotheses (attributional rules) shown in Figure 7 are:

Rule1: [x1=1..3] & [x2=1..4] & [x3=1..4]

Rule2: [x2=3..4] & [x4=1..3]

Both rules include the individual that represents the function minimum: [x1=3] & [x2=3] & [x3=3] & [x4=3], in a short notation: (3,3,3,3).

Problem 2. Find the minimum of the Rastrigin function:

$$Ras(x_1, x_2,..x_n) = n*10 + \sum_{i=1}^{n}(x_i^2 - 10*\cos(2*\pi*x_i))$$

in which the number of arguments, *n*, was set to 100, and each x was bounded between –5.12 and 5.12.

The Rastrigin function has many local optima, and it is easy to miss the global solution (Figure 8). In thi s experiment, both uniLEM and duoLEM versions were employed, and their results were compared with the best available result from a conventional evolutionary method, which was obtained by a parallel GA with 16 subpopulations and 20 individuals per subpopula tion (Muhlenbein, Schomisch, and Born, 1991). This result is shown by the point PGA in Figure 9. The LEM2' results were also compared with the performance of ES.

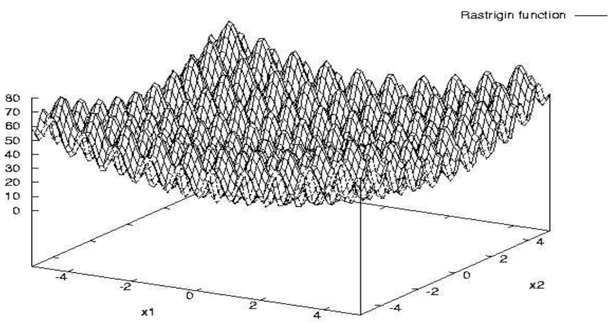

Figure 8. A 2D projection of the Rastrigin function.

The results of uniLEM, duoLEM, ES and the Parallel GA are shown in Figure 9.

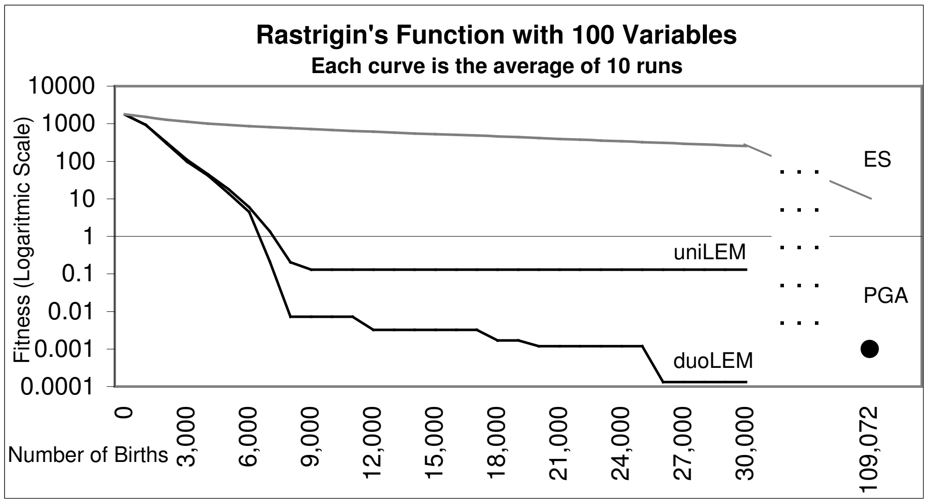

Figure 9. Results obtained by ES, LEM2's uniLEM and duoLEM versions, and a Parallel GA for the Rastrigin function with 50 variables.

Figure 9 illustrates the evolutionary pro cess conducted by uniLEM, duoLEM, and ES. It also shows a point indicating the best result obtained by the parallel GA. Each curve represents an average of 10 runs. The y-axis represents the fitness using a logaritmic scale, and the x-axis represents the number of births. As one can see, both uniLEM and duoLEM relatively quickly. DuoLEM reached the global minimum with δ=0.0001 in all 10 runs after about 26000 evaluations. UniLEM found the global minimum 7 times out of 10 (hence the average of the fitness function is higher than in the case of duoLEM). The parallel GA, which achieved the best result found in the literatu re on this problem, required 109072 evaluations to achieve δ=0.001 (it used 8 subpopulations, each with 20 individuals). Thus, the speedup of duoLEM over parallel GA (PGA) was more than 420%. We also investigated the rate of convergence to the optimum ob tained of these four algorithms by repeating the experiment for 20, 50, and 100 variables.

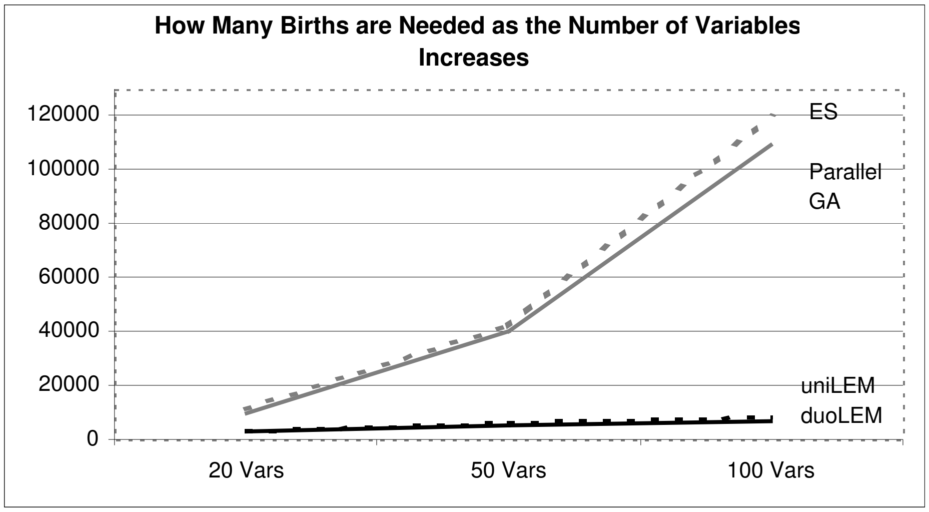

Figure 10. The number of births needed to reach the optimum as a function of the number of arguments.

Figure 10 shows the dependence o f the *evolution duration* (measured by the number of births required to reach the near-optimal solution) on the number of function arguments for different methods. As seen in the figure, the evolution duration in case of LEM2 has only slightly increased wit h the number of arguments, while in the case of ES and Parallel GA it has increased much faster.

Problem 3. Find the minimum of the Gaussian Quartic function:

$$Gauss(x_1, x_2, .., x_n) = \sum_{i=1}^{n} i x_i^4 + Gauss(0,1)$$

in which the number of arguments, n, was set to 10, 50 and 100, and each x was bounded between –5.12 and 5.12. This is a simple unimodal function padded with noise (Figure 11). The Gaussian noise ensures that the algorithm never gets the same value on the same point. Algorithms that do not do well on this test function will do poorly on noisy data. In this experiment uniLEM was compared with ES.

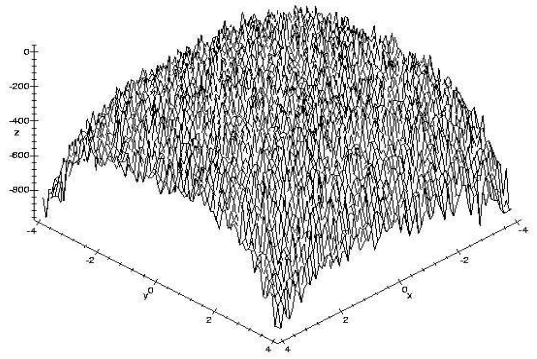

Figure 11. A 2D projection of the Gaussian Quartic function.

Table 4 presents the result of comparing LEM2 in uniLEM version with ES using different population sizes. Results are shown for different deltas.

(# vars)	**10**			**50**			**100**		
	δ0	**δ0.01**	**δ0.1**	**δ0**	**δ0.01**	**δ0.1**	**δ0**	**δ0.01**	**δ0.1**
LEM 100 .3 .3	1000	800	700	4900	4400	3900	21400	10500	10100
ES 100 .7	3600	3200	2900	40100	40100	36700	432860	391979	92455
Speedup LEM/ES	300%	400%	400%	800%	900%	900%	2000%	3700%	900%

Table 4. Results and relative speedups of LEM2 over ES for the Gaussian Quartic function.

This experiment confirms the results of the problem 2, where it was shown that the speedup of LEM vs. Darwinian Evolutionary Algorithms increases as the number of dimensions increases. Th is is to be attributed to the fact that blind operators such as mutation and recombination tend to be less effective with large search spaces. This experiment also shows the ability of LEM to work with noisy functions.

4.2 Design of Heat Exchangers

In or der to test LEM on a practical problem, we applied it to the optimization of heat exchanger designs under various technical and environmental constraints. To this end, we developed a specialized system, ISHED1, that customized LEM to this problem. To expla in this application, let us briefly explain the problem. In an air conditioning unit, the refrigerant flows through a loop. It is superheated and placed in contact with cooler outside air in the condenser unit, where it transfers heat out and liquefies. Co ming back inside to the evap orator, it comes into contact with the warmer interior air that is being pushed through the heat exchanger, as a r esult cooling the air while heating and evaporating the r efrigerant. The heat exchanger consists of an array of parallel tubes through which the refrigerant flows back and forth. Different orderings of the flow of the refrigerant through the individual tubes may have a profound effect on the air conditioner's cooling ability.

ISHED1 applies a version of duoLEM tailored to this problem. Individuals in a population represent designs (structures) of heat exchangers. Each design is defined by a vector that characterizes the arrangement of tubes on the path from the input and the output. In Darwinian Evolution mode, ISHED1 employs eight *structure-modifying* operators, which make changes in the structures (analogous to mutation operators in evolutionary algorithms). For example, one operator may create a split in a refrigerant path by moving the source of a tube's refrigerant closer to the inlet tube; second operator may swap the tubes in the structure; and another operator may graft a path of tubes into another path, etc. (Kaufman and Michalski, 2000b). The application of these operators is domain knowledge driven, that is, operators are applied according to known technical constraints.

Machine learning mode in ISHED1 is also tailored to the heat exchanger design task. The hypotheses generated describe abstractions of the individual structures. They specify only the location of inlet, outlet and split tubes. Beyond that, the instantiation module may choose among the different structures that fit the learned template, and generate the most plausible one according to the real-world background knowledge. ISHED1 uses high and low fitness thresholds of 25% to select the H-group and L-group. Once rules are generated, an elitist strategy is used to form the next generation of proposed architectures. The best architecture found so far, as well as all members of the H-group are passed to the next generation, along with various instantiations of the learned rules.

An ISHED1 run proceeds as follows. Given instructions characterizing the environment for the target heat exchanger, an initial population of designs (specified by the user or randomly generated), and parameters for the evolutionary process, ISHED1 evolves populations of designs using combination of Darwinian and machine learning operators for a specified number of generations. ISHED1 returns a report that includes the best designs found and their estimated quality (capacity). Throughout the execution, design capacities are determined by a heat exchanger simulator (Domanski, 1989).

Many experiments have been conducted. Initial experiments concerned a poblem with a known expert solution (design) regarding the heat exchanger size and airflow pattern. The best design found by ISHED1 was comparable to the expert designs widely used by industry. Further experiments utilized different exchanger sizes and airflow patterns. The latter changes were especially significant, because the commercially built air conditioners typically do not take into account an uneven airflow. When confronted with such situations, their cooling ability suffers. In the case of non-uniform airflow, the ISHED1-designed heat exchangers performed significantly better than the currently-used expert-designed structures (Kaufman and Michalski, 2000b).

An example of the output from an ISHED1 run is shown in Figure 12. This run was done in a v erbose mode, and as such, the log details every structure tested, every operator applied, and the rules learned. The figure only shows a very small sample of the full output in order to give the reader a flavor of ISHED1 in action. Added comments are given in italics.

```
Exchanger Size: 16 x 3
Population Size: 15   Generations: 40
Operator Persistence: 5
Mode Persistence: GA-probe=2 SL-probe=1
Initial population:
Structure #0.3:  17 1 2 3 4 5 6 7 8 9 12 13 29 15 31 I 18 33 20 36 22 38 24 40 26 42 11 2
         7 45 14 47 16 34 35 19 37 21 39 23 41 25 43 44 28 46 30 48 32:  5.5376
Structure #0.8:  17 1 20 3 4 22 6 24 8 26 10 28 27 15 16 32 33 2 18 19 5 38 7 40 9 42 11
         44 13 46 30 48 34 35 36 I 21 37 23 39 25 41 27 43 29 45 31 47:  Capacity = 5.2099
and 13 others

Selected Members:  3, 2, 3, 7, 9, 3, 9, ...
Operations: NS(23, 39), SWAP(8), SWAP(28), ...,
          SWAP(29), SWAP(25), SWAP(1)
Below is one of the structures created by the application of a SM operator in Darwinian mode (by swapping the two tubes following
tube 29 in Structure #0.8)
Generation 1:
Structure #1.13: 17 1 20 3 4 22 6 24 8 26 10 28 27 15 16 32 33 2 18 19 5 38 7 40 9 42 11 4
        13 45 30 48 34 35 36 I 21 37 23 39 25 41 27 43 46 29 31 47:  Capacity=5.2093
and 14 others.


Selected Members:  6, 15, 11, 3, 13, 1, ...
. . . . . .
The program soon shifts into Symbolic Learning Mode:
Generation 5: Learning mode
Learned rule:
    [x1.x2.x3.x4.x5.x6.x7.x8.x9.x11.x12.x13.x14.x15.x17.x18.x19.x20.x21.x22.x23.x24.x25.x2
    6.x27.x28.x29.x30.x31.x32.x33.x34.x35.x36.x37.x38.x39.x40.x41.x42.x43.x44.x45.x46.x47.
    x48=regular] & [x10=outlet]&[x16=inlet] (t:7,u:7,q:1)

An example of a generated structure:
Structure #5.1:  17 1 2 3 4 5 6 7 8 9 12 29 45 30 31 I 18 33 20 36 22 38 24 40 26 42 11 27
        13 15 47 48 34 35 19 37 21 39 23 41 25 43 44 28 46 14 32 16:  Capacity=5.5377
........
Below is a structure from the 21st generation:
Generation 21: Learning mode
Structure #21.15 2 18 4 1 6 3 5 7 8 9 12 13 45 15 31 I 33 17 35 36 22 39 24 40 42 25 11 44
        30 46 32 47 34 19 20 37 21 23 38 41 26 43 28 27 29 14 48 16:  5.5387
and 14 others

Selected Members:  11, 4, 4, 13, 15, 10, 12, 13, 15, 15, 12, 2, 3, 5, 10.
........
ISHED1 continues to evolve structures, and finally achieves:
Generation 40:
Structure #40.15: 33 17 2 41 4 5 6 9 7 8 12 29 46 45 47 I 1 34 20 36 22 38 24 3
        42 43 44 27 13 15 32 16 18 11 19 37 21 32 23 25 40 26 28 35 30 14 48 31:
        Capacity=6.3686
```

Figure 12. An excerpt from the log of an ISHED1 run.

5 Summary

This paper presented a selection of recent results f rom the research on Learnable Evolution Model that employs machine learning to speed up evolutionary computation. The results were obtained by system LEM2, which can operate in two versions, uniLEM and duoLEM. The uniLEM version executes repeatedly Mach ine Learning mode, which uses rule learning and instantiation as basic operators (the Startover operator allows the system switch the population). The duoLEM version alternates between

Machine Learning and Darwinian Evolution mode. The latter mode applies a conventional evolutionary computation algorithm.

LEM2, described in this paper, is an improvement of the earlier system, LEM1. It implements a multiple rule instantiation (catering to multiple global optima), adaptive anchoring discretization (an aut omatic adjustment of the precision in discretizing continuous attributes), and a uniLEM version. The results from LEM2 have confirmed previous strong results (Michalski and Zhang, 1999), and demonstrate that the LEM methodology can be highly useful for some problems.

ISHED1 is a version of LEM tailored to a class of problems in engineering design (optimization of heat exchangers). It applies task-specific operators in a LEM-type control environment. Results have confirmed significant benefits from integrating Darwinian Evolution and Machine Learning modes of evolution. By doing so, ISHED1 was able to achieve or exceed the cooling capacity of commercially designed systems under a variety of conditions, with only a moderate amount of domain knowledge.

The processs of generating new individuals by hypothesis generation and instantiation used in LEM is computationally more intensive than the process of generating new individuals by mutation and/or recombination. This is offset by the reduction, som etimes very significant, in the number of evaluations needed to reach the evolution goal. Thus, LEM appears to be particularly well -suited for evolutionary computation problems in which evaluating fitness of individuals in a population is a costly and/or time-consuming operation (as is, e.g., in the case of heat-exchanger design).

The LEM methodology is at an early stage of development and opens many interesting problems for further research. They include a theoretical and experimental investigation of the trade-offs inherent in LEM, an implementation of more advanced versions of LEM, an experimentation with different combinations of conventional evolutionary algorithms and machine learning algorithms, and testing in variety of practical domains.

Acknowledgments

The authors thank Jeff Bassett and Paul Wiegand for their help in plotting the Rastrigin function.

This research has been conducted in Machine Learning and Inference Laboratory at George Mason University. The Laboratory's research that enabled the work presented in the paper has been supported in part by the National Science Foundation under Grants No. IIS-9904078 and IRI-9510644.

References

Baeck, T., Fogel, D.B. and Michalewicz, Z. (eds.) (1997). *Handbook of Evolutionary Computation*. Oxford: Oxford University Press.

Cervone, G. (1999). An Experimental Application of the Learnable Evolution Model to Selected Optimization Problems. *Master's Thesis*, Dept. of Computer Science, George Mason University, Fairfax, VA.

Cervone, G. and Michalski, R.S. (2000). Design and Experiments with the LEM2 Implementation of the Learnable Evolution Model. *Reports of the Machine Learning and Inference Laboratory* , George Mason University, Fairfax, VA (to appear).

Cervone, G. and Coletti, M. (2000). EC++, a Generic C++ Library for Evolutionary Computation. *Reports of the Machine Learning and Inference Laboratory*, George Mason University, Fairfax, VA. (to appear).

Coletti, M., Lash, T., Mandsager, C., Michalski, R.S., and Moustafa, R. (1996). Comparing Performance of the Learnable Evolution Model and Genetic Algorithms on Problems in Digital Signal Filter Design. *Proceedings of the 1996 Genetic and Evolutionary Computation Conference* .

Domanski, P.A. (1989). EVSIM -An Evaporator Simulation Model Accounting for Refrigerant a nd One Dimensional Air Distribution. NISTIR 89-4133.

Eshelman, L. J. and Schaffer, J. D. (1993). Real -Coded Genetic Algorithms and Interval Schemata. *Foundation of Genetic Algorithms 2*, San Mateo, CA.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Op timization and Machine Learning* . Addison - Wesley.

Holland, J. (1975). *Adaptation in Artificial and Natural Systems* . Ann Arbor: The University of Michigan Press.

Kaufman, K.A. and Michalski, R.S. (2000a). The AQ18 System for Machine Learning and Data Mining: User's Guide. *Reports of the Machine Learning Laboratory* , MLI 00-3, George Mason University, Fairfax, VA.

Kaufman, K.A. and Michalski, R.S. (2000b). Applying Learnable Evolution Model to Heat Exchanger Design. *Proceedings of the Twelfth International Conf erence on Innovative Applications of Artificial Intelligence*, Austin, TX (to appear).

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolutionary Programs* . Springer Verlag.

Michalski, R.S. (1998). Learnable Evolution: Combining Symbolic an d Evolutionary Learning. *Proceedings of the Fourth International Workshop on Multistrategy Learning,* organized by the University of Torino, Desenzano del Garda, Italy, June 11-13, pp.14-20.

Michalski, R.S. (2000). LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning. *Machine Learning* 38(1-2), pp. 9-40.

Michalski. R.S. and Cervone, G. (2000). Adaptive Anchoring Quantization of Continuous Variables for Learnable Evolution. *Reports of the Machine Learning and Inference Laboratory* , Ge orge Mason University, Fairfax, VA (to appear).

Michalski. R.S. and Zhang, Q. (1999). Initial Experiments with the LEM1 Learnable Evolution Model: An Application to Function Optimization and Evolvable Hardware. *Reports of the Machine Learning and Inference Laboratory*, MLI 99-4, George Mason University, Fairfax, VA.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press.

Muhlenbein, H., Schomisch, M. and Born, J. (1991). The Parallel Genetic Algorithm as Function Optimizer, *Proceedings of the Fourth Int'l Conference on Genetic Algorithms and their Applications.*

Ravise, C. and Sebag, M. (1996) An Advanced Evolution Should Not Repeat Its Past Errors. *Proceedings of the Thirteenth International Conference on Machine Learning*.

Reynolds, R.G. (1994). An Introduction to Cultural Algorithms. *Proceedings of the third Annual Conference on Evolutionary Programming*.

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function, *Computer Journal* 3:175, 1960.

Schlierkamp-Voosen, D. and Muhlenbein, H. (1994). Strategy Adaptation by Competing Subpopulations, *Parallel Problem Solving from Nature*, *Proceedings of the Third Workshop, PPSN III, Jerusalem.*

Sebag, M. and Schoenauer, M. (1994). Controlling Crossove r Through Inductive Learning. *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, Springer-Verlag.

Sebag, M., Schoenauer M., and Ravise C. (1997). Inductive Learning of Multation Step -size in Evolutionary Paramter Optimization, *Proceedings of the Sixth Annual Conference on Evolutionary Programming.*