Experimental Validations of the Learnable Evolution Model

Guido Cervone
Kenneth K. Kaufman
Ryszard S. Michalski*

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

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

Abstract

A recently developed approach to evolutionary computation, called Learnable Evolution Model or LEM, employs machine learning to guide processes of generating new populations.

The central new idea of LEM is that it generates new individuals by processes of hypothesis generation and instantiation, rather than by mutation and/or recombination, as in conventional evolutionary computation methods. The hypotheses are generated by a machine learning program from examples of high and low performance individuals.

When applied to problems of function optimization and parameter estimation for nonlinear filters, LEM significantly outperformed the evolutionary computation algorithms used in experiments, sometimes achieving two or more orders of magnitude of evolution speed-up in terms of the number of generations (or births). An application of LEM to the problem of optimizing heat exchangers has produced designs equal to or exceeding the best human designs.

1 Introduction

Current methods evolutionary computation employ various forms of mutation and/or recombination operators to generate new individuals. Because these operators are semi-random, such methods often suffer from low efficiency (e.g., Holland 1975; Goldberg 1989; Michalewicz 1996; Mitchell 1996; Baeck, Fogel and Michalewicz 1997; Banzhaf et al. 1998).

A new approach, called the *Learnable Evolution Model* (LEM), speeds up evolutionary computation by introducing a learning process to evolution (Michalski 1998; 2000). Specifically, at selected steps of evolution, LEM seeks hypotheses differentiating between groups of high and low performance individuals, that is, individuals that score high and low on the given fitness evaluation criterion. These groups are selected from the current and possibly also past populations.

In LEM1 and LEM2, early systems implementing the methodology (Michalski and Zhang 1999; Cervone 1999), hypotheses are generated by AQ-type learning systems (AQ15 and AQ18, respectively). The AQ-type learning has proven to be highly suitable for LEM. The following sections briefly describe the LEM methodology, and then illustrate it by a sample of results from its application to function optimization and heat exchanger design.

2 An Overview of the LEM methodology

The Learnable Evolution Model or LEM is fundamentally different from the Darwinian-type model that underlies most of the current methods of evolutionary computation. The central engine of evolution in LEM is *Machine Learning mode*, which creates new individuals by processes of generalization and instantiation rather than mutation and/or recombination as in the Darwinian-type evolutionary computation methods.

Machine Learning mode consists of two processes: *hypothesis generation*, which determines a hypothesis characterizing differences between high-fitness and low-fitness individuals in one or more past populations, and *hypothesis instantiation*, which generates new individuals by instantiating the hypothesis in various ways. Machine Learning mode thus produces new individuals not through semi-random Darwinian-type operations, but rather through a deliberate reasoning process involving generation and instantiation of hypotheses about populations of individuals. Thus, in LEM, new individuals are *genetically engineered*, in the sense that they are determined according to descriptions learned from the analysis of the current and possibly past generations.

LEM may alternate between Machine Learning mode and *Darwinian Evolution* mode (executing one of the conventional evolutionary computation methods), or may rely entirely on Machine Learning mode.

LEM differs not only from the Darwinian-type evolution but also from the Lamarckian-type of evolution, because in generating new individuals it takes into consideration not only the experience of single individuals, but the experience of one or more populations of individuals.

An evolutionary process in LEM starts with an initial population, which is generated randomly or according to some rules. In analogy to nature, this population may represent "phenotypes," or "genotypes" that are used to produce "phenotypes." Below is a simplified form of LEM (a full version is in (Michalski 2000a)):

1. *Generate a population*
2. *Invoke Machine Learning mode:*

   a) Derive extrema: Split the current population (or the union of the current and selected past populations) into three groups: H-group (high-performance), L-group (low-performance), and the rest, based on the fitness function.

   b) Create a hypothesis: Apply an inductive learning method to hypothesize a description of the H-group that differentiates it from the L-group.

   c) Generate new population: Generate new individuals by instantiating the learned hypothesis in different ways, and combine the new individuals with those in the H-group. Select from the combined set a new population.

   d) Go to step 2*a)*, and continue repeating Machine Learning mode until the *Machine Learning mode termination condition* is met, in which case perform one of the following steps:

      (i) If the *LEM termination condition* is met, end the evolution process.

      (ii) Repeat the process from step 1. This is called a *start-over* operation.

      (iii) Go to step 3.
3. Invoke *Darwinian Evolution mode* and apply one of the existing Darwinian-type evolutionary computation methods. Repeat and continue this mode until the *Darwinian Evolution mode termination condition* is met.
4. Go to step 2, and then continue alternating between steps 2 and 3 until the LEM termination condition is met.

The best individual existing when the LEM termination condition is met is the output result of the evolution.

The Machine Learning mode termination condition is met when a plateau of the fitness function is reached while the LEM termination condition is not yet satisfied. In this case, LEM may execute the start-over operation [step *2d (ii)*] or switch to Darwinian Evolution mode. If at this point LEM always chooses the start-over operation, the evolution process is based solely on a repetitious application of Machine Learning mode. This version of LEM is called uniLEM. For the purpose of distinction, LEM's version that works in both modes is called duoLEM.

The main parameters of LEM are those that control the way the H-group and the L-group are selected and the number of new individuals that ought to be instantiated from each rule found. Other parameters control the persistence of executing each mode, the start-over operation, and termination conditions (Cervone 1999, Michalski 2000a).

Selecting H- and L-groups can be done according to a *fitness-based method*, a *population-based method*, or a combination of the two. The *fitness-based* method partitions the population using two fitness thresholds, HFT ("High Fitness Threshold") and LFT ("Low Fitness Threshold"), which specify portions of the total fitness value range in the population that are used to determine the H- and L-groups. The *population-based* method partitions the population using parameters HPT (the "high population threshold") and LPT (the "low population threshold") that specify the portions of the population to be used as H-group and L-group group, respectively. Figure 1 illustrates the latter method using a *fitness profile function* that maps individuals ordered by their fitness into the fitness value.

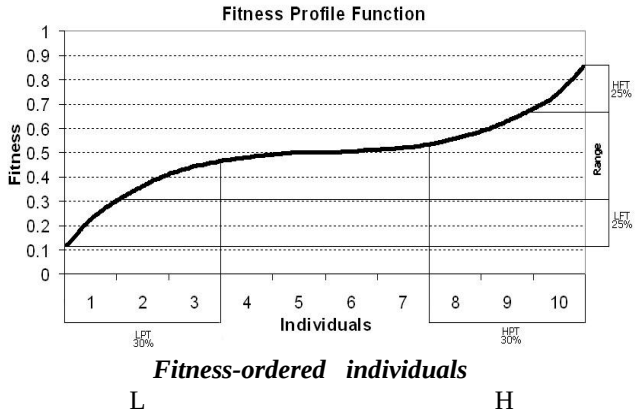

Figure 1. A fitness profile function and the HPT and LPT thresholds.

The H-group and L-group are then passed as positive and negative training examples to the AQ attributional learning program. AQ was selected because it has many feature particularly useful for LEM, such as internal disjunction and conjunction in the representation language, the ability to generate rules at different levels of generalization, and others (Michalski, 1999, 2000). AQ determines rulesets that differentiate between the H-group and L-group. As illustrated in Figure 2, these rulesets

describe a subspace of the search space that is hypothesized to contain the global optimum (or optima).

New individuals are selected from this subspace by an operation of *ruleset instantiation*. This operation creates new individuals by instantiating the variables in the rules in different ways. The learned rules typically include only a subset of the initial variables, that is, make no constraints on some variables. Variables not included in the rules are thus assigned values based on the individuals in the population.

The number of new individuals generated through instantiation is determined by the *rule fitness* [called in (Cervone, 1999), the weighted t-weight of the rule]. The rule fitness is the sum of the fitnesses of individuals covered by the rule. In calculating the rule fitness, the range of fitnesses of individuals was mapped into discrete units 1 to 5, using the χ^2 method (Cervone, 1999).

An H-group description represents a hypothesis that the area in the landscape identified by it contains individuals with a higher fitness than that of the individuals outside of that area. Such a description can thus be interpreted as a *qualitative differential* that approximates the direction of change of the fitness landscape. Selecting individuals from the area indicated this description corresponds to climbing up an extrapolated fitness landscape. This qualitative differential achieves a *qualitative zero* at the extreme points of the fitness landscape, or in the areas where it is unchanging. Thus, the qualitative zero is indicated by a flat fitness profile function and the consequent impossibility of dividing a population into distinct H- and L-groups.

The power of LEM seems to stem from computing such qualitative differentials and using them to guide the evolution process. Since qualitative differentials can be repeatedly computed in parallel, i.e., determined simultaneously in many places of the fitness landscape, LEM has higher chances to efficiently find the global optimum than methods that rely only on mutation or crossover. Moreover, if the fitness landscape has several global optima, LEM may be able to find all or a subset of them simultaneously. To do so, the machine learning method used in Machine Learning mode must be able to construct disjunctive descriptions of H-groups. If the H-group description correctly hypothesizes the direction of the landscape change, the evolution process will proceed rapidly. This is demonstrated by quantum leaps or "insight jumps" of the fitness function.

The process of computing qualitative differentials can be executed in such a way that in each iteration, the generated descriptions describe a subset of the previously described region of the search space. Figure 2 illustrates such a progressive partitioning of the search space (different shading indicates areas indicated by descriptions obtained in different generations). The symbol "1" indicates the globally optimal solution.

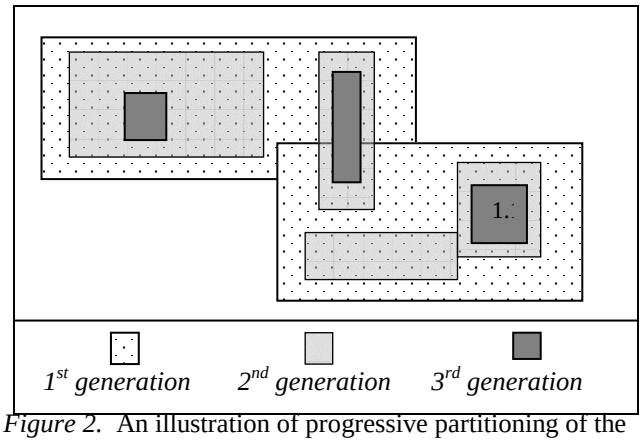

Figure 2. An illustration of progressive partitioning of the search space.

If the generated hypotheses accurately predict the region with the global optimum, such iterative partitioning of the search space leads to a rapid progress of the evolution.

3 LEM1, LEM2 and ISHED

LEM1 was the first preliminary LEM implementation described in (Michalski and Zhang, 1999) and it was developed combining the aq15c machine learning program to GA1 and GA2, two simple evolutionary algorithms. (De Jong, 1999). Despite the names lead in thinking GA1 and GA2 are traditional Genetic Algorithm, they implement a deterministic selection mechanism and a real-value representation of the variables. The main differences between the two, is that GA1 generates new individuals only through a uniform gaussian mutation operator, while GA2 implements in addition a uniform crossover.

The first application of LEM1 was to optimize a set of well-know functions, and analyze how the machine learning program improves the efficiency of the algorithm. The second application was to design a linear digital filter. (Coletti et al, 1999).

LEM2, is the newest implementation of the Learnable Evolution Model and it was programmed using EC++, a generic Evolutionary Computation Library (Cervone and Coletti, 2000). LEM2 introduced several improvements, and fixed some of the problems that arise with the early implementation. The main improvements are:

A. Employment of a new method for discretizing continuous variables, called Adaptive Anchoring Discretization, briefly, ANCHOR (Michalski and Cervone, 1999). This method replaced the χ^2 method used in LEM1 (Zhang, 1999). This method allows to gradually

increasing the resolution for the representation of continuous variables, and it leaded to drastic improvements in several problems.

B. Creation of new individuals by instantiating multiple rules rather than only the strongest rule in a ruleset generated by AQ18. This feature allows the system to explore in parallel several subareas of the search space rather than one. This feature is important in the case of multi-modal landscapes.

C. The rule instantiation takes into consideration the *rule mass*, defined as the sum of fitnesses of examples covered by the rule (the current implementation takes into consideration the rule mass per example covered)

D. Employment of the fitness-based selection, in addition to the population-based selection used in LEM1. This way, the number of examples selected as training examples is not a fixed percentage of the population, but depends on the behavior of the fitness profile function.

E. Dynamic adjustment of the cost of variables in the learning process. Each time a variable is included in a ruleset generated by the learning program, its cost is increased. This way, the system will give a preference to other variables when learning a ruleset in the next generation. This feature has proven to be useful for optimizing functions with large number of variables (> 50 variables).

F. Implementation of a simple version of the Startover operator. Specifically, when the fitness profile function is flat for a certain number of generations (defined by the *fitness_probe* parameter), the evolution is re-started from a new, randomly generated population. The best individual generated so far is, however, preserved ("elitism"). This feature has proved to be useful in cases where the system converges to a local optimum.

G. The introduction of the uniLEM mode, in which the evolution process executes only Machine Learning mode, that is, is guided solely by machine learning process, and Darwinian-type operators of mutation and cross-over are not applied.

H. The introduction of the population lookback and the description lookback in the process of generating H and L groups, as described in (Michalski, 2000). In the experiments described below, the population lookback was set to 1, and the description lookback parameter was set to 0 (no past rules are taken into consideration when generating new rules).

LEM2 was applied to optimize highly dimensional problems, and it performance compared to standard Darwinian type Evolutionary Algorithms (Cervone, 1999).

ISHEDis an implementation of the methodology for a specific domain of applications, namely the design of heat exchange systems.

4 Experiments

In this section we present some of the most distinguishing results that were achieved with the LEM methodology.

4.1 Designing Digital filters

LEM-1 with GA1 and GA2 were compared on a different type of problems, namely problems related to the design of digital filters. We present here just a sample of results (for more details, see Coletti et al, 1999). The fitness function was defined by equations specifying linear and nonlinear filters described in (Yao and Sethares, 1994).

Problem:. Determine optimal parameters of nonlinear filters defined by the equation:

$$y(k)=\left[\frac{3-0.3y(k-1)u(k-2)}{5+0.4y(k-2)u^2(k-1)}\right]^2+(1.25u^2(k-1)-2.5u^2(k)))\ln\left(\left|1.25u^2(k-2)-2.5u^2(k)\right|\right)+n(k)$$

where: k – is the sample index (or time), $n()$ – is a noise component ranging from -.25 to .25, and $u()$ – an inserted function (sin, step, random)

In this study, we assumed that coefficients -0.3, 0.4, 1.25, and -2.5 in the above equation are variables. The problem was to find their correct values from a set of $<vector_i, y(vector_i)>$ pairs, where $vector_1$ is a specific assignment of values to variables. Individuals in the population are thus vectors with four real-valued variables ("genes"). When substituted in the equation, the individual's genes yield a value of *y* that is compared with the known correct value. The fitness of an individual is inversely proportional to the difference between the result and correct value. The individual whose gene coefficients give the lowest error is assigned the highest fitness.

In the experiments, three different sets of input data to LEM-1, GA1, and GA2 were used. In Machine Learning mode, LEM-1 used the population-based method with HPT and LPT both equal 30%. The population of each generation was 20; the *learn-probe* was 3, the *learn-threshold* was 0.01*; dar-length* was *3, dar-threshold* was 0.01. Each variable was discretized into 200 ranges. Each program was executed 10 times, each time using a different input data. Runs differed in the seeds used for starting a random number generator. Presented results are averages of results obtained in these runs. LEM-1 and genetic algorithm GA1 used the same seeds.

Yao and Sethares used a uniformly distributed random input over the interval (-2.5, 2.5). In addition to this input, a unit step function 2.5u(k) and a sine wave 2.5sin(□/10)

were used for comparison. The landscape function generated an output array based on a 200 sample input sequence and stored it for comparison against the populations. Populations were generated, and the fitness of each individual was calculated by computing the mean-square error between the known values and the output generated by the individual's genes. The fitness function was defined as in (Yao and Sethares, 1994), namely, as the reciprocal of the mean-square error over the 200 sample window:

$$Fitness = \frac{1}{MeanSquareError} = \frac{200\,samples}{\sum_{200\,samplewindow}(individual - known)^2}$$

LEM-1, GA1, GA2 were applied ten times using uniform noise, sine, and step function inputs. The initial populations were generated randomly. The convergence rate varied greatly between populations and generations due to the random initial conditions. It was difficult to obtain a meaningful average performance, because a few runs would dominate the average. Therefore, for comparing performance we used learning curves that converged the fastest for each of the three systems for different input functions. For a non-linear filter, experiments were performed with uniform random input, unit step input, and sine wave input. In all cases, LEM-1 significantly outperformed GA1 and GA2. For illustration, Figures 8, 9 and 10 show results for the case of a nonlinear filter with a uniform noise input.

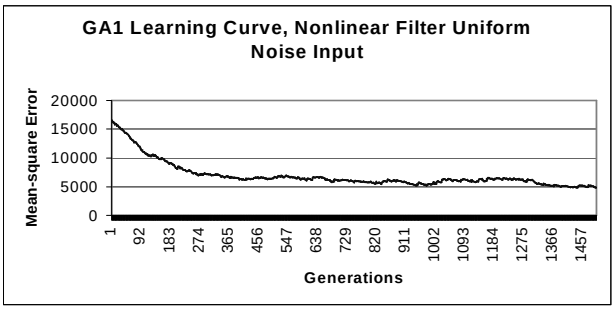

Figure 3. A GA1 revolutionary process for nonlinear filter with uniform random noise input.

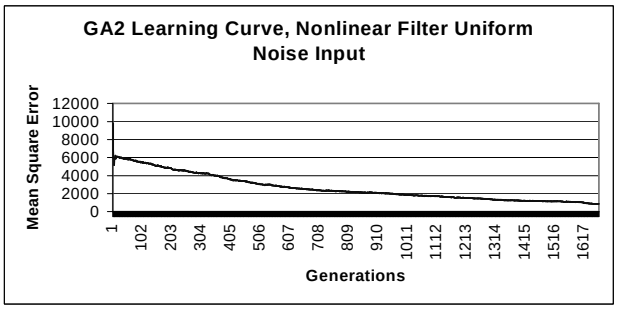

Figure 4. A GA2 evolutionary process for nonlinear filter with uniform random noise input.

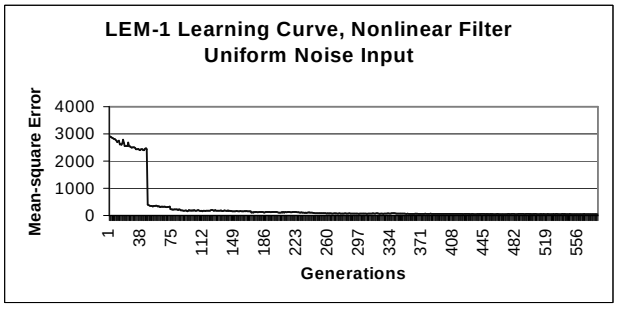

Figure 5. LEM-1 evolutionary process for nonlinear filter with uniform random noise input.

As Figures 3, 4, and 5 show, the convergence of GA1 and GA2 was relatively slow. The effect of the machine learning mode is demonstrated by a dramatic drop (an "insight jump") in the mean-square error when the system learned good rules for generating the next generation of individuals.

LEM-1 toggles between Machine Learning mode and Darwinian Evolution mode roughly 10-800 times throughout the course of a typical experiment. A dramatic drop in the mean-square error usually occurred within the first 100 generations. Because four genes were used to represent the four parameters of the filter, the error surface generated by the mean-square error is four-dimensional.

Such an error surface creates difficulties for traditional search techniques. These techniques, for example, the gradient descent and LMS, are subject to finding local minima, and they would have to run in parallel to achieve the robustness of the evolutionary computation approach. LEM-1 alleviates much of the computational cost of the genetic algorithm approach by accelerating the evolutionary process through a series of symbolic learning steps.

4.2 Optimization Problems

Below is one of many testing results. This result concerns the application of LEM to the problem of minimizing the Rosenbrock function (denoted as f_2), in which the number of arguments, *n*, was set to 100:

$$f_2(x_1, x_2, .. x_n) = \sum_{i=0}^{n} (100 \cdot (x_{i+1} - x_i)^2 + (x_i - 1)^2)$$

For comparison, an evolutionary strategy method, ES, was also applied to the same problem.

This is a rather complex optimization problem because the Rosenbrock function has a very narrow and sharp ridge and runs around a parabola, so the variables are interrelated (Figure 6).

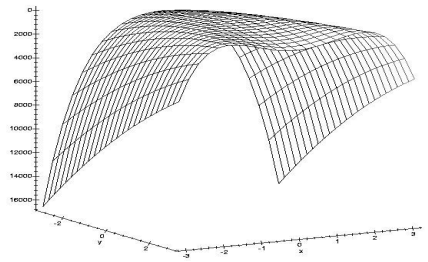

Figure 6. An inverted 2D graph of the Rosenbrock function.

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.

In Figure 7, LEM *a,b,c* means that the method was LEM, the population size was *a*, and the High and Low population thresholds (for class assigament) were *b* and *c*, respectively. ES *a,b* means that the method was evolutionary strategy with population size *a* and mutation rate *c*. ES employs a real-value representation, a deterministic selection, and the binary tournament method for the selection of the survivors. Each parent is cloned (produces only one child), and then mutated using uniform mutation (Cervone and Michalski 2000). Optionally, the Uniform Crossover operator or the One Point Crossover operators are used to generate more individuals. Finally all the new individuals compete with a randomly selected parent.

As shown in Figure 7, LEM2 was significantly less dependent on the input parameters than ES, and converged to the function minimum (the value "0") much faster than ES .

LEM2 was also compared with the best available result previously published using this function, however the only data available were relative to the original problem which involves a much smaller number of variables (2 and 4).

The results from the following experiments are summarized in a table that shows the number of evaluations needed to come δ-close to the global optimum, and the relative speedups. To evaluate the performance of the 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. Using δ, we define the *speedup* of algorithm A over B, 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.

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

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

In the case of two variables, the best result was achieved using the CHC+BLX algorithm using 4893 evaluations (Back, Hoffmeister and Schwefel, 1991). LEM2 found the global minimum using only 101 births, that is, about fifty times fewer (the speedup of nearly 5000%). Table 7 summarizes the results.

With four variables, the best published result was achieved by a breeder GA, which required about 250,000 evaluations (Schlierkamp-Voosen and Muhlenbein, 1994). LEM2 found the global solution (δ=0) with only 281 births (evaluations), that is, required about 750 times

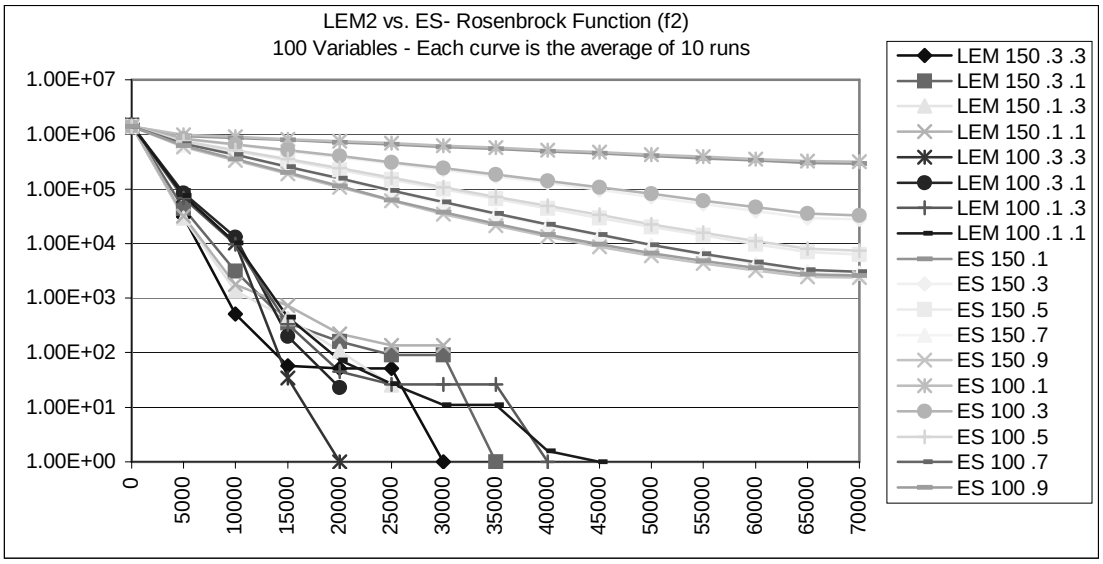

Figure 7. A graphical comparison of performance of LEM2 and ES methods.

fewer births than GA that found δ=0.1 solution (the speedup at least of 75,000). Table 2 summarises. This is the highest speedup obtained by LEM2 over a genetic algorithm for this set of problems.

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

Rosenbrock function minimization 4 vars	
LEM2	δ=0: 281
GA	δ=0.1: 77,000
Speedup LEM2/GA	27,500%

This strong results indicates that LEM2 was able very quickly locate the area of the landscape with the global optimum. In achieving this result, LEM2 was helped by the ANCHOR method that gives preference to values that represented this optimum.

4.3 An Application to Heat Exchanger Design

LEM was also tested on real-world problems, such as parameter estimation for digital filters (Coletti et al. 1999), and the optimization of heat exchangers. For the latter application, we have developed ISHED-1, a LEM implementation specifically tailored to a class of design problems (Kaufman and Michalski 2000a). ISHED-1 conducts an evolutionary optimization process to determine the best arrangement of the evaporator tubes in the heat exchanger of an air conditioning system under given technical and environmental constraints.

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 liquifies. Coming back to the evaporator, it comes into contact with the warmer interior air that is being pushed through the heat exchanger, as a result cooling the air while heating and evaporating the refrigerant. The heat exchanger consists of an array of parallel tubes through which the refrigerant flows back and forth.

ISHED-1 is able to apply background knowledge based on the nature of the problem in order to constrain its search for the best ordering of the tubes search to plausible architectures. A user-defined parameter imposes limitations on the lengths of most tube bends. Additionally, the program enforces six real-world constraints on generated designs, ranking from suggested to essential. The program rejects structures that violate a required constraint, and only under special circumstances (namely when designing a more coherent architecture is very difficult) generates structures from scratch that violate the more lenient constraints.

For example, a constraint limiting splits in refrigerant paths is based on the unacceptable drops in refrigerant pressure that will occur if a single path undergoes multiple splits. Another constraint requiring inlets and outlets to be on the same side of the heat exchanger manifold is based on the structural requirements of the air conditioning unit, and a third constraint forbids looping in the refrigerant path.

An ISHED-1 run proceeds as follows: Given instructions characterizing the environment for the sought heat exchanger design, an initial population of designs (either specified by the user, randomly generated, or a

combination of the two), and parameters for the evolutionary process, it evolves populations of designs using a synthesis of specially designed Darwinian and symbolic evolution operators for a specified number of generations, and returns a report that includes the best designs (architectures) found and their estimated capacity.

Troughout the execution, design capacities are determined by a heat exchanger simulator (Domanski 1989).

During the course of ISHED-1 development, many experiments with the system were conducted. The best ISHED-produced architectures conformed intuitively to expectations of the general form of a successful architecture in the given airflow environment, and some performed far better than currently-used expert-designed structures in situations of non-uniform airflow.

5 Conclusions

6 References

Back, T., Hoffmeister, F., and Schwefel, H. (1991). A Survey of Evolution Strategies, *Proceedings of the Fourth International Conference on Genetic Algorithms and their Applications,* Morgan Kaufmann.

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.

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*, MLI 00-2, George Mason University, Fairfax, VA.

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 (GECCO)*.

De Jong, K.A., *Evolutionary Computation: Theory and Practice*, MIT Press, 1999 (to appear).

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization 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. (2000). ISHED-1: Applying the LEM Methodology to Heat exchanger Design. *Reports of the Machine Learning Laboratory*, George Mason University, Fairfax, VA (to appear).

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

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

Michalski, R.S. (1983). A Theory and Methodology of Inductive Learning. In Michalski, R.S. Carbonell, J. and Mitchell, T. eds. *Machine Learning: An Artificial Intelligence Approach*. Palo Alto: TIOGA Publishing Co., 83-134.

Michalski, R.S. (1998). Learnable Evolution: Combining Symbolic and Evolutionary Learning. *Proceedings of the Fourth International Workshop on Multistrategy Learning (MSL'98)*, 14-20.

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

Michalski, R.S. (2000). Natural Induction: A Theory, Methodology, and Applications to Machine Learning and Knowledge Mining. *Reports of the Machine Learning Laboratory*, George Mason University, Fairfax, VA (to appear).

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

Michalski. R.S. and Zhang, Q. (1996). 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.

Muhlenbein, H., Schomisch, M. and Born, J. (1996). The Parallel Genetic Algorithm as Function Optimizer, *Proceedings of the Fourth International Conference on Genetic Algorithms and their Applications*, Morgan Kaufmann.