The LEM3 Implementation of Learnable Evolution Model and Its Testing on Complex Function Optimization Problems

Janusz Wojtusiak and Ryszard S. Michalski*

George Mason University
4400 University Drive MSN 5B2
Fairfax, VA 22030 USA
(*) Also with Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

{jwojt, michalski}@mli.gmu.edu

ABSTRACT

Learnable Evolution Model (LEM) is a form of non-Darwinian evolutionary computation that employs machine learning to guide evolutionary processes. Its main novelty are new type of operators for creating new individuals, specifically, *hypothesis generation,* which learns rules indicating subareas in the search space that likely contain the optimum, and *hypothesis instantiation,* which populates these subspaces with new individuals. This paper briefly describes the newest and most advanced implementation of learnable evolution, LEM3, its novel features, and results from its comparison with a conventional, Darwinian-type evolutionary computation program (EA), a cultural evolution algorithm (CA), and the estimation of distribution algorithm (EDA) on selected function optimization problems (with the number of variables varying up to 1000). In every experiment, LEM3 outperformed the compared programs in terms of the evolution length (the number of fitness evaluations needed to achieved a desired solution), sometimes more than by one order of magnitude.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning – *Concept Learning, Induction,* G.1.6 [**Optimization**].

General Terms

Algorithms, Performance, Design, Experimentation, Theory.

Keywords

Evolutionary Computation, Learnable Evolution Model, Function Optimization, Machine Learning

1. INTRODUCTION

Research on non-Darwinian evolutionary computation is concerned with developing algorithms in which the creation of new individuals in the population is guided by an "intelligent agent," rather than done merely by random or semi-random change operators, such as mutation and/or crossover, employed in the "Darwinian-type" evolutionary methods. The selection of individuals for the new generation from among those generated by the intelligent agent can be done using standard methods of selection, or can also engage such an agent.

The Learnable Evolution Model (LEM), introduced in [9, 10], and the topic of this paper, employs a learning program for directing the process of creating new individuals. Specifically, at each step of evolution, the method creates general hypotheses indicating regions in the search space that likely contain the optimal solution (or alternative optimal solutions), and then instantiates these hypotheses to generate new individuals. Early implementations of the model, LEM1 and LEM2, gave very promising results on selected function problems (e.g., [9], [2]). We have also developed domain-oriented implementations, ISHED and ISCOD, that were tailored to problems of optimizing heat exchanger designs. They also produced highly satisfactory results, as they generated designs that matched or improved upon human designs [3].

An implementation of Learnable Evolution Model for Multi-objective Optimization, LEMMO, developed independently [5], is based on rules derived from decision trees learned by the C4.5 program. LEMMO was recently applied to a water quality optimization problem. The decision tree representation of the hypotheses is, however, significantly more limited than the attributional rule representation in LEM implementations, and is also more difficult to instantiate.

Work related to LEM includes research on *cultural algorithms* [15, 16], which use additional information about solutions to guide mutation and recombination operators. The cultural algorithms perform a constrained optimization process in which constraints are created during the evolutionary computation. The constraints, called beliefs, reside in a belief space that is updated during the evolution process. Individuals that are stored in an optimization space are modified so that they satisfy the beliefs. The belief space is built based on statistical information about individuals, which usually consists of intervals containing the fittest individuals.

Other related work concerns *Estimation of Distribution Algorithms* (EDAs), methods that use statistical inference, usually Bayesian or Gaussian networks, to generate distributions of high-performing individuals selected from one population [7, 8, 13, 14]. LEM significantly differs from EDAs, however, as it employs symbolic learning rather than statistical methods, and seeks rules for distinguishing between high- and low-performing individuals, while EDAs seldom use contrast sets. It also uses the fitness function not only for selecting individuals for learning but also during the learning process itself, while most EDAs use it

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

solely for selecting individuals; an exception is a method described in [13]. LEM does this by learning significance-based descriptions.

This paper describes briefly LEM3, the latest and most advanced implementation, and its comparative evaluation on a range of function optimization problems. LEM3 employs the most recent AQ-type learning program, AQ21 [5], and includes several significant improvements over earlier versions. These include the abilities to represent solutions using a wide range of different attribute types, to take into consideration these types in the hypothesis formation process, to control this process according to the problem at hand, as well as new methods for selecting different actions (modes) in the process of evolutionary computation, and to instantiate hypotheses in several new ways.

2. DESCRIPTION OF LEM3

This section describes the top-level structure of LEM3. It contains several components that are also found in traditional evolutionary algorithms, such as generation of an initial population, selection of individuals for a new population, and evaluation of individuals.

Components that are unique to LEM3 are concerned with guiding evolutionary computation through machine learning. This is done by selecting at each step of evolution the highest and lowest performing individuals in the population, the H- group and L-group, respectively, and then employing the AQ21 learning program to generate a hypothesis that differentiates between the two groups. The hypothesis is then instantiated in various ways to generate new individuals. Figure 1 presents the top-level algorithm underlying LEM3.

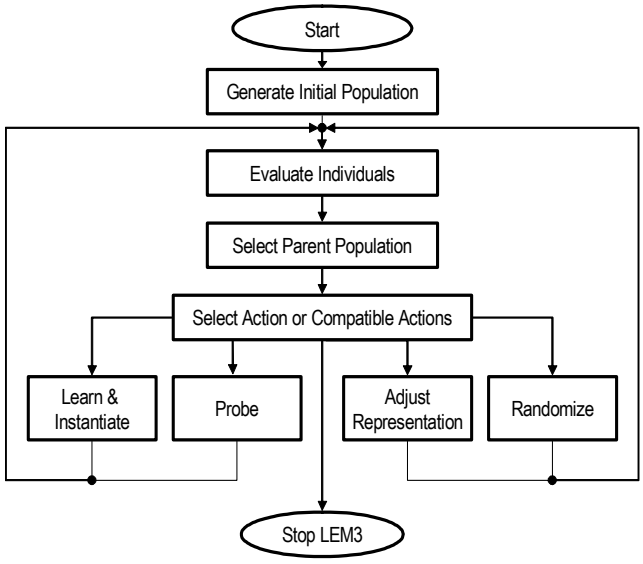

Figure 1: The top-level structure of LEM3.

The following sections describe algorithms underlying the major LEM3 components.

2.1 Evaluate Individuals

This component determines the value of the fitness function for every individual in the population. This may be a simple operation if the fitness is defined by a mathematical formula. In many applications, however, fitness evaluation may require a time consuming or costly process of running a simulator, solving a set of complex equations, or even performing an experiment. Such situations occur, for example, when designing heat exchangers [3], optimal non-linear filters, and aircraft wing shapes.

Due to the capabilities of the AQ21 learning program, LEM3 allows a user to describe individuals (problem solutions) and the fitness function in terms of not only numeric attributes, but also in terms of other attribute types, such as nominal, rank, cyclic (e.g., days of the week), structured (representing hierarchies), interval, and ratio [11]. Thus, one does not need to design an ad-hoc attribute encoding, as sometimes done, for example, in genetic algorithms, but instead can directly use attributes as specified in problem definition. This feature extends LEM3's applicability to domains in which individuals are characterized by a combination of qualitative and qualitative properties. For example, when optimizing the design of a physical device, attributes characterizing it may include numerical ones that describe its length, width, height, as well as symbolic ones that characterize the material it is made of (structured) or modes of operations (nominal).

2.2 Select Parent Population

Once individuals are evaluated, a new population is created by combining new individuals and individuals from the old population. In general, the creation of the new population involves two steps: (1) creation of a (possibly large) temporary population and (2) selection of individuals from it into the new population.

The temporary population consists of all newly created individuals and also, depending on parameters, individuals from the old population. If the number of created individuals is smaller than the desired population size, the program selects individuals from the previous population to meet the target population size. It is usually convenient to add all individuals from the previous population into the temporary group. This way, there is no need to keep track of past populations for more than one iteration, because they were already represented in previous steps.

Individuals are selected from the temporary population into the new population using methods developed in evolutionary computation. LEM3 implemented three methods: selection of the best individuals, also known as rank-based selection; tournament selection; and proportional selection, also known as roulette wheel selection. At the end of the selection process, a new population is created that serves as the parent population for the next steps. LEM3 also keeps an elite consisting of one or more (depending on the elite-size parameter) individuals with the highest score of the fitness function. Note that using rank-based selection from the temporary population guarantees elitism.

2.3 Select One or More Actions

The next step is to choose and apply an action to create new individuals from the parent population. As shown in Figure 1, these actions include learn and instantiate, probe, adjust representation and randomize. Which action or combination of actions is performed is decided by the "Select Action or Compatible Actions" module, which uses an *Action Profiling Function* (APF) introduced in LEM3.

How does the "Select Action or Compatible Actions" decide which action to perform? Initially, by default, it selects the "Learn and Instantiate" action. When after a number of iterations no progress is observed, as defined by the *learn-probe* and *learn-threshold* parameters, the program switches to the "Probe" action which is applied once and program returns to "Learn and Instantiate" mode. The *learn-probe* parameter defines the maximum number of iterations the action is performed even if the progress is unsatisfactory, as defined by *learn-threshold* parameter, which specifies the minimal improvement of fitness value of the best individual to accept progress as satisfactory. The program counts how many times the "Probe" action was applied, and if the number reaches *mutation-probe*, it switches to the "adjust representation" action, which is applied once before returning to "Learn and Instantiate". *Representation-probe* sets a limit on the number of times the representation is adjusted before switching to the "Randomize" action and randomly generating new individuals. Figure 2 presents pseudocode describing the above process.

```
Increment learn-probe-counter
If learn-probe-counter >= learn-probe
   Learn-probe-counter = 0
   If mutation-probe-counter < mutation-probe
      Increment mutation-probe-counter
      Mutate individuals (Probe)
   Else if representation-probe-counter < representation-probe
      Increase representation-probe-counter
      Mutation-probe-counter = 0
      Adjust discretization
      Mutate individuals
   Else if randomize- counter < randomize-Probe
      Increment randomize-probe-counter
      Representation-probe-counter = 0
      Mutation-probe-counter = 0
      Rollback discretization
      Add the best individuals to a list of local optima
     Randomize
   Else
      Stop LEM3
```

Figure 2: Pseudocode describing action selection in LEM3.

The chosen order of *mutation*, *adjust representation*, and *randomize* operations is deliberate. Mutation is performed in order to introduce diversity into a population and assure that the program does not get stuck near a local or global optimum.

The latter situation may occur when the learning program cannot learn hypotheses because the training set is uniform, and thus can not create different H- and L-groups. Next, the precision of the representation of individuals is increased by adjusting discretization. If the increase in precision not does help, it may mean that the program has found an optimum. However, the optimum may be local, so it is desirable to perform additional iterations with new, randomly generated populations in order to explore different parts of the search space and test the found best solution for optimality. The above four actions are explained in greater detail in the following sections.

2.3 Learn and Instantiate Action

The "Learn and Instantiate" action is the most important component of the Learnable Evolution Model. This action creates new individuals by performing three steps: (1) selecting the training set for the learning program (2) learning a hypothesis characterizing subspaces that likely contain the optimum, and (3) instantiating the hypothesis in various ways to create new individuals.

Step (1) selects high-performing (H-group) and low-performing (L-group) individuals from the population, according to the given fitness function. These individuals serve as positive and negative examples, respectively, for the AQ21 learning program. There are two methods of creating these groups. The first one, *Fitness-Based Selection*, employs high and low fitness thresholds in the range from the highest to the lowest fitness value observed in the current population. For example, if High and Low Fitness Thresholds (HFT and LFT) are chosen to be 25%, then individuals whose fitnesses are in the highest 25% of the range and the lowest 25% of the range are included in the H-group and L-group, respectively. The second method, *Population-Based Selection*, selects a specified percentage of individuals from the population for each group, regardless of the distribution of fitness values. These percentages are defined by the High Population Threshold (HPT) and Low Population Threshold (LPT). For example, if HPT and LPT are both 30%, then the 30% of the individuals with the highest fitness and the 30% with the lowest fitness are included in the H- and L-group, respectively.

The H- and L-groups are then passed as positive and negatives examples to a learning program, which in LEM3 is AQ21. This program is the newest implementation of the AQ learning, an inductive learning method that produces hypotheses in the form of sets of *attributional rules*[13]. The simplest form of such a rule is:

CONSEQUENT ← PREMISE

where CONSEQUENT and PREMISE are conjunctions of *attributional conditions* (a.k.a. *selectors*). An attributional condition defines a relation between an attribute and attribute values that satisfy that condition. Here is an example of an attributional rule:

[design = acceptable] ← [weight = 2..5] &
[shape= rhombus v triangle] &
[height < 3]

The rule states that a design is classified as acceptable if its weight is between 2 and 5, its shape is rhombus or triangle, and its height is less than 3 (units of weight and height are defined in the attribute domain).

A hypotheses learned by AQ21 usually consist of a number of such rules. This representation of a hypothesis is very useful for LEM, because such individual rules can be easily instantiated. They are also have high expressive power and are easy to interpret and understand.

The main operator used for generating rules in AQ is *extension-against*. Provided with a positive example, called a *seed*, and a negative example, this operator generates a *partial star*, which is defined as the set of maximal generalizations of the seed that do not cover the negative example. The intersection of all partial stars of the seed against every negative example is called a *star*, which is a set of maximal generalizations of the seed that do not cover any negative example. In order to prevent the exponential growth to the size of the star, AQ employs a *beam search* that limits the number of generalizations (single attributional rules) that are retained for the next step from each extension-against and intersection operation. AQ selects the best rule from the generated

star, and selects a new seed from positive examples uncovered by the previously selected rules. This process continues until all positive examples are covered. Such an algorithm guarantees that the learned ruleset is complete and consistent with regard to the training data, provided that training examples are consistent (do not represent more than one class simultaneously). Figure 3 presents the basic AQ algorithm in pseudocode.

```
HYPOTHESIS = null
While not all H-group examples are covered
  Select uncovered positive example e+ and use it as a seed
  Generate star G(e+, L-group)
  Select the best rule, R, from the star according to a given
     criterion of optimality, and add it to HYPOTHESIS
  Remove examples covered by R from H-group
```

Figure 3: Pseudocode of the basic form of the AQ algorithm.

The best rule is selected from a star using a *Lexicographical Evaluation Function* (LEF) that combines several elementary criteria, as specified by the user [18]. For further details on AQ learning and some of its more extended forms, see, e.g. [6].

The instantiation process (Step 3) generates new individuals that satisfy the learned hypothesis. When instantiating a rule to create an individual for the new population, the program faces two problems: (a) which values to assign to attributes that are specified in the rule (b) which values to assign to attributes not present in the rule. Depending on the attribute type and user-defined parameters, different probability distributions can be used to select values for the attributes specified in the rule. This can be done, for example, using uniform distribution over ranges of values in the rules, a normal distribution for numerical attributes with the mean in the middle of the range and the variance defined by the user, or a distribution that rewards individuals maximally distant from the closest negative example.

Selection of values of attributes not specified in the rule is a more intricate problem that can have many different solutions. One way is to select a random value from the entire attribute domain, which will result in individuals consistent with the rule; however, it is easy to show cases in which this method will lead to poor results. For instance, assume that we are optimizing a function with two attributes x and y. Both attributes are continuous and defined on the range -5 to 5. Suppose that the function optimum is at the point (0, 0), and that AQ21 learned a rule [x = 0]. The method will generate individuals with x = 0, and with y distributed over the range [-5, 5]. In the next iteration, AQ21 will learn rules containing only the attribute y, since there is no longer any differentiation among the x-values. During the instantiation phase, the program will now assign values of the attribute x randomly, which means that the information from the previous iteration is lost. Such a process will thus not converge to the optimal solution.

Another method is to select a value from a randomly selected existing individual. The individual can be selected from the H-group only, from non-L-group individuals, or from the entire past population. Experiments have shown that when selecting values from the H-group, the program tends to lose diversity of individuals, and may converge very quickly to a point that may not the globally optimal solution. The default method used by LEM3 selects individuals from the whole population probabilistically, in proportion to their fitness levels. A pseudocode of the instantiation algorithm used in LEM3 is presented in Figure 4.

```
For each rule in a ruleset (hypothesis) to be instantiated
  Compute the number of individuals to be created
  For each individual to be created
    Create the individual
    For each attribute
      If the attribute is specified in the rule
        Select a random value satisfying the rule
      Else Select a random individual from the previous
        population and use its value
```

Figure 4: Basic instantiation algorithm.

2.4 The Probe Action

The probe action executes Darwinian-type operators in order to generate new individuals. Two probing operators are implemented in LEM3, namely, mutation and crossover. Because in LEM3 representation of variables depends on their type, and individuals may be built of variables of many types, the crossover operator is based on selection of whole attributes' values. After selecting two parents, two new individuals are created by taking values from the parents. The mutation operator in LEM3 is more complex. LEM3 effectively uses the semantics of different variable types, which are also used by the mutation operator. For example, a random change of structured variables reflects represented hierarchies.

2.5 The Representation Change Action

Adjusting the representation space of solutions may include removing irrelevant variables, adjusting domains of variables, and creating new variables that are more relevant to the optimization problem. Although all three types of operations are being investigated in the LEM methodology, LEM3 currently implements only adjustment of attribute discretization.

The program uses an adaptive discretization method that changes the attribute precision when it is required. The method that is used by default is a version of *Adaptive Anchoring Discretization* (ANCHOR) [12], which discretizes continuous attributes with a granularity size changing in the process of evolution. The method starts with an initial very rough discretization. Once it starts converging toward a possible solution, the precision of numeric attributes is increased in the intervals suggested by the learned hypothesis.

2.6 The Randomize Action

This action adds randomly generated individuals to the new population, or replaces the entire population by a new, randomly generated population. The randomize action aims at adding diversity to the current population or to start the evolution from scratch. It is applied when the program appears to be stuck at a local optimum, and needs to explore other parts of the search space. Such a situation is detected by the no-progress condition, when learn & instantiate, mutation, cross-over, and discretization adjustments do not lead to the improvement of the current solution, and the solution need to be tested further before the evolutionary process ends. New individuals are created either (1) randomly in the entire space, (2) randomly in the parts of the space that were not explored so far, or (3) randomly by

maximizing distance from the local optima found so far. The first method is the simplest one, and is equivalent to a full restart of the LEM3 algorithm. The second method requires keeping track of all values of attributes that appeared in past individuals, in order to distribute individuals over the parts of the space not yet explored. The third method builds distributions based on a list of optima found so far. Individuals that are farther from the found best solutions have a higher probability of being selected.

3. EXPERIMENTAL RESULTS

The goal of these experiments was to test the performance of LEM3, and compare its results with those obtained by other evolutionary computation methods, including the previous LEM implementation (LEM2), a conventional evolutionary algorithm (EA), cultural algorithms (CAs), and Estimation of Distribution Algorithms (EDAs). In these experiments, LEM3, LEM2 and EA were applied to a group of benchmark function optimization problems. To compare LEM3 with CAs and EDAs, LEM3 was applied to problems for which results from CAs and EDAs are available in the literature. The problems involved optimization of the *Rastrigin*, *Griewangk*, and *Rosenbrock* functions of different numbers of variables, ranging from 2 to 1000, depending on compared programs. These functions were chosen because they are frequently used for testing evolutionary algorithms. They are described, for example, in [17].

EA is an implementation of a conventional, Darwinian-type evolutionary algorithm taken from library *EO* (*Evolutionary Objects*) *0.9.3a* that can be downloaded from URL: http://eodev.sourceforge.net [4]. The EO library was selected because it contains an implementation of a Darwinian-type evolutionary algorithm that can work with large numbers of variables (we tried other programs, but they support far fewer variables), is well described in the available tutorial, and is easily downloadable from the internet.

LEM3 and EA were applied to optimizing functions of between 10 and 1000 variables. LEM2 was applied to optimizing functions of 10 and 100 variables (it does not work with 200 or more variables). Each experiment involving LEM3, LEM2 and EA on optimizing a function of a given number of variables was repeated 10 times with a different starting population. To make a fair comparison, the same starting populations were used by each program.

The LEM3, LEM2 and EA results are reported for *δ-close solutions,* which are at a normalized distance from the optimal solution. A δ-close solution, *s*, is a solution for which function *δ(s)*, defined as:

$$\delta(s) = \frac{|opt - v(s)|}{|opt - init|}$$

reaches an assumed δ-target value. In this equation, *init* is the evaluation ("fitness value") of the best solution in the initial population, *opt* is the globally optimal value, and v(s) is the evaluation of the found solution s. Such a measure works for both maximization and minimization problems, that is, for problems in which the optimal solution reaches the maximal or the minimal value of performance evaluating function (the "fitness function).

This definition of a δ-close solution suggests two possible ways of analyzing performance of evolutionary computation methods. One is to determine the *evolution length,* denoted FE(δ<=k), and defined as the smallest number of fitness function evaluations needed to achieve a solution, *s* for which δ(s) <= k, by the best individual in the population, where k is a number between 0 and 1. This is the main measure used to report results in this paper. The second way is to determine δ(s) for the best solution, s, found after a specified number of fitness evaluations.

For example, if the fitness value of the best individual in the initial population is 10 and during the process of minimization the program achieved value 0.1, while the optimal value is 0, δ is 0.01, indicating that the program found a solution within 1% distance from the optimal solution, normalized by the fitness value of the best individual in the initial population. Figure 5 illustrates the concept of a δ-close solution.

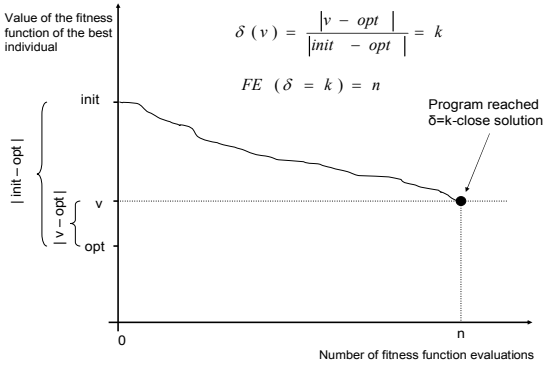

Figure 5: Illustration of a δ-close solution.

In the experiments, tested programs were compared using their default parameters. This reflects real-world situations in which only one run of the optimization method is done, because the evaluation of the fitness function is very difficult. LEM3, LEM2 and EA were executed with population size 100. Both LEM3 and LEM2 used the ANCHOR adaptive discretization method. An evolutionary speedup is defined as a ratio of evolution lengths of two programs for a given value of δ.

EA was executed with the following parameters: the probability of mutation was 0.1, the probability of crossover was *0.6*, and the selection method was *tournament*. It used two types of crossover, standard, which creates new individuals by taking values form two parents, and hypercube crossover, which uniformly selects a point in the hypercube spanned by the two parent individuals (for details see website http://eodev.sourceforge.net).

The authors will be glad to provide actual starting populations, programs, scripts used to run experiments, actual result files, and all other relevant information to those interested in reproducing the presented results or trying other methods.

3.1 Function Optimization Problems

Three well-known benchmark function optimization problems were used for testing LEM3 and comparing its performance with that of other methods. The first problem was to minimize the Rastrigin function defined by the following formula:

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

The function has a large number of local optima, and one global optimum equal to zero. It is reached when all the variables equal zero.

The second problem was to minimize the Griewangk function, defined by the following formula:

$$f(x_1,\ldots,x_n) = 1 + \sum_{i=1}^{n} \frac{x_i}{4000} - \prod_{i=1}^{n} \cos(x_1/\sqrt{i})$$

Similarly, this function has a large number of local optima, and one global minimum equal to zero, which is reached when all the variables equal zero.

The third benchmark problem was to minimize the generalized Rosenbrock function, which is defined by the following formula:

$$f(x_1,\ldots,x_n) = \sum_{i=1}^{n-1}(100*(x_{i+1}-x_i^2)^2 + (x_i-1)^2)$$

The function has one global optimum equal to zero. It is reached when all the variables equal one. The function has a ridge along the line for which values of all attributes are equal.

We selected the three problems because they are well-known benchmark problems that are hard and scalable [17].

3.2 Comparing LEM3 with CA

This section presents results from comparing results from LEM3 with results from Cultural Algorithm, CA, on the Rastrigin, Griewangk, and Rosenbrock functions of 2, 3, and 5 variables. The comparison was limited to only such a small number of variables because only for these numbers were results from CAs reported in the literature [15]. Results are presented in Table 1. They are means of the results from 40 runs.

To compute speedup, whose definition is based on the value of δ, we assumed that the best individuals in the initial populations used by the programs were the same, and to stopped LEM3 when it found any fitness value better than one found by CA. The same assumptions were used when computing speedups for EDA in the next section. The programs converged to very close fitness values, thus comparison of evolution lengths is meaningful.

Table 1: Comparison of LEM3 with CA on the Rastrigin, Griewangk, and Rosenbrock functions.

Function # vars.	Method	Best fitness Value	Evolution Length	LEM3/CA Speedup
Rastrigin 5 vars.	LEM3	0	687	**~728**
	CA	5.4532e-05	~500,000	
Griewangk 3 vars.	LEM3	0	1,521	**~53**
	CA	1.0E-10	~79,900	
Rosenbrock 2 vars.	LEM3	0	219	**~243**
	CA	1.0e-10	~53,200	

Table 1 presents the best results obtained by different variants of cultural algorithms, as reported in [15]. As can been seen, LEM3 very significantly outperformed CA: over 700 times for the Rastrigin function, almost 250 times for the Rosenbrock function, and over 50 times for the Griewangk function. Moreover, in all cases, LEM3 found exact solutions (δ=0), while CAs found only approximate solutions, as reported in [15].

3.3 Comparing LEM3 with EDA

This section presents a comparison of LEM3 with different implementations of Estimation of Distribution Algorithm (EDA) reported in [1]. Only the best result from different version of EDA is presented. The result was obtained using $EMNA_{global}$, an EDA method based on multivariate normal distribution. For a detailed description of the method, refer to [7]. We used reported results on the Griewangk and Rosenbrock functions of 10 and 50 variables. The presented values are mean for 10 runs.

Table 2: Comparison of LEM3 with EDA on the Rastrigin, Griewangk, and Rosenbrock functions.

Function # vars.	Method	Best fitness Value	Evolution Length	LEM3/EDA Speedup
Griewangk 10 vars.	LEM3	0	1,305	**~ 231**
	EDA	0.051166	301,850	
Griewangk 50 vars.	LEM3	0	4,005	**~ 54**
	EDA	8.7673E-6	216,292	
Rosenbrock 10 vars.	LEM3	1.2	1,389	**~ 118**
	EDA	8.6807	164,519	
Rosenbrock 50 vars.	LEM3	46.74	7,875	**~ 15**
	EDS	48.8234	275,663	

LEM3 outperformed the compared EDA for both functions of 10 and 50 variables, LEM3 was in average about 174.5 times faster for 10 variables, and about 34.5 times faster for 50 variables when compared with results reported [1]. The presented results are preliminary and further investigation is needed to fully compare the methods especially for larger numbers of variables.

3.4 Comparing LEM3 with LEM2 and EA

This section presents results from comparing LEM3 with LEM2, the previous implementation of Learnable Evolution Model, and EA representing Darwinian-type method on the above three function optimization problems. It is important to note that in all experiments with LEM3, LEM2 and EA, the default parameter settings were used and none of the program parameters were fine-tuned to achieve better results. In real world problems, especially in solving hard optimization problems, there is frequently only one run of the method made, because of the high cost (time or difficulty) of evaluating fitness. In such cases the users cannot change parameters and try again. For comparison purposes all experiments were repeated 10 times with different starting populations which were the same for all programs.

The comparative results of LEM3, LEM2 and EA for the Rastrigin, Griewangk, and Rosenbrock functions for δ=0.1- and δ=0.01-close solutions are presented in Tables 3 - 8. The presented results show the superiority of LEM3 over other compared programs in terms of the evolution length (the number of fitness function evaluations needed to achieve a δ-close solution). The speedup LEM3/EA varied for different problems. The average speedup LEM3/EA was about 16.5 times for functions of 100 or more variables for δ=0.1 and δ=0.01.

A very important result of the experiments was that the evolutionary speedup of LEM3 over the Darwinian-type method, EA, usually grew with the number of variables of the tested functions up to about 500-1000 variables, depending on the type of function being optimized, and then tended to stabilize. This feature makes LEM3 particularly attractive for optimizing complex systems with large number of controllable variables when the fitness evaluation is non-trivial, time consuming or costly. The reason for speedup stabilization at very high numbers of variables is hypothesized in the Summary.

A question arises as to whether LEM3 has advantage over EA and other programs also in terms of execution time. A simple experiment demonstrated that whenever evaluation of fitness function takes longer than a small fraction of second, the increased computational time of hypothesis formulation and instantiation is compensated by the shorter evolutionary length, and LEM3 wins also in terms of execution time [20].

Table 3: LEM3. LEM2 and EA evolution length and speedup on optimizing Rastrigin function for δ=0.1.

# of Variables	Evolution Length for δ=0.1			LEM3/EA Speedup
	LEM2	EA	LEM3	
10	374	2,673	415	~4
100	2,451	28,402	2,270	~13
200	--	56,465	3,302	~17
300	--	82,809	4,113	~20
400	--	106,687	4,820	~22
500	--	128,184	5,252	~24
600	--	152,291	5,652	~27
700	--	184,172	6,053	~28
800	--	191,768	6,440	~30
900	--	208,246	7,491	~28
1000	--	244,408	7,481	~33

Table 4: LEM3, LEM2 and EA evolution length and speedup on optimizing Rastrigin function for δ=0.01.

# of Variables	Evolution Length for δ=0.01			LEM3/EA Speedup
	LEM2	EA	LEM3	
10	8,59	12,419	1,000	~12
100	6,723	114,445	5,298	~22
200	--	283,523	7,705	~37
300	--	409,591	10,471	~39
400	--	584,363	12,708	~46
500	--	631,218	16,195	~40
600	--	727,158	22,173	~33
700	--	1,134,610	26,375	~43
800	--	884,545	30,124	~29
900	--	1,214,476	37,026	~33
1000	--	1,418,323	43,090	~33

Table 5: LEM3 and EA evolution length and speedup on optimizing Griewangk function, δ=0.1.

# of Variables	Evolution Length for δ=0.1		LEM3/EA Speedup
	EA	LEM3	
10	2,579	268	~10
100	24,611	1,797	~14
200	50,145	2,985	~17
300	75,345	4,370	~17
400	101,810	5,401	~19
500	126,057	6,547	~19
600	151,382	7,227	~21
700	177,221	8,161	~22
800	202,317	9,001	~22
900	226,499	9,959	~23
1000	251,233	10,780	~23

Table 6: LEM3, and EA evolution length and speedup on optimizing Griewangk function, δ=0.01.

# of Variables	Evolution Length for δ=0.01		LEM3/EA Speedup
	EA	LEM3	
10	7,367	3,223	~2
100	52,632	10,486	~5
200	105,453	20,003	~5
300	157,320	29,799	~5
400	211,341	40,215	~5
500	263,801	51,564	~5
600	314,888	59,881	~5
700	369,915	72,437	~5
800	422,357	86,017	~5
900	473,310	97,606	~5
1000	525,096	112,600	~5

Table 7: LEM3, LEM2 and EA evolution length and speedup on optimizing Rosenbrock function for δ=0.1.

# of Variables	Evolution Length for δ=0.1			LEM3/EA Speedup
	LEM2	EA	LEM3	
10	275	541	325	~2
100	918	3,367	1,906	~2
200	--	5,699	2,625	~2
300	--	8,547	3,518	~2
400	--	11,690	4,038	~3
500	--	14,960	4,519	~3
600	--	15,606	5,013	~3
700	--	19,448	5,491	~4
800	--	22,731	5,710	~4
900	--	25,216	6,835	~4
1000	--	28,468	6,851	~4

Table 8: LEM3, LEM2 and EA evolution length and speedup on optimizing Rosenbrock function for δ=0.01.

# of Variables	Evolution Length for δ=0.01			LEM3/EA Speedup
	LEM2	EA	LEM3	
10	492	2,027	682	~3
100	2,348	26,944	3,495	~8
200	--	57,588	4,922	~12
300	--	89,280	6,158	~18
400	--	120,056	9,872	~12
500	--	145,984	12,655	~12
600	--	178,358	15,951	~11
700	--	209,274	16,931	~12
800	--	234,348	22,843	~10
900	--	259,168	25,065	~10
1000	--	296,879	29,691	~10

4. SUMMARY

The presented LEM3 system is the most advanced implementation of the Learnable Evolution Model to date and it employs the most recent AQ-type learning program. In some aspects, the algorithms implemented in LEM3 go beyond the methodology described in [10]. For example, LEM3 introduces the Action Profiling Function and new instantiation algorithms.

An experimental application of LEM3 to very complex function optimization problems (with up to 1000 variables) achieved a superior performance over EA, a standard Darwinian-type method. Comparisons with published results on Estimation of Distribution Algorithms, and Cultural Algorithms also show a clear superiority of LEM3. LEM3 showed a high scalability that could not be achieved with previous implementations. Extensive experiments with LEM3 thus have confirmed that it is a powerful new optimization system that outperforms other evolutionary computation systems in terms of evolution length (number of fitness evaluations) and in terms of the expressiveness of the language it offers for describing individuals in a population (due to a wide range of attribute types individually handled by LEM3).

Our research also revealed a weakness of the current LEM3 implementation in the case of optimizing functions with very large numbers of variables. When the number of variables is very large (usually above 500-1000, depending on the optimized function) the speed up of LEM3 over EA starts to level out or even decrease. While a conclusive explanation of this phenomenon awaits more research, we believe that it is due to the fact that at the end-phase of the LEM3 process in problems with the increasing number of variables, the number of variables that are instantiated semi-randomly, rather than from rules, is also increasing. This means that at that phase, the role of learning is strongly diminishing and LEM3 is increasingly behaving like a Darwinian-type algorithm. While the above problem is a worthy challenge for further research on the LEM methodology, it should be mentioned that most practical problems have fewer variables than 500. Future research will investigate theoretical aspects of the LEM methodology, such as its complexity, convergence speed, and areas of applicability for which it is the most suitable.

Based on many experiments performed, one can draw one general conclusion that the LEM methodology can be particularly advantageous in the application areas in which fitness evaluation is time consuming or costly.

5. ACKNOWLEDGEMENTS

Authors express their gratitude to Ken Kaufman and Jarek Pietrzykowski for their valuable comments on the earlier version of this paper.

This research has been conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's research has been supported in part by the National Science Foundation under Grants No. IIS-0097476 and IIS-9906858, and in part by the UMBC/LUCITE #32 grant.

6. REFERENCES

[1] Bengoextea, E., Miquelez, T., Larranaga, P., and Lozano, J.A., "Experimental Results in Function Optimization with EDAs in Continuous Domain," In Pedro Larranaga and Jose A. Lozano *Estimation of Distribution Algorithms*, Kluwer Academic Publishers, 2002.

[2] Cervone, G., Kaufman, K. and Michalski, R. S., "Recent Results from the Experimental Evaluation of the Learnable Evolution Model," *Proc. of the Genetic and Evolutionary Computation Conference*, GECCO-2002, 2002.

[3] Domanski, P.A., Yashar, D., Kaufman K. and Michalski R.S., "An Optimized Design of Finned-Tube Evaporators Using the Learnable Evolution Model," *International Journal of Heating, Ventilating, Air-Conditioning and Refrigerating Research*, 10, April, 2004, pp 201-211.

[4] Evolutionary Objects Library, downloadable from the website: http://eodev.sourceforge.net

[5] Jourdan, L.; Corne, D.; Savic, D.; and Walters, G., "Preliminary Investigation of the 'Learnable Evolution Model' for Faster/Better Multiobjective Water Systems Design," *Proceedings of The Third Int. Conference on Evolutionary Multi-Criterion Optimization*, EMO'05, 2005.

[6] Michalski, R. S. and Kaufman, K., "The AQ19 System for Machine Learning and Pattern Discovery: A General Description and User's Guide," *Reports of the Machine Learning and Inference Laboratory*, MLI 01-2, George Mason University, Fairfax, VA, 2001.

[7] Larranaga, P., Lozano, J. A., and Bengoetxea, E., "Estimation of Distribution Algorithms Based on Multivariate Normal and Gaussian Networks," *Technical Report KZZA-IK-1-01*, Dept. of Computer Science and Artificial Intelligence, University of Basque Country, 2001.

[8] Larrañaga, P. and Lozano, J. (eds.), *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Kluwer Academic Publishers, 2002.

[9] Michalski, R.S., "Learnable Evolution: Combining Symbolic and Evolutionary Learning," *Proceedings of the Fourth International Workshop on Multistrategy Learning (MSL'98)*, Italy, June 11-13, 1998.

[10] Michalski, R.S., "LEARNABLE EVOLUTION MODEL Evolutionary Processes Guided by Machine Learning," *Machine Learning*, Vol. 38, 2000, pp. 9-40.

[11] Michalski, R.S., "ATTRIBUTIONAL CALCULUS: A Logic and Representation Language for Natural Induction," *Reports of the Machine Learning and Inference Laboratory*, MLI 04-2, George Mason University, Fairfax, VA, April, 2004.

[12] Michalski, R.S. and Cervone, G., "Adaptive Anchoring Discretization for Learnable Evolution Model," *Reports of the Machine Learning and Inference Laboratory*, MLI 01-3, George Mason University, Fairfax, VA, 2001.

[13] Miquelez, T., Bengoetxea, E., and Larranaga, P., "Evolutionary Computation Based on Bayesian Classifiers," *International Journal of Applied Mathematics and Computer Science*, 14, 2004.

[14] Mühlenbein, H and Paaß, G., "From Recombination of Genes to the Estimation of Distributions I. Binary Parameters," *Proceedings of The 4th International Conference on Parallel Problem Solving from Nature*, Berlin, Germany, September 22-26, 1996.

[15] Reynolds, R. G. and Zhu, S., "Knowledge-Based Function Optimization Using Fuzzy Cultural Algorithms with Evolutionary Programming," *IEEE Transactions on Systems, Man, and Cybernetics,* 31, 2001.

[16] Reynolds, R. G., Peng, B., "Cultural Algorithms: Modeling of How Cultures Learn to Solve Problems," *16th IEEE International Conference on Tools for Artificial Intelligence (ICTAI'04)*, Florida, 2004.

[17] Whitley, D., Soraya, R., Dzubera, J., Mathias, K. E., "Evaluating Evolutionary Algorithms," *Artificial Intelligence*, 85, 1996.

[18] Wojtusiak, J., "AQ21 User's Guide, " *Reports of the Machine Learning and Inference Laboratory*, George Mason University, MLI 04-3, Fairfax, VA, 2004.

[19] Wojtusiak, J., "The LEM3 Implementation of Learnable Evolution Model: User's Guide, " *Reports of the Machine Learning and Inference Laboratory*, George Mason University, MLI 04-5, Fairfax, VA, 2004.

[20] Wojtusiak, J., and Michalski R.S., "The LEM3 System for Non-Darwinian Evolutionary Computation and its Application to Complex Function Optimization," *Reports of the Machine Learning and Inference Laboratory*, George Mason University, MLI 05-2, Fairfax, VA, 2005.