Machine Learning, 38, 9–40, 2000.

LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning

RYSZARD S. MICHALSKI michalski@gmu.edu
Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA and Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Editors: Floriana Esposito & Lorenza Saitta

Abstract. A new class of evolutionary computation processes is presented, called *Learnable Evolution Model* or *LEM*. In contrast to Darwinian-type evolution that relies on mutation, recombination, and selection operators, LEM employs machine learning to generate new populations. Specifically, in *Machine Learning mode*, a learning system seeks reasons why certain individuals in a population (or a collection of past populations) are superior to others in performing a designated class of tasks. These reasons, expressed as inductive hypotheses, are used to generate new populations. A remarkable property of LEM is that it is capable of quantum leaps ("insight jumps") of the fitness function, unlike Darwinian-type evolution that typically proceeds through numerous slight improvements. In our early experimental studies, LEM significantly outperformed evolutionary computation methods used in the experiments, sometimes achieving speed-ups of two or more orders of magnitude in terms of the number of evolutionary steps. LEM has a potential for a wide range of applications, in particular, in such domains as complex optimization or search problems, engineering design, drug design, evolvable hardware, software engineering, economics, data mining, and automatic programming.

Keywords: multistrategy learning, genetic algorithms, evolution model, evolutionary computation

1. Introduction

Recent years have witnessed significant progress in the development and applications of machine learning methods, in particular, in scaling them up to cope with large datasets (e.g., Clark & Niblett, 1989; Cohen, 1995; Dietterich, 1997; Mitchell, 1997; Michalski, Bratko, & Kubat, 1998). There has also been significant progress in the area of evolutionary computation (e.g., Koza, 1994; Michalewicz, 1996; Baeck, Fogel, & Michalewicz, 1997; Banzhaf et al., 1998). As machine learning and evolutionary computation have complementary strengths and limitations, a question arises if their integration could not lead to a new powerful multistrategy learning and problem solving methodology. This question has motivated research whose first results are presented in this paper.

All conventional methods of evolutionary computation draw inspiration from the principles of Darwinian evolution, which is governed by "...one general law, leading to the advancement of all organic beings, namely, multiply, vary, let the strongest live and the weakest die" (Darwin, 1859). In generating new populations, the Darwinian-type evolutionary computation uses such operators as mutation (an asexual reproduction with variation), crossover (a recombination or sexual reproduction), and selection (survival of the

fittest). These operators are simple to execute and domain-independent; thus, they can be employed without any knowledge of the problem area (e.g., Holland, 1975; Goldberg, 1989; Michalewicz, 1996; Mitchell, 1996). Consequently, evolutionary computation based on the principles of Darwinian evolution has been applied to a very wide range of problems, including many kinds of optimization and search problems, automatic programming, engineering design, game playing, machine learning, evolvable hardware, and others.

The Darwinian-type evolutionary computation is, however, semi-blind: the mutation is a random modification of the current solution, the crossover is a semi-random recombination of two solutions, and the selection is a form of parallel hill climbing. In this type of evolution, the generation of new individuals is not guided by lessons learned from the past generations, but is a form of a trial and error process executed in parallel. Consequently, computational processes based on Darwinian evolution tend to be inefficient. Low efficiency has been the major obstacle in the application of the Darwinian-type evolutionary computation to highly complex problems.

The novel idea proposed in this research is to employ machine learning to guide the generation of new populations in an evolutionary process. The proposed approach, called *Learnable Evolution Model* (briefly, LEM) executes a *Machine Learning* mode that seeks reasons why certain individuals in a population are superior to others in performing a designated class of tasks. These reasons, expressed as inductive hypotheses, are then used to create a new generation of individuals. A remarkable aspect of LEM is that it can exhibit evolutionary "insight jumps," in which the fitness value experiences a quantum leap, apparently due to the discovery of the right direction of evolution. LEM can thus be termed as a form of "intelligent evolution." The next section describes LEM in more detail.

2. The learnable evolution model

2.1. Top-level LEM algorithm

The Learnable Evolution Model or LEM is fundamentally different from the Darwinian-type model underlying current methods of evolutionary computation. The central engine of evolution in LEM is Machine Learning mode which creates new populations by employing hypotheses about high fitness individuals found in past populations. Specifically, Machine Learning mode consists of two processes: *hypothesis generation*, which determines hypotheses characterizing differences between high-fitness and low-fitness individuals in one or more past populations, and *hypothesis instantiation*, which generates new individuals on the basis of the learned hypothesis. Machine Learning mode thus produces new individuals not through semi-random Darwinian-type operations, but through a deliberate reasoning process involving generation and instantiation of hypotheses about populations of individuals.

LEM may alternate between Machine Learning mode and *Darwinian Evolution* mode, or may rely entirely on Machine Learning mode. Darwinian Evolution mode implements one of the conventional methods of evolutionary computation that employ some form of mutation, recombination (optionally), and selection to generate new populations. If LEM executes Machine Learning mode only, it repetitively applies machine learning to evolve

new populations. This variant of LEM's operation was proposed in conversations with the author's student, Guido Cervone. When working in both modes, LEM switches from one mode to another, whenever a *mode termination condition* is met. The evolution process continues until a *LEM termination condition* is met (a solution is accepted as satisfactory or the allocated resources are exhausted).

LEM differs not only from Darwinian-type evolution but also from Lamarckian-type[1] of evolution. This is so because the process of generating new individuals takes into consideration not only the experience of single individuals, but the experience of an entire population, or a collection of populations. Although LEM is presented here as a form of evolutionary computation, it can be viewed as a new general form of evolutionary processes, whose principles are potentially applicable to guiding any form of evolution, including biological or social.

An evolutionary process in LEM starts with some initial population of individuals, which, in analogy to nature, may represent "phenotypes," or "genotypes" that are used to produce "phenotypes." In the case of evolutionary computation, individuals may thus represent problem solutions or prescriptions for generating such solutions. The solutions may be values of variables that optimize a function, designs, concept descriptions, plans, strategies, or any entities that are actors in an evolutionary process. Here is a general form of the LEM process:

1. *Generate a population*: Randomly, or according to a method, create a starting population.
2. *Execute Machine Learning mode*:
   a) *Derive extrema*: Select from the current population of individuals (solutions) two groups: *High performance group*, or briefly *H-group*, and *Low performance group*, or *briefly L-group*, based on the values of the fitness function. The union of H-group and L-group can be a subset of the population or span the whole population.
   b) *Create a hypothesis*: Apply a machine learning method to create a description of the H-group, that differentiates it from the L-group, and, optionally, a description of the L-group that differentiates it from the H-group. When learning a description of the H-group (optionally, also the L-group), take into consideration the evolutionary history, that is, either past populations or descriptions of past populations (see Section 2.2).
   c) *Generate a new population*: Generate new individuals by instantiating the learned description of the H-group and combine these individuals with those in the H-group. Select from the combined set a new population. The description instantiation is done either randomly or according to some *description instantiation rules*. A simple form of step (c) is to replace non- H-group individuals by randomly generated instantiations of the H-group description.
   d) Go to step 2a) and continue repeating *Machine Learning mode until the Machine Learning mode termination condition* is met (e.g., the best fitness value in a sequence of *learn-probe* populations does not exceed by more than the *learn-threshold* the best fitness value found previously). When the Machine Learning mode termination condition is met, take one of the following actions:
      (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. *Execute Darwinian Evolution mode*
   Apply a Darwinian-type evolutionary method, that is, apply some form of mutation, crossover (optionally) and selection operators to generate a new population. Continue this mode until the *Darwinian Evolution mode termination condition* is met (e.g., the best fitness in a sequence of *dar-probe* populations does not exceed by more than *dar-threshold* the best fitness found previously).
4. *Alternate*: Go to step 2, and then continue alternating between step 2 and step 3 until the LEM termination condition is met. The best individual obtained is the result of evolution.

The fundamental assumption underlying LEM (as well as other forms of evolution) is that there is a method for evaluating performance of individuals in evolving populations. Such a method, called a fitness function, assigns a quantitative or qualitative value to each individual. This value characterizes the individual's performance from the viewpoint of the assumed class of goals or tasks. The ability to determine the fitness value of an individual, or an approximation of this value, is a precondition for the LEM application. The output domain of a fitness function can be continuous (e.g., can represent degrees to which individuals perform a designated task), or discrete (representing distinct classes of performance).

The parameters *learn-probe* and *dar-probe* control persistence in continuing a given mode; specifically, they define the number of generations in Machine Learning and Darwinian Evolution modes, respectively, which are performed despite an unsatisfactory progress of the evolution process in the given mode. What constitutes unsatisfactory evolutionary progress is defined by the *learn-threshold* and *dar-threshold* parameters. These parameters specify threshold ratios of the best fitness function value in the sequence of learn-probe and dar-probe populations to the best fitness in the previous sequence of populations in Machine Learning and Darwinian Evolution mode, respectively.

The *learn-threshold* and *dar-threshold* parameters are typically equal or greater than one, but they can also be smaller than one (in which case, a given mode would continue even if the best value in the sequence of *learn-probe* or *dar-probe* populations was smaller than the best-so-far value). A mode termination condition can thus be tightened or relaxed by modifying values of the above parameters.

The Machine Learning mode termination condition is met when a plateau of performance is reached. If at this point the LEM termination condition is not yet met, LEM executes the start-over operation [step 2d(ii)] or switches to Darwinian Evolution mode. If LEM always chooses the start-over operation, the evolution process is based solely on a repetitious application of Machine Learning mode. Such a version of LEM operation is called uniLEM, as it does not involve a separate Darwinian Evolution mode. For the purpose of distinction, the LEM version that applies two separate modes is called duoLEM.

When LEM chooses the start-over operation, it moves back to step 1. A simple way to execute step 1 is to generate a new starting population randomly. This is typically done when executing step 1 for the first time. When executing it as a start-over operation, such a random population generation method or some alternative method can be used. Here are some alternative methods:

A. *Select-elite.* Randomly generate individuals but include them in the new population only if their fitness value is above a certain threshold (e.g., above the mean fitness of the

individuals in the last generated population, or above the fitness of a randomly selected individual in the past population).

B. *Avoid-past-failures.* This method employs a description of the L-group(s) generated in Machine Learning mode (specified as an option in step 2(b)). A new population is created by randomly generating new individuals, but they are included in the new population only if they do not satisfy the description of the L-group. This method avoids computing the fitness value of randomly generated individuals, but requires matching them against the L-group description.

C. *Use-recommendations.* This method requires storing a set of past H-group descriptions whose instantiations led to a jump in the maximal-so-far fitness in the evolution process. Such descriptions are called *consequential* and a rapid increase of the top fitness value is called an *insight jump*. A new starting population is generated by instantiating one or more consequential descriptions.

D. *Generate-a-variant.* This method generates a new start-over population by making modifications to the individuals of the last population. Such modifications may be random mutations, or may be results of specialized description modification operators tailored for a given application domain.

Machine Learning mode can potentially employ any learning method that can generate descriptions discriminating between classes of individuals in a population. If individuals are described by attribute-value vectors, one can use, for example, a rule learning method, a decision tree learner, or a neural net. If individuals are described by structural descriptions, a structural (relational) learning method would be needed, for example, an inductive logic programming system (e.g., Lavrac & Dzeroski, 1994), or INDUCE relational learning system that learns descriptions in *annotated predicate calculus* (Michalski, 1983). In the experiments presented here, Machine Learning mode employed an AQ learner (by which we mean a learning system that employs some from of the AQ algorithm). As described in Section 3, AQ learners are particularly suitable for implementing LEM.

In the duoLEM version, a separate Darwinian Evolution mode is executed at selected steps of evolution. In this mode, any conventional evolutionary algorithm can be potentially applied, for example, a genetic algorithm, an evolutionary strategy, or an evolutionary program. It should also be noted that although the above general LEM algorithm placed Machine Learning mode (step 2) before Darwinian Evolution mode (step 3), this order is not necessary. LEM can start with Darwinian Evolution mode, and then alternate between Darwinian Evolution and Machine Learning modes until the LEM termination condition is met.

2.2. Extrema generation

In order to execute Machine Learning mode, one needs to select an H-group and an L-group from the population. These groups represent subsets of the population that are used as positive and negative examples for a machine learning method. Typically, they will be subsets

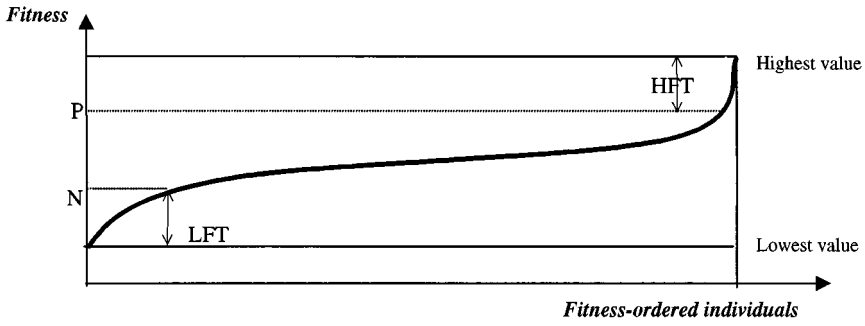

Figure 1. An example of the fitness profile function of a population.

representing extreme or near extreme values of the fitness value in the population. The selection of such groups can be done in different ways. One way is to use the *fitness-based* method in which the population is partitioned according to two fitness thresholds. These thresholds, called HFT ("High Fitness Threshold") and LFT ("Low Fitness Threshold"), are expressed as percentages, and specify portions of the total fitness value range in the population that are used to determine the H-group and the L-group. These parameters can be illustrated by a *fitness profile function* which maps individuals of the population arranged in an ascending order into their fitness value (figure 1).

Individuals whose fitness value is not lower than the HFT% from the highest fitness value (i.e., is above point P in figure 1) are put into the H-group, and those whose fitness value is not higher than LFT% from the lowest fitness value (i.e., is below point N) are put into the L-group. For example, if HFT and LFT are both 25%, then individuals with a fitness value no lower than 25% below the highest fitness, and no higher than 25% above the lowest fitness value are included in the H-group and L-group, respectively. This method may work well if the fitness profile function behaves similarly on both sides of the middle. If the fitness profile function is flat on one side and steep on the other side, then this method may produce highly uneven sizes of the H-group and L-groups. For example, if it is flat at the beginning (figure 2), then there may be many more negative examples than positive, and if it is flat at the end, there may be many more positive examples than negative. This situation may be not favorable for producing high quality descriptions. Also, when using a fitness-based method, the size of the H-group and L-group will vary from population to population, as it depends on the shape of the fitness profile function.

Another way to select an H-group and an L-group is the *population-based* method, in which the population is partitioned according to a priori defined portions of the population. This is done by using two parameters, HPT (the "high population threshold") and LPT (the "low population threshold") expressed as percentages. The HPT and LPT parameters indicate portions of the population to be used as the H-group (to the right of P), and the L-group (to the left of N) group, respectively (figure 2). When the population-based method is used, it is possible that some individuals will belong to both the H-group and the L-group. In such a case, a method for resolving inconsistency needs to be used. AQ learners, which we used in the experimental studies of LEM, provide several methods for resolving example

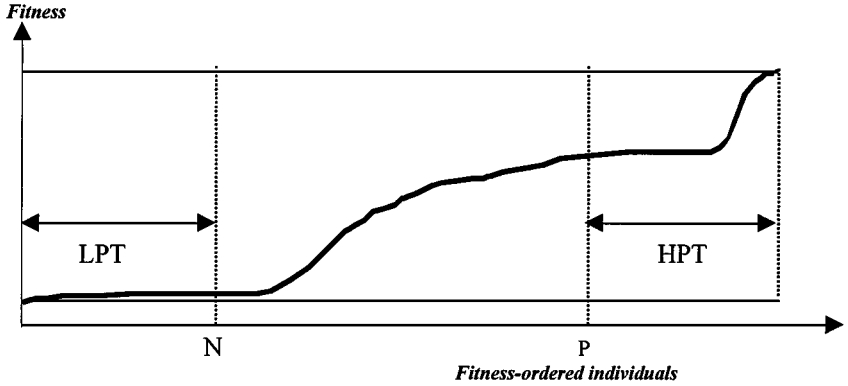

Figure 2. A fitness profile function that is flat at the beginning.

inconsistency (see Section 3.3). The fitness-based and population-based methods can also be used in combination.

If any of above methods is applied to the entire population, then this a *global approach* to extrema formation. In contrast, a *local approach* applies one of the above methods in parallel to different subsets of the population. For example, an AQ learner (Section 3.3) produces a set of rules (a ruleset), each characterizing a subset of the population. A local approach is to determine extrema in each subset (subgroups), and apply machine learning to each pair ⟨H-subgroup, L-subgroup⟩ separately. Such an approach may speed-up the evolution process.

The above methods for extrema determination can be enhanced by employing elitism, i.e., by always including in the H-group the *best-so-far individual* (the individual with the highest fitness at this point of evolution), and possibly also including the *worst-so-far* individual in the L-group. This simple technique, analogous to elitism used in conventional methods of evolutionary computation, facilitates generation of descriptions that preserve the memory of the highest and the lowest peak of the fitness landscape (i.e., the search space with the fitness value assigned to each individual) that has been encountered during the evolution process.

When the range of fitness values in a population is zero or near zero, this is a signal that the population consists of individuals whose fitness reached an extreme value (maximum or minimum) or resides at a flat region between peaks of the fitness landscape. If at this point the LEM termination condition is still not satisfied, the evolution process can either switch to Darwinian Evolution mode or repeat Machine Learning mode with a new starting population (the start-over operation described previously).

In the above methods, the H-group and L-group were selected only from the current population in the evolution process, that is, the past history of evolution was ignored. Such an approach may not work well because H-group description that does not take into consideration past L-groups is likely to be too general. Similarly, a description of the L-group that does not take into consideration past L-groups is likely to be too specific.

To avoid the above problem, below are examples of methods that could be used for taking into consideration the evolutionary history:

a. *Population-lookback.* This method keeps a record of the L-groups determined in past populations. A set-theoretic union of the past L-groups plus the L-group in the current population is the actual L-group supplied to the machine learning system. The number of past L-groups to be taken into consideration is specified by the *p-lookback* parameter. There is no strong need to store past H-groups because the current H-group presumably contains the best individuals determined so far. A possible advantage of storing past H-groups is to use them for resolving ties when choosing a description among alternatives.

b. *High-group description-lookback.* This method maintains a collection of descriptions of the past H-groups. The number of H-group descriptions to be taken into consideration is specified by the *d-lookback* parameter. The current H-group description is used to generate new candidate individuals and the past H-group descriptions serve as preconditions for accepting a candidate (see Section 2.5).

c. *Low-group description-lookback.* This method maintains a collection of descriptions of L-groups and uses them as constraints in the process of generating H-group descriptions.

d. *Incremental specialization.* This method applies an incremental learning algorithm to maintain one updated description of the H-group. The input to such an algorithm is a description of the previous H-group, and the new H-group and L-group consisting of individuals derived from that description. The output is a new, updated description of the H-group. Since the new H-group and L-group are instantiations of the description of the previous H-group, the description update is a specialization operation.

If this method is repeated at every step of evolution, the last updated description reflects the whole history of population evolution. To limit the influence of past populations, one may apply incremental specialization only for a fixed number of steps, defined by the *incremental lookback or i-lookback* parameter. After this number of steps has been performed, the current description is discarded and the process starts anew with a freshly generated H-group and L-group.

When the option of generating L-group descriptions is exercised, an incremental learning process can be simultaneously conducted to maintain an updated description of the L-groups. In this case, the description update is a generalization operation. Incremental specialization is an elegant method for taking into consideratin the evolutionary history. In order to apply this method, however, a machine learning system must be able to incrementally specialize hypotheses according to new inputs. Not all machine learning methods can work in such a fashion.

Example enfolding

A simple and efficient way to approximate the incremental specialization method is to construct highly specific descriptions the H-groups, that is, descriptions that closely enfold the H-group. This method is particularly easy to apply with an AQ learner because such learners have a parameter that controls the generality of generated descriptions. One can obtain the most specific, the most general, or an intermediate description just by properly setting

a program parameter. Highly specific descriptions are called *characteristic* (Michalski, 1983) because they specify characteristic features common to all individuals in a group. By learning characteristic descriptions of the H-group one can decrease the danger of overgeneralization that may occur due to forgetting past examples. Because this danger increases with the continuation of the evolution process, a simple strategy is to learn discriminant descriptions at the beginning of the evolution process and then switch to learning characteristic descriptions.

2.3. Fitness function change

In some evolutionary processes, the fitness function may not be constant through the entire process, but may be changing over time. A change of the fitness function may be gradual (a "fitness function drift") or abrupt (a "fitness function shift"). In nature, environmental conditions may undergo both types of changes.

If the fitness function is changing during the evolution process, some high- fitness individuals in a previous population may become low-fitness individuals in a future population, and the other way around. One way to address this problem is through the *p-lookback*, *d-lookback* or *i-lookback* parameter, depending on the method used for considering the evolutionary history. A simple alternative is the example enfolding method, which helps to cope with a fitness function change by avoiding making large generalization steps.

Research on concept shift or drift done in the area of machine learning provides a source of ideas for addressing the fitness function drift problem. The paper by Maloof & Michalski (1999) addresses some aspects of this problem and provides references to other work.

2.4. Extrema descriptions as qualitative differentials

In Machine Learning mode, given an H-group and an L-group, a machine learning method generates a description that discriminates between these groups (Michalski, 1983). In LEM, one can use just a description of the H-group, or, optionally, a pair ⟨H-group description, L-group description⟩. An H-group description represents a hypothesis that the area in the landscape referred to by this description contains individuals with a higher fitness than that of the individuals outside of this area. Similarly, the pair ⟨H-group description, L-group description⟩ represents a pair of hypotheses that indicate likely locations of individuals with higher and lower fitnesses, respectively.

A H-group description can thus be interpreted as a *qualitative differential* that roughly approximates the direction of change of the fitness landscape. Selecting individuals that satisfy a H-group description thus 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. The qualitative zero is indicated by a flat fitness profile function and the consequent impossibility of determining an H-group and an L-group.

The power of the LEM idea seems to stem from computing such qualitative differentials and using them to guide the evolution process, rather than relying on a semi-blind Darwinian-type evolutionary operators. Since qualitative differentials can be repetitively computed in

parallel, that is, determined simultaneously in many places of the fitness landscape, LEM has a better chance to efficiently find the global optimum than methods that rely only on mutation and/or crossover. Moreover, if the fitness landscape has several global optima, LEM may be able to find all or a subset of them simultaneously (assuming that Machine Learning mode constructs disjunctive descriptions of the successive H- groups). A powerful machine learning method can generate such descriptions in spaces spanned over large numbers of variables (hundreds or thousands) and for highly irregular, non-differentiable landscapes. Thus, LEM methodology can be potentially applied to highly complex problems.

A particularly attractive characteristic of LEM-based evolutionary processes is that they frequently exhibit quantum leaps of the fitness function (Section 4), unlike Darwinian-type processes that are typically characterized by gradual changes of the fitness function. These quantum leaps are called "insight jumps" as they can be interpreted as indicators that a correct direction of search has been found.

2.5. LEM as progressive partitioning of the fitness landscape

The search conducted in Machine Learning mode can be interpreted as progressive partitioning of the search space. An H-group description hypothesizes a region where finding a solution is likely, and an L-group description hypothesizes a region where finding a solution is less likely. Each subsequent H-group description (or a pair of ⟨H-group description, L-group description⟩ hypothesizes a new partition of the search space.

This process can be easily illustrated by using an AQ learner (Section 3) that applies incremental specialization method In this case, an H-group description is a collection of hyper-rectangles in a multidimensional search space (or rectangles in a 2D space). In figure 3, two largest rectangles represent an H-group description of the first generation (each rectangle corresponds to one rule in the description).

An H-group description created in the next generation is a specialization of this description, thus, it is a subset of the previous region (distributed over different subareas). Rules of the same generation are represented in figure 3 by similarly shaded rectangles. The incremental specialization method thus creates a cascade of progressively more specialized

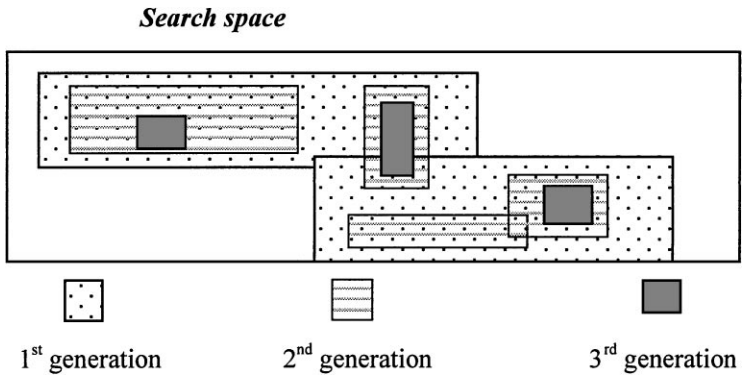

Figure 3. Exemplary search regions defined by de scriptions of 1st, 2nd and 3rd generation.

(nested) regions in the fitness landscape. A fully successful termination of such a process produces a collection of rules such that each rule describes an area where the fitness landscape achieves one of the global peaks (in the case of multi-modal problems).

It should be noted that descriptions of H-groups are generated through induction from examples. Therefore, they are only hypotheses about areas with HIGH performance solutions. It is possible that in a given pass of the algorithm, the area with the highest fitness individual (a global solution) may be missed. In such a case, a start-over operation creates a different starting population, and a new series of inductive inference steps is executed. Alternatively, Darwinian Evolution mode can be used.

2.6. Description instantiation

Let us now turn to the problem of generating new individuals that satisfy an H-group description, that is, to the *description instantiation* problem. In the case of incremental specialization, example enfolding, or population lookback method, one needs to consider only a single description of H-group (or, optionally, a single pair ⟨H-group description, L-group description⟩).

In the case of description lookback method with *d-lookback* greater then 0, one needs to consider a collection of descriptions, each of which distinguishes between HIGH and LOW individuals in one of the past populations. New individuals should satisfy all H-group descriptions. One technique to achieve this effect is to use the most recent description to generate candidates for new population and to apply other descriptions as enabling conditions in a cascade fashion. Thus, regardless of the method for handling evolutionary history, one can limit consideration to only one description (or one pair of descriptions).

A description instantiation operation depends on the form of the description language employed by the machine learning method. To illustrate this step, let us assume, for example, that the description of H-group individuals is in the form of *attributional rulesets*. A ruleset is equivalent to a DNF description (disjunctive normal form) in *attributional calculus* (Section 3).

Given a ruleset, an instantiation operation is done by generating individuals in such a way that each of them satisfies at least one rule in the ruleset. It is desirable that at least two different individuals are generated from each rule. The reason for this is that having at least two examples facilitates a computation of the qualitative differential discussed in Section 2.3.

A description instantiation is done by assigning different combinations of values to variables in the rules of a ruleset. Each assignment must satisfy all conditions in at least one of the rules. For example, if one of the conditions is [color is blue or red or yellow], then new solutions can take any of the colors from the set {blue, red, yellow}. These values can be chosen randomly or according to some probability distribution function (which, for example, may represent the distribution of colors in the H-group). If one of the conditions is [length $\geq$5 cm] then solutions may have different lengths but always greater than or equal to 5 cm. The choice of specific values can be random or done according to a probability distribution function over the range ⟨5 cm. . . *Maximum Value*⟩. Such a probability distribution function may reflect the "maximizing-the-distance-to-the-negative" heuristic that favors values that

are further away from the values in the description of the H-group. For example, if an H-group description has a condition [length = 5..15 cm] then the probability of selecting a particular length may peak in the center of the range (as values outside the range can be viewed as belonging to an L-group).

A question arises as to how to assign values to "free" attributes, that is, to attributes that are not present in a description. This can be done in a variety of ways, for example, by randomly choosing values from the attribute domain, by choosing only values that are present in the examples of the H-group, or by choosing them according to some probability distribution over the attribute domain. Which method should be used in various situations is an open problem for further research. In our experiments, free attributes were assigned values present in the examples of the H-group or chosen randomly from the attribute domain.

In analogy to proportional selection, which is frequently used in Darwinian-type methods, LEM may use a *proportional instantiation*. To explain this method, let us assume that an H description is in the form of a set of rules (a ruleset), and each rule is assigned two parameters:

- the *scope*, defined as the number of all possible individuals that satisfy the rule,
- the *fitness score*, defined as the average of the fitness values of the H-group individuals described by this rule.

The proportional instantiation method generates from each rule a number of instantiations that is proportional to a monotonic function of its scope and the fitness score.

2.7. New population generation

As outlined in step 2c of the LEM method, a new population is created by combining individuals from the past population with those generated anew in Machine Learning or Darwinian Evolution modes. In Darwinian Evolution mode, new individuals are generated by applying mutation and/or (optionally) crossover operators to the current population. In Machine Learning mode, new individuals are generated by description instantiation.

To generate a new population, a number of methods can be used. A *direct replacement* method replaces all non-H-group individuals in the current population by those generated through an H-group description instantiation. A more general approach is to allow the union of H-group individuals ("inferential parents") and new individuals generated through instantiation to initially exceed the assumed population size. Then, from the combined set a desirable subset is selected that constitutes the new population. The field of evolutionary computation has developed a number of such selection methods.

One method is *fitness proportional selection* (also called a roulette wheel selection or stochastic sampling with replacement). In this method, the probability of selecting an individual for the new population is proportional to its fitness. Another method, *rank-based selection*, orders individuals according to their fitness and selects the highest ranked. There is also a stochastic *tournament method* in which a pair of individuals is selected in a fitness proportional way and the individuals compete with each other. The one with the higher fitness is selected. Then another pair is drawn, and this process continues until enough

individuals are selected. For a comprehensive description of various selection methods developed in the field of evolutionary computation consult, for example, Goldberg (1989), Michalewicz (1996).

2.8. Adaptive anchoring quantization

Symbolic learning methods such as AQ that are used in LEM1 and LEM2 systems (Section 3) are oriented toward problems with discrete variables. To apply these methods with continuous variables, such variables need to be discretized. If the problem solution requires a representation of variables with a certain precision, then the quantization must permit such a precision. The precision needed for representing a solution is, however, usually unknown.

A brute force solution to this problem would be to quantize all the variables with the maximal precision that one guesses may be needed. Such a solution faces two major problems. One is that the guess may be incorrect and the chosen precision will be insufficient, that is, the system will not be able to find the correct solution regardless of how long the evolution process continues. The second problem is that such a method may result in unnecessary large attribute domains, which can significantly impede the performance of the learning system.

An alternative solution to this discretization problem is what we call *adaptive anchoring discretization*. The term *adaptive* means that the quantization precision is dynamically adapted to the problem at hand in the process of evolution. The term *anchoring* is used to signify that the domains of attributes consist of consecutively more precise discrete values that are rounded to the nearest whole numbers (anchors). This feature is reflective of human preference for representing numbers simply, that is, with the minimum number of digits that is sufficient for a given problem. The method proceeds by making discretizations that are consecutively more precise, but only within the ranges that are hypothesized to need a higher precision. Adaptive quantization can achieve an arbitrary level of precision, but avoids an excessive precision, and by that decreases the computational complexity of the learning process (Michalski, 1999).

2.9. A simple illustration of a AQ-based LEM search

This section illustrates a simple form of LEM using an example of a search problem in a discrete space. The space is presented in a planar form using *General Logic Diagram or GLD* (Michalski, 1978; Zhang, 1997). In the diagrams in figure 4, each cell represents a vector (an individual) in a search space spanned over four variables: x, y, w, and z, with domain sizes 2, 2, 2, and 3, respectively. The initial population is visualized by a collection of cells marked by dark dots (figure 4A). The numbers next to the dots indicate the fitness value of the individual. The search goal—the global solution of the search problem—is represented by the cell with an x (the fitness value of the solution is 9).

In this example, we assume that descriptions discriminating between an H-group and an L-group are in the form of attributional rules (see Section 3.2). Such rules are learned by the AQ-type learning programs employed in experiments with LEM1 and LEM2 described in the next section.

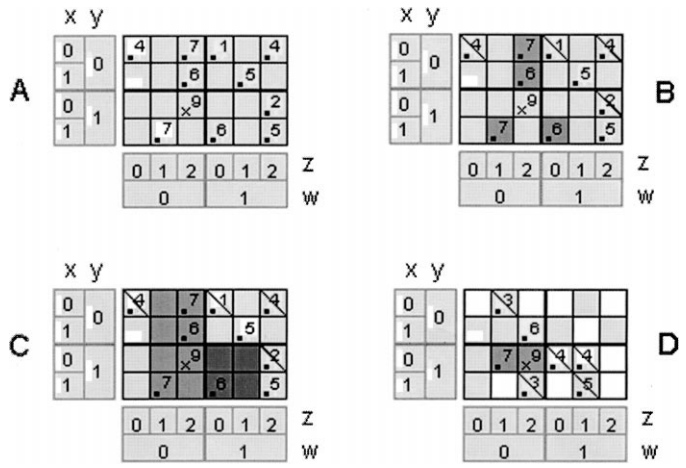

Figure 4. An illustration of a simple LEM search process.

Figure 4(B) presents H-group individuals (the gray-shaded cells) and L-group individuals (crossed cells) determined from the initial population. The shaded areas in Figure 4C depict two attributional rules that discriminate between the H-group and the L-group (the qualitative differential). These rules are: [w = 0] & [z = 1 v 2] and [x = 1] & [w = 1] & [z = 0 v 1].

Figure 4(D) shows individuals in the H-group (shaded cells) and the L-group (crossed cells) which were generated through an instantiation of rules in figure 4C. The shaded area in figure 4D shows a single rule that discriminates between these groups: [x = 1] & [y = 0] & [w = 0] & [z = 1v2]. This rule was obtained through incremental specialization of the parent rule. The new rule covers the goal cell. It is easy to see that the global solution will be found in the next rule learning step.

3. An initial AQ-based implementation of LEM

3.1. An overview

To test some of the proposed ideas, we implemented a simple, preliminary system, LEM1. In Machine Learning mode, LEM1 employs an AQ-type learner (specifically, AQ15; Wnek et al., 1995, Kaufman & Michalski, 1999). In Darwinian Evolution mode, LEM1 uses a simple genetic algorithm, GA2 (Section 3.4). At present, we are in the process of completing and experimenting with a new system, LEM2. In Machine Learning mode, LEM2 uses AQ18 learner (an enhancement of AQ15c; Kaufman & Michalski, 1999), and in Darwinian Evolution mode uses an evolutionary algorithm EV. The following subsections describe briefly the AQ learning process and evolutionary algorithms used in the experiments. Section 4 presents a sample of results from applying LEM1 and the current version of LEM2 to selected problems.

Before we briefly describe AQ learning, it is appropriate that we justify why we chose it for implementing the LEM methodology. AQ learning has several important characteristics that make it particularly attractive for implementing LEM. First of all, it generates descriptions in the form of *attributional rules* (next section), which are more general than those used in typical rule learners. Unlike conventional decision rules that are conjunctions of attribute-value conditions, or conditional-value, attributional rules involve conditions that may include also internal disjunctions of attribute values, ranges of values, internal conjunctions of attributes, and other constructs. Such conditions make the representational language not only more expressive, but also facilitate the instantiation of rules into collections of examples (individuals) that vary slightly. This feature is attractive for implementing evolutionary processes.

Another characteristic of AQ learning is that given a set of examples, it can generate different types of attributional rulesests characterizing these. Depending on the program parameters, these rulesets may vary in the degree of generality and in the internal organization. For example, they can be highly general (which may be desirable at the early stages of the evolution), highly specific (which may be desirable at later stages), or of intermediate generality. The ability to control the degree of generality is particularly useful for applying the example enfolding method described in Section 2.2 (which we used in experiments with LEM2). The internal organization of a ruleset can also vary: rules in a ruleset may be logically intersecting, logically disjoint or ordered (a decision list). A ruleset can also optimize many different criteria of description quality. The flexibility in the type of descriptions generated by AQ learning facilitates testing and experimentation with LEM.

As mentioned earlier, individual attributional rules correspond to subspaces in the fitness landscape (recall figure 3). By repetitively generating rulesets characterizing successive H-groups, LEM can conduct a parallel search through many subspaces, which increases the probability of finding the global solution.

Another attractive feature of AQ learning is that it performs a form of *natural induction*, by which we mean an inductive inference process that strives to produce hypotheses that appear natural to people, that is, are easy to understand and interpret. This feature helps to perform analyses and to develop insights into the LEM-based evolutionary processes.

To explain AQ learning and other aspects of implemented LEM systems, the next section briefly describes attributional calculus (Michalski, 1999).

3.2. Attributional calculus

Attributional calculus is a simple to implement but highly expressive description language. It has well-defined syntax and semantics, and its representational power is between propositional calculus and first order predicate logic. From the viewpoint of LEM, its most important construct is an attributional rule, which is in the form:

$$\textbf{\textit{CONDITION}} => \textbf{\textit{DECISION}}$$

where ***CONDITION*** (also called a *complex*) is a conjunction of *attributional conditions* (also called *selectors*), and ***DECISION*** is an elementary *attributional condition*. An attributional condition is in the form:

$$[\textbf{\textit{L rel R}}]$$

where ***L*** (*left side*) is an attribute, or one or more attributes with the same domain, joined by "&" or "v", called *internal conjunction* and *disjunction*, respectively, ***R*** (*right side*) is a value, a list of values from the domain of attribute(s) in ***L***, joined by the symbol "v" (called internal disjunction), a pair of values joined by ".." (called *range*), or an attribute with the same domain as attribute(s) in ***L***, and ***rel*** is a relational symbol from the set $\{=, \neq, \geq, >, <, \leq\}$.

Brackets [ ] are optional, and can be omitted whenever it causes no confusion. An attributional condition [***L rel R***] is *true* (or *satisfied*) if ***L*** is in relation ***rel*** to ***R***. A condition is called *elementary* if ***L*** is a single attribute, ***rel*** is not $\neq$, and ***R*** is a *single value*; otherwise, it is called *composite*. Here are examples of attributional conditions and their interpretation:

Elementary:

[size = medium]	(*the size of the object under consideration is medium*)
[weight > 50]	(*the weight is greater than 50*)

Composite:

[color = red v white v blue]	(*the color is red or white or blue*)
[width & length= 5..8]	(*the width and length are both between 5..8*)
[temperature = 2..10 v 25..50]	(*the temperature is between 2 and 10 or 25 and 50*)
[length & width > depth]	(*the length and width are each greater than the depth*)

Note that attributional conditions are easy to interpret and easy to translate into logically equivalent natural language expressions. Attributional calculus is based on the variable-valued logic system VL1, which has been specifically developed for machine learning (Michalski, 1973). For more advanced features of attributional calculus see (Michalski, to appear).

3.3. AQ learning

Given a set of positive and negative training examples of a decision class, an AQ learner generates a set of attributional rules (a ruleset) characterizing this class. Training examples are in the form of *attributional events*, which are vectors of attribute values. Events in the decision class (or class, for short) for which a ruleset is generated are considered positive examples, and events in all other classes are considered negative examples.

When a ruleset for another class is generated, positive and negative example labels are changed accordingly. Attributional rules are generated from examples according to algorithm AQ, which pioneered the progressive covering (also called separate-and-conquer), approach to rule learning (e.g., Michalski, 1969; 1973; to appear).

For completeness, a brief description of a simple form of the AQ algorithm follows:

1. *Seed selection:* Select randomly a positive example and call it a *seed*.
2. *Star generation:* Generate a *star* of the seed, defined as a set of general attributional rules (complexes) that cover the seed and any other positive examples, but do not cover

negative examples. In the general case, a rule can cover negative example if it optimizes a *description quality criterion* (see, e.g., Kaufman & Michalski, to appear).

3. *Rule selection:* Select the highest quality rule from the star according to a given description quality criterion. Such a criterion can be tailored to the requirements of the problem domain. For example, a quality criterion may require selecting the rule that covers the largest number of positive examples, covers no negative examples, and has the lowest computational cost among other equivalent rules in the star.
4. *Coverage update:* Remove examples covered by the rule from the set of positive examples and select a new seed from the remaining positive examples. If there are no positive examples left, return the generated set of rules. Otherwise, go to Step 1.

The most important step of the algorithm is star generation (Step 2), which involves a repetitive application of the *extend-against* generalization operator (Michalski, 1983), and logically multiplying out the resulting collections of partial rules (Michalski, to appear). If properly implemented, such a process can be executed highly efficiently. For example, recent implementations of AQ-type of learning have been effectively applied to problems with hundreds of attributes and tens of thousands of examples.

AQ learning programs, such as AQ15c (used in LEM1), and AQ18 (used in LEM2), have several special features. AQ15c includes the ability to learn a range of different types of attributional rulesets (e.g., intersecting, disjoint, ordered, characteristic, and discriminant), to adapt inductive reasoning to different types of attributes (nominal, rank, cyclic, numerical, and structured), to learn from noisy and/or inconsistent data, to learn incrementally, and to match rules with examples using a strict or flexible matching method. AQ18 includes several additional features, such as the ability to discover strong patterns in data (which optimize a multi-criterion measure of description quality), and automatic constructive induction. The latter feature enables the program to automatically search for a better representation space when the original one is found inadequate. For more details on the recent work on AQ learning consult (Kaufman & Michalski, 1999; Michalski, to appear).

As mentioned earlier, an H-group and an L-group in LEM may logically intersect, that is, contain inconsistent examples. AQ learning programs provide three simple methods for handling such inconsistency: *ignore* (inconsistent examples are ignored), *include-in-positive* (inconsistent examples are included only in the set of positive examples), and *include-in-negative* (inconsistent examples are included only in the set of negative examples). One can also employ a statistical (ROC) method for resolving inconsistency (Michalski & McCormick, 1971).

3.4. Darwinian Evolution mode in LEM1 and LEM2

In Darwinian Evolution mode, LEM1 employs a simple genetic algorithm GA2 (Michalski & Zhang, 1999). GA2 uses a real-value representation, uniform mutation, proportional selection, and uniform crossover. Each gene has 1/L chance of being mutated, where L is the number of attributes in the representation of an individual (the number of genes in a genome). Mutation is done by increasing or decreasing a value of the gene by a fixed unit (.1 or 10%). Mutations are not allowed to produce values outside of the boundaries

of the attribute domain. The uniform crossover operator takes two parent individuals and produces a child, whose genes are inherited from each parent with equal probability. The initial population is generated by a floating point random number generator. There is no elitism. In testing experiments, LEM1 was compared with genetic algorithm GA2 run alone, and another genetic algorithm, GA1. GA1 is a simplified version of GA2 that does not include a crossover operator.

LEM2 employs the AQ18 learning program in Machine Learning mode (without constructive induction) and the EV evolutionary algorithm in Darwinian Evolution mode (Cervone & Michalski, to appear). EV is an introductory evolutionary algorithm which applies uniform random reproduction selection with M parents and N children, and binary tournament survival. In the experiments described here M = 100, N = 10, and a mutation rate is 0.5. In testing experiments, LEM2 was compared with EV run alone, and in some cases also with two other algorithms:

—EP, a simple "Evolution Program" with M parents, N children, a deterministic reproduction selection, and binary tournament survival (M = 100, N = 10).
—ES, a simple "Evolution Strategy" with M*parents, M*brood-size children, deterministic reproduction selection, and truncation survival (M = 100, brood-size = 10).

Programs GA1, GA2, EV, ES and EP were provided by Professor Ken De Jong. In the experiments reported here, LEM2 was run with HFT = 30% and LFT = 30%. When LEM2 was run in the uniLEM mode, the learning program was set to create general rulesets in the first four generations, and then switched to determine specific rulesets (the example enfolding method). LEM2 used an adaptive attribute quantization procedure which starts with the domain size of 5 for all attributes, and then iteratively increases the domain size by 5 whenever the evolution progress is insufficient (no progress after Machine Learning mode has been repeated five times). For more details see (Cervone & Michalski, to appear).

4. Experiments

This section briefly describes selected results from initial experimental studies that compared LEM1 with GA1 and GA2 in solving two types of problems. One type concerned function optimization and the second type concerned parameter estimation in nonlinear digital filter design. To show that LEM methodology can scale-up to more complex problems, we also present selected results from applying LEM2 to the same type function optimization problems but with a much larger number of variables.

4.1. Experimental study 1: simple function optimization

This study applied LEM1 and genetic algorithms GA1 and GA2 to problems of optimizing five functions f_1, f_2, f_3, f_4, and f_5, described in (De Jong, 1975; Goldberg, 1989). These problems have been popular in the field of evolutionary computation because they represent different kinds of difficulties in function optimization. Specifically, they concern the optimization of functions that vary in terms of such characteristics as: continuous or

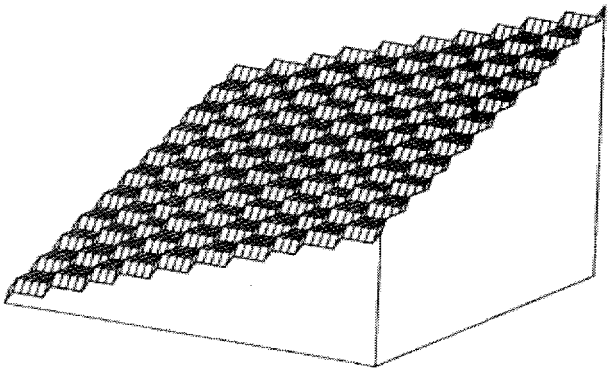

Figure 5. Inverted two-dimensional graph of f_3. (Reprinted with permission of Kenneth De Jong)

discontinuous, convex or nonconvex, unimodal or multimodal, low dimensionality or high dimensionality, deterministic or stochastic.

In this study, LEM1, GA1 and GA2 were applied to determine both the minimum (the original formulation of the problems) and the maximum of each function. We present here only results from optimizing two functions: f_3, a discontinuous function that poses a rather difficult problem for evolutionary algorithms, and f_4, a high dimensionality function with noise. Other results, which tended to be similar to the ones presented here, are reported in (Michalski & Zhang, 1999). In the experiments, LEM1 used the population-based method for creating HIGH and L-groups with HPT = 25% and LPT = 25%. (The HPT and LPT parameters were chosen as a guess; there was no effort to optimize them in any way). In learning H-group descriptions, LEM1 did not take into consideration the evolutionary history (i.e., the lookback parameters were zero).

Problem 1. Find the maximum of function f_3 of five variables, illustrated in figure 5. (Results from seeking the minimum of the same function but with 100 variables are given in Study 3, Problem 4).

$$f_3(x_1, x_2, x_3, x_4, x_5) = \sum_{i=1}^{5} \text{integer}(x_i) \quad -5.12 \leq x_i \leq 5.12$$

Maximum: 25

Figure 6 presents graphically the results from applying GA1, GA2 and LEM1 to Problem 1. In this case, the fitness value corresponds to the function value. As we can see, LEM1 dramatically outperformed GA1 and GA2 in solving this problem. It reached the maximum already around the 60th generation, while GA2 was over 50% away from the maximum after the 500th generation, and GA1 performed even worse (by the number of generations is meant the ratio of the number of births to the population size; here 20). To test the possibility that the low performance of evolutionary algorithms might be attributable to the poor choice of mutation rate, in Study 2, a similar experiment was conducted with different mutation rates.

To characterize the relative performance of the tested algorithms, we introduced a measure called "Relative-Distance-to-Target," defined as the ratio of the difference between the target

Table 1. The δ-close counts for Problem 1.

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

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

Table 2. The speed-up of LEM1 over GA1 and GA2 for different values of δ in Problem 1.

	LEM1 speed-up for different δ								
	0.0%	1.0%	2.0%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
LEM1/GA1	$\gg$172	$\gg$172	$\gg$222	$\gg$222	$\gg$222	$\gg$227	$\gg$227	$\gg$227	$\gg$238
LEM1/GA2	$\gg$172	$\gg$172	$\gg$222	$\gg$222	$\gg$222	$\gg$227	$\gg$227	$\gg$227	$\gg$238

GA1 and GA2 solutions have not become δ-close to the maximum within 10,000 generations. "$\gg$N" means that the speedup LEM/GAi is at least N.

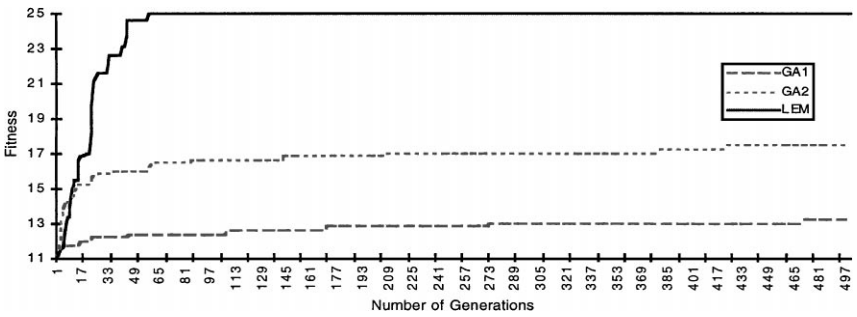

Figure 6. Evolution process within 500 generations (10,000 births) for function f_3.

value and the obtained value, to the target value, expressed in percentage, after a given number of generations. This measure was used to determine the *δ-close count*, defined as the number of generations in the evolutionary process after which the Relative-Distance-to-Target of the solution reaches the value δ (Table 1).

By dividing the δ-close count for GA1 and GA2 by the δ-close count for LEM1, we estimated the LEM1's "evolution speed-up" over GA1 and GA2, respectively. For example, a speed-up LEM/GA equal 10 means that LEM reaches the given δ-close fitness function value in 10 times fewer generations than GA. Table 2 shows the speed-up of LEM1 over GA1 and GA2 for different values of δ.

Problem 2: Find the maximum of the function f_4 of 30 continuous variables with Gaussian noise:

$$f_4(x_1, x_2, \ldots, x_{30}) = \sum_{i=1}^{30} i x_i^4 + \text{Gauss}(0, 1), \quad -1.28 \leq x_i \leq 1.28$$

Maximum: approximately 1248.225.

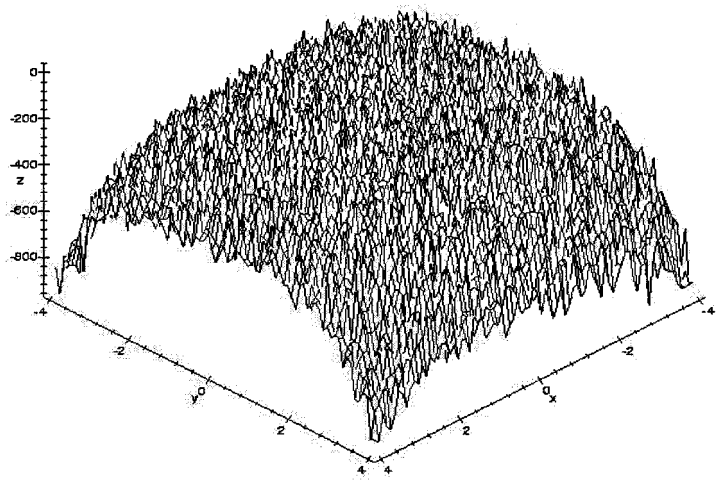

Figure 7. An inverted two-dimensional projection of the function f_4 with 30 variables.

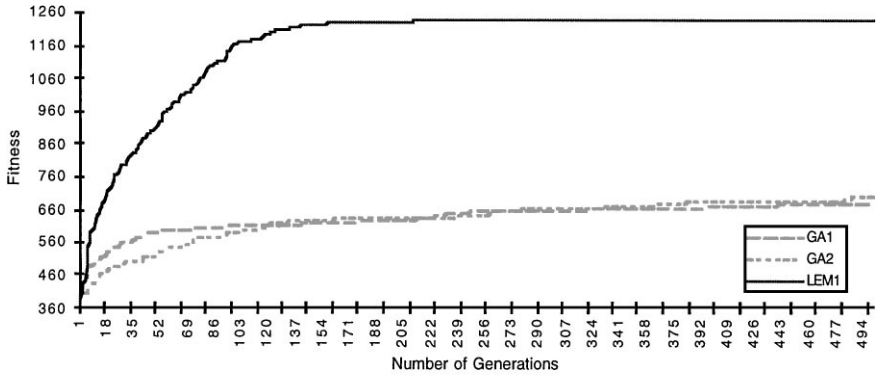

Figure 8. An evolution process for finding the maximum of f_4 using GA1, GA2 and LEM1.

Figure 7 presents an inverted two-dimensional projection of f_4.

Results from applying GA1, GA2, and LEM1 to Problem 2 are illustrated in figure 8. In this problem, the fitness value corresponds to the function value. As figure 8 shows, LEM1 significantly outperformed GA1 and GA2. It came very close to the maximum after the 150th generation, while GA1 and GA2 were still over 50% away from the maximum after the 500th generation.

Table 3 shows the speed-up of LEM1 over GA1 and GA2 for different values of δ.

Table 3. The speed-up of LEM1 over GA1 and GA2 for different values of δ.

	LEM speed-up ratio for different δ							
	1.0%	2.0%	3.0%	4.0%	5.0%	6.0%	7.0%	8.0%
GA1/LEM	$\gg$20.37	$\gg$66.23	$\gg$75.76	$\gg$83.33	$\gg$86.21	$\gg$90.91	$\gg$102.0	$\gg$103.1
GA2/LEM	$\gg$20.37	$\gg$66.23	$\gg$75.76	$\gg$83.33	$\gg$86.21	$\gg$90.91	$\gg$102.0	$\gg$103.1

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

4.2. Experimental study 2: designing digital filters

In this study, LEM1 was compared with GA1 and GA2 on a different type of problems, namely the parameter estimation for digital filter design. We present here a sample of results (for more details, see Coletti et al., to appear). The fitness function was defined by equations specifying linear and nonlinear filters presented in (Yao & Sethares, 1994).

Problem 3. Determine optimal parameters of nonlinear filters defined by the equation:

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

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)

To pose a problem for evolutionary computation, it was assumed that coefficients -0.3, 0.4, 1.25, and -2.5 in the above equation are variables. The problem then was to find their correct values using samples $\{\langle vector_i, y(vector_i)\rangle\}$, where $vector_I$ is a specific assignment of values to variables and $y(vector_i)$ is the value of the equation for this assignement.

Individuals in an evolving population are thus vectors with four real-valued variables ("genes"). When substituted in the equation, individuals generate a value of y that is compared with the value computed when correct coefficients are used in the equation. The fitness of an individual is inversely proportional to the difference between the result and the correct value. The individual that specifies coefficients giving the lowest error is thus assigned the highest fitness.

The parameters of algorithms GA1 and GA2 were:

Gene representation (type of attributes)	Real
Number of genes per individual (number of variables)	4
Gene landscape (constraint on range)	-30 to 30
Number of individuals per generation	60
Mutation rate	0.25
Maximum number of births	100,000
Maximum number of generations	1,500

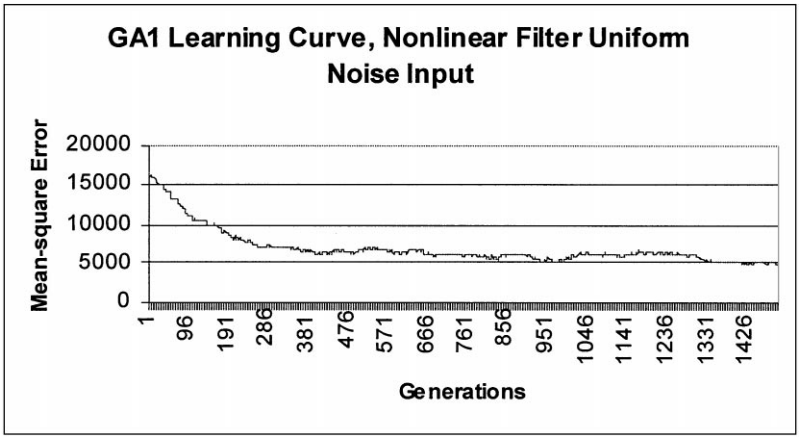

Figure 9. GA1 learning curve, nonlinear filter, uniform noise input.

Experiments used three different sets of samples as input data to LEM1, GA1, and GA2. In Machine Learning mode, LEM1 used the population-based method with HPT and LPT both equal to 30%. The population of each generation was 20; the *learn-probe* was 3, the *learn-threshold* was 0.01; *dar-probe* was 3, *dar-threshold* was 0.01.

Each variable was discretized into a fixed number of 200 ranges. Each program was executed 10 times, each time using a different input data. The seeds used for starting a random number generator differed for each run. Presented results are averages obtained in these runs. LEM1 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 $2.5\sin(\Pi/10)$ 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 correct values and the output values generated by the program. The fitness function was defined as in (Yao & Sethares, 1994), namely, as the reciprocal of the mean-square error over the 200 sample window:

$$\text{Fitness(Candidate)} = \frac{1}{\text{MeanSquareError}} = \frac{200}{\sum_{200}(\mathit{Candidate} - \mathit{KnownValue})^2}$$

LEM1, GA1, GA2 were applied ten times using uniform noise, sine, and step function inputs. The initial populations for each run were generated randomly. The convergence rates varied significantly with the change of initial conditions. The average performance was thus not very representative, as it was frequently dominated by a few runs. Since the goal of the experiments was to minimize the error, the learning curve that fastest converged to zero was selected from among the results obtained by each method for each input function.

For a non-linear filter, experiments were performed with uniform random input, unit step input, and sine wave input. In all cases, LEM1 significantly outperformed GA1 and GA2.

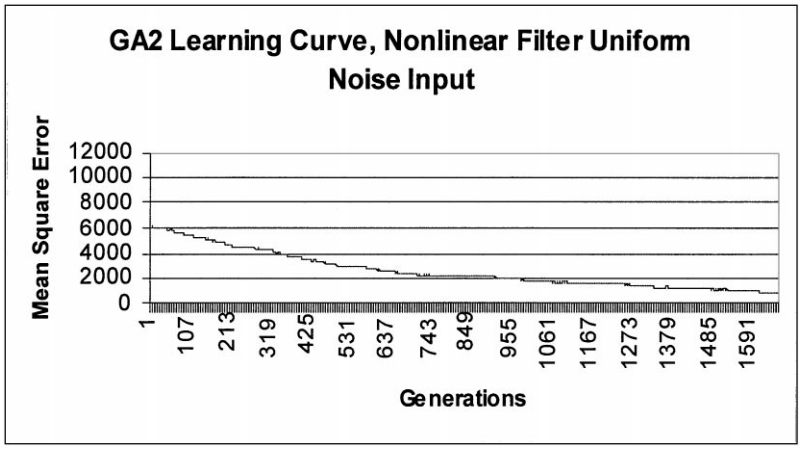

Figure 10. GA2 learning curve, nonlinear filter, uniform random noise input.

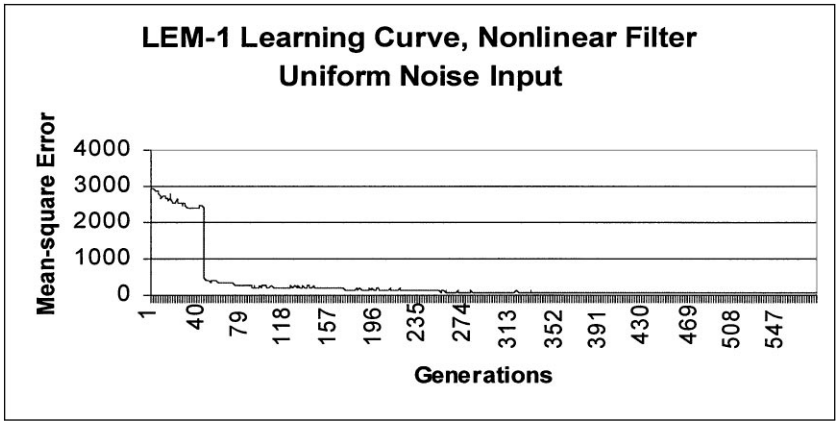

Figure 11. LEM-1 learning curve, nonlinear filter, uniform random noise input.

For illustration, figures 9–11 show results for the case of a nonlinear filter with uniform noise input. Results from other runs were similar.

As figures 9–11 show, the convergence of GA1 and GA2 was relatively slow. The effect of Machine Learning mode is seen 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. LEM1 switched 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. Experiments have shown that LEM1 can significantly accelerate the evolutionary process due to the symbolic learning steps.

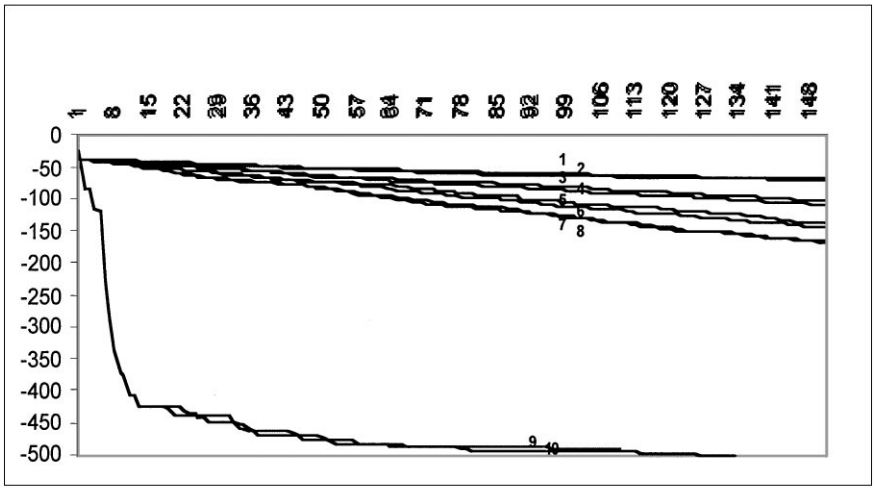

Figure 12. An evolutionary process for finding the minimum of f_3 with 100 variables using EP, EV and ES evolutionary computation methods and LEM2 in uniLEM and duoLEM versions.

4.3. Experimental study 3: more complex function optimization

Problem 4. Find the minimum of function f_3 (the same as in Problem 1, but with 100 variables; see figure 7 for a 2D graph):

$$f_3(x_i) = \sum_{i=1}^{100} \text{integer}(x_i), \quad -5.12 \le x_i \le 5.12$$

$$\text{Min}(f_3) = f_3(-5.12, -5.12, \ldots, -5.12) = -500$$

To this problem, we applied LEM2 and three evolutionary computation methods: EV, EP and ES, using different mutation rates. Results are presented in figure 12. The horizontal axis in figure 12 represents the number of births in thousands and the vertical axis represents the function value. In the caption of figure 12, EP p, c, m; EV p, c, m; and ES p, m stand, respectively, for "Evolutionary Program," "Evolutionary algorithm," and "Evolutionary strategy," where p denotes the number of parents, c denotes the number of children, and m is the mutation rate.

LEM2 was run in both the uniLEM and the duoLEM versions. The uniLEM version used the simplest, random population method in the start-over operation. The duoLEM version used EV with 0.5 mutation rate in Darwinian Evolution mode. AQ18 was run 8335 times, and EV was run 499 times (with population size 100 and 10 children).

In Machine Learning mode, the number of births was (100 − #individuals) in the H-group, as determined by HFT = 30%). The duoLEM2 and uniLEM2 versions exhibited a

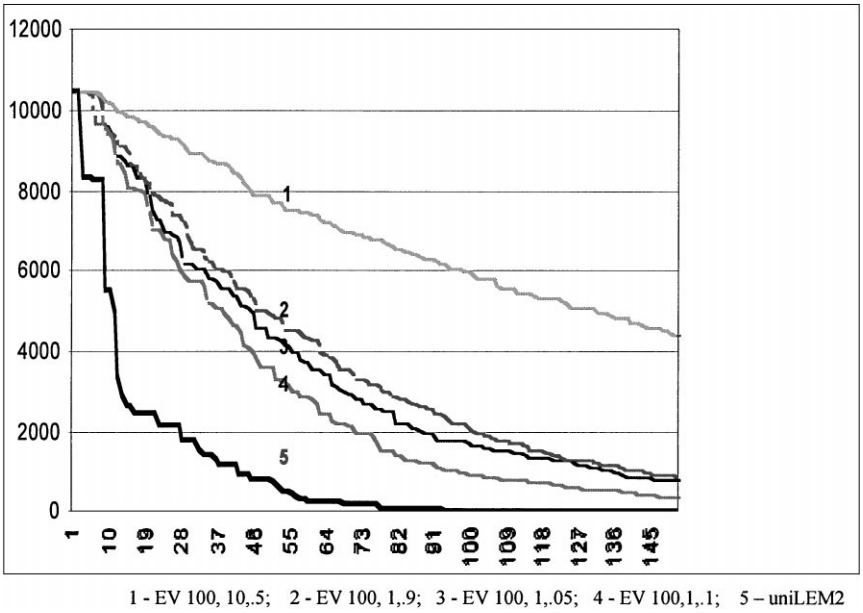

Figure 13. An evolution process for finding the minimum of f_4 using the EV evolutionary computation method and LEM2 in the uniLEM version.

very similar performance. They both converged toward the minimum dramatically faster than the evolutionary methods, regardless of their mutation rate.

Problem 5. Find the minimum of function f_4 of 100 variables with a Gaussian noise:

$$f_4(x_1, x_2, \ldots, x_{100}) = \sum_{i=1}^{100} i x_i^4 + \text{Gauss}(0, 1), \quad -1.28 \leq x_i \leq 1.28$$

Minimum: 0

The function f_4 is the same as the function in Problem 1, but involves 100 variables (recall figure 7 with an inverted 2D projection). To this problem, we applied LEM2 and the EV evolutionary method with different mutation rates and varying numbers of parents and children. Results are illustrated in figure 13.

In the caption of figure 13, EV p,c,m stands for the evolutionary algorithm EV with p parents, c children, and m mutation rate. The horizontal axis represents the number of births in thousands and the vertical axis represents the function value. The figure shows that the mutation rate had a significant effect on the performance of EV. LEM2 significantly outperformed EV, regardless of its mutation rate. The evolutionary process in LEM2 exhibited a few insight jumps at early phases of evolution. For more results from experimentation with LEM2, see (Cervone & Michalski, to appear).

4.4. Discussion of experimental results

In all experiments, LEM1 and LEM2 significantly outperformed evolutionary computation methods employed in our studies. While these results are highly encouraging, they should not be used, however, for making any strong generalizations but viewed as preliminary. The results from the evolutionary computation methods might have been stronger if more sophisticated methods were used. On the other hand, LEM1 and an introductory version of LEM2 used in the experiments are only rudimentary implementations of the LEM methodology. They are missing many features that we plan to include in the future implementation of LEM.

5. Relation to other research

The described LEM methodology is to the author's knowledge an original development. In searching through the literature, the author found several papers describing efforts to apply machine learning to evolutionary computation, but they are very different from LEM. There has also been work done on the application of evolutionary computation to machine learning. Some of the work along these two directions is briefly reviewed below.

Sebag & Schoenauer (1994) applied machine learning to adaptively control the crossover operation in genetic algorithms (implementing their own version of AQ-type learning). In their system, a machine learning method develops rules characterizing relevant crossovers. These rules are then used to bias the selection of crossover operators. Sebag, Schoneauer, & Ravise (1997a) used inductive learning for determining mutation step-size in evolutionary parameter optimization. Ravise & Sabag (1996) described a method for using rules to prevent new generations from repeating past errors. In a follow-up work, Sebag, Schoenauer, & Ravise (1997) proposed keeping track of past evolution failures by using templates of unfit individuals, called "virtual losers." An evolution operator, which they call "flee-mutation," aims at creating individuals different from the virtual losers.

Grefenstette (1991) developed a genetic learning system, SAMUEL, that implements a form of Lamarckian evolution. The system was designed for sequential decision making in a multi-agent environment. A strategy, in the form of *if-then* control rules, is applied to a given world state and certain actions are performed. This strategy is then modified either directly, based on the interaction with the environment, or indirectly by changing the rules' strength within the strategy. The changes in a strategy are passed to its offspring. This is a Lamarckian-type process that takes into consideration the performance of a single individual when evolving new individuals.

As to the application of evolutionary computation to machine learning, most research concerned the improvement of propositional concept learning. An indirect form of such application is to evolve the "best" subsest of attributes from a collection of original attributes in order to concept learning (Forsburg, 1976; Vafaie & De Jong 1992). Another form concerns an improvement of the learning method itself (e.g., Vafaie & De Jong, 1991; De Jong, Spears, & Gordon, 1993; Giordana & Neri, 1995; Greene & Smith 1993; Janikow, 1993; Hekanaho, 1997). There have also been efforts to use genetic algorithms to evolve a population of biases for a machine learning algorithm. For example, Turney (1995) applied a genetic algorithm to evolve weights assigned to attributes in the C4.5 program in

order to derive the minimum cost decision tree. Evolutionary algorithms have also been applied to improve relational learning, e.g. (Augier, Venturini, & Kodratoff, 1995; Hekanaho, 1998).

6. Conclusion

The learnable evolution model (LEM) presented here applies machine learning to guide evolutionary processes. LEM can work as a combination of Machine Learning and Darwinian Evolution modes (duoLEM), or in Machine Learning mode only (uniLEM). In Machine Learning mode, at each step of evolution, a learning system creates a hypothesis characterizing differences between high performing and low performing individuals in the population. This hypothesis is then instantiated to generate new individuals.

Among surprising findings stemming from the LEM development, two appear to be particularly significant. One is that due to machine learning, an evolutionary process can be dramatically speeded up over the Darwinian type of evolution. Another finding is that the process of evolving a target individual (a solution), which is conceptually opposite to inductive generalization, can be directly helped by such inductive generalization. Specifically, at each step of evolution a method of inductive generalization hypothesizes consecutively smaller subspaces of the search space which are likely to contain the solution.

The paper presented a number of ideas and techniques for implementing the LEM methodology. Some of them have been used in the preliminary systems LEM1 and LEM2. These systems employ AQ-type attributional learning which was indicated as particularly advantageous for LEM. In pilot experiments, LEM1 and LEM2 strongly outperformed evolutionary methods used in the study, both on problems of function optimization and parameter estimation. The power of LEM is attributable to the fact that discriminant descriptions created by machine learning can be interpreted as qualitative differentials of the fitness landscape and used to hypothesize the desirable direction of evolution. Our experiments demonstrated that the employment of the qualitative differentials can significantly speed-up the evolutionary process as compared to the semi-blind, parallel trial and error Darwinian-type evolution. Such speedups frequently involved quantum leaps of the fitness function. These leaps, termed insight jumps, signify the discovery of the correct direction of evolution and stand in contrast with gradual improvements that are typical of Darwinian-type evolutionary processes.

The price for such an evolutionary speedup is a higher computational complexity of individual steps of evolution. Instead of performing simple Darwinian operators, each step of LEM (in Machine Learning mode) involves determining qualitative differentials that involve more complex computations. Their complexity depends on the machine learning method used. It may be noted, however, that if LEM is found effective for solving a given class of problems then its complexity could be reduced by an efficient software and/or hardware implementation of the learning method. For example, a parallel implementation of the AQ algorithm can reduce its complexity from linear to logarithmic.

Experiments indicate that LEM employing current AQ learners (such as AQ15c or AQ18) can scale-up to highly complex problems. For example, LEM2 (a preliminary version not optimized for efficiency) took less than 20 minutes on the SUN SPARC 20 workstation to

solve the optimization problem with 100 continuous variables (Section 4.3, Problem 4). Such an efficiency can be explained by the fact that qualitative differentials that are in the form of attributional rulesets delineate hyper-rectangular subsets of the search space. By performing an incremental specialization of these rulesets, the search process explores consecutively smaller, nested subspaces. This facilitates a rapid convergence of the evolution process toward the solution (or a set of solutions).

It may also be worth noting that in the case of attributional ruleset learning, qualitative differentials are determined simultaneously in many subareas of the fitness landscape. Each subarea correponds to one rule of a ruleset. Such a parallel search increases chances for finding the global optimum. When the fitness landscape has several global optima, the system may be able to determine all optima or a subset of them during a single evolutionary process.

Although initial results from experimenting with LEM are highly promising, that are many unresolved problems that require further research. In particular, there is a need for systematic theoretical and experimental investigations of the methodology in various implementations, understanding their tradeoffs, strengths and limitations, and determining the most appropriate areas for their application. Some applications of LEM may require new types of learning systems, thus LEM may also stimulate research in the field of machine learning. As LEM bridges the fields of evolutionary computation and machine learning, different implementations may involve different combinations of methods from these two fields. This creates a plethora of possible new research directions.

LEM has a potential for a wide range of applications, in particular, to such domains as highly complex optimization or search problems, engineering design, drug design, evolvable hardware, software engineering, economics, data mining, and automatic programming. Concluding, LEM represents a new class of evolutionary processes that opens challenging avenues for further research and may impact many practical applications.

Acknowledgments

This paper is a significantly modified and extended version of the invited paper presented by the author at the 4th International Workshop on Multistrategy Learning, held in Desenzano del Garda, June 11–14, 1998. The author is indebted to many people for their assistance, discussions, and valuable feedback in conducting the presented research.

In developing the idea of LEM, the author was inspired by research on evolvable systems done by Hugo de Garis and his group at ATR in Japan. Qi Zhang implemented LEM1 and performed experiments on its application to optimization problems described in Study 1. Mark Coletti, Craig Mandsager, Tom Lash, and Rida Mustafa, author's students in the CS580 Artificial Intelligence class, performed experiments on applying LEM1 to problems in digital filter design described in Study 2. Guido Cervone implemented LEM2 and performed experiments described in Study 3. Guido's assistance and collaboration helped to improve this paper in several ways. He and his friend Domenico Napoletani also prepared the 3D image presented in figure 6.

Ken Kaufman and Zbigniew Michalewicz provided a thoughtful criticism and comments that helped to improve the final version of the paper. Ken De Jong provided the evolutionary computation algorithms used in the experiments. His long-term dedicated investigations in

the area of evolutionary computation have stimulated the author's interests in this area. Elizabeth Marchut-Michalski has sustained the author in good spirits during intensive and sometimes frustrating efforts to finish this work in time.

Deeply felt words of gratitude go to all those mentioned above.

The author would also like to express very special thanks to Ephraim P. Glinert from the National Science Foundation for providing grant support for this work when it was most critically needed.

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

Note

1. After Chevalier de Lamarck, the title of Jean Baptiste Pierre Antoine de Monet, French naturalist (1744–1829), who is the author of the theory that adaptive responses to environment cause structural changes capable of being inherited.

References

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

Augier, S., Venturini, G., & Kodratoff, Y. (1995). Learning first order logic rules with a genetic algorithm. *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining* (pp. 21–26). Montreal, Canada: AAAI Press.

Baeck, T., Fogel, D. B., & Michalewicz, Z. (1997). (Eds.), *Handbook of evolutionary computation*. Oxford University Press.

Banzhaf, W., Nordin P., Keller R. E., & Francone, F. D. (1998). *Genetic programming: An introduction*, San Francisco, CA: Morgan Kaufman Publishers, Inc.

Baldwin, J. M. (1896). A new factor in evolution. *American naturalist* (Vol. 30) (pp. 441–451, 536–553).

Bloedorn, E. & Michalski, R. S. (1998). Data-driven constructive induction: A methodology and its applications, *IEEE Intelligent Systems*, special issue on feature transformation and subset selection. Huan Liu & Hiroshi Motoda (Eds.), March–April.

Cervone, G. & Michalski, R. S. (to appear). Design and experiments with LEM2 implementation of the Learnable Evolution Model, *Reports of The Machine Learning and Inference Laboratory*, George Mason University.

Clark, P. & Niblett, R. (1989). The CN2 induction algorithm, *Machine Learning*, 3.

Cohen, W. W. (1995). Fast effective rule induction. *Proceedings of the Twelfth International Conference on Machine Learning*.

Coletti, M., Lash, T., Mandsager, C. Michalski, R. S., & Moustafa, R. (1999). Comparing the learnable evolution model with genetic algorithms in the area of digital filter design, *Reports of The Machine Learning and Inference Laboratory*, George Mason University, MLI 99–5.

Darwin, C. (1859). *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*, London, John Murray.

de Garis, Hugo. (1996). CAM-BRAIN: The evolutionary engineering of a billion neuron artificial brain by 2001 which grows/evolve at electronic speeds inside a cellular automata machine (CAM). *Lecture notes in computer science*, Vol. 1062. Towards evolvable hardware (pp 76–98). Springer-Verlag.

de Garis, H., Korkin, M., Gers, F., Nawa, E., & Hough, M. (to appear). Building an Artificial Brain Using an FPGA Based CAM-Brain Machine, *Applied Mathematics and Computation Journal* (special issue on Artificial life and robotics, artificial brain, brain computing and brainware, North Holland.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Ph.D. Thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor.

De Jong, K. A., Spears, W. M., & Gordon, F. D. (1993). Using genetic algorithms for concept learning, Machine Learning, *13*, 161–188.

De Jong, K. A. (to appear). *Evolutionary computation: theory and practice*. MIT Press.

Dietterich, T. G. (1997). Machine-learning research: four current directions, *AI Magazine*, *18*(4).

Esposito, F., Michalski, R. S., & Saitta, L. (Eds.) (1998). *Proceedings of the Fourth International Workshop on Multistrategy Learning*, Desenzano del Garda.

Forsburg, S. (1976). AQPLUS: *An adaptive random search method for selecting a best set of attributes from a large collection of candidates*, Internal Technical Report, Department of Computer Science, University of Illinois, Urbana.

Giordana A. & Neri, F. (1995). Search-intensive concept induction. *Evolutionary Computation*, *3*(4), 375–416.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

Grefenstette, J. (1991). Lamarckian learning in multi-agent environment. In R. Belew & L. Booker (Eds.). *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, GA: Morgan Kaufmann (pp. 303–310).

Greene D. P. & Smith, S. F. (1993). Competition-based induction of decision models from examples. *Machine Learning*, *13*, 229–257.

Hekanaho, J. (1997). GA-based rule enhancement concept learning. *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (pp. 183–186). Newport Beach, CA: AAAI Press.

Hekanaho, J. (1998). DOGMA: A GA-based relational learner, *TUCS Technical Reports Series*, Report No. 168.

Hinton, G. E. & Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems*, *1*, 495–502.

Holland, J. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press.

Janikow, C. Z. (1993). A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, *13*, 189–228.

Kaufman, K. & Michalski, R. S. (1999). Learning from inconsistent and noisy data: the AQ18 approach, *Foundations of Intelligent Systems*, 11th International Symposium, ISMIS'99, Warsaw, Poland, Spring.

Koza, J. R. (1994). *Genetic programming II: automatic discovery of reusable programs*. The MIT Press.

Lavrac, N. & Dzeroski, S. (1994). *Inductive logic programming: techniques and applications*. Ellis Horwood.

Maloof, M. & Michalski, R. S. (1999). AQ-PM: a system for partial memory learning. *Proceedings of Intelligent Information Systems VIII*, Ustron, Poland.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs*, 3rd edn. Springer Verlag.

Michalski, R. S. (1969). On the quasi-minimal solution of the general covering problem, *Proceedings of the V International Symposium on Information Processing (FCIP 69)*, Yugoslavia, Bled (Vol. A3) Switching Circuits, (pp. 125–128).

Michalski, R. S. (1978). A planar geometrical model for representing multi-dimensional discrete spaces and multiple-valued logic functions. *Reports of the Department of Computer Science*, No. 897, University of Illinois at Champaign-Urbana.

Michalski, R. S. (1973). Discovering classification rules using variable-valued logic system VL1. *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford, CA (pp. 162–172).

Michalski, R. S. (1983). A theory and methodology of inductive learning, *Artificial Intelligence*, *20*(2) 111–161.

Michalski, R. S. (1994). Inferential theory of learning: developing foundations for multistrategy learning, In R. S. Michalski & G. Tecuci (Eds.), *Machine learning: a multistrategy approach* (Vol. IV) San Mateo, CA, Morgan Kaufmann.

Michalski, R. S. (1998). Learnable evolution: combining symbolic and evolutionary learning. *Proceedings of the 4th International Workshop on Multistrategy Learning*, Decenzano del Garda, Italy.

Michalski, R. S. (to appear). Natural induction: theory, methodology and its application to machine learning and data mining. *Reports of the Machine Learning and Inference Laboratory*, George Mason University.

Michalski, R. S. & Cervone, G. (to appear). Adaptive anchoring quantization of continuous variables: the ANCHOR method. *Reports of the Machine Learning and Inference Laboratory*, George Mason University.

Michalski, R. S., Bratko, I., & Kubat, M. (1988). *Machine learning and data mining: methods and applications*. John Wiley and Sons.

Michalski, R. S. & McCormick, B. H. (1971). Interval generalization of switching theory. *Proceedings of the Third Annual Houston Conference on Computer and System Science*, Houston, Texas.

Michalski, R. S., Mozetic, I., Hong, J., & Lavrac, N. (1986). The AQ15 inductive learning system: an overview and experiments. *Reports of the Intelligent Systems Group*, No. 86-20, UIUCDCS-R-86-1260, Department of Computer Science, University of Illinois, Urbana.

Michalski, R. S. & 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*, George Mason University.

Mitchell, M. (1996). *An introduction to genetic algorithms*, Cambridge, MA: MIT Press.

Mitchell, T. M. (1997). Does machine learning really work. *AI Magazine, 18*(3).

Ravise, C. & Sebag, M. (1996). An advanced evolution should not repeat its past errors. In L. Saitta (Ed.), *Proceedings of the 13th International Conference on Machine Learning* (pp. 400–408).

Sebag, M. & Schoenauer, M. (1994). Controlling crossover through inductive learning In Y. Davidor, H. P. Schwefel & R. Manner (Eds.), *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, LNVS (Vol. 866) (pp. 209–218). Springer-Verlag.

Sebag, M., Schoenauer M., & Ravise, C. (1997a). Inductive Learning of multation step-size in evolutionary paramter optimization. *Proceedings of the 6th Annual Conference on Evolutionary Programming*, Indianapolis. (pp. 247–261). LNCS (Vol. 1213).

Sebag, M., Shoenauer, M., & Ravise, C. (1997b). Toward civilized evolution: developing inhibitions, *Proceedings of the 7th International Conference on Genetic Algorithms* (pp. 291–298).

Turney, P. D. (1995). Cost-sensitive classification: empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research, 2*, 369–409.

Vafaie, H. & De Jong, K. A. (1991). Improving the performance of a rule induction system using genetic algorithms, *Proceedings of the First International Workshop on Multistrategy Learning*, WV: MSL-91, Harpers Ferry.

Vafaie, H. & De Jong, K. A. (1992). Genetic algorithms as a tool for feature selection in machine learning. *Proceedings of the 4th International Conference on Tools with Artificial Intelligence*, Arlington, VA.

Wnek, J., Kaufman, K., Bloedorn, E., & Michalski, R. S. (1995). Inductive learning system AQ15c: the method and user's guide, *Reports of the Machine Learning and Inference Laboratory*, MLI 95-4, George Mason University, Fairfax, VA.

Yao, L. & Sethares, W. (1994). Nonlinear parameter estimation via the genetic algorithm. *IEEE Transactions on Signal Processing, 42*(4), 927–935.

Zhang, Q. (1997). Knowledge visualizer: a software system for visualizing data, patterns and their relationships. *Reports of the Machine Learning and Inference Laboratory*, MLI 97-14, George Mason University, Fairfax, VA.

Received December 22, 1998

Accepted June 18, 1999

Final manuscript June 18, 1999