Learning and Evolution:
An Introduction to Non-Darwinian Evolutionary Computation

Ryszard S. Michalski

Machine Learning and Inference Laboratory
School of Computational Sciences
George Mason University, Fairfax, VA, USA
and
Institute of Computer Science
Polish Academy of Sciences, Warsaw, Poland

michalski@gmu.edu

Abstract. The field of evolutionary computation has drawn inspiration from Darwinian evolution in which species adapt to the environment through random variations and selection of the fittest. This type of evolutionary computation has found wide applications, but suffers from low efficiency. A recently proposed non-Darwinian form, called *Learnable Evolution Model* or *LEM*, applies a learning process to guide evolutionary processes. Instead of random mutations and re-combinations, LEM performs hypothesis formation and instantiation. Experiments have shown that LEM may speed-up an evolution process by two or more orders of magnitude over Darwinian-type algorithms in terms of the number of births (or fitness evaluations). The price is a higher complexity of hypothesis formation and instantiation over mutation and recombination operators. LEM appears to be particularly advantageous in problem domains in which fitness evaluation is costly or time-consuming, such as evolutionary design, complex optimization problems, fluid dynamics, evolvable hardware, drug design, and others.

1 Introduction

In his prodigious treatise "On the Origin of Species by Means of Natural Selection," Darwin conceived the idea that the evolution of species 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 such biological or natural evolution, new organisms are created via asexual reproduction with variation (mutation) or via sexual reproduction (recombination). The underlying assumption is that the evolution process is not guided by some "external mind," but proceeds through semi-random modifications of genotypes through mutation and recombination, and progresses to more advanced forms due to the principle of the "survival of the fittest."

In Darwinian evolution, individuals thus serve as holders and transmitters of their genetic material. Their life experiences play no role in shaping their offspring's properties. Jean-Baptiste Lamarck's[1] idea that traits learned during the lifetime of an individual could be directly transmitted to progeny has been rejected as biologically viable because it is difficult to construe a mechanism through which this could occur.[2] Many scientists believe, however, that there is another mechanism through which learned traits might influence evolution, namely, the so-called *Baldwin effect* (Baldwin, 1896). This effect stems from the fact that, due to learning, certain individuals can survive even though their genetic material may be suboptimal. In this way, some traits that otherwise would not survive are passed on to the next generations. Some researchers argue that under certain conditions such learning may actually slow genetic change and thus slow the progress of evolution (Anderson, 1997).

More than a century after Darwin introduced his theory of evolution, computer scientists adopted it as a model for implementing evolutionary computation (e.g., Holland, 1975; Goldberg, 1989; Michalewicz, 1996; Koza et al. 1999). Their efforts have led to the development of several major approaches, such as genetic algorithms, evolutionary strategy, genetic programming, and evolutionary programs. These and related approaches, viewed jointly, constitute the rapidly growing field of evolutionary computation (see, e.g., Baeck, Fogel, M. Mitchell; 1996, Banzhaf et al., 1999; Zalzala, 2000).

Methods of evolutionary computation based on principles of Darwinian evolution use various forms of mutation and/or recombination as variation operators. These operators are easy to implement and can be applied without any knowledge of the problem area. Therefore, Darwinian-type evolutionary computation has found a very wide range of applications, including many kinds of optimization and search problems, automatic programming, engineering design, game playing, machine learning, evolvable hardware, and many others.

The Darwinian-type evolutionary computation is, however, semi-blind: the mutation is a random, typically small, modification of a current solution; the crossover is a semi-random recombination of two or more solutions; and selection is a sort of parallel hill climbing. In this type of evolution, the generation of new individuals is not guided by principles learned from past generations, but is a form of the trial and error process executed in parallel. Consequently, computational processes based on Darwinian evolution tend to be not very efficient. Low efficiency has been the major obstacle in applying Darwinian-type evolutionary computation to highly complex problems. The objective of many research efforts in this area has been thus to increase the efficiency of the evolutionary process.

---

[1] Jean-Baptiste Lamarck, a French naturalist (1744-1829), who proposed a theory that the experience of an individual can be encoded in some way and passed to the genome of the offspring.

[2] Recent studies show that Lamarckian evolution appears to apply in the case of antibody genes (Steel and Blanden, 2000).

In modeling computational processes after principles of biological evolution, the field of evolutionary computation has followed a long-practiced tradition of looking to nature when seeking technological solutions. The imitation of bird flying by mythological Icarus and Daedalus is an early example of such efforts. In seeking technological solutions, the "imitate-the-nature" approach, however, frequently does not lead to the best engineering results. Modern examples of successful solutions that are not imitations of nature include balloons, automobiles, airplanes, television, electronic calculators, computers, etc.

This paper discusses a recently proposed, non-Darwinian form of evolutionary computation, called *Learnable Evolution Model* or *LEM*. In LEM, new individuals are created by hypothesis formation and instantiation, rather through mutation or recombination. This form of evolutionary computation attempts to model "intellectual evolution"---the evolution of ideas, technical solutions, human organizations, artifacts, etc.---rather than biological evolution. In contrast to Darwinian evolution, an intellectual evolution is guided by an "intelligent mind," that is, by humans who analyze advantages and disadvantages of previous generation of solutions and use the developed understanding in creating next generation of solutions. It is due to the intellectual evolution that the process of evolving the automobile, airplane or computer from primitive prototypes to modern forms was astonishingly rapid, taking just few human generations.

The idea and the first version of the LEM methodology were introduced in (Michalski, 1998). A more advanced and comprehensive version is in (Michalski, 2000). Its early implementation, LEM1, produced very encouraging results on selected function optimization problems (Michalski and Zhang, 1999). Subsequent experiments with a more advanced implementation, LEM2, confirmed earlier results and added new highly encouraging ones (e.g., Cervone et al., 2000, Cervone, Kaufman and Michalski, 2000).

The following sections briefly describe LEM and its relationship to Darwinian-type evolutionary computation, and then summarize results of testing experiments.

2 LEM vs. Darwinian Evolutionary Computation

Darwinian-type evolutionary algorithms can be generally viewed as stochastic techniques for performing parallel searches in a space of possible solutions. They simulate natural evolution by creating and evolving a population of individuals until a termination condition is met. Each individual in the population represents a potential solution to a problem. Such a solution can be represented as a vector of parameters, an instantiation of function arguments, an engineering design, a concept description, a control strategy, a pattern, a computer program, etc. A precondition for applying an evolutionary algorithm is the availability of a method for evaluating the quality (fitness) of individuals from the viewpoint of the given goal.

A general schema of an evolutionary computation consists of the following steps:

1. **Initialization**
   t := 0
   Create an initial population P(t) and evaluate fitness of its individuals.
2. **Selection**
   t := t+1
   Select a new population from the current one based on their fitness: P(t) := Select(P(t-1))
3. **Modification**
   Apply *change operators* to generate new individuals: P(t) := Modify(P(t))
4. **Evaluation**
   Evaluate fitness of individuals in P(t)
5. **Termination**
   If P(t) satisfies the *termination condition*, then END, otherwise go to step **2.**

Different evolutionary algorithms differ in the way individuals are represented, created, evaluated, selected and modified. They may also use different orders of steps in the above schema, employ single or multiple criteria in fitness evaluation, assume different termination conditions, and simultaneously evolve more that one population. Some algorithms (specifically, genetic algorithms) make a distinction between the *search space* and the *solution space*. The search space is a space of encoded solutions ("genotypes"), and the solution space is the space of actual solutions ("phenotypes"). Encoded solutions have to be mapped onto the actual solutions before the solution quality or fitness is evaluated.

As mentioned earlier, in Darwinian-type (henceforth, also called conventional) evolutionary algorithms, change operators are typically some forms of mutation and/or recombination. Mutation is a unary transformation operator that creates new individuals by modifying previous individuals. Recombination is an n-ary operator (where n is typically 2) that creates new individuals by combining parts of n individuals. Both operators are typically semi-random, in the sense that they make random modifications within certain constraints.

The selection operator selects individuals for the next population. Typical selection methods include proportional selection (the probability of selecting an individual is proportional to its fitness), tournament selection (two or more individuals compete for being selected on the basis of their fitness), and ranking selection (individuals are sorted according to their fitness and selected according to probabilities associated with different ranks on the sorted list). The termination condition evaluates the progress of the evolutionary process and decides whether to continue it or not.

Learnable Evolution Model, briefly, LEM, also follows this general schema. Its fundamental difference from Darwinian-type algorithms lies in step 4, as it generates new individuals in very different way. In contrast to semi-random change operators employed in Darwinian-type algorithms, LEM conducts a reasoning process in

generating new individuals. Specifically, it applies operators of *hypothesis formation* and *hypothesis instantiation*.

The operator of hypothesis formation selects from a population a group of high-performing individuals, called the H-group, and a group of low-performing individuals, called the L-group, according to their fitness. The H-group and L-group may be selected from the current population or from a sequence of past populations. These groups can be selected using a *population-based* method, a fitness-based method, or a combination of the two. The population-based method applies High and Low Population Thresholds (HPT and LPT) in selecting individuals, and fitness-based method applies High and Low Fitness Thresholds (HFT and LFT). The thresholds can be fixed or may change in the process of evolution. For details, see (Michalski, 2000).

The H-group and L-group are then supplied to a machine learning program that generates a general hypothesis distinguishing between high performing from low performing individuals. Such a hypothesis can be viewed as a theory explaining the differences between the two groups. Alternatively, it can be viewed as a characterization of the sub-areas of the search space that are likely to contain the top performing individuals (the best solutions). Once such a hypothesis has been generated, the algorithm generates new individuals that satisfy the hypothesis.

In principle, any inductive learning method can be used for hypothesis formation. LEM1 and LEM2 implementations of the LEM methodology has used the AQ-type learning method (specifically, AQ15 and AQ18, respectively; see Wnek et al., 1995; Kaufman and Michalski, 2000b). This method appears to be particularly advantageous for LEM, because it employs *attributional calculus* as the representation language (Michalski, 2000b). Attributional calculus adds to the conventional logic operators new operators, such as *internal disjunction*, *internal conjunction*, *attribution relation,* and the *range operator*, which are particularly useful for characterizing groups of similar individuals. Attributional calculus stands between propositional calculus and predicate calculus in terms of its representational power.

New individuals are generated by a hypothesis instantiation operator that instantiates the given hypothesis in various ways. To very simply illustrate, suppose that a hypothesis was generated by an AQ-type learning program and expressed in the form of two *attributional rules* (these rules are in a simplified form to facilitate explanation):

Rule 1: [x = a v c] & [y = 2.3 .. 4] & [z > 5] (sup=80)
Rule 2: [x = b v d v e] & [z = 3.5 .. 6.4] (sup=15) (1)

where the domains of attributes x, y, and z are: D(x) = {a,b,c,d,e,f}, and D(y) and D(z) range over real numbers between 0 and 10.

Rules in (1) characterize two subareas of the search space that contain high performing individuals. The first rule states that high performing individuals appear in the area in which the variable x has value a or c, the variable y takes value between 2.3 and 4, and the variable z takes value greater than 5. The parameter sup (support) indicates that this rules covers 80 individuals in the H-group. The second rule describes an alternative set of conditions, namely, that high performing individuals appear also in the area in which differ x takes value b or d or e, and z takes values from the real interval between 3.5 and 6.4. The second rule covers sup=15 individuals in the H-group. Note that Rule 2 does not include variable y. This means that this variable was found irrelevant for differentiating between high and low performing individuals.

The hypothesis (1) is a generalization of the set of individuals in the H-group. Thus, it may potentially cover many other, unobserved individuals. The instantiation operator instantiates the hypothesis in different ways, that is, generates different individuals that satisfy conditions of the rules. For example, using hypothesis (1), the operator may generate such individuals as:

<a, 2, 6>, <c, 3.5, 9.1>, <a, 2.1, 6.4> (based on Rule 1)
<d, 2, 6>, <e, 5.5, 4.3>, <b, 2.2, 4.5> (based on Rule 2) (2)

Since variable y is not present in Rule 2, any values of y could be selected from D(y) to instantiate this rule. In our experiments, variables not present in the rule were instantiated to values selected randomly from among those that appeared in individuals of the training set (H-group and L-group).

The newly generated individuals are combined with the previous ones, and a new population is selected using some selection method. Again, an H-group and L-group are generated and operations of hypothesis generation and instantiation are repeated. The process continues until a *LEM termination condition* is met, e.g., the (presumably) global or a satisfactory solution has been found.

The above-described process of creating new individuals by operators of hypothesis formation (through inductive generalization) and hypothesis instantiation (by generating individuals satisfying the hypothesis) constitutes the *Machine Learning Mode* of LEM. A general form of LEM includes two versions: *uniLEM*, which repetitively applies the Machine Learning mode until a termination condition is satisfied, and *duoLEM*, which toggles between Machine Learning and Darwinian Evolution mode, switching from one mode to another when the termination condition for the given mode is satisfied (when there is little progress in executing the mode). The Darwinian Evolution Mode executes one of the existing conventional evolutionary algorithms.

A comprehensive explanation of various details of the LEM methodology and its variants is in (Michalski, 1999).

3 A Simple Illustration of LEM

To illustrate LEM, let us consider a very simple search problem in a discrete space. The search space is spanned over four discrete variables: x, y, w, and z, with domains {0,1}, {0,1}, {0,1}, and {0,1,2}, respectively. Figure 1A, presents this space using the General Logic Diagram or GLD (Michalski, 1978; Zhang, 1997). Each cell of the diagram represents one individual. For example, the uppermost cell marked by 7 represents the vector: <0, 0, 0, 2>. The initial population is visualized by cells marked by dark dots (Figure 1A). The numbers next to the dots indicate the fitness value of the individual. The search goal is to determine individuals with the highest fitness, represented by the cell marked by an x (with the fitness value of 9).

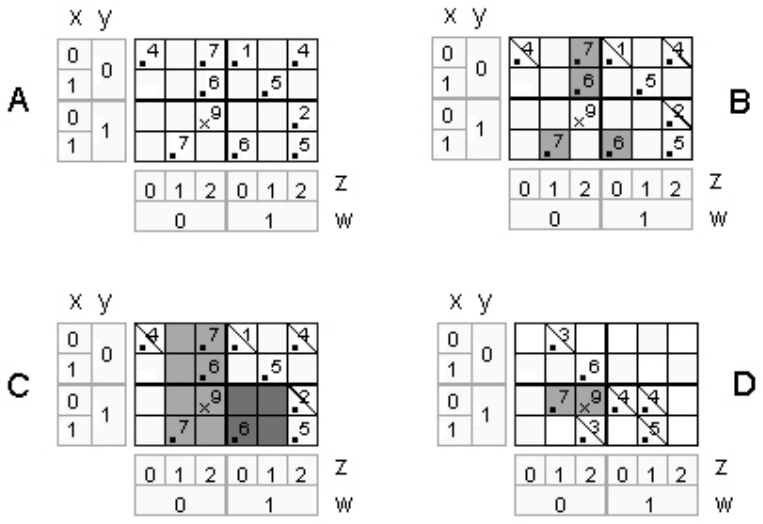

Figure 1. The search space and four states of the LEM search process.

We assume that descriptions discriminating between an H-group and an L-group are in the form of attributional rules learned AQ-type learning programs. Figure 1B presents the H-group individuals (the gray-shaded cells) and L-group individuals (crossed cells) determined from the initial population. The shaded areas in Figure 1C represent two attributional rules discriminating between the H-group and the L-group: [w = 0] & [z = 1 v 2] and [y = 1] & [w = 1] & [z = 0 v 1].

Figure 1D shows individuals in the H-group (shaded cells) and the L-group (crossed cells) generated by instantiating rules in Figure 1C. The shaded area in Figure 1D represents a rule that discriminates between these groups: [x=0] & [y=1] & [w=0] & [z=1v2]. This rule was obtained through incremental specialization of the parent rule, and covers two individuals. The global solution will be located in the next iteration.

4 Summary of Testing Experiments

To test the LEM methodology, it has been implemented in a general-purpose form in programs LEM1 (Michalski and Zhang, 1999) and LEM2 (Cervone, 1999). It was also employed in program ISHED1, specifically tailored to problems of optimizing heat exchangers (Kaufman and Michalski, 2000a). Both LEM1 and LEM2 were applied to a range of function optimization problems. LEM1 was also successsulfy applied to a problem in filter design (Colleti et. al, 1999). LEM2 was tested in a wide range of experiments dealing with optimizing different types of functions with different numbers of arguments, ranging from 4 to180 continuous variables.

In all experiments LEM2 strongly outperformed conventional evolutionary computation algorithms employed in the study, frequently achieving two or more order of magnitude speedups in terms of the number of births (or function evaluations). Results from LEM2 were also significantly better than the best results from conventional evolutionary algorithms published on a website. These and other recent results have been described in (Cervone et. al, 2000a; Cervone et al., 2000b). Results from experiments with ISHED1 were presented in (Kaufman and Michalski, 2000a). According to the collaborating expert, ISHED1's heat exchanger designs were comparable to the best human designs in the case of uniform flow of refrigerant, and were superior to the best human designs in the case of non-uniform flow.

5 Conclusion

Experimental studies conducted so far have strongly demonstrated that the proposed Learnable Evolution Model can significantly speed up evolutionary computation processes in terms of the number of births (or fitness evaluations). These speed-ups have been achieved at the cost of higher complexity of operators generating new individuals (hypothesis formation and instantiation). An open problem is thus to study trade-offs associated with the LEM application to different problem domains. It is safe to say, however, that LEM is likely to be highly advantageous in problem areas in which computation of the evaluation function is costly or time-consuming. Such areas include engineering design, complex optimization problems, fluid dynamics, evolvable hardware, drug design and automatic programming.

Another limiting aspect of LEM is that in order to apply it, the machine learning system must be able to work with the given representation of individuals. For example, if individuals are represented as attribute-value vectors, rule and decision tree learning

systems can be applied. If they are represented as relational structures, a structural learning system must be employed.

Concluding, among the open problems for further research on LEM are to understand the benefits, trade-offs, advantages and disadvantages of LEM versus Darwinian-type evolutionary algorithms in different problem domains.

Acknowledgments

The author thanks Guido Cervone, Ken Kaufman and Liviu Panait for an excellent collaboration on the LEM project and the experimental validation of the LEM methodology. This research has been conducted in Machine Learning and Inference Laboratory at George Mason University. The Laboratory's research on this project has been supported in part by the National Science Foundation under Grants No. IIS-9904078 and IRI-9510644.

References

Anderson R.W. (1997), The Baldwin Effect, in *Handbook of Evolutionary Computation*, Section C3.4.1, pp. C3.:4:1-C3.4:15., IOP Publishing Ltd and Oxford: Oxford University Press.

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

Baldwin, J.M. (1896), A New Factor in Evolution, *American Naturalist*, vol. 30, pp.441-51.

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

Cervone, G., Michalski, R.S., Kaufman K., Panait, L. (2000), Combining Machine Learning with Evolutionary Computation: Recent Results on LEM, *Proceedings of the Fifth International Workshop on Multistrategy Learning* (MSL2000), Michalski, R.S. and Brazdil, P. B. (eds.), Guimaraes, Portugal, June 5-7.

Cervone, G., Kaufman, K.A., and Michalski, R.S. (2000), Experimental Validations of the Learnable Evolution Model, *Proceedings of the 2000 Congress on Evolutionary Computation*, La Jolla, California.

Coletti, M., Lash, T., Mandsager, C., Michalski, R.S., and Moustafa, R. (1999), Comparing Performance of the Learnable Evolution Model and Genetic Algorithms on Problems in Digital Signal Filter Design. *Proceedings of the 1999 Genetic and Evolutionary Computation Conference (GECCO)*.

Darwin, C. (1859), On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life, John Murray, London.

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

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

Kaufman, K. A. and Michalski, R.S. (2000a), Applying Learnable Evolution Model to Heat Exchanger Design*, Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence* (AAAI-2000/IAAI-2000), Austin, Texas.

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

Koza, J.R., Bennett, F. H. III, Andre D., Keane M. A. (1999), *Genetic Programming III: Darwinian Invention and Problem Solving,* Morgan Kaufmann Publishers, San Francisco, CA.

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

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

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

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

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

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press.Back, Thomas, *Optimization by Means of Genetic Algorithms*, ENCORE.

Steele E. J. and Blanden R. V. (2000), Lamarck and Antibody Genes, *Science*, Vol. 288, No. 5475, pp. 2318.

Wnek, J., Kaufman, K., Bloedorn, E. and 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.