RECENT RESULTS FROM THE EXPERIMENTAL EVALUATION OF THE LEARNABLE EVOLUTION MODEL

by

G. Cervone
K. Kaufman
R. S. Michalski

Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2002, 2002.

GECCO - 2002

2002 GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE

LATE-BREAKING PAPERS

New York, New York
July 11-13, 2002

Erick Cantú-Paz
Lawrence Livermore National Laboratory

Editor

Recent Results from the Experimental Evaluation of the Learnable Evolution Model

Guido Cervone
Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA

Kenneth A. Kaufman
Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA

Ryszard S. Michalski
Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA

Abstract

The Learnable Evolution Model (LEM) represents a form of non-Darwinian evolutionary computation that is guided by a learning system. Specifically, LEM "genetically engineers" new populations via hypothesis formation and instantiation. This paper presents results from new studies in which LEM was systematically tested on a range of optimization problems and a complex real world design task. The study involved LEM2, a new implementation oriented toward function optimization, and LEMd-ISHED, an implementation oriented toward engineering design. LEM2 strongly outperformed tested evolutionary algorithms in terms of the *evolution length*, measured by the number of fitness function evaluations needed to reach the solution. This evolutionary speedup also translated to an *execution speedup* whenever the fitness evaluation time was above a small threshold (a fraction of a second). Experiments with LEMd-ISHED on problems of optimizing heat exchangers (evaporators) produced designs that matched or exceeded designs produced by human experts.

1 INTRODUCTION

Most current methods of evolutionary computation employ various forms of mutation and/or recombination for creating new individuals. In the Learnable Evolution Model (LEM), individuals are created by a process of hypothesis generation and instantiation. Specifically, LEM employs a machine learning program that at each step of evolution learns descriptions differentiating high-performance and low-performance individuals. These descriptions are then instantiated in different ways to generate new individuals. (Michalski, 1998, Cervone, 1999, Michalski 2000).

This paper presents results from a systematic testing of two new LEM implementations LEM2---oriented toward function optimization, and LEMd-ISHED—oriented toward a class of engineering design problems. In function optimization experiments, LEM2 was compared with selected conventional algorithms both in terms of the *evolution length* (the number of fitness function evaluations or births needed to achieve a desirable solution) and the *evolution time* (the computation time to achieve the solution). Among its new features, LEM2 includes an *adaptive anchoring discretization* method, called ANCHOR, for adaptively discretizing continuous variables (Michalski and Cervone, 2001).

In engineering design experiments, LEMd-ISHED was applied to a problem of designing optimized evaporators in air conditioning units (in the name of the system, LEMd stands for the LEM method tailored for design problems, and ISHED indicates the specific problem class. Henceforth, for simplicity, we will use "ISHED" whenever we talk about the specific system).

2 AN OVERVIEW OF THE LEM METHODOLOGY

The Learnable Evolution Model (LEM) represents a fundamentally different approach to evolutionary computation than conventional Darwinian-type evolutionary algorithms. In Darwinian algorithms, new individuals are generated through various forms of mutation and/or recombination operators. Such operators are easy to execute and apply to a wide range of problems. They are, however, semi-random, and take into consideration neither the experience of individuals in a given population (as in Lamarckian-type evolution), nor the experience of populations through the history of the evolution process. Thus, Darwinian-type evolutionary algorithms tend to be inefficient, and this diminishes their effectiveness in complex real-world problems.

The novel idea introduced in LEM is that an evolutionary computation can be guided by hypotheses created by a machine learning program that identify areas in the search space that most likely include the sought optimum, or

optima. Such hypotheses are created on the basis of the current and, optionally, also past populations of individuals. A general form of LEM may also include periods of conducting a Darwinian form of evolution, when it appears to be useful. Such a form is implemented in the *duoLEM* version, which integrates the intrinsic LEM mode of operation employing machine learning, called *Machine Learning mode*, and a conventional evolutionary mode, called *Darwinian Evolution mode*, which executes some conventional evolutionary algorithm (in which new individuals are generated by a form of mutation and/or recombination operators).

LEM can also be run in a uniLEM version in which, the *Machine Learning mode*, described above, is the sole method for generating new populations.

In Machine Learning mode, at each step of evolution, a population is divided into three groups of individuals: High-performing individuals (H-group) that score high on the fitness function, Low-performing individuals (L-group) that score low on the fitness function, and the rest. The creation of these groups can be done using various methods (Michalski, 2000). The population from which these groups are selected may be the current population, or a combination of the current and past populations. The selected H-group and L-group are supplied to a learning program that creates general hypotheses distinguishing between these groups. The hypotheses are then instantiated in various ways to produce new, candidate individuals. The candidate individuals compete in terms of their fitness with previously generated individuals for the inclusion in the new population.

Thus, unlike in Darwinian-type evolutionary computation, the generation of new individuals in LEM may take into consideration not only properties of individuals, but also properties of populations of individuals, and even of the evolution history. Initial experiments have shown that guiding evolutionary processes by hypotheses generated on the basis of whole populations can lead to a dramatic evolutionary speedup (Michalski, 1998; Cervone, 1999; Michalski, 2000).

In executing duoLEM, one mode runs until a *mode termination criterion* is met, and control is then switched to the other mode. The mode termination criterion is met when there is insufficient improvement of the fitness function after a certain number of populations, or the allocated computational resources are exhausted. The main justification for a duoLEM approach is that operators of hypothesis creation and instantiation are more computationally costly than conventional evolutionary operators, but are more powerful in selecting individuals. By allowing the interchangeable execution of both Darwinian and Machine Learning modes, duoLEM can utilize the best features of both strategies, and also facilitates comparative studies of the two. A detailed description of the methodology can be found in (Michalski, 2000).

The following sections describe recent experiments applying LEM2 to function optimization problems, the ISHED implementation, and ISHED's application to the design of optimized heat exchangers.

3 FUNCTION OPTIMIZATION

LEM2 was applied to systematically test the LEM methodology on function optimization problems. The problems involved several functions widely used by the Evolutionary Computation community for benchmarking different evolutionary computation algorithms, and thus were particularly attractive for doing a comparative study of the LEM2 performance.

Due to space limitations, we present here just a sample of representative results concerning the optimization of the Rastrigin function. This function was selected because the results from its optimization by conventional, Darwinian-type methods are readily available, thus enabling a comparison of results. To test the scalability of LEM2 to complex optimization problems, we have extended the number of variables in the problem.

The comparison of LEM2's results with those obtained by Darwinian-type evolution algorithms was made both in terms of *evolution length*, defined as the number of evaluations (or births) needed to determine the target solution, and *evolution time*, defined as the execution time required to achieve this solution. The reason for measuring both characteristics is that LEM and Darwinian type algorithms represent a tradeoff between the complexity of the population generating operators and the evolution length. Operators of hypotheses generation and instantiation used in LEM2 are more computationally costly than mutation and crossover, but LEM2's evolution length is typically much shorter than that of Darwinian algorithms. Using the concept of evolution length, we analyzed the *evolutionary speedup* of compared algorithms. To define this concept, note that the function to be optimized can be transformed (by inversion and/or adding a constant) into a form in which the function optimum is the maximum of the transformed function, and the function minimum is 0. Thus, the optimized function can be considered to be a positive-only fitness function, and the optimization problem becomes a problem of finding a solution with the maximum fitness.

Let us introduce parameter δ as a measure of the relative distance between the highest fitness solution found by an algorithm and the globally highest fitness solution. Specifically, let δ be the ratio of the difference between the fitness of the globally maximal solution and the best solution found by the algorithm, divided by the fitness of the globally maximal solution. For example, if the global maximum of a function is 100 and the best solution found is 99, then δ is 0.01. By *δ-close solution* is meant a solution that differs from the global optimal by at most δ. For the purpose of evaluating performance of LEM2 and other algorithms in function optimization, it is assumed that δ is a controllable parameter, and the evolution process continues until a δ-close solution (also called the *target solution*) is found.

The relative performance of two algorithms is characterized by two measures: the *evolution speedup* and the *execution speedup*. The evolution speedup of algorithm A over B for a given δ is defined as the ratio, expressed in percentage, of the number of births (or fitness evaluations) required by B to the number of births required by A to achieve the δ-close solution. The execution speedup of algorithm A over B for a given δ is defined as the ratio of the total computation time required by B to the total computation time required by A to achieve the target solution.

We tested different numbers of function arguments (variables), specifically, 10 or 20, 50 and 100. We measured the evolution length and the execution time of the programs compared. Experiments were performed with different sizes of the population, and repeated 10 times (runs) with different random initial populations. The results represent the highest fitness solution from the 10 runs performed by LEM2 and, for comparison, by ES, a program implementing an evolutionary strategy method.

The ES method was chosen because it is widely used in the field of evolutionary computation. ES employs a real-valued vector representation of individuals and deterministic selection (i.e., each parent is selected, and then mutated a fixed number of times). The mutation is done according to a Gaussian distribution, in which the mean is being mutated, and the standard deviation is a controllable parameter, called the *mutation rate*. Each variable has a $1/L$ probability of being mutated, where L is the total number of variables defining an individual. New individuals and their parents are sorted according to their fitness, and the *popsize* highest-fitness individuals are included in the next generation, where *popsize* is a numeric parameter denoting a fixed population size.

The ES program was implemented according to the description in (Baeck, Fogel and Michalewicz, 1997). It was implemented anew, rather than employing some existing program, as this has allowed us to closely integrate it with LEM2. Such integration facilitates running LEM2 and ES with identical parameters, that is, with identical initial populations, the same random number seed, and the same mutation and recombination operators when executing duoLEM and ES. We found on the Web an archive of the alleged best solutions for benchmark function optimization, achieved using several different algorithms, and for those solutions we determined comparable solutions from LEM2.

The experiment described below concerned the minimization of the Rastrigin function:

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

for the number of arguments, n, set to 20, 50 and 100, and ranging between –5.12 and 5.12.

The Rastrigin function has many local optima, and it is easy to miss the global solution. In these experiments, both uniLEM and duoLEM versions were employed, and their results were compared with previously published results obtained by a parallel GA with 16 subpopulations and 20 individuals per subpopulation (Muhlenbein, Schomisch, and Born, 1991).

Experiments compared LEM2, ES, and a parallel genetic algorithm, under different parameter settings, in terms of the evolution length and the evolution speedup for different values of δ. In the case of 20 variables, LEM2 showed an evolution speedup of about 10 over the other two strategies; for 50 variables, the speedup was about 15, and for 100 variables, it was in the 15-20 range depending on the value of δ (Cervone, Kaufman and Michalski, 2002).

We determined the dependence of the evolution length on the number of function arguments (variables) for the different methods (Figure 1). As shown there, the evolution length of ES and PGA increases with the number of variables significantly faster than that of LEM2 (the target solution was defined by $\delta = 0.1$). This result confirms the previous result that LEM2's advantage in terms of the evolution length grows with the complexity of the function (in terms of number of variables).

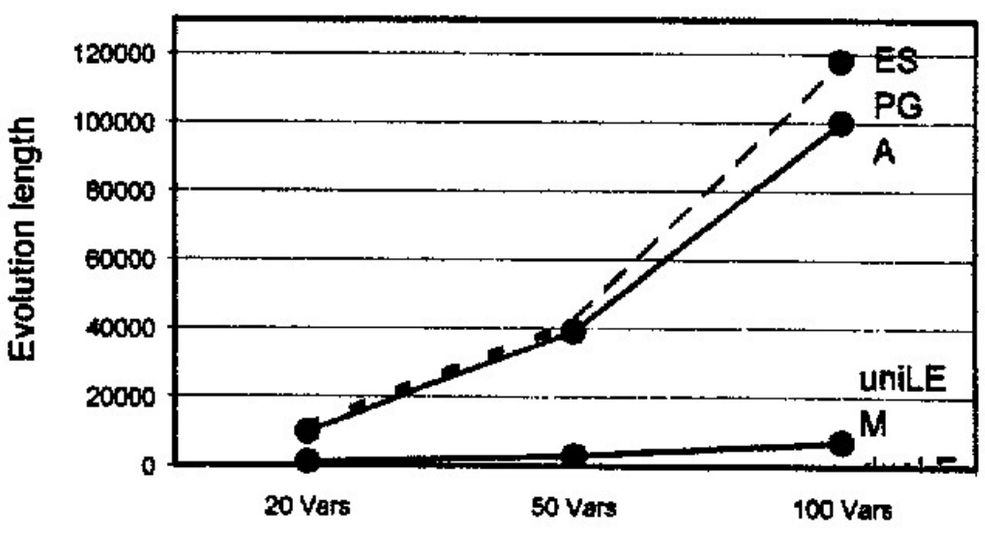

Figure 1: The increase of the evolution length (number of births) with the increase of the number of variables when optimizing the Rastrigin function using ES, parallel GA, and LEM2 (both uniLEM and duoLEM).

We also conducted experiments to determine the dependence of the execution speedup of LEM2 over ES on the evaluation delay. Figures 2, 3 and 4 present the results. As shown there, the evaluation delay, ED, needed to achieve ES =1, was in every case very small (it varied between 0.0043 and .0025 seconds). For larger evaluation delays, the execution speedup grew quickly and converged to the evolution speedup (9 for 20 variables, 13 for 50 variables, and 15 for 100 variables).

These results confirm the hypothesis that the speedup of LEM2/ES increases with the evaluation delay, and converges to the evolutionary speedup.

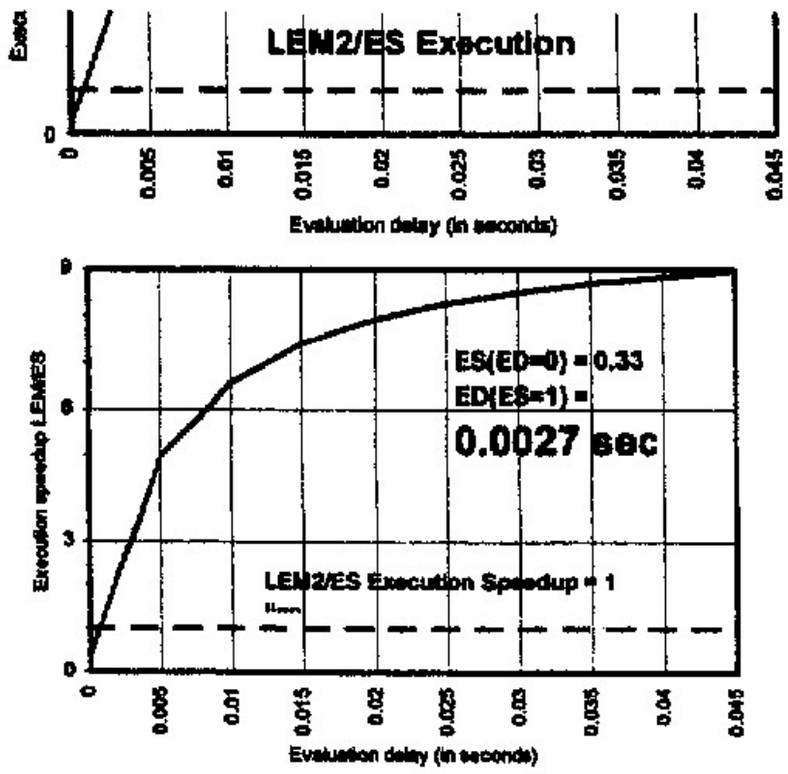

Figure 2: The dependence of the execution speedup of LEM2/ES for the Rastrigin function of 20 variables on the evaluation delay parameter.

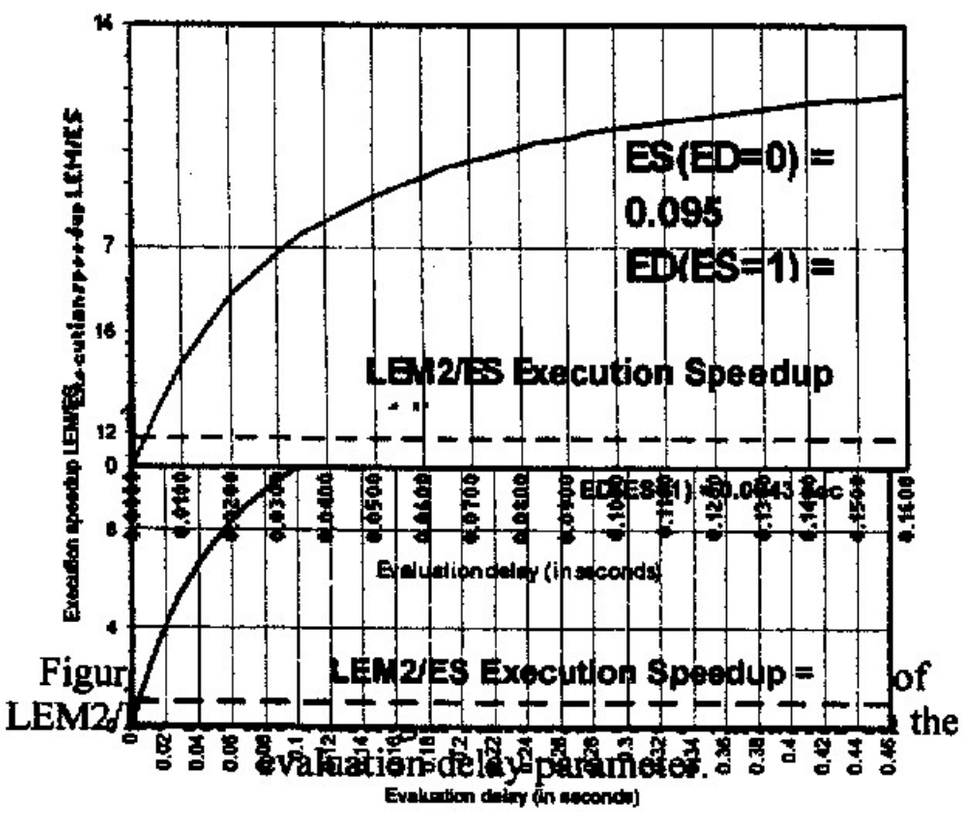

Figure 4: The dependence of the execution speedup of LEM2/ES on the evaluation delay for the Rastrigin function of 100 variables.

4 DESIGN PROBLEM

Complex design optimization problems appear to be an application domain for which the search capabilities of evolutionary computation algorithms are often very useful. They differ from the function optimization problems described above in one important respect. Specifically, in function optimization problems, the values of any of the variables may be usually set or modified in any fashion (within their defined ranges) without any harm to the integrity of the proposed solution. In design problems, however, variables will typically represent positions or configurations of components, and are subject to various constraints. Thus, arbitrary changes to them will often result in solutions that are either physically or practically infeasible. Even if we may assume that the fitness evaluator will recognize such infeasibility and return an appropriate score, the high density of such designs will likely hamper the evolution process, whether Darwinian or machine learning-based.

This motivated the development of LEMd, a LEM implementation oriented toward design problems. In contrast to LEM, LEMd includes domain-specific representation of variables. The optimized variables represent real-world aspects of the design configuration, and are subject to domain-specific constraints.

Instead of generic Darwinian operators, such as random mutations and recombinations, LEMd employs domain-specific *Design Modification (DM)* operators that make various changes in the candidate designs that are potentially useful according to the domain knowledge.

The algorithms for generating individuals, and instantiation of learned rules employ domain knowledge-based formulas and constraints tailored toward generating feasible designs.

LEMd assumes also that the quality of initial and subsequently generated designs can be evaluated in some way, for example, by a design simulator. LEMd employs duoLEM, but the Darwinian Evolutionary mode is done in a special way, because in practical design applications an expert often has domain knowledge that can be used to define meaningful design variations. As mentioned above, to generate such variations, Design Modification (DM) operators need to be defined based on the expert advice or knowledge. These operators perform design modifications that are tried in the process of evolutionary computation, rather than the random mutations and/or recombinations used in conventional evolutionary computation. Since in real-world problems, various domain constraints are known, LEMd also allows the user to define various constraints and applies them in both Darwinian and Machine Learning modes of operations.

In LEMd, individuals represent different designs under consideration. Each design is defined by a vector of attributes that characterizes it. A Control Module takes the current design population and determines which of LEM's evolutionary modes to apply. A new population of candidates is generated: through DM operators in

Darwinian Evolutionary mode, and through rule learning and instantiation in Machine Learning mode. The created population is then passed to the simulator for evaluation. The designs and their evaluations are passed to the Control Module for the next generation (iteration).

4.1 EVAPORATOR DESIGN OPTIMIZATION: LEMd-ISHED

To test the LEMd methodology in a real-world application domain, we implemented it in the LEMd-ISHED, or, briefly, the ISHED (Intelligent System for Heat Exchanger Design) program tailored to the problem of optimizing evaporators in air conditioning units under given environmental and technical constraints. To give the reader a better understanding of the complexity of this problem, a brief explanation follows.

In an air conditioner, the refrigerant flows through a loop. It is superheated and placed in contact with cooler outside air (in the condenser unit), where it transfers thermal energy (heat) out and liquifies. Returning to the evaporator, it comes into contact with the warmer interior air that is being pushed through the evaporator, as a result cooling the air and heating and evaporating the refrigerant. An evaporator consists of arrays of parallel tubes through which the refrigerant flows back and forth.

The path the refrigerant takes through these tubes will affect both the its temperature when it reaches a given tube, and the temperature of the air after it passes over the tube. The amount of heat transfer (cooling) the air conditioner will provide is the aggregate of the heat transfer provided by each of its evaporator's tubes. These terms aree a function of the temperature and volume per unit time of both the air and the refrigerant coming into contact at that tube. Different orderings of the tubes will change the characteristics of the refrigerant passing through each tube, and the results of prior air/refrigerant interactions will affect both substances' temperatures at later interactions, as will other factors. For instance, the refrigerant will lose pressure (and velocity) while passing through the bends between tubes; it thus helps if adjoining tubes are physically close to each other.

By changing the path of the refrigerant flow, one can therefore change the amount of heat transferred between air and refrigerant. The more that can be transferred overall, the more efficiently the interior air will be cooled to the desired temperature.

The optimization problem involves determining an arrangement of tubes that produces the highest evaporator capacity under given technical and environmental constraints. Because of the nature of the problem and the feasible ways of internally representing evaporator structures, both evolutionary modules utilize problem-specific customization. Traditional genetic operators, random mutations and crossovers, would be unworkable in this domain; therefore, we implemented eight domain-specific design modifying (DM) operators based on discussions with a domain expert. The DM operators change the characteristics of candidate evaporators in ways likely to lead to *admissible* new structures, that is, structures satisfying the given constraints. A selected operator is tried repeatedly with different operands in order to generate a feasible design, until it either succeeds or "times out" (based on control parameters specifying the allowed number of iterations), in which case another operator, hopefully more applicable, will be tried.

For example, one operator may create a split in a refrigerant path by moving the source of a tube's refrigerant closer to the inlet tube (Figure 5), a second operator may swap the tubes in the structure, another operator may graft a path of tubes into another path, etc. In Figure 5, solid arrowed lines represent the initial connections, with dashed ones representing new ones created by the DM operator. Each tube has only one source, so the dashed links replace links from their destination tubes' prior sources. An arrow that is open-ended on one side represents an inlet or outlet flow (flow into or out of the evaporator from/to other parts of the refrigerant circuitry). The application of these operators is domain knowledge driven, that is, operators are applied according to known technical constraints.

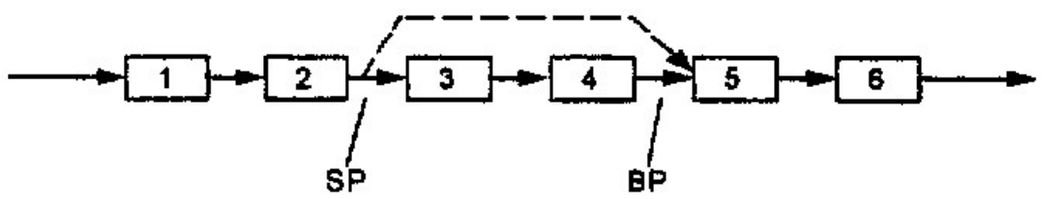

Figure 5: Application of the SPLIT Operator, SPLIT(2,5).

The second strategy, based on symbolic learning, examines the characteristics of both well- and poorly-performing designs, and automatically creates hypotheses (in the form of attributional rules) that characterize the well-performing architectures. These hypotheses are then applied to generate a new population of designs.

Machine learning mode in ISHED is also tailored to the evaporator design task. The hypotheses generated describe abstractions of the individual structures. Specifically, they specify only the location of inlet, outlet and split tubes. Beyond that, the instantiation module may choose among the different designs that fit the learned template, and generate the most plausible one according to the real-world background knowledge. Once rules are generated, an elitist strategy is used to form the next generation of proposed architectures..

An ISHED run proceeds as follows. Given instructions characterizing the environment for the target evaporator, an initial population of designs (specified by the user and/or randomly generated), and parameters for the evolutionary process, ISHED evolves populations of designs using a combination of Darwinian and machine learning operators for a specified number of generations (Kaufman and Michalski, 2000). The programs return a report that includes the best designs found and their estimated quality (capacity). Throughout the execution,

design capacities are determined by an evaporator simulator (Domanski, 1989).

4.2 EXPERIMENTS ON EVAPORATOR OPTIMIZATION

ISHED experiments were performed under different sets of conditions, such as different refrigerants, evaporator sizes and shapes, and airflow patterns. Industrially available air conditioning systems typically perform very efficiently if the airflow is fairly uniform. However, their efficiency drops off sharply if that is not the case; the side of the unit over which more air flows has a heavier cooling burden, so for best performance it needs to carry more and colder refrigerant. Manufacturers generally have not built models for such a situation.

During the course of ISHED development, many experiments with the system were conducted. The initial experiments concentrated on a well-known problem, using a common evaporator size and a fairly uniform airflow pattern. ISHED designs provided results comparable to the industry standard. One concern in some of the ISHED-generated designs was that after many generations of Darwinian evolution, designs would become chaotic in terms of their inter-tube connections (and the simulator wasn't fully reflecting the detrimental effect of this). Nonetheless, using available tools, an engineer could easily smooth some of the connections, hopefully at little or no cost to performance.

In later experiments, the refrigerant was changed, and the airflow pattern was defined as highly non-uniform. Under such conditions, industry-standard heat exchangers do not perform well. The best ISHED-produced architectures conformed intuitively to expectations of what a successful architecture in a non-uniform airflow should look like, and indeed performed far better than the currently-used expert-designed structures. Subsequent experiments varied the size and shape of the evaporator and similar results were observed.

The experiments during all stages of this work served to confirm the ability of ISHED to generate improved designs. There appears to be promise that the LEMd methodology would exhibit similar success in other design tasks.

5 RELATED WORK

The LEM methodology represents a new approach to evolutionary computation. Its relation to other evolutionary computation methods was briefly described in (Michalski, 2000). Unlike most methods of evolutionary computation that draw inspiration from Darwinian evolution, LEM attempts to model what we call an *intellectual evolution*. In intellectual evolution, which governs the development of human artifacts, the generation of new populations is based on the results of the analysis of the advantages and disadvantages of past populations.

The closest methods to LEM appear to be cultural evolution algorithms, as they utilize top performing individuals and use generalized beliefs in the evolutionary process (e.g., Reynolds, 1994; Rychtyckyj and Reynolds, 1999; Saleem and Reynolds, 2000). The approach taken in cultural algorithms differs, however, significantly from LEM. Unlike LEM, cultural evolution is a process of dual inheritance consisting of a "micro-evolutionary level," which involves individuals described by traits and modified by conventional evolutionary operators, and a "macro-evolutionary" level, in which individuals generate "mappa" representing generalized beliefs that are used to modify the performance of individuals in the population. LEM is different from cultural evolution algorithms in both, the way learning process is implemented and in the way its results are used in the process of evolution.

Sebag and Schoenauer (1994) applied AQ-type learning to adaptively control the crossover operation in genetic algorithms. In their system, the rules are used for the selection of the crossover operator. Sebag, Schoneauer and Ravise (1997a) used inductive learning for determining mutation step-size in evolutionary parameter optimization. Ravise and Sabag (1996) described a method for using rules to prevent new generations from repeating past errors. In a follow-up work, Sebag, Schoenauer and Ravise (1997b) 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 process that takes into consideration the performance of a single individual when evolving new individuals.

Another approach that extends the traditional Darwinian approach can be found in the GADO algorithm (Rasheed, 1998). GADO is an evolutionary algorithm developed for complex engineering problem optimization. It differs from traditional genetic algorithms primarily in the way new individuals are generated. It uses five different crossover operators, three of which are introduced in this algorithm: Line crossover, double line crossover, and guided crossover. However, unlike LEM, the algorithm does not create any generalizations of the current population, and therefore is significantly different.

6 CONCLUSION

The results presented in this paper confirm that LEM offers a powerful new methodology for non-Darwinian

evolutionary computation. Two implementations of LEM have been tested on selected and evaluated, LEM2 – on function optimization problems and ISHED – on an engineering design problem concerning optimization of tube arrangements in evaporators in air conditioners. In all function optimization experiments, LEM2 outperformed selected Darwinian-type evolutionary algorithms (mostly ES) in terms of the evolution length, sometimes achieving speedups of two or more orders of magnitude. In the evaporator design domain, ISHED was able to find solutions that were better or comparable to the best designs used in the industry.

Since operators of hypothesis generation and instantiation used by LEM2 are significantly more computationally costly than operators of mutation and recombination used in ES, the evolution speedup does not always result in the execution speedup. It was shown experimentally that if the fitness evaluation time is above a small threshold (tens of milliseconds), LEM2 also outperforms ES in terms of the execution time. The execution time speedup grows with fitness evaluation time, asymptotically converging to the evolution speedup. The most remarkable result of experiments is that the evolutionary speedup advantage of LEM2 over ES (evolutionary strategy) grew rapidly with the complexity of the optimized function. It is likely that a similar advantage would be obtained with regard to other Darwinian-type evolutionary computation algorithms.

Experiments also revealed a weakness in the LEM2 implementation that sometimes appears late in the evolutionary processes. When most of the variables reach their global optimum value, the program may take a long time to find the absolute optimal value for the few remaining variables. This problem is currently handled using the *Start-Over* operator. Further research is needed to determine a better method for handling this problem.

The LEM methodology is at a very early stage of development, and poses many interesting new research problems. They include a theoretical and experimental investigation of the trade-offs inherent in LEM, an implementation of more advanced versions of LEM, experimentation with different combinations of conventional evolutionary algorithms and machine learning algorithms, and testing the methodology in different application domains. Among important research topics are also the development of methods for applying LEM to dynamic landscapes, and to optimization problems with complex constraints.

Concluding, the experiments described here have strongly confirmed the earlier results that LEM can very significantly reduce the length of evolutionary computation over Darwinian-type algorithms, and thus can be particularly useful in domains where the fitness evaluation is time-consuming or costly. They also indicated a pattern, potentially highly significant for practical applications, in which the LEM advantage over Darwinian-type evolutionary computation appears to increase with the complexity of the optimization problem.

Acknowledgments

The authors thank Dr. Piotr Domanski from the National Institute of Standards and Technology for his invaluable help and consultation in the development of ISHED, and his simulator for evaluating evaporator designs. Thanks are also due to Domenico Napoletani of the Department of Mathematics at the University of Maryland for his help in plotting graphs of the functions used in experiments.

This research was conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's research has been supported in part by the National Science Foundation under Grants No. IIS-0097476 and IIS-9906858, and in part by the UMBC/LUCITE # 32 grant. Any opinions, findings and conclusions or recomendations expressed in the material on this site are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

References

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

Cervone, G. (1999). An Experimental Application of the Learnable Evolution Model to Selected Optimization Problems. *Reports of the Machine Learning and Inference Laboratory*, George Mason University, Fairfax, VA, MLI-1999-12

Cervone, G, Kaufman, K.A., Michalski, R.S. (2002). Validating Learnable Evolution Model on Selected Optimization and Design Problems. *Journal of Machine Learning Research* (submitted).

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

Grefenstette, J. (1991) Lamarckian Learning in Multi-agent Environment. *Proceedings of the Fourth International Conference on Genetic Algorithms*, R. Belew and L. Booker (Eds.), San Mateo, CA: Morgan Kaufmann, pp. 303-310.

Kaufman, K.A. and Michalski, R.S. (2000). Applying Learnable Evolution Model to Heat Exchanger Design. *Proceedings of the Twelfth International Conference on Innovative Applications of Artificial Intelligence*, Austin, TX, pp. 1014-1019.

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

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

Michalski. R.S. and Cervone, G. (2001). Adaptive Anchoring Discretization of Continuous Variables for the

Learnable Evolution Model. *Reports of the Machine Learning and Inference Laboratory*, George Mason University, Fairfax, VA, MLI-2001-5.

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

Rasheed, K. (1998) GADO: A Genetic Algorithm for Continuous Design Optimization. *Technical Report DCS-TR-352, Department of Computer Science*, Rutgers University, New Brunswick, NJ, 1998. Ph.D. Thesis.

Ravise, C. and Sebag, M. (1996). An Advanced Evolution Should Not Repeat Its Past Errors, *Proceedings of the 13th International Conference on Machine Learning*, L. Saitta (ed.), pp. 400-408.

Reynolds, R.G. (1994). An Introduction to Cultural Algorithms. *Proceedings of the 3rd Annual Conference on Evolutionary Programming*.

Rychtyckyj, N. and Reynolds, R.G., (1999). Using Cultural Algorithms to Improve Performance in Semantic Networks, *Proceedings of the Congress on Evolutionary Computation*, Michalewicz, Z., Schoenauer, M.. Yao, X., and Zazala, A. (eds.), Washington, DC, pp. 1651-1663.

Saleem S. and Reynolds, R.G., (2000). Cultural Algorithms in Dynamic Environments, *Congress on Evolutionary Computation 2000 (CSC00), vol. 2*, La Jolla, CA, pp.1513-1520.

Sebag, M. and Schoenauer, M., (1994) Controlling Crossover Through Inductive Learning, *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, Springer-Verlag, LNVS 866, pp. 209-218.

Sebag, M., Schoenauer M., and Ravise C., (1997a) Inductive Learning of Multation Step-size in Evolutionary Paramter Optimization, *Proceedings of the 6th Annual Conference on Evolutionary Programming*, LNCS 1213, pp. 247-261, Indianapolis.

Sebag, M., Shoenauer, M., and Ravise, C., (1997b) Toward Civilized Evolution: Developing Inhibitions, *Proceedings of the 7th International Conference on Genetic Algorithms*, pp.291-298, 1997.