The LEM3 System for Non-Darwinian Evolutionary Computation and Its Application to Complex Function Optimization

Janusz Wojtusiak
Ryszard S. Michalski

MLI 05-2
P 05-5
October, 2005

THE LEM3 SYSTEM FOR NON-DARWINIAN EVOLUTIONARY COMPUTATION AND ITS APPLICATION TO COMPLEX FUNCTION OPTIMIZATION

Janusz Wojtusiak and Ryszard S. Michalski*

Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA 22030-4444, USA
{jwojt, michalski}@mli.gmu.edu
http://www.mli.gmu.edu

(*) Also Institute of Computer Science
Polish Academy of Sciences

Abstract

LEM3 is the newest implementation of Learnable Evolution Model (LEM), a non-Darwinian evolutionary computation methodology that employs machine learning to guide evolutionary processes. Due to a deep integration of different modes of operation and the use of the advanced machine learning system AQ21, the LEM3 system is a highly efficient and effective implementation of the methodology. LEM3 supports different attribute types for describing individuals in the population, such as nominal, rank, structured, interval and ratio, which makes it applicable to a wide range of practical problems. It also implements very efficient methods for switching between different modes of operation and operators controlling the generation of new individuals. This paper describes the underlying LEM3 algorithm, results from LEM3 testing on selected benchmark function optimization problems (with the number of variables varying from 10 to 1000), and its comparison with EA, a conventional, Darwinian-type evolutionary computation program. In every experiment, without exception, LEM3 outperformed EA in terms of the evolution length (the number of fitness evaluations needed to achieved a desired solution), sometimes very significantly. It also outperformed the previous LEM2 implementation.

Keywords: Function Optimization, Learnable Evolution Model, Machine Learning, Non-Darwinian Evolutionary Computation

Acknowledgments

The authors express their gratitude to all people who contributed to LEM research and this paper. Dr. Kenneth Kaufman reviewed this paper and provided valuable suggestions during the development of LEM3. Jarek Pietrzykowski provided useful comments regarding this paper and discussed with the authors various aspects of LEM3. John Ashley and Scott Mitchell, students in the CSI 777 and CSI 873 classes taught by the second author, provided helpful comments about LEM3 implementation.

The presented research has been conducted in the Machine Learning and Inference Laboratory of George Mason University, whose research activities are supported in part by the National Science Foundation Grants No. IIS 9906858 and IIS 0097476, and in part by the UMBC/LUCITE #32 grant. The findings and opinions expressed here are those of the authors, and do not necessarily reflect those of the above sponsoring organizations.

1 INTRODUCTION

Research on non-Darwinian evolutionary computation is concerned with developing algorithms in which the creation of new individuals in the population is guided, at least partially, by an "intelligent agent," rather than done solely by random or semi-random change operators, such as the mutations and/or crossovers employed in "Darwinian-type" evolutionary methods. Each new generation is selected from the generated individuals by some selection method, as in Darwinian evolution. The Learnable Evolution Model (LEM), described in (Michalski, 2000; Michalski and Zhang, 1999), employs an AQ-type learning program for creating hypotheses for guiding the evolutionary process.

Experiments with initial implementations of LEM have demonstrated that it may very significantly speed up the evolutionary process over Darwinian-style evolutionary algorithms in terms of the *evolution length*, defined as the number of generations (or fitness evaluations) needed to achieve a desired solution. Most importantly, these experiments also have indicated that the LEM advantage grows with the complexity of the problem, as measured by the number of variables to optimize. These findings indicate that LEM may be particularly attractive for solving very complex optimization problems in which the fitness evaluation is time-consuming or costly.

The first implementation of Learnable Evolution Model, LEM1, employed the AQ15 learning program, and included a relatively small subset of the LEM methodology (Michalski and Zhang, 1999). Its experimental testing produced, however, very promising results. A more advanced implementation, LEM2 used the AQ18 learning program and was tested on several benchmark problems (Cervone, Kaufman and Michalski, 2003). Domain-oriented LEM implementations, called ISHED and ISCOD, were tailored to problems of optimizing heat exchanger designs, and also produced highly promising results (Kaufman and Michalski, 2000a; Domanski et al., 2004).

The latest implementation, LEM3, described in this paper, employs the newest implementation of AQ learning, AQ21 (Wojtusiak, 2004a; Wojtusiak, 2004b). In contrast to previous implementations, LEM3 deeply integrates different modes of LEM operation, that is, it integrates them at the level of data structures.

Due to the integration with AQ21 and use of its data structures, LEM3 allows one to describe individuals in a population in terms of a wide range of attribute types, such as nominal, rank, structured, absolute, and ratio. In guiding evolution, it can control the generality of the hypotheses generated at different stages of evolution, and also adapt learning parameters, such as description type, to the problem. LEM3 also implements a new method for executing the end-phase of the evolutionary computation, and adds various extensions to the original LEM methodology, such as new hypothesis instantiation algorithms, and an *Action Profiling Function* (*APF*) that allows switching between modes of execution or applying them in parallel. As in the previous version, continuous attributes are handled by a very efficient method, ANCHOR, that incrementally discretizes them into variable-length intervals appropriate to the progress of evolution (Michalski and Cervone, 2001).

Section 2 describes the LEM3 algorithm, including an overview of specific methods described in detail in later sections. Sections 3 to 8 present descriptions of, respectively, the AQ21 learning program, hypothesis instantiation methods that create new individuals (candidate solutions), action selection, probing mode that applies conventional operators for generating new solutions, the ANCHOR method for adaptively discretizing continuous variables, and the randomization and start-over operators. Section 9 describes the most important implementation issues and the integration of AQ21 within LEM3. An example of LEM3 execution in presented in Section 10. Section 11 describes experimental results from applying LEM3 and, for comparison, a Darwinian-type method, EA, and the previous implementation, LEM2, to selected benchmark function optimization problems. Related research and future plans are presented in Sections 12 and 13, respectively.

2 DESCRIPTION OF THE LEM3 ALGORITHM

This section presents in detail the LEM3 algorithm, the most recent implementation of the Learnable Evolution Model.

The LEM3 algorithm contains several components also found in traditional evolutionary algorithms, such as generation of an initial population, selection of individuals for a new population, and evaluation of individuals. Other LEM3 components are concerned with guiding evolutionary computation through machine learning. This is done by selecting at each step of evolution the highest and lowest performing individuals in the population, the H- group and L-group, respectively, and then employing the AQ21 program to generate a hypothesis that differentiates between the two groups. The hypothesis is then instantiated in various ways to generate new individuals,

The LEM3 algorithm is presented in pseudocode in Figure 1 and in flowchart form in Figure 2.

```
Generate  initial population
Loop until the Stop Condition is satisfied
    Evaluate individuals
    Select parent population
    Select one or more of the following actions:
        Learn and instantiate hypothesis that discriminate high and low performing
                individuals in the parent population (Learning Mode)
        Generate new individuals through Darwinian-type operators (Probing Mode)
        Change the representation of individuals
        Randomize the  population (either partially or via a start-over evolution process)
Compute statistics and display results
End LEM3
```

Figure 1: LEM3 Algorithm in pseudocode.

2.1 Generate Initial Population

The first step of the algorithm generates an initial population of individuals, which LEM3 can do in three ways. It can generate an initial population randomly, create it according to constraints defined by the user, or import it from an external source. The latter capability is especially useful when there is a need to run LEM3 with a preexisting population, e.g., developed by experts in the domain of application. An initial population can be also assembled through a combination of the above methods.

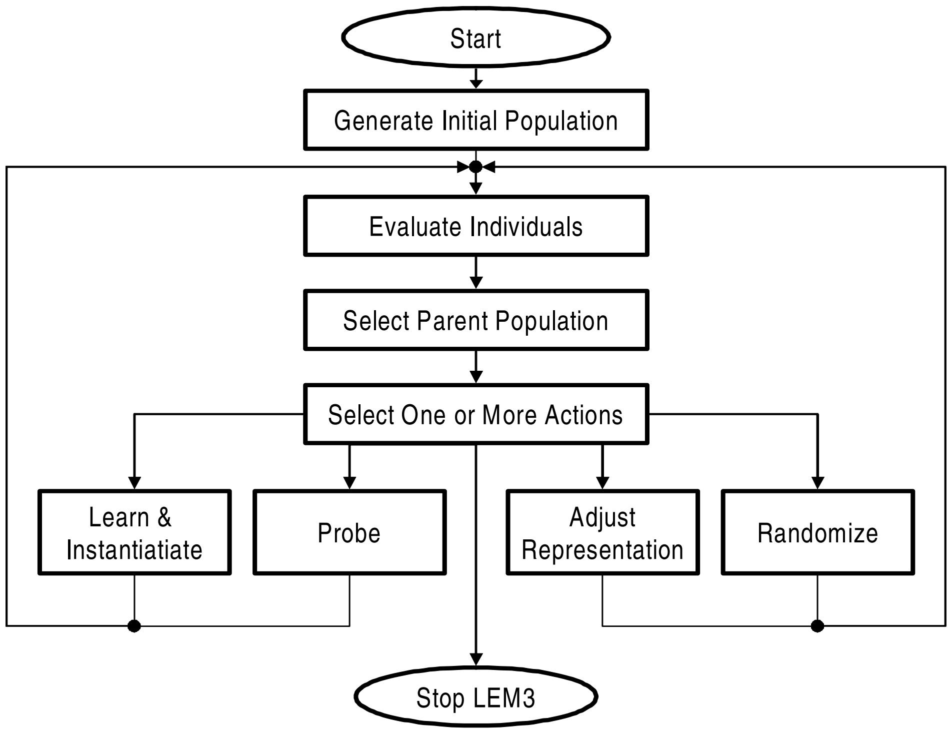


Figure 2: Flowchart of the LEM3 algorithm.

2.2 Evaluate Individuals

This step determines the value of the fitness function for every individual in the population. This may be a simple operation if the fitness is defined by a mathematical formula. In many applications, however, fitness evaluation may require a time consuming or costly process of running a simulator, solving a set of complex equations, or even performing an experiment. For example, such situations occur in the case of designing heat exchangers (Kaufman and Michalski, 2000a; Domanski et al. 2004), optimal non-linear filters (Coletti et al., 1999), aircraft wing shapes (Oyama, 2000), and some applications described in (Bentley and Corne, 2002; Gen and Cheng, 2000).

Due to the use of the AQ21 machine learning program, LEM3 allows a user to define the fitness function in terms not only of numeric attributes, as is usually done in conventional evolutionary algorithms, but also in terms of other attribute types, such as nominal, rank, cyclic (for example days of the week), structured (representing hierarchies; Larson and Michalski, 1977), interval, and ratio (Michalski, 2004; Michalski and Wojtusiak, 2005b). This feature extends LEM3 applicability to domains in which fitness is measured not only by some numerical score, but also by a combination of qualitative and qualitative properties.

2.3 Generate New Population

After a successful instantiation process, new individuals need to be incorporated into the new population. This is done by copying new individuals directly into the new population, or by combining new individuals and individuals from the old population. In general, the creation of new population works in two steps: (1) creation of a (possibly large) temporary population and (2) selection of individuals into the new population.

The temporary population consists of all newly instantiated individuals and also, depending on parameters, individuals from the old population. If the number of instantiated individuals is smaller than the desired population size, the program needs to take some or all individuals from the previous population in order to meet the target population size. It is usually desirable regardless to always add all individuals from the previous population into the temporary group. There is no need to keep track of past populations from more than one iteration, because they were already used in previous steps, and are reflected in the old population.

Selection of individuals into the new population is done using methods known from evolutionary computation. Three methods used in LEM3 are: select the best individuals, also known as rank-based selection; tournament selection; and proportional selection also known as roulette wheel selection described, for example in (Bäck, Fogel and Michalewicz, 2000). As a result of selection the final new population is created.

2.4 Select One or More Actions

LEM3 works in different modes of operation. Each action presented in Figure 2 represents one mode of operation that may consist of a number of different operations (e.g., in learning mode three operations are executed, namely example selection, hypothesis creation, and hypothesis instantiation). The program can select one action (mode of operation), or actions can be executed in parallel (e.g. Learning Mode and Probing Mode). Selection of actions is controlled by the Action Profile Function (APF). A detailed description of methods for selecting actions is presented in Section 5.

2.5 Execute Learning Mode

In learning mode, the most important part of the Learnable Evolution Model, new individuals are created through hypothesis formulation and instantiation. Individual creation is a three step process: (1) selection of example individuals, (2) hypothesis formulation, and (3) hypothesis instantiation.

Based on the fitness of the individuals, step (1) selects high-performing (H-group) and low-performing (L-group) individuals from the population. These groups serve as positive and negative examples, respectively, for the AQ21 learning program.

There are two methods of creating these groups (Michalski, 2000). The first one, Fitness-Based Selection, defines high and low fitness thresholds in the range from the highest to the lowest fitness value observed in the current population. For example, if High and Low Fitness Thresholds are both 25%, then individuals whose fitnesses are in the highest 25% of the range and the lowest 25% of the range are included in the H-group and L-group, respectively

The second method, Population-Based Selection, selects a specified percentage of individuals from the population for each group, regardless of the distribution of fitness values. These percentages are defined by the High Population Threshold (HPT) and Low Population Threshold (LPT). For example, if HPT and LPT are both 30%, then the 30% of the individuals with the highest fitness and the 30% with the lowest fitness are included in the H- and L-group, respectively. This issue is discussed in more detail in (Michalski, 2000).

The central component of the LEM3 algorithm is a learning module (step 2) that induces general descriptions discriminating H-group from L-group individuals. In LEM3, this function is performed by AQ21 whose learning module generates hypotheses in the form of attributional rules (Michalski, 2004). Such a representation of hypotheses is beneficial to LEM because of the rules' high expressive power, the ease of instantiation of learned hypotheses, and the understandability of the hypotheses by experts. Previous LEM implementations employed earlier versions of AQ learning programs; LEM1 used AQ15c, (Wnek et al., 1996) and LEM2 used AQ18 (Kaufman and Michalski, 2000b). AQ21 and its use in LEM3 are described in Section 3 of this paper.

Instantiation (step 3) is a process of generating new individuals that satisfy a given hypothesis. The instantiation process is the main way to create new individuals in Learnable Evolution Model, and correct instantiation of learned hypotheses is therefore crucial for evolution. A detailed description of the methods of instantiating attributional rules used in LEM3 is presented in Section 4.

2.6 Execute Probing Mode

The probing mode executes Darwinian-type operators in order to generate new individuals. The two operators implemented in LEM3 are crossover and mutation, described in detail in Section 6.

2.7 Change Representation

Continuous variables in LEM3 can be automatically discretized using Adaptive Anchoring Discretization (Michalski and Cervone, 2001), or can be represented in their original continuous form. Adaptive anchoring increases the precision of the discretized variables in the most promising areas, neighborhoods of the fittest individuals. Execution of this operator is controlled by the Action Profile Function mentioned above. A description of the discretization method used is presented in Section 7.

2.8 Randomizing

This action adds randomly generated individuals to a population in order to introduce diversity, or replaces the entire population in a start-over process. A description of the randomizing action is presented in Section 8.

3 AQ21 MACHINE LEARNING SYSTEM

In this section we describe AQ21, the machine learning system used in hypothesis generation in LEM3. It also describes how AQ21 is used within the LEM3 algorithm.

Programs from the AQ family learn hypotheses in the form of *Attributional Rules* (Wojtusiak, 2004a, Michalski, 2004, Michalski and Kaufman, 2001). The simplest form of attributional rules is:

$$\text{CONSEQUENT} \leftarrow \text{PREMISE}$$

where consequent and premise are complexes -- conjunctions of *attributional conditions* (a.k.a. *selectors*). Here is an example of such a rule:

```
[part = acceptable] ← [weight = 2..5] &
                      [color = red v yellow] &
                      [height <  3 ]
```

which means that a part is acceptable if its weight is between 2 and 5 (units are presumably defined in the attribute domain), its color is red or yellow, and its height is less than 3.

Hypotheses learned by AQ are in the form of *attributional rulesets*, defined as sets of rules with the same consequent. The main operator used in the AQ algorithm is *extension-against*. Provided with a positive example and a negative example, this operator generates a *partial star* defined as the set of maximal generalizations of the positive example that do not cover the negative example. The intersection of all partial stars from one positive example against the different negative examples is called a *star* -- a set of maximal generalizations of the positive example that do not cover any negative example. In order to prevent an exponential growth to the size of the star, AQ employs a *beam search* that limits the number of rules to be retained from each iteration. The positive example that was used to generate the star is called a *seed*. AQ selects the best rule from a generated star, selects a new seed from the uncovered positive examples, and similarly generates stars until all positive examples are covered. Such an algorithm guarantees that the learned ruleset will be complete and consistent, provided that examples of different classes are always distinct in the representation space. Figure 3 presents the basic AQ algorithm in pseudocode.

```
HYPOTHESIS = <null>
While not all H-group examples are covered by HYPOTHESIS
    Select an uncovered positive example $e^+$ and use it as a seed
    Generate star $G(e^+, L\text{-group})$
    Select the best rule R from the star, and add it to HYPOTHESIS
```

Figure 3: Pseudocode of the AQ algorithm as used in LEM3.

Selection of the best rules is done using a *Lexicographical Evaluation Function* (LEF) that specifies user-defined criteria for rule preference (e.g. Michalski, 2004a; Wojtusiak, 2004a).

In LEM3, the consequent part of rules takes the form [group = H], indicating that the rules are describing high performing individuals (positive examples). The negative examples are in the L-group that consists of lowest performing individuals. The following example shows a rule learned by AQ21 during optimization of the Rosenbrock function of 10 variables.

```
[group=H]    ←    [x0=-0.5..1.5: 28,19] &
                  [x4=-0.5..2.0: 15,16] &
                  [x5=-1.5..1.5: 18,12] &
                  [x8=-0.5..1.5: 30,28] &
                  [x9=-0.5..1.5: 27,22]: p=12,n=0
```

Figure 4: An example of an attributional rule learned by AQ21[1] in LEM3

The rule says that attribute x0 may have a value between -0.5 and 1.5, x4 may have a value between -0.5 and 2.0, and so forth. The pair of numbers after ":" in each condition indicates the positive and negative support (coverage) for this condition. For example, the condition specifying the value of x0 was itself satisfied by 28 of the positive training examples (H-group) and 19 of the negative training examples (L-group.) Attributes x1, x2, x3, x6 and x7 are not included in the rule. The numbers p and n indicate the coverage of the entire rule (12 positive and 0 negative examples).

In some situations, especially at the end of the evolutionary process, the diversity of the individuals may be very low in a population, and the learning module (AQ21) will not be able to learn any rules. Such a situation occurs, for example, when the entire population converges to a set of identical individuals. Usually, this means that the program has found an optimum, either local or global. To see if this is a local or global optimum, LEM3 performs special exploratory steps controlled by the *Action Profile Function*.

4 INSTANTIATION OF LEARNED HYPOTHESES

The learned hypotheses are used to generate new individuals by a process of rule instantiation. Because basic attributional rules are conjunctions of conditions that define ranges (or sets) of attributes values, the instantiation of such rules is a relatively easy process.

When instantiating a rule for a member of the new population, the program faces two problems, what values to assign to attributes that are in the rule, and what values to assign to attributes not present in the rule. The latter case is exemplified by the rule in Figure 4, which does not include attributes x1, x2, x3, x6 and x7.

Three algorithms for instantiating attributional rules are used in LEM3

4.1 Instantiation Algorithm 1

This is the simplest algorithm for generating new individuals by instantiating the attributional rules. It takes all rules from the ruleset learned by AQ21, and for each rule computes the number of individuals to be generated. The total number of individuals that are created can be either constant during the run or vary over time; it is defined by the user as a parameter. The number of individuals to be created can be the same for all rules, or can be computed proportionally according to a measure of the rules' significance that is defined as the sum of the fitness values of the high performing individuals covered by the rule.

[1] AQ21 displays more information describing rules, see (Wojtusiak, 2004a).

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

Figure 5: Instantiation Algorithm 1

For each newly created individual, the program has to assign values for all attributes, both those included in the rule being applied and those not included in the rule. Depending on the attribute type and user defined parameters, different distributions can be used to select random values for the attributes specified in the rule; it can be done uniformly, using normal distribution for numerical attributes with mean equal to the middle of the range and user-defined variance, using maximum distance from negative examples, or using projections of positive and negative examples.

Selection of values of attributes not specified in the rule is a more intricate problem with many potential solutions. One way is to select a random value from the entire attribute domain. This will result in individuals consistent with the rule; however, it is easy to show cases in which this will result in poor performance. For instance let us assume that we are optimizing a function with two attributes x and y. Both attributes are continuous and defined on the range -5 to 5, and the function optimum is at the point (0, 0). Let us further assume that AQ21 has found the rule [x = 0]. The program will then generate individuals with x = 0 in all cases, and with y distributed over the range [-5, 5]. In the next iteration, AQ21 will learn rules containing only the attribute y, since there is no longer any differentiation among the x-values. During the instantiation phase, the program will assign values of the attribute x randomly, which means that the information from the previous iteration is lost. Thus, the rules may not converge to the solution.

Another method of value selection is to select the value from a randomly selected existing individual. The individual can be selected from the entire population, the H-group, or non-L-group individuals. Experiments have shown that when selecting values from the H-group, the program tends to lose diversity of individuals, and may converge very quickly to a point that is not the target solution. The method that LEM3 uses by default selects individuals from the whole population probabilistically in proportion to their fitness levels.

The presented instantiation algorithm has a very severe weakness. It does not work well for multimodal functions. Events selected to provide values of attributes that are not specified in a rule may be located in different parts of the event space, thus causing new individuals to be generated in the wrong parts of the space. As an extension to the algorithm, a constraint may be added to ensure that generated events must match the rule that is instantiated, but this also may not keep the evolution process from straying in all cases. The second instantiation algorithm was designed to deal with the described problems.

4.2 Hypothesis Instantiation Algorithm 2

This algorithm selects parent individuals and then modifies them according to learned rules. As mentioned before, the second instantiation algorithm is designed to deal with problems that appear using the first instantiation algorithm. The difference between this and the previous algorithm is in the way an individual from the old population is selected. This algorithm is especially good for multimodal functions.

```
Compute the fitness of all individuals (see below)
Loop while more individuals remain to be created
    Select probabilistically an individual (parent) based on its fitness
    Create a list of rules satisfied by the selected individual
    Select a rule from this list probabilistically in proportion to its significance
    Create a new individual:
        For all attributes
            If the attribute is specified in the selected rule
                Select a random value satisfying the rule
            Else Select the value from the parent
```

Figure 6: Instantiation Algorithm 2

The typical quality measure of individuals is simply their fitness. This is not, however, the only possibility; for example, a quality measure may take also into consideration the number of rules that match the individual. When AQ21 is working in *Theory Formation* (TF) mode (i.e., learning covers that are complete and consistent with respect to the training data), it is guaranteed that all individuals from the H-group will satisfy some rules. Although LEM uses by default the TF mode, it is also possible to learn rules in *Pattern Discovery* (PD) mode, in which stronger patterns may be favored over a complete and consistent ruleset. In such a case, some individuals from the H-group may not be covered completely by any rules, and it is thereby important to select only individuals that are covered by at least one rule. Values of attributes that are specified in the rule are selected according to Algorithm 1.

4.3 Hypothesis Instantiation Algorithm 3

Regardless of the learning mode, AQ21 guarantees that each rule will cover at least one high-performing individual. Using that information, it is possible to combine the two algorithms presented above.

```
For all rules
    Compute the number of individuals to be created
    For all individuals to be created
        Select probabilistically an individual covered by the rule
        For all selectors in the rule
            Modify value of the individual within the selector
```

Figure 7: Instantiation Algorithm 3

The third instantiation algorithm, similarly to Algorithm 1, computes the number of individuals to be instantiated for each rule. A significant difference from the first algorithm is that the program does not select random values according to the rules, but instead modifies an existing individual that is covered by the rule. This guarantees that all rules will be instantiated, and multimodal fitness functions will be treated appropriately

4.4 Instantiation of Alternative Hypotheses

The AQ21 program has the unique feature that it can learn not only one ruleset per class, but also a number of alternative descriptions/rulesets for the same class (Michalski 2004b; Wojtusiak, 2004a). In LEM, when the number of attributes may be much larger than the number of examples, it is highly probable that the program can generalize the positive examples in different ways. LEM3 handles alternative covers learned by AQ21 in two ways: (1) the intersection of the covers can be instantiated, or (2) the union of the covers can be instantiated. Once the program computes either the intersection or union, one of the three algorithms described above is used to instantiate.

By using the intersection method (1), we instantiate in an area covered by at least one rule from each alternative ruleset. Suppose that RS1, RS2, …, RSn are alternative rulesets describing the high performing individuals. Ruleset RS1 consists of k1 rules: RS1 = { $R_{1,1}$, $R_{1,2}$, … $R_{1,k1}$ }, ruleset RS2 consists of k2 rules RS2 = { $R_{2,1}$, $R_{2,2}$, … $R_{2,k2}$ }, and so on. A ruleset is defined as a disjunction or rules that are conjunctions of selectors, so the intersection of rulesets is equivalent to a conjunction of rulesets and is given by the following formula:

$$\bigwedge_{i=1..N} RS_i = \bigwedge_{i=1..N} (R_{i,1} \vee R_{i,2} \vee \ldots \vee R_{i,ki})$$

Using De Morgan's and absorption laws the intersection RS can be easily computed. In fact, computation of such an intersection is one of the most common operations in the AQ algorithm applied during the star generation phase. In LEM3 we use this feature of AQ21 to compute the intersection. Intersection of alternative rulesets for a given class is also a ruleset. Moreover if AQ21 works in TF mode and all rulesets are complete and consistent, the intersection is also a complete and consistent ruleset. To prove this, it is sufficient to mention two facts: by our assumption, none of the rules in the rulesets cover any negative examples, so their intersection cannot cover any such examples; and each positive example is covered by at least one rule from in each ruleset. , so they will be covered in the intersection. Let E be a positive event that is covered by rules $R_{1,m1}$, $R_{2,m2}$, … $R_{n,mn}$. It follows that $\bigwedge_{1..n} R_{i,mi}$ covers the example *E*. Instantiation of the intersection of alternative rulesets speeds up the evolution process by limiting the area covered by learned rules. It is however dangerous, since intersected rulesets may be too specialized and the program may converge to a point that is not necessarily an optimal solution.

The second method is to take the union of alternative rulesets. The union is defined using the following formula:

$$\bigvee_{i=1..N} RS_i = \bigvee_{i=1..N} (R_{i,1} \vee R_{i,2} \vee \ldots \vee R_{i,ki})$$

In this case computation of RS is trivial and requires only the use of absorption laws in order to remove unnecessary rules. Similarly to the case of intersection, it can be proven that the union of complete and consistent covers is also a complete and consistent cover.

Unlike intersection, the union expands the area in which new individuals are instantiated. This slows down the evolution process, but increases the chance that the target solution is covered.

5 ACTION SELECTION

Execution of different modes of operation is a unique feature of LEM3 that distinguishes it from other implementations of the Learnable Evolution Model and from many other evolutionary computation methods. As mentioned above, the two basic modes that guide the evolution process are learning mode, which uses hypothesis creation and instantiation, and probing mode, which employs Darwinian-type operators such as mutation and crossover. In addition to the two modes of operation, LEM3 employs additional actions such as adjusting discretization and randomizing (start-over) to help the evolution process.

An important question to be answered is how to switch between the modes of operation and when each action should be executed. The two modes of operation can be executed in parallel, or the program can switch between them as defined in *duoLEM*. Other operations such as changing representation need to be executed separately. To control the application of different actions, LEM3 defines an *Action Profiling Function* (*APC*) that based on the performance of different types of operators decides which operators to apply in the next step. It also decides how many new individuals to create by each mode. For example if the total number of new individuals to be created is 100, the APC may decide to generate 70 by learning mode, 25 by probing mode and 5 by randomizing. The APC should adapt during the evolution process to reflect which operators are the most relevant for the optimization problem. Assigning individuals to learning and probing mode can be done by two simple rules:

```
If average-learning-fitness >> average-probing-fitness then
    Increase number of individuals in learning mode
If average-learning-fitness << average-probing-fitness then
    Increase number of individuals in probing mode
```

where averages are computed over individuals created in one or more iterations of the respective modes.

During the evolution process, it may happen that over a number of iterations, the program makes no progress in terms of value of the fitness function. This situation can be identified through the use of two program parameters, *learn-probe* and *learn-threshold*. *Learn-probe* defines the maximum number of iterations that are performed even if there is unsatisfactory progress, as defined by *learn-threshold*, the minimal acceptable increase of fitness of the best individual. In such a situation, we say that the no-progress condition is met, and several possible operations need to be applied.

If the no-progress condition is met, operations *mutation*, *adjust discretization*, and/or *start-over* are invoked. LEM3 tries to apply *mutation* for *mutation-probe* iterations. If there is still no progress, the program then tries to adjust discretization for *discretization-probe* iterations. If there is still no progress, LEM3 tries to run the start-over operation for up to *start-over-probe* times.

```
If no progress
    Increment learn-probe-counter
        If learn-probe-counter >= learn-probe
            Learn-probe-counter = 0
            If mutation-probe-counter < mutation-probe
                Increment mutation-probe-counter
                Mutate individuals
                Evaluate modified individuals
            Else if discretization-probe-counter < discretization-probe
                Increase discretization-probe-counter
                Mutation-probe-counter = 0
                Adjust discretization
                Mutate individuals
                Evaluate modified individuals
            Else if start-over-probe-counter < start-over-Probe
                Increment start-over-probe-counter
                Discretization-probe-counter = 0
                Mutation-probe-counter = 0
                Rollback discretization
                Add the best individuals to a list of local optima
                Start-over
                Evaluate individuals
            Else
                Stop LEM3
```

Figure 8: No-progress condition pseudocode

The order of *mutation*, *adjust discretization*, and *start-over* operations is not accidental. Mutation is performed in order to introduce diversity into a population, and to be sure that the program did not get stuck “close” to the optimal value (this could be local or global optimum). It is usually the case that AQ21 is unable to learn hypotheses because of a lack of diverse examples. The next step increases the precision of the search by adjusting discretization. If the change of precision does not make any difference, it may mean that the program has found an optimum. However the optimum may be local and it may be desirable to start over with a new random population, and to explore different parts of the space. All three operations are explained in detail in next Sections.

6 PROBING MODE

In *Probing Mode* LEM3 executes Darwinian-type operators to create new individuals. It can be executed in parallel with *learning mode*, or LEM3 can switch between the two as controlled by the *Action Profile Function* (*APF*) described earlier. The APF decides not only when probing mode is to be executed, but also which Darwinian-type operators are to be invoked. In addition to two operators, crossover and mutation, described below, the APF allows the user to define specialized operators based on expert knowledge in the optimization domain.

6.1 Crossover operator

LEM3 implements a single point crossover operator that selects two parent individuals and creates a new individual using the parents' values of attributes. The algorithm is executed in three steps:

1. *Select parent individuals*
2. *Define a split point*
2. *Create a new individual*

Selection of parents is done probabilistically, proportionally to their fitness. The split point defines how many variables' values will be taken from the first parent, and how many from the second one. For example if there are 100 variables, and the split point is 0.3, the first 30 values will be assigned from the first parent, and the last 70 will be assigned from the second parent. The split point is generated randomly with uniform distribution.

A newly created individual thus has assigned values from both parents: the first part from the first parent and second part from the second parent.

6.2 Mutation Operator

In LEM3 the primary goal of the mutation operation is not to find better individuals, but rather to introduce diversity into a population. It may, however, produce better individuals in terms of fitness function value, but the direct goal is to distribute individuals in such a way that the AQ21 program will have enough examples to generate hypotheses by generalization of the examples. It is usually the case that individuals generated through mutation are included in the L-group of low performing individuals. Such individuals provide constraints for learning, and allow the AQ21 program to generalize. It is especially important when the program has already converged to one value of an attribute (that may or may not be correct), so the attribute will not be included in any rule. Moreover, during the instantiation process for attributes not included in rules, LEM3 uses values from randomly selected existing individuals, implying that without introducing diversity LEM3 will not make any progress for the attribute.

The mutation methodology in LEM3 is more complex than in standard evolutionary algorithms or genetic algorithms, due to the need to deal with the different types of attributes that are implemented in LEM3 and AQ21, namely *nominal, rank*, *discretized continuous*, *cyclic, structured* , *ratio,* and *absolute* (Wojtusiak, 2004a; Michalski, 2004).

The LEM3 program can also use information about the population or evolution history to mutate individuals in the "right" direction. However, in practice it usually happens that mutation is invoked when the population has converged to one point and no such information can be taken from the current population. One possibility is to assign higher probability to values that are not present, or are present in smaller numbers of individuals. This method can be justified because it provides more diversity to the population. However, it can be assumed that the population has converged to a point that is close to the optimum, and thus only its close neighborhood should be explored via mutation. Below we provide a description of the mutation operation for different types of attributes:

The domain of a *nominal* attribute is an unordered set, so the concept of distance between its values does not apply (we can assume that a distance between any two values is the same). A reasonable way of changing attributes of this type is to assign a value randomly.

An *absolute* attribute domain is a totally ordered set, meaning that it is possible to create a distribution based on how far a value is from the actual attribute value. The distance between values in the domain is based on the count of values between them plus one. For example in domain “length” having values {very short, short, medium, long, very long} the distance between values “short” and “very long” is three. Mutation can be done using the normal distribution with the additional assumption that the distance between successive values is equal. For experimental purposes, one may also want to use a uniform distribution.

A *ratio* type attribute domain is a special case of continuous linear domain, for example, real numbers. In LEM3, values of this type are stored with double precision. Values of a continuous attribute can be changed in a number of ways. They can be changed uniformly over the entire domain range, or in a range specified as a parameter. It is also natural to use the normal distribution with mean equal to the value taken from the parent individual.

A *discretized continuous* attribute domain type is a special case of the *ratio* domain [6]. Real values are discretized to predefined points (representing specific intervals). The discretized intervals are not necessarily equally distributed, especially when an adaptive discretization method such as *ANCHOR* (Michalski and Cervone, 2001) or ChiMerge (Kerber, 1992) is used (the default discretization method in LEM3 is ANCHOR). Mutation of the discretized continuous attributes can be done as with either continuous attributes or absolute attributes. In the first method, the middle of an interval is changed according to a selected distribution, as a continuous value, and discretized back to one of the intervals. It is important to ensure that the selected distribution will go beyond the current interval; otherwise no change will be ever made to the attribute. The second method uses the ordering of intervals, 1, 2, 3, … k, to apply mutation as for absolute attributes. In such a case, information about length of the intervals is ignored.

A *cyclic* attribute domain can be treated in the same way as an absolute domain; it represents a linear domain whose last value wraps around to its first (e.g., days of the week). The only computational difference between cyclic and linear is that “distance” between two points should be computed in both ways -- counting up from the first point to the second, and from the second to the first, and the minimum should be used. Similarly to linear domains, the distance is computed by counting elements in the list of possible values.

A *structured* attribute domain represents a generalization hierarchy. The mutation can be done in two ways, as in *nominal* domains, and using information about the hierarchy. It seems reasonable to take advantage of the structure that is defined in the domain. To compute distances between nodes, a simple distance in the tree can be used, defined as the number of edges that need to be followed from one node to another. Note that in true trees, there is only one possible path between two nodes, so such a measure is well defined.

Two important issues governing the mutation process are which attributes should be changed during the mutation and what parameters of mutation should be used. To solve the first problem, LEM3 uses two methods: randomly selecting attributes to be changed; and selecting attributes that are currently not specified in the learned rules. It is also possible to use a combination of the methods. The program can create a list of attributes that are not included in the current ruleset, and if the list contains fewer attributes than a given threshold, the first method is used.

In the LEM3 implementation, each attribute has its own mutation parameters. For many applications, attributes are from different domains, and must have separate mutation parameters. This is also desirable when adaptive methods are used, for example, in methods that adapt mutation ratio during the evolution process. Similar techniques are known and used in the field self-adaptive genetic algorithms.

7 ADJUSTING REPRESENTATION SPACE

In general, adjusting the representation space of solutions may include ignoring irrelevant variables, adjusting domains of variables, and creation of new variables that are more relevant to the optimization problem. Although all three types of operations are being investigated in the LEM methodology, LEM3 currently implements only adjusting discretization of variables described in paragraphs below.

After the mutation operation, “adjust discretization” is the second operation comprising the no-progress condition in LEM3. The program uses adaptive discretization to change the precision of the attributes when it is required. The method that is used by default is a modified version of *Adaptive Anchoring Discretization* (Michalski and Cervone, 2001), which discretizes the continuous variable with a granularity size adapted to the problem. It is also possible to employ other methods of discretization in LEM3. For example, one could use ChiMerge (Kerber, 1992). This method, however, is based on frequencies of values, and is not designed for use in evolutionary computation.

The modified ANCHOR method starts with an initial discretization of the space that is denser in areas closer to zero and less dense farther from zero. For example an attribute whose domain is a range [-2000, 300] will be discretized into intervals whose boundaries are: {-2000, -1000, -900, ..., -100, -90, ..., -10, -9, … -1, 0, 1, ..., 9, 10, 20, …, 90, 100, 200, 300}.

The modified ANCHOR method increases precision in the neighborhood of the best individuals in the current population. The increase is done for all attributes discretized by the method. In many cases when the program reaches the no-progress condition, there is only one such point representing a local optimum to which the program converged. For multi-modal functions for which LEM3 seeks simultaneously a number of solutions or, more generally, when more than one best individual is selected, precision is increased in the neighborhood of all local optima. Let us suppose that for the attribute from the previous example, LEM3 decides to increase precision in the neighborhood of the value 100. As a result the following will be boundaries of intervals in the new discretized domain: {-2000, -1000, -900, ..., -100, -90, ..., -10, -9, … -1, 0, 1, ..., 9, 10, 20, …80, *90, 91, …, 99, 100, 110, …, 190, 200,* 300}, where the new intervals are shown in *italics*.

After adjusting the discretization, it is necessary to generate randomly new individuals within the area in which the discretization was improved. This will give the AQ program examples distributed over the new discretization space after applying the mutation operator to individuals that were the basis for the adjust discretization operation.

8 RANDOMIZATION AND START-OVER

The Randomize operation can either add new (usually few) randomly generated individuals to the current population in order to increase diversity, or restart the evolution process by applying the *start-over* operation.

LEM3 without the *start-over* operation can be viewed as a local optimization algorithm that cannot get out from local optima if it already converged (unless mutation is applied with the uniform distribution).

The *start-over* operation generates a new population to restart the evolution process from the beginning. This operation is designed for when the program finds a local optimum and needs to search other parts of the optimization space. Such a situation is detected by the no-progress condition given that neither mutation, which locally modifies the best individuals, nor adjust discretization, which increases search precision, improves global solution. New individuals are created (1) randomly in the entire space, (2) randomly in the parts of space that were not explored so far, and (3) randomly by maximizing distance from the local optima found so far. The first method is the simplest one, and is equivalent to a full restart of the LEM3 algorithm. The second method requires keeping track of all values of attributes that appeared in past individuals. The information is used to distribute individuals over the not explored yet parts of the space. The third method builds distributions based on a list of local optima found so far. The farther from a found solution, the higher the probability that an example is used.

9 THE LEM3 IMPLEMENTATION

The basis for LEM3 implementation is the learning module of the AQ21 program. Our goal was to fully integrate AQ21's learning module with the evolution module in LEM. The full integration not only reduces learning time, but also provides access to all functions implemented in AQ21, and reuses its data structures. The integration is based on the simple fact that an individual can be represented in the same way as an event in AQ learning. Moreover, these individuals are used as examples when LEM applies machine learning to generate hypotheses that describe high-performing individuals.

The following sections describe the main AQ21 data structures that are shared with LEM3 to store individuals and learned hypotheses. Details of attribute types in AQ learning and their representation are presented in (Michalski and Wojtusiak, 2005b).

9.1 Representation of Individuals in LEM3

It was mentioned above that LEM3 uses AQ21 data structures to store individuals. This can be done because an individual is nothing else than list of values of attributes that describe a point in an optimization space. The point can represent a design, an argument of a multidimensional function, a set of parameters, or any object in the optimization space. Moreover individuals are used as examples for learning when determining hypotheses that distinguish between high and low performing individuals. In Learnable Evolution Model, the optimization space and event space for learning are the same. In the following paragraphs we briefly present the representation of events in AQ learning that is also used to represent individuals in LEM.

To represent an event, it is necessary to store both discrete and continuous attributes. The discrete attributes are stored in bitstrings, where each value of the discrete domain is represented by exactly one bit. The concatenation of bits for all domains represents the discrete part of the event. Continuous attributes are represented directly as numeric values. A vector of real values is used to represent all numeric attributes.

Because LEM3 uses Adaptive Anchoring Discretization, attributes to which the discretization is applied are stored using bits even if they were originally continuous. LEM3 can represent numeric attributes both ways – discretized and not discretized.

9.2 Representation of learned hypotheses in LEM3

AQ's learning module outputs learned hypotheses in the form of attributional rulesets. A ruleset, a collection of rules with the same consequent, is represented using an ordered set of rules. Each rule is represented using data structures similar to these used to represent individuals (events in AQ learning) – vectors of bits and vectors of real values. For discrete attributes, each value from the domain of the attribute is assigned exactly one bit that is set to 1 when the value is present and 0 when it is not. An extra bit is added to each attribute, and is used to help represent meta-values (Michalski and Wojtusiak, 2005a), although LEM3 does not use this feature of AQ21. For continuous values, each attribute is assigned two real values – for lower and upper bounds. Thus, an individual can be viewed as a special case of a rule condition in which exactly one value per attribute is present. The following example demonstrates the representation used in LEM3 and AQ21.

Let x1, x2 and x3 be attributes that define the optimization space. Attributes x1 and x2 are of nominal type with domains {r, g, b, w} and attribute x3 is a continuous (ratio) attribute. Suppose that [group = high] $\leftarrow$ [x1=r,g] & [x2=b] &[x3=5..7] is a rule learned during the optimization process. The consequent part of the rule is represented by a complex (conjunction of attributional conditions) that has all three attributes active, two of them represented by bitstrings (x1 and x2) and one represented by a range (x3). The following is a representation of the consequent of the presented rule:

Bits: 1100000100   Ranges: (5, 7)

The domain size of both nominal attributes is 4, so 5 bits are used to represent each of the attributes (an extra bit is used to represent meta-values, as mentioned above). Bits 1 to 5 are used to store values for the first selector (attribute x1) and bits 6 to 10 are used to store values for the second selector (attribute x2). There is only one continuous attribute (x3); therefore only one range is used.

It should be mentioned that even if not all attributes are used in a rule, bits or ranges are assigned to all attributes. This means that the size of a complex in a given representation space is constant, even if it has only one selector. Inactive discrete selectors have all value bits switched to 1 and inactive continuous selectors are represented by special ranges equivalent to (-∞, +∞).

10 AN APPLICATION OF LEM3 AND EA TO A SIMPLE PROBLEM

This section presents an example of applying LEM3 and a conventional, Darwinian-type algorithm, *Evolutionary Algorithm* (EA) to a simple function optimization problem. The problem is simple enough to be illustrated graphically using *Generalized Logic Diagrams* (e.g. Michalski, 1978) but sufficiently complex to show some important aspects of the LEM3 algorithm. The optimization problem is to find all global maxima of the sample function:

$$f(x_0, x_1, x_2, x_3) = 16 - x_0^2 - 8 * \cos(2 * x_1) - x_2^2 - x_3^2$$

Domains of all attributes are ranges [-2, 2]. The cosine part was added to the function in order to make two equal global optimal solutions to the problem. A two dimensional version of the function, given by the formula $f(x_0, x_1) = 8 - x_0^2 - 4 * \cos(2 * x_1)$ is illustrated in Figure 9. The factors 4*n and 2*n (values 16, 8 and 8, 4 in expressions above), where n is the number of attributes, are used the function for scalability and to guarantee that its value is not negative.

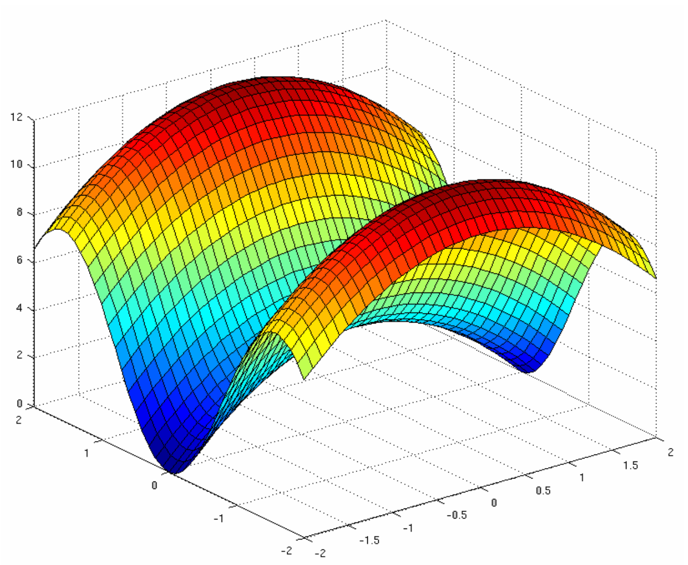
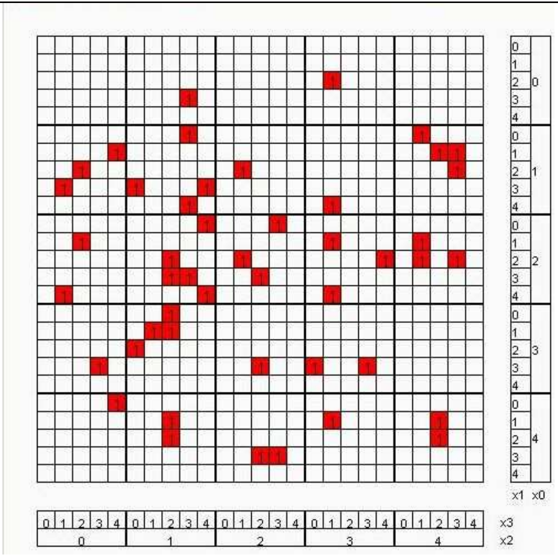
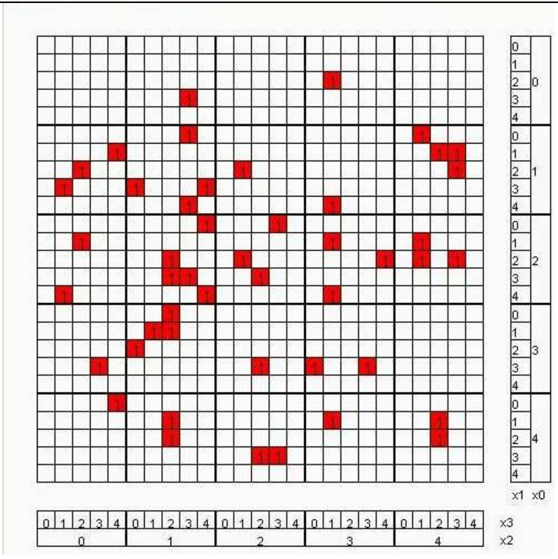
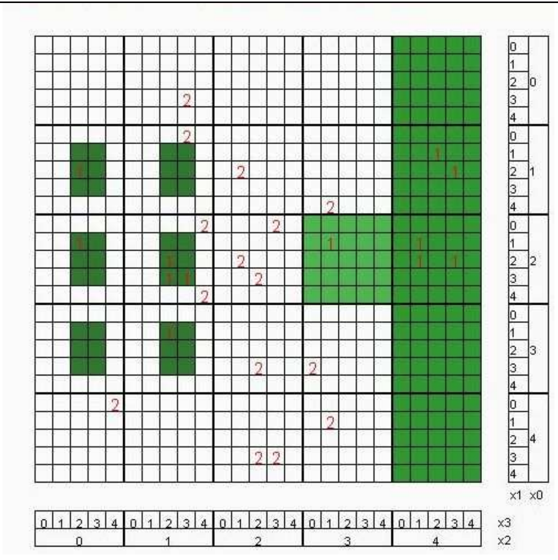


Figure 9: A plot of the function f(x_0, x_1).

The following figures demonstrate consecutive steps of executing LEM3. The figures are Generalized Logic Diagrams (GLDs) representing four dimensional spaces spanned over discretized variables x0,…,x4. For each generation, two diagrams are presented for LEM3, one with the current population and one with selected solutions in the H-group and L-group) and the learned rules. For comparison, steps of executing EA for this problem are also illustrated.

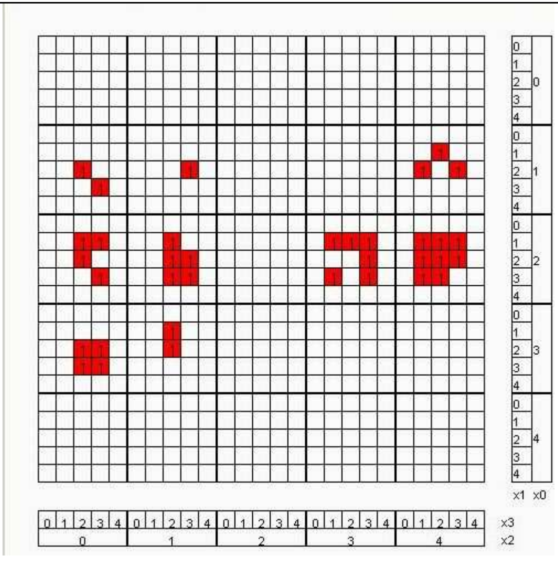
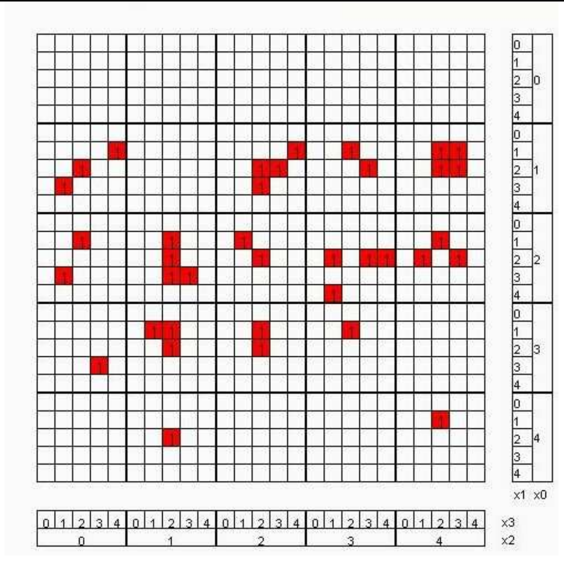
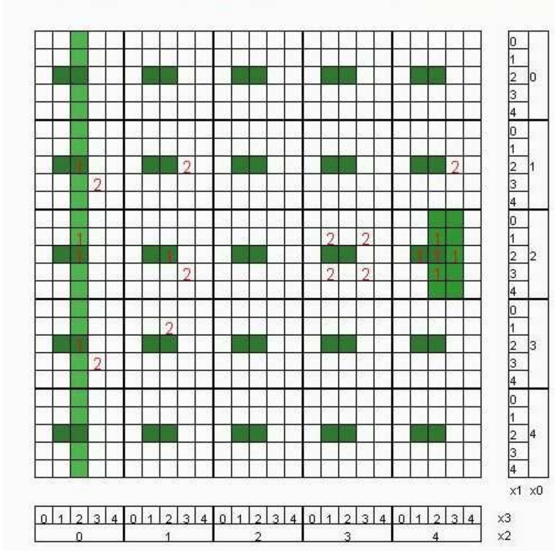
Generation 1:

In the first iteration, the population is randomly initialized over the entire space as shown on Figure 10. LEM3 selects examples for the H- and L-groups indicated as respectively “1” and “2” on the same figure. Provided with the examples AQ21 learns rules presented as shaded areas in Figure 12. Initial populations in LEM3 and EA are the same.

LEM3	**EA**
 **Figure 10: Randomly generated initial population (the same for both programs).**	 **Figure 11: Randomly generated initial population (the same for both programs).**
 **Figure 12: Learned hypothesis and H- and L-group individuals in LEM3.**	EA applies mutation and crossover to generate new individuals. EA probability of mutation is *0.1* EA probability of crossover is *0.1* EA selection method is *tournament.*

Generation 2:

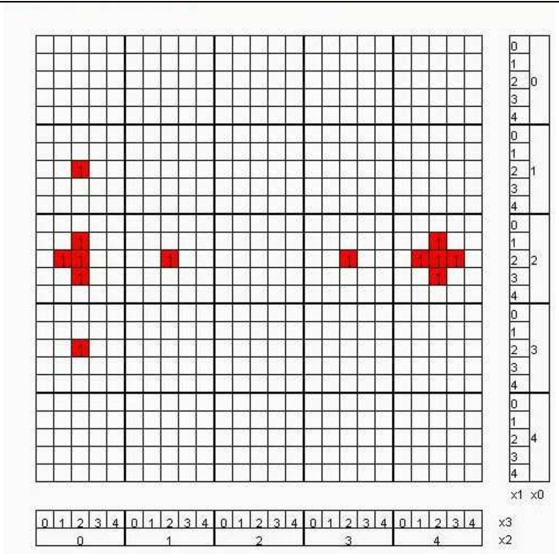
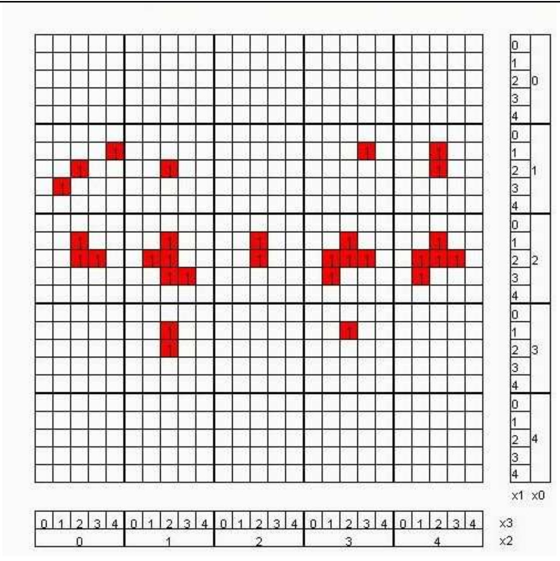
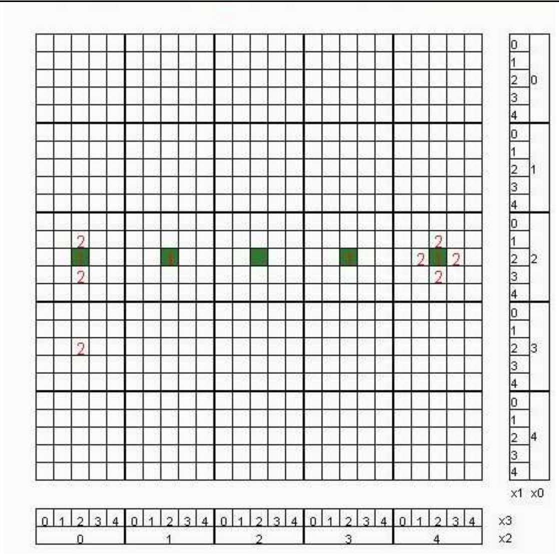
Instantiated individuals are combined with the old individuals and a new population is selected (Figure 13). In the instantiated individuals both solutions of the function have already been found – individuals (2, 2, 0, 2) and (2, 2, 4, 2), which represent the values of real solutions (0, 0, -2, 0) and (0, 0, 2, 0). Although the program has found the solutions, it still needs for all individuals to converge to the solutions in order to satisfy the LEM3 stop condition (the only individuals are solutions). Figure 15 shows high and low performing individuals selected from population and learned rules (shaded areas). Evolutionary algorithm slowly converges toward the solutions as shown in Figure 14.

LEM3	EA
 **Figure 13: LEM3 Population in the second generation (100 fitness evaluations).**	 **Figure 14: EA Population in the second generation (85 fitness evaluations).**
 **Figure 15: Learned hypothesis and H- and L-group individuals in LEM3.**	EA applies mutation and crossover to generate new individuals. EA probability of mutation is *0.1* EA probability of crossover is *0.1* EA selection method is *tournament.*

Generation 3:

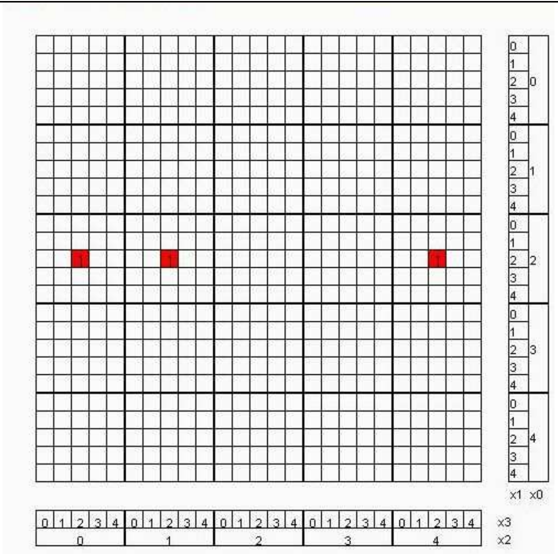
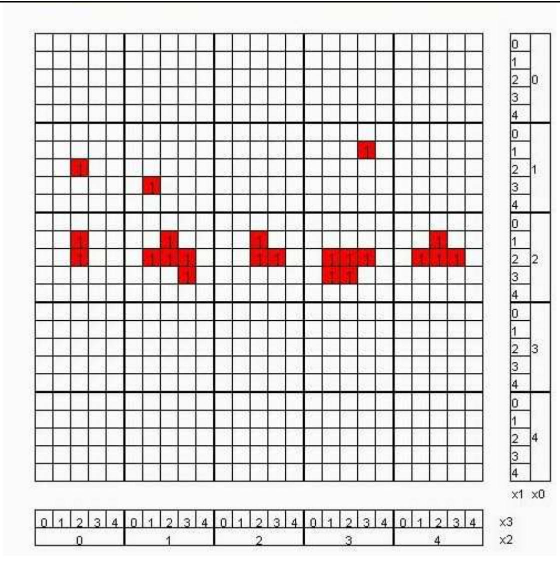
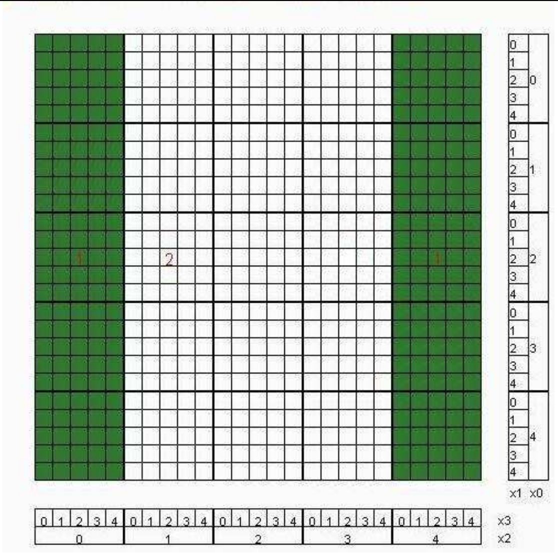
In the third generation, LEM3 individuals converge closer to the solutions as shown on Figure 16. After selecting L- and H-groups from the population, one rule is learned as shown on Figure 18.

EA also found the solutions, but the population is much more distributed over the space (Figure 17).

LEM3	EA
 **Figure 16: LEM3 Population in the third generation (150 fitness evaluations).**	 **Figure 17: EA Population in the third generation (130 fitness evaluations).**
 **Figure 18: Learned hypothesis and H- and L-group individuals in LEM3.**	EA applies mutation and crossover to generate new individuals. EA probability of mutation is *0.1* EA probability of crossover is *0.1* EA selection method is *tournament.*

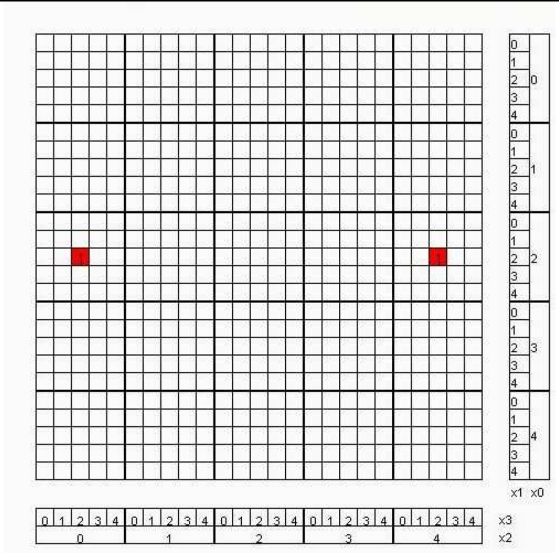
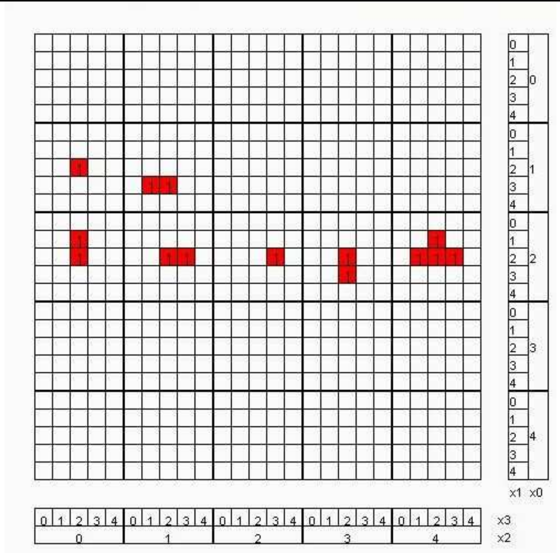
Generation 4:

The fourth generation in LEM3 consists of individuals that converged to the two solutions and one individual that is not a solution (Figure 19). The only individual that is not a solution is used as the L-group, and all other individuals are included in the H-group. One simple rule describes the H-group against the L-group (Figure 21). EA slowly converges towards solutions (Figure 20).

LEM3	EA
 **Figure 19: LEM3 population in the fourth generation (200 fitness evaluations).**	 **Figure 20: EA Population in the fourth generation (168 fitness evaluations).**
 **Figure 21: Learned hypothesis and H- and L-group individuals in LEM3.**	EA applies mutation and crossover to generate new individuals. EA probability of mutation is *0.1* EA probability of crossover is *0.1* EA selection method is *tournament.*

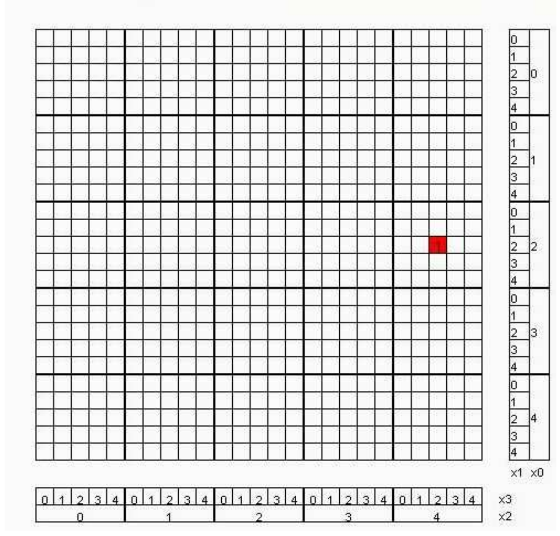
Generation 5:

Finally, in the fifth generation (250 fitness evaluations) all individuals generated by LEM3 converged to the two solutions (Figure 22). This ends the evolution process in LEM3. EA did not converge to the solutions yet (Figure 23).

LEM3	EA
 **Figure 22: Two global optima found by LEM3 in the last, fifth generation (250 fitness evaluations).**	 **Figure 23: EA population in the fifth generation (214 fitness evaluations).**

Generation 8:

In the eighth generation, EA converged to one of the two solutions shown in Figure 24. The total number of fitness function evaluations needed by LEM3 was 250, and by EA was 346. Also, LEM3 found both optima, and EA only one of the two.

LEM3	EA
LEM3 already converged to both optimal solutions after fifth generation, as shown in Figure 22.	 **Figure 24: One of the two optima found by EA in the eighth generation (346 fitness evaluations).**

While the advantage of LEM3 over EA in solving this simple problem (only 4 variables describe the individuals) is relatively large (EA needs about 100 more fitness evaluations than LEM3), a more impressive advantage of LEM3 is seen in problems with larger number of variables, where LEM3 is on average over 16 times faster than EA (see Section 11 for details). This is so because LEM3's advantage grows with the number of variables.

In real-world and more complex applications LEM3 will start applying methods described in the no-progress condition to ensure that program is not stuck close to the solutions, the discretization level is correct, and there are no other solutions in unexplored areas of the optimization space. The presented simple example was chosen for demonstration of LEM3 and its comparison with EA because of ease of the visualization that because of the complexity of GLDs can grow exponentially worse with a larger number of attributes.

11 COMPARATIVE RESULTS FROM APPLYING LEM3, EA AND LEM2

The goal of these experiments was to compare the performance of LEM3, LEM2, and EA, representing a conventional, Darwinian type evolutionary computation algorithm on a range of function optimization problems. The problems involved optimization of the *Rastrigin*, *Griewangk*, and *Rosenbrock* functions of different numbers of variables, ranging from 10 to 1000. These functions were chosen because they are often used for testing evolutionary algorithms and are described, for example, in (Whitley et al., 1996). EA is an implementation of Evolutionary Algorithm taken from library *EO* (*Evolutionary Objects*) *0.9.3a* that can be downloaded from URL: http://eodev.sourceforge.net. The EO library was selected because it is an advanced implementation of Darwinian-type evolutionary algorithm, it supports large numbers of variables (we tried other programs that support far fewer variables), it is well described in available tutorials, and it is easily downloadable from the internet.

LEM3 and EA were applied to optimizing functions of 10, 100, 200, 300, …, 1000 variables. LEM2 was applied to optimizing functions of 10 and 100 variables (its limit is fewer than 200 variables). Each experiment involving optimizing a function of a given number of variables was repeated 10 times with a different starting population. To make a fair comparison, the same starting population was used in each program.

The results are reported for *δ-close solutions* that are characterized by a *normalized distance from the optimal solution*. The δ-close solution, *s*, is a solution for which function *δ(s)*, defined as:

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

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

This definition of δ-close solution suggests into two possible ways of analyzing performance of evolutionary computation methods. First, one may consider the problem of how many fitness function evaluations are needed to achieve a given δ=k by the best individual in the population, denoted as *FE(δ=k),* where k is a number between 0 and 1. The measure is the main way of reporting results in this paper. Secondly, one may consider the problem of finding *δ(v)* after given number of fitness evaluations, where *v* is the fitness value of the best individual after a given number of fitness function evaluations. Figure 25 illustrates concept of the δ-close solution.

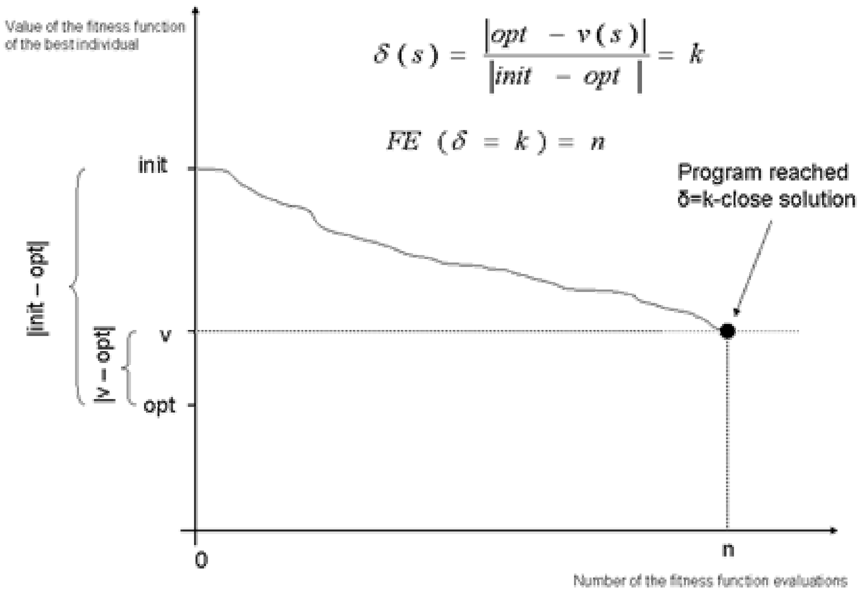

Value of the fitness function of the best individual
$\delta(s) = \frac{|opt - v(s)|}{|init - opt|} = k$
$FE(\delta = k) = n$
init
Program reached δ=k-close solution
|init – opt|
|v – opt|
v
opt
0
n
Number of the fitness function evaluations

Figure 25: Illustration of a δ-close solution.

For example if the fitness value of the best individual in the initial population is 100 and during the process of minimization the program achieved value 0.1, and the optimal value is 0 then δ=0.001, indicating that program found a solution within 0.1% distance from the optimal solution, normalized by the fitness value of the best individual in the initial population.

In the presented experiments, LEM3 was executed using all default parameters. Most importantly, high and low population thresholds were both *0.3*, learning probe was *4*, mutation probe was *10*, discretization probe was *2*, discretization method was *ANCHOR*, and survival selection was *rank-based*.

In the presented experiments, LEM2 was executed with the following parameters: high and low thresholds were both *0.3*, discretization method was *ANCHOR*, and mode was *DUOLEM* (utilize both learning and probing modes). All other LEM2 parameters used default values.

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

On request, the authors will provide actual starting populations, programs, scripts used to run experiments, actual result files, and all other information needed to reproduce the presented results.

11.1 Optimizing the Rastrigin Function

Optimizing (minimizing) the Rastrigin function is a well-known problem used in testing evolutionary algorithms. As shown in Figure 26, the function has a large number of local optima, and one global optimum equal to zero. It is reached when all the variables equal zero. A general expression of the Rastrigin function is:

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

A plot of the two-variable Rastrigin function is presented in Figure 26.

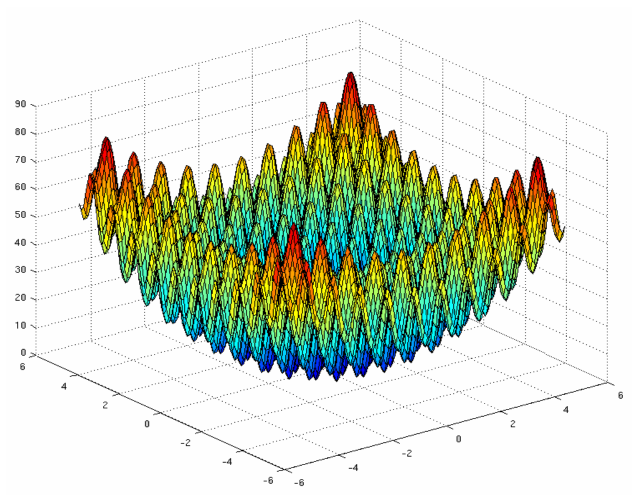


Figure 26: The Rastrigin function of 2 variables.

The experiments below are examples presented to demonstrate comparison of the programs on the Rastrigin function. A full summary of the experiments is presented in Section 11.1.5.

11.1.1 Experiment 1: Optimizing the Rastrigin Function of 10 Variables

Figure 27 and Figure 28 present results from optimizing the Rastrigin function of 10 variables by LEM3, LEM2 and EA. Both LEM2 and LEM3 converged relatively fast at the early stage of evolution, and then slowed down when approaching the optimum. LEM3 reached a δ=0.1-close solution after 415 fitness evaluations, a δ=0.01-close solution after 1000 fitness evaluations, and the optimum after 1208 fitness evaluations. LEM2 reached a δ=0.1-close solution after 374 fitness evaluations, a δ=0.01-close solution after 853 fitness evaluations, and the optimum after 1225 fitness evaluations. EA reached a δ=0.1-close solution after 2673 fitness evaluations, and a δ=0.01-close solution after 12,419 fitness evaluations.

Based on the above numbers, the evolution speedup of LEM3 over EA for δ=0.1 is about 4 times, and for δ=0.01 is about 10 times.

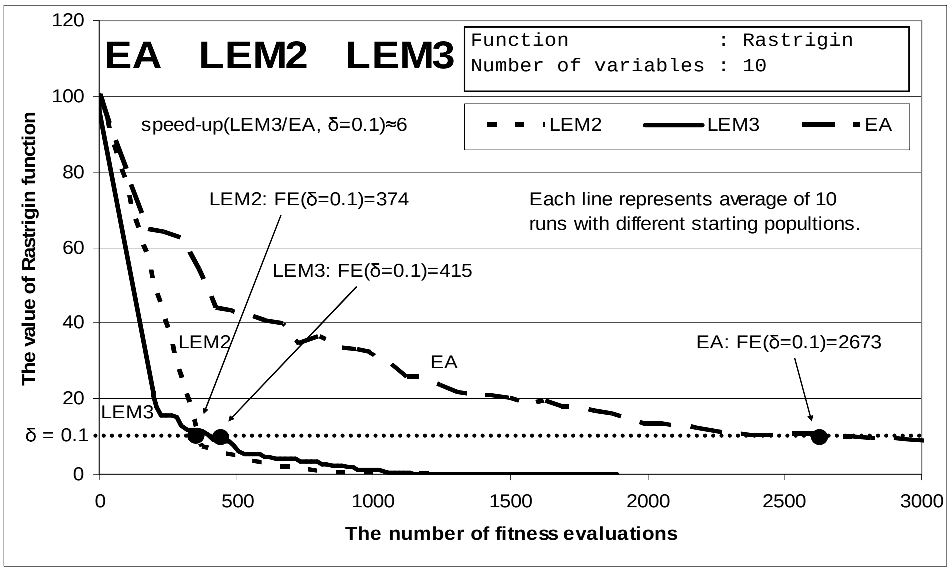


Figure 27: The LEM3, LEM2 and EA evolutionary computation in minimizing the Rastrigin function of 10 variables.

Figure 27 presents graphs representing the average of 10 runs of LEM3, LEM2 and EA on minimizing the Rastrigin function of 10 variables. Both LEM2 and LEM3 found δ=0.1-close solution after fewer than 500 fitness function evaluations, and EA converged to δ=0.1 close solution after about 2700 fitness evaluations. The dotted line represents a δ=0.1 distance from the solution.

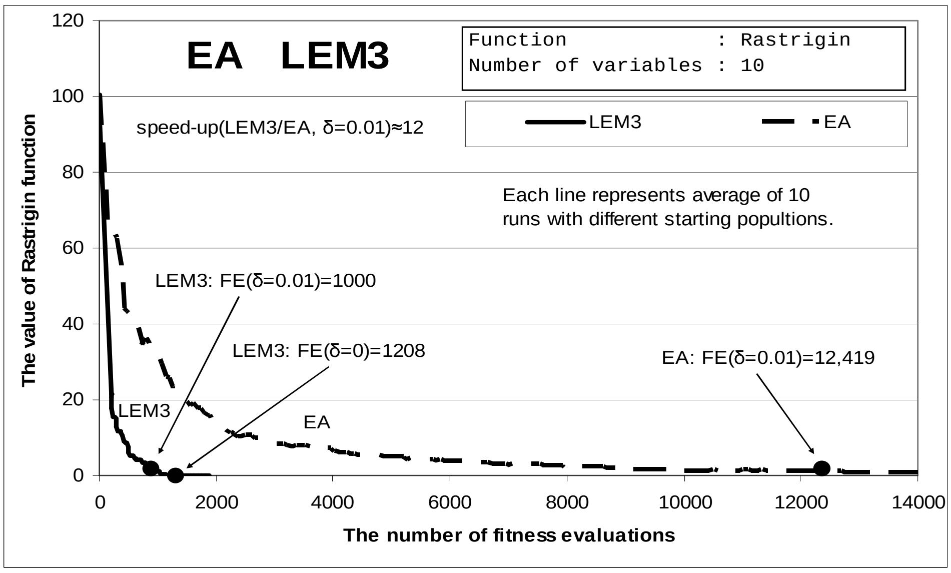


Figure 28: The LEM3 and EA evolutionary computation in minimizing the Rastrigin function of 10 variables.

Figure 28 presents graphs representing the average of 10 runs of EA and LEM3 on minimizing the Rastrigin function of 10 variables on a larger scale up to 14,000 fitness evaluations. LEM3 found a δ=0.01-close solution after 1000 fitness evaluations and the exact solution after about 1200 fitness function evaluations. EA converged to a δ=0.01-close solution after over 12,000 fitness evaluations.

11.1.2 Experiment 2: Optimizing the Rastrigin Function of 100 Variables

Figure 29 and Figure 30 present results from optimizing the Rastrigin function of 100 variables by LEM3, LEM2 and EA with population 100. All three programs converged relatively quickly in the early stage of evolution, and then slowed down when approaching the optimum (EA slowed down much earlier than LEM2 and LEM3). LEM3 reached the optimum after 7569 fitness evaluations, and LEM2 converged to an approximate solution with δ=0.01 after about 6000 fitness evaluations (in all ten of the runs). LEM2 did not find exact solution in this experiment.

EA converged continuously toward the solution and reached an approximate solution with δ=0.01 after 114,445 fitness evaluations.

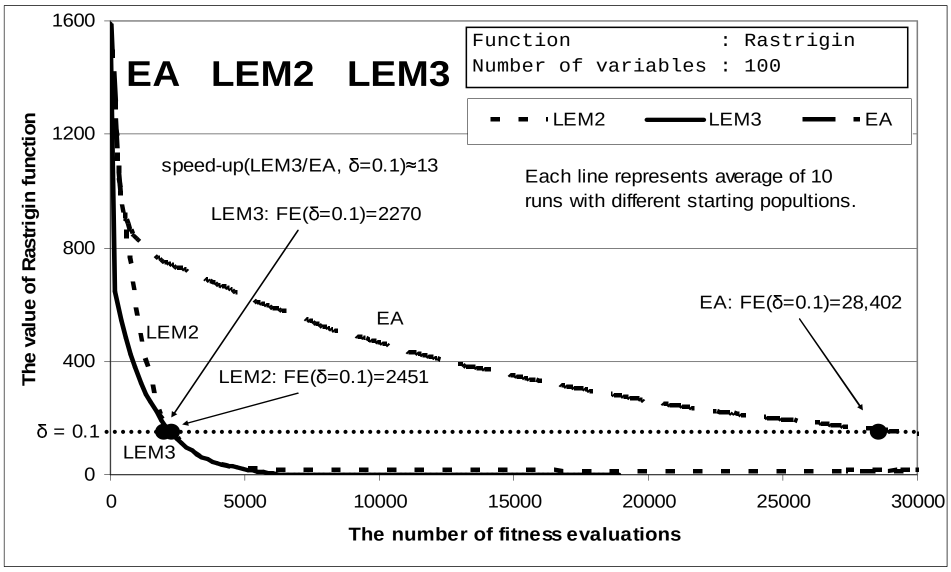


Figure 29: The LEM3, LEM2 and EA evolutionary computation in minimizing the Rastrigin function of 100 variables.

Figure 29 presents graphs representing the average of 10 runs of LEM3, LEM2 and EA on minimizing the Rastrigin function of 100 variables. Both LEM2 and LEM3 converged quickly up to δ=0.01, and EA converged much slower. LEM2 and LEM3 reached δ=0.1-close solutions after fewer than 2500 fitness evaluations. After almost 30,000 fitness evaluations, EA achieved a δ=0.1-close solution. The dotted line represents the δ=0.1 distance from the solution.

Figure 30 presents graphs representing the average of 10 runs of EA and LEM3 on minimizing the Rastrigin function of 100 variables on a larger scale. As mentioned above, LEM3 reached the optimum after 7569 fitness evaluations, and EA found a δ=0.01-close solution after over 114,000 fitness evaluations.

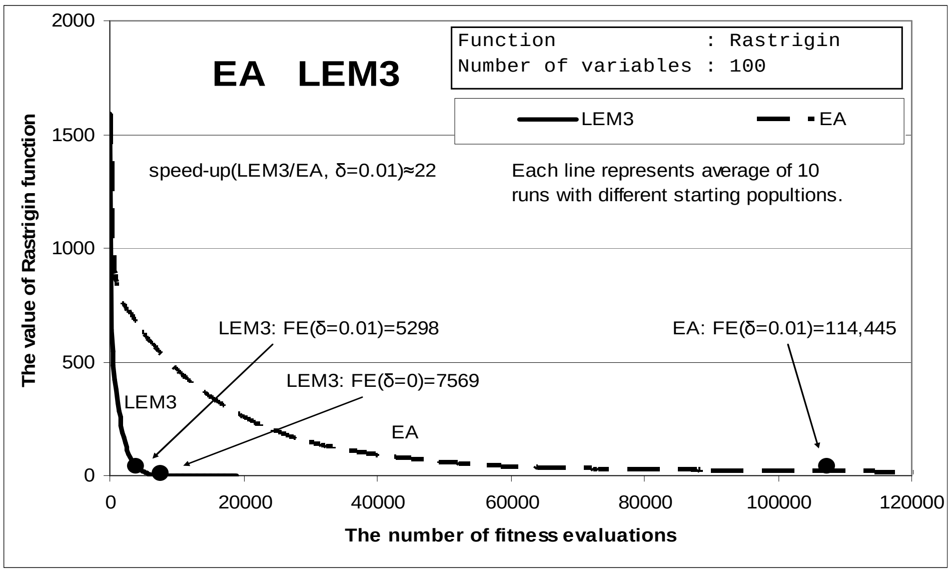


Figure 30: The LEM3 and EA evolutionary computation in minimizing the Rastrigin function of 100 variables.

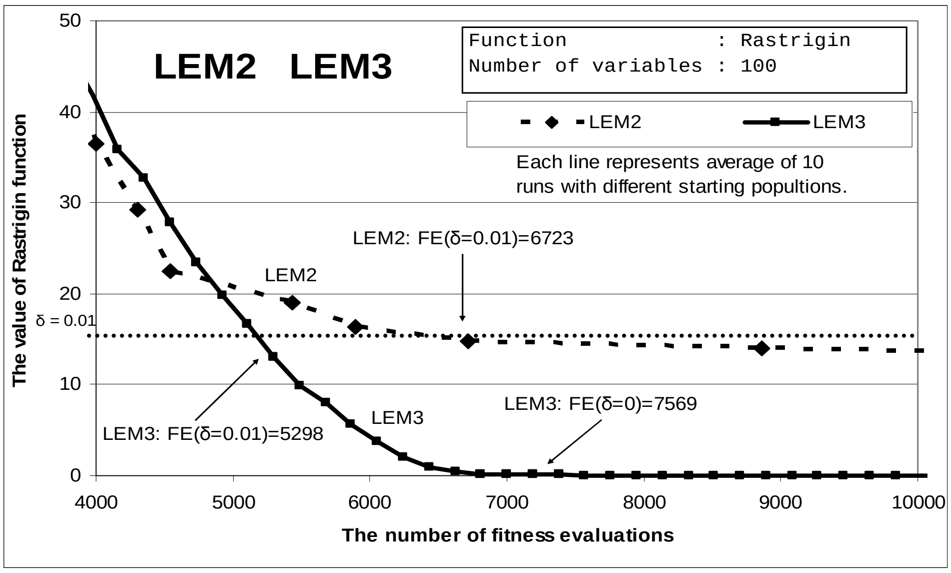


Figure 31: The end-phase of LEM3 and LEM2 evolutionary computation in minimizing the Rastrigin function of 100 variables.

Figure 31 presents a comparison of the end-phase (after 4000 fitness evaluations) performance of LEM3 and LEM2 in minimizing the Rastrigin function of 100 variables. All LEM3 runs found the exact solution (δ=0) after between 6000 and 10,000 fitness evaluations. No LEM2 runs found the exact solution; they converged to an approximate solution with δ=0.01. The dotted line represents the δ=0.01 distance from the solution.

11.1.3 Experiment 3: Optimizing the Rastrigin function of 500 Variables

In this experiment, LEM3 was compared with EA. LEM2 was been compared because its limit on the number of variables. Figure 32 presents results of the optimization of the Rastrigin function of 500 variables by LEM3 and EA. Both programs converged relatively fast in the early stage of evolution, and then slowed down when approaching the optimum (EA slowed down much earlier than LEM3). LEM3 reached a δ=0.1-close solution after 5,252 fitness evaluations and a δ=0.01-close solution after 16,195 fitness evaluations. EA reached a δ=0.1-close solution after 128,184 fitness evaluations (in all of the 10 runs). Thus, the evolutionary speedup of LEM3 over EA with δ=0.1 is about 24.

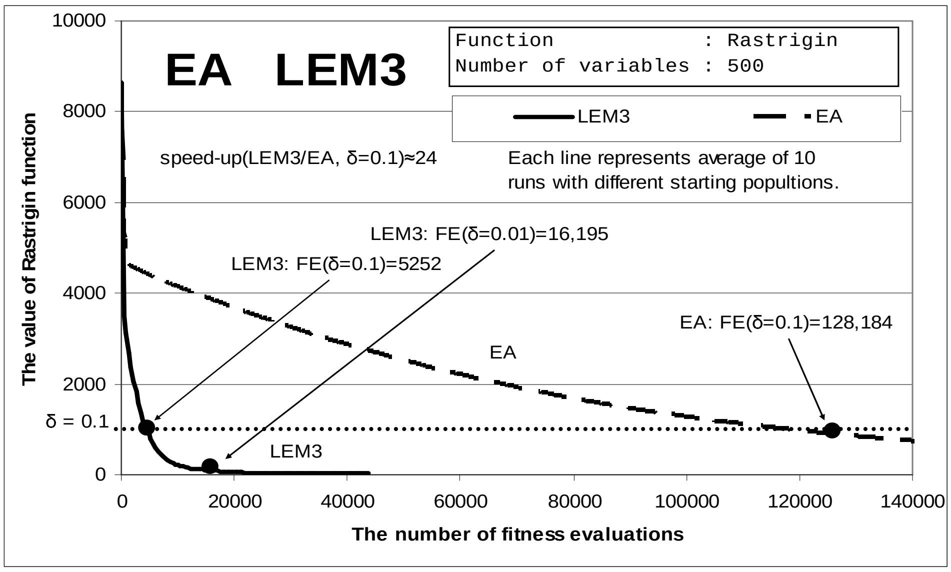


Figure 32: The LEM3 and EA evolutionary computation in minimizing the Rastrigin function of 500 variables.

11.1.4 Experiment 4: Optimizing the Rastrigin Function of 900 Variables

In this experiment, LEM3 was compared with EA. Figure 33 presents graphs representing results from optimizing the Rastrigin function of 900 variables by LEM3 and EA. Both programs converged relatively fast at the early stage of evolution, and then slowed down when approaching the optimum (EA slowed down much earlier than LEM3).

LEM3 reached a δ=0.1-close solution after 7,491 fitness evaluations and a δ=0.01-close solution after 182,366 fitness evaluations. This indicates that LEM3 slowed down significantly at the end of evolution. EA reached an approximate solution with δ=0.1 after 208,246 fitness evaluations and a δ=0.01-close solution after 1,214,476, an order of magnitude slower than LEM3.

Based on the above numbers, the evolution speedup of LEM3 over EA for δ=0.1 is about 28 times and for δ=0.01 is about 7 times.

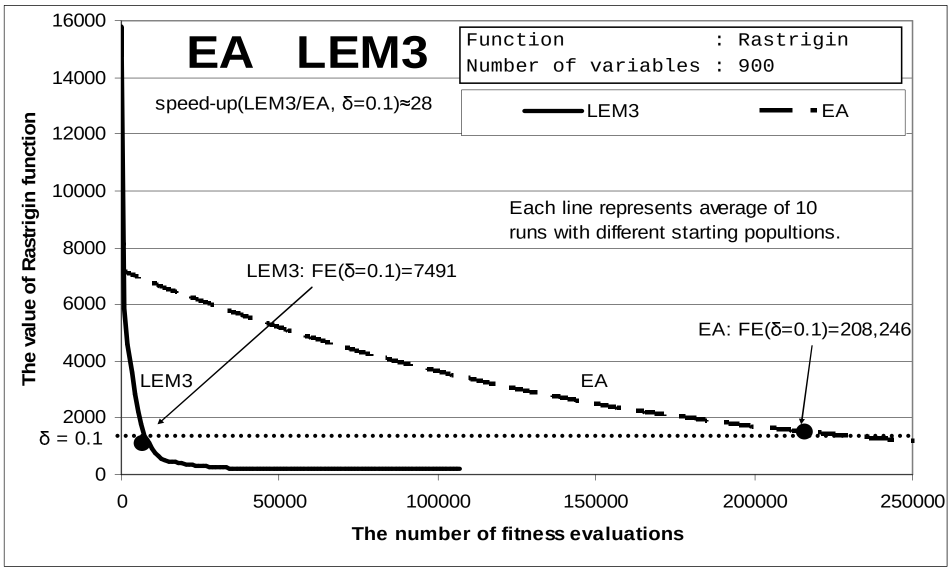


Figure 33: The LEM3 and EA evolutionary computation in minimizing the Rastrigin function of 900 variables.

11.1.5 Summary of Experimental Results with the Rastrigin Function

A comparison of LEM3 and EA on the function on terms of evolution length with δ=0.1, and δ=0.01 is presented in Tables 1 and 2 below, and graphically presented in Figure 34 and Figure 35. In practice, the programs can be applied to functions with even more variables, especially LEM3, since it converges very fast for large numbers of variables.

Table 1 shows that the advantage of LEM3 over EA for the Rastrigin for δ=0.1 grows with the number of variables. For example, for 10 variables, the evolutionary speedup (the ratio of the number of fitness evaluations by EA and LEM3) is about 4 and for 1000 variables, the speedup is about 33.

Table 1: LEM3 and EA evolution length and evolutionary speedup on optimizing the Rastrigin function of different numbers of variables, δ=0.1.

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

Figure 34 below presents graphically the results from the above table. It shows that the number of fitness function evaluations for EA grows much faster than for LEM3 with increasing numbers of attributes.

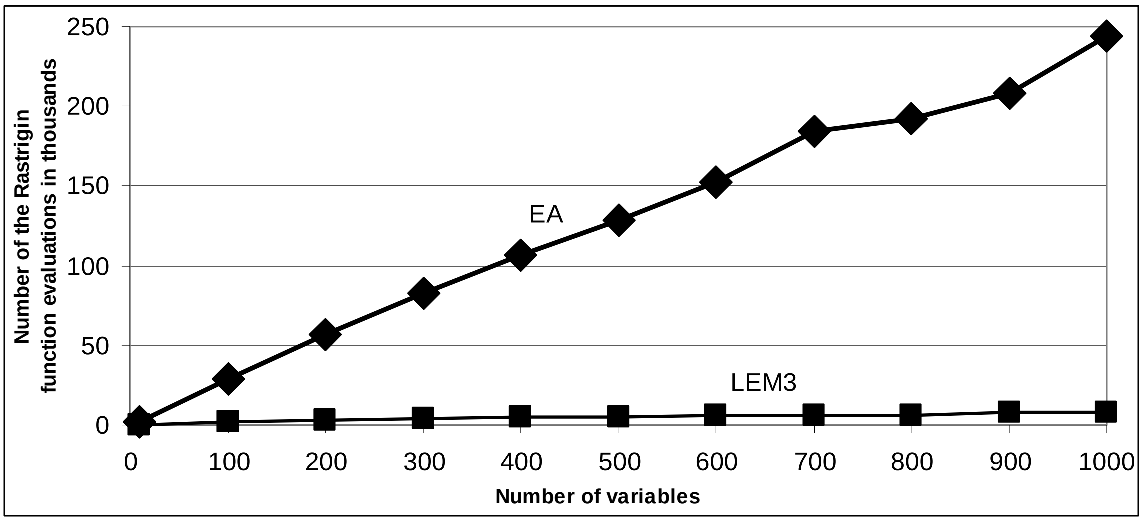


Figure 34: Evolutionary speedup on minimizing the Rastrigin function of 10 to 1000 variables for δ =0.1.

Table 2 below presents the numbers of fitness evaluations needed by LEM3 and EA to converge to δ=0.01-close solutions for the Rastrigin function of different numbers of variables. The evolution speedup grows up to 400 variables, where it reaches 46, and decreases for more variables. Despite the decreasing evolution speedup, LEM3 remains about an order of magnitude faster than EA in terms of evolution length.

Table 2: LEM3 and EA evolution length and evolutionary speedup on optimizing the Rastrigin function of different numbers of variables, δ=0.01.

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

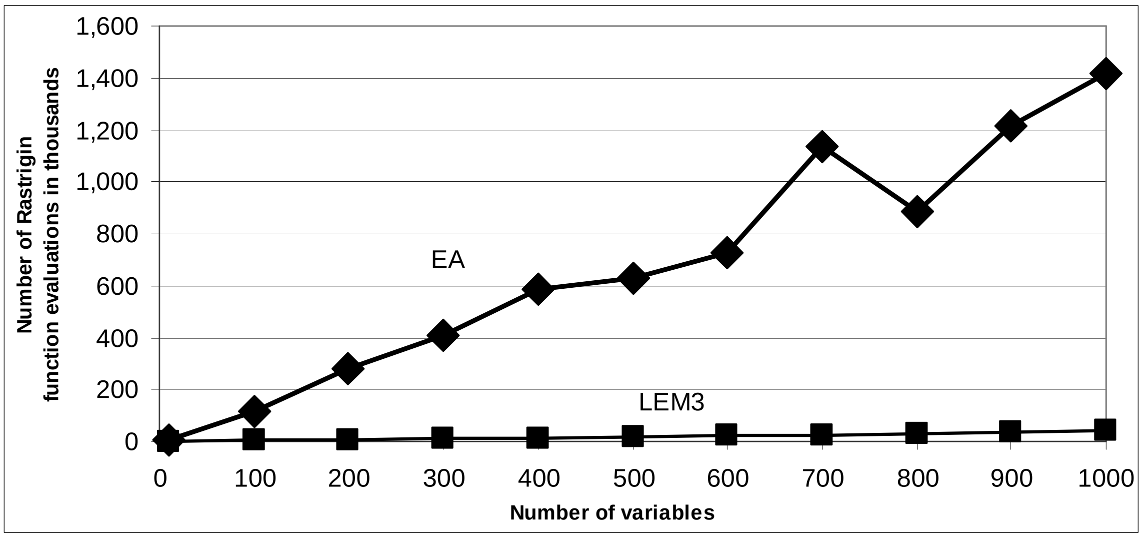


Figure 35: Evolutionary speedup on minimizing the Rastrigin function of 10 to 1000 variables for δ =0.01.

The number of fitness evaluations needed to δ-optimize the function (that is, to find a solution δ-close to the optimum) grows with the number of variables up to a point when because of lack of diversity, LEM3 starts behaving similarly to a standard Darwinian-type method by and has to apply mutation every iteration. This opens a very important and interesting area for future research, namely how to modify LEM3 to improve optimization at the end of evolution process.

It is interesting that for 700 variables, EA requires more fitness function evaluation than for 800 variables, but continues following the original tendency for 900 and 1000 variables. This shows the high sensitivity of the algorithm on starting populations, even if the experiment was repeated 10 times with different starting populations (the same starting populations were used in LEM3).

11.2 Optimizing the Griewangk Function

Optimizing (minimizing) the Griewangk function is a well-known problem used in testing evolutionary algorithms. The function has a large number of local optima and one global optimum equal to zero. It is reached when all the variables equal zero. The domain for all variables in the preformed experiments was [-5.12, 511].

The general n-dimensional Griewangk function is given by the expression $f(x_1,\ldots, x_n) = 1 + \sum_{i=1}^{n} \frac{x_i}{4000} - \prod_{i=1}^{n} \cos(x_1 / \sqrt{i})$. A plot of its 2 dimensional case is presented in Figure 36 (local view near the minimum).

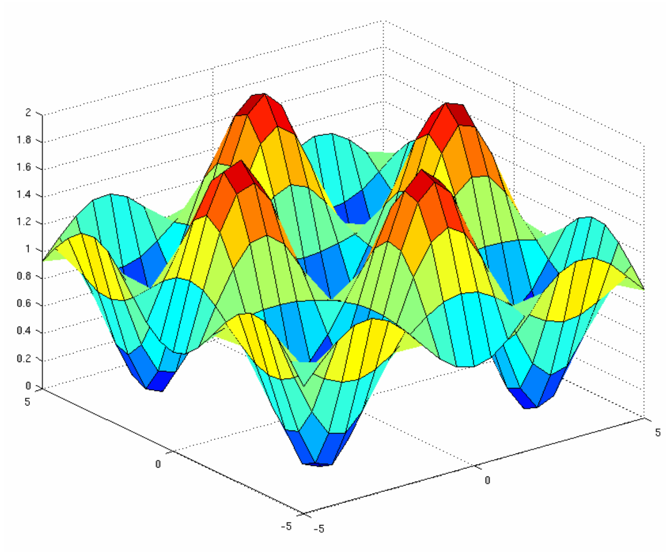


Figure 36: The Griewangk function of 2 variables (local view near minimum).

Experiments presented in this section show comparison LEM3 and EA on the Griewangk function of 10, 100, 500, and 100 variables. A summary of experiments for δ=0.1 and δ=0.01-close solutions is presented in Section. LEM2 was not compared in this set of experiments because the Griewangk function is not in its set of testing functions.

11.2.1 Experiment 5: Optimizing the Griewangk Function of 10 Variables

Figure 37 presents graphs representing results from optimizing the Griewangk function of 10 variables by LEM3 and EA. Both programs converged relatively quickly at the early stage of evolution (LEM3 much faster than EA), and then slowed down when approaching the optimum (EA slowed down much earlier than LEM3).

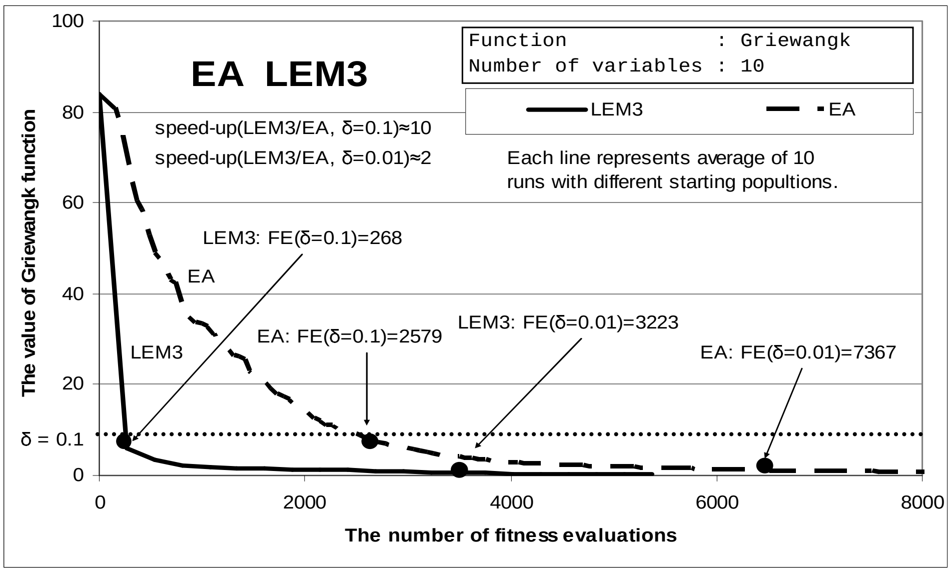


Figure 37: The LEM3 and EA evolutionary computation in minimizing the Griewangk function of 10 variables.

LEM3 reached a δ=0.1-close solution after 268 fitness evaluations, a δ=0.01-close solution after 3223 fitness evaluations, and δ=0.001 after 15,309 fitness evaluations. EA reached a δ=0.1-close solution after 2579 and δ=0.01 after 7367 fitness evaluations.

The above numbers indicate that the speedup of LEM3 over EA for δ=0.1 is about 10 times, and the speedup of LEM3 over EA for δ=0.01 is about 2 times.

11.2.2 Experiment 6: Optimizing the Griewangk Function of 100 Variables

Figure 38 presents graphs representing results from optimizing the Griewangk function of 100 variables by EA and LEM3 with population 100. LEM3 converged very fast from the beginning of evolution and slowed down when approaching the optimum. EA converged slowly from the beginning of evolution. LEM3 reached a δ=0.1-close solution after 1,797 fitness evaluations, a δ=0.01-close solution after 10,486 fitness evaluations, and a δ=0.001-close solution after 26,366 fitness evaluations. EA reached a δ=0.1-close solution after 24,611 fitness evaluations, a δ=0.01-close solution after 52,632 fitness evaluations, and a δ=0.001-close solution after 89,258 fitness evaluations.

The above numbers indicate that the speedup of LEM3 over EA for δ=0.1 is about 14 times, the speedup of LEM3 over EA for δ=0.01 is about 5 times, and the speedup of LEM3 over EA for δ=0.001 is about 3 times.

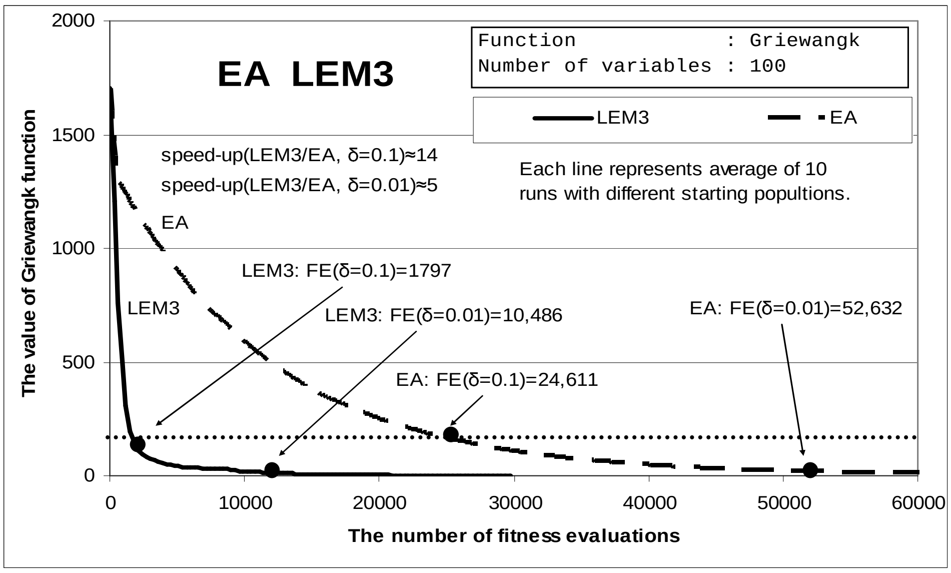


Figure 38: The LEM3 and EA evolutionary computation in minimizing the Griewangk function of 100 variables.

11.2.3 Experiment 7: Optimizing the Griewangk Function of 500 Variables

Figure 39 presents graphs representing results from optimizing the Griewangk function of 500 variables by EA and LEM3. LEM3 converged very fast from the beginning of evolution and slowed down when approaching the optimum. EA converged slowly from the beginning of evolution. LEM3 reached a δ=0.1-close solution after 6547 fitness evaluations and a δ=0.01-close solution after 51,564 fitness evaluations. EA reached a δ=0.1-close solution after 126,057 fitness evaluations and a δ=0.01-close solution after 263,801 fitness evaluations.

The above numbers indicate that the speedup of LEM3 over EA for δ=0.1 is about 19 times and the speedup of LEM3 over EA for δ=0.01 is about 5 times.

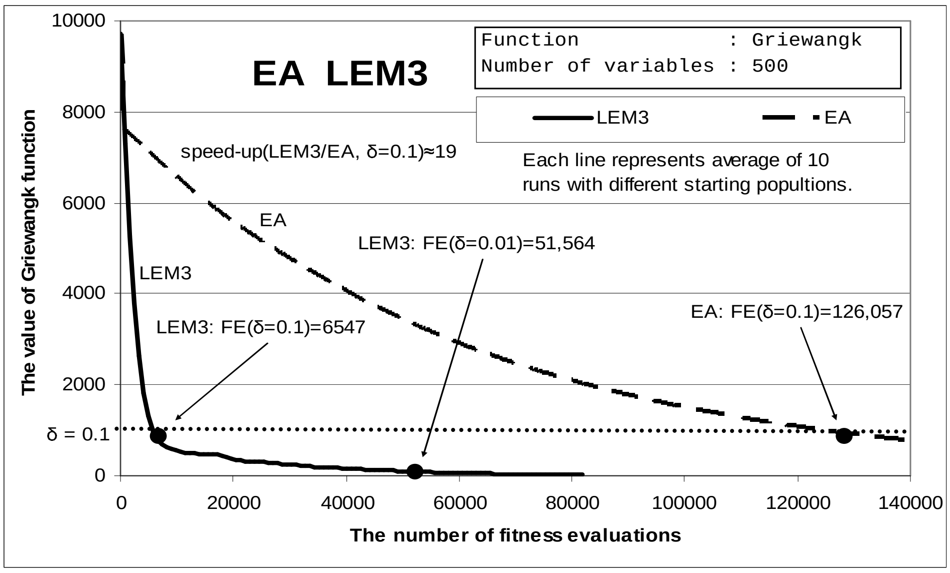


Figure 39: The LEM3 and EA evolutionary computation in minimizing the Griewangk function of 500 variables.

11.2.4 Experiment 8: Optimizing the Griewangk Function of 1000 Variables

Figure 40 presents graphs representing results from optimizing the Griewangk function of 1000 variables by EA and LEM3. LEM3 converged very fast from the beginning of evolution and slowed down when approaching the optimum. EA converged slowly from the beginning of evolution. LEM3 reached a δ=0.1-close solution after 10,780 fitness evaluations and a δ=0.01-close solution after 112,600 fitness evaluations. EA reached a δ=0.1-close solution after 251,233 fitness evaluations and a δ=0.01-close solution after 525,096 fitness evaluations.

The above numbers indicate that the speedup of LEM3 over EA for δ=0.1 is about 23 times and the speedup of LEM3 over EA for δ=0.01 is about 5 times.

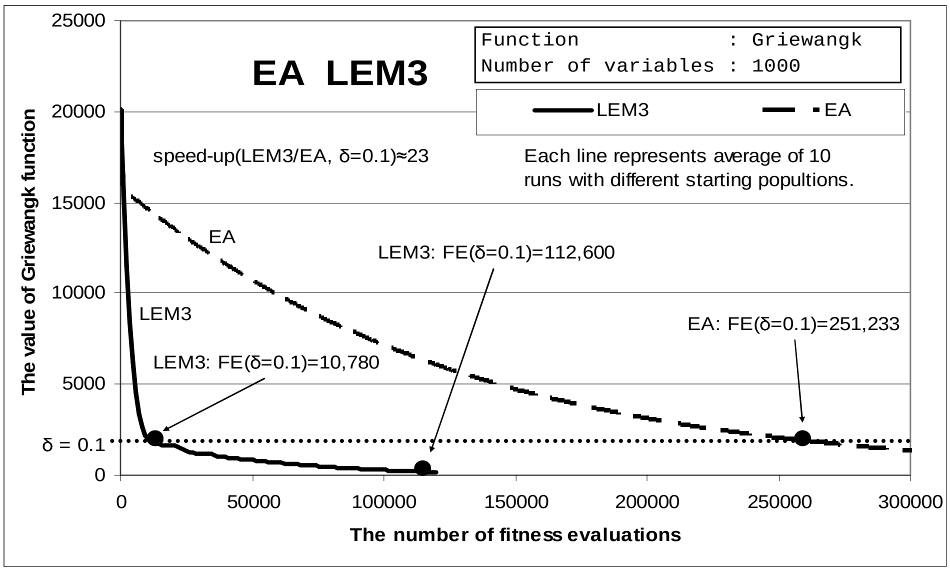


Figure 40: The LEM3 and EA evolutionary computation in minimizing the Griewangk function of 1000 variables.

11.2.5 Summary of Experimental Results with the Griewangk Function

Results from applying LEM3 and EA to optimizing this function in terms of the evolution length for δ=0.1-close solutions are presented in Table 3 and for δ=0.01-close solutions are presented in Table 4.

Table 3: LEM3 and EA evolution length and evolutionary speedup on optimizing the Griewangk function of different numbers of variables, δ=0.1.

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

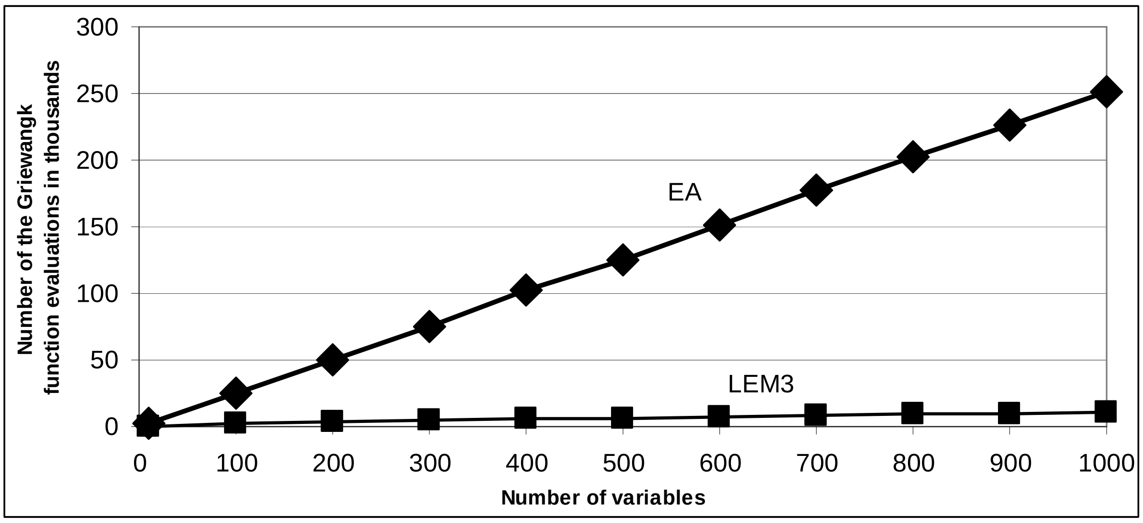


Figure 41: Evolutionary speedup on minimizing the Griewangk function of 10 to 1000 variables for δ =0.1.

Table 4: LEM3 and EA evolution length and evolutionary speedup on optimizing the Griewangk function of different numbers of variables, δ=0.01.

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

When optimizing the Griewangk function for δ=0.1-close solutions, LEM3's advantage over EA grew from about 10 times for the function of 10 variables to about 23 times for the function with 1000 variables. For the same function for δ=0.01-close solutions, the speed up of LEM3 was about 5 times when the number of variables was at least 100. This is because of the fact that at the end of evolution LEM3 needs to apply its mutation operator to introduce diversity in a population, and therefore it behaves like the Darwinian-type method. Without sufficient diversity in the population, it is impossible to apply the learning process in LEM. This problem is addressed in the description of current research issues in the Learnable Evolution Model in Section 13.

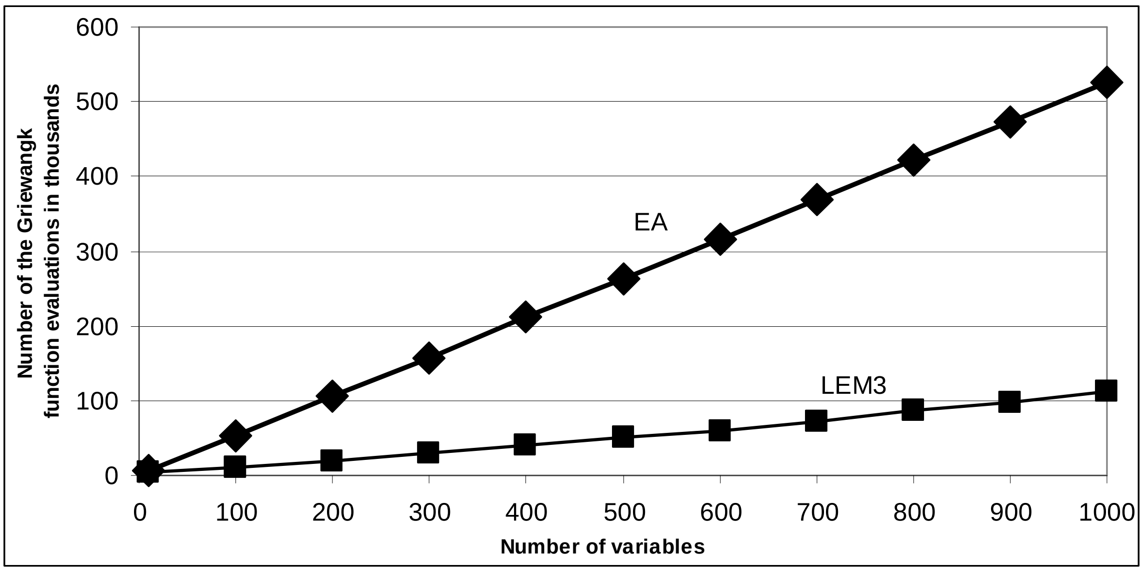


Figure 42: Evolutionary speedup on minimizing the Griewangk function of 10 to 1000 variables for δ =0.01.

11.3 Optimizing the Rosenbrock Function

Optimizing (minimizing) the Rosenbrock function is a well-known problem used in testing evolutionary algorithms. The function has one global optimum reached when values of all attributes equal one. The Rosenbrock function is a hard optimization problem due to the high correlation of variables and the almost flat ridge on which the optimum is located.

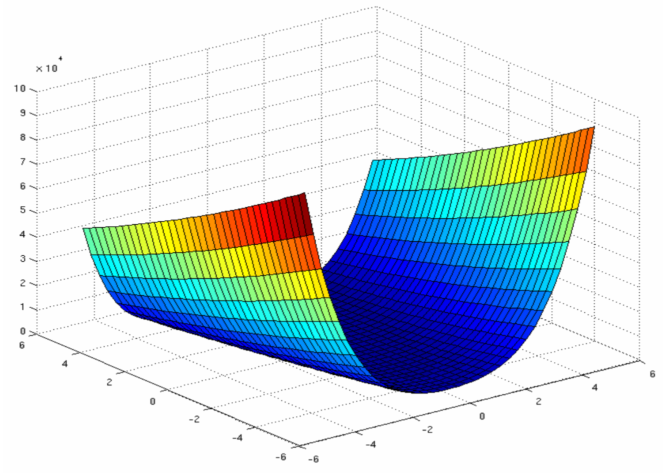


Figure 43: The Rosenbrock function of 2 variables.

The function is given by equation $f(x_1,...,x_n)=\sum_{i=1}^{n-1}(100*(x_{i+1}-x_i^2)^2+(x_i-1)^2)$, and a plot of its 2-dimensional case is presented in Figure 43. The Rosenbrock function reaches a minimum equal to 0 at the point (1, 1, …1). Please note that in the figure, domains of both variables are [-10, 10] for better visualization, but in the presented experiments, we used domains [-2, 10].

11.3.1 Experiment 9: Optimizing the Rosenbrock Function of 10 Variables

In this experiment LEM3 was compared with EA and LEM2. Figure 44 presents graphs representing results from optimizing the Rosenbrock function of 10 variables by LEM3, LEM2, and EA. LEM3 reached a δ=0.1-close solution after 325 fitness evaluations, a δ=0.01-close solution after 682 fitness evaluations, and a δ=0.001-close solution after 1365 fitness evaluations. LEM2 reached a δ=0.1-close solution after 275 fitness evaluations, a δ=0.01-close solution after 492 fitness evaluations, and a δ=0.001-close solution after 1180 fitness evaluations. EA reached a δ=0.1-close solution after 541 fitness evaluations, a δ=0.01-close solution after 2027 fitness evaluations, and a δ=0.001-close solution after 8602 fitness evaluations.

The above numbers indicate that the speedup of LEM3 over EA is about 2 times for δ=0.1, about 3 times for δ=0.01 is, and about 6 times for δ=0.001. All presented numbers are averaged over 10 executions of each program.

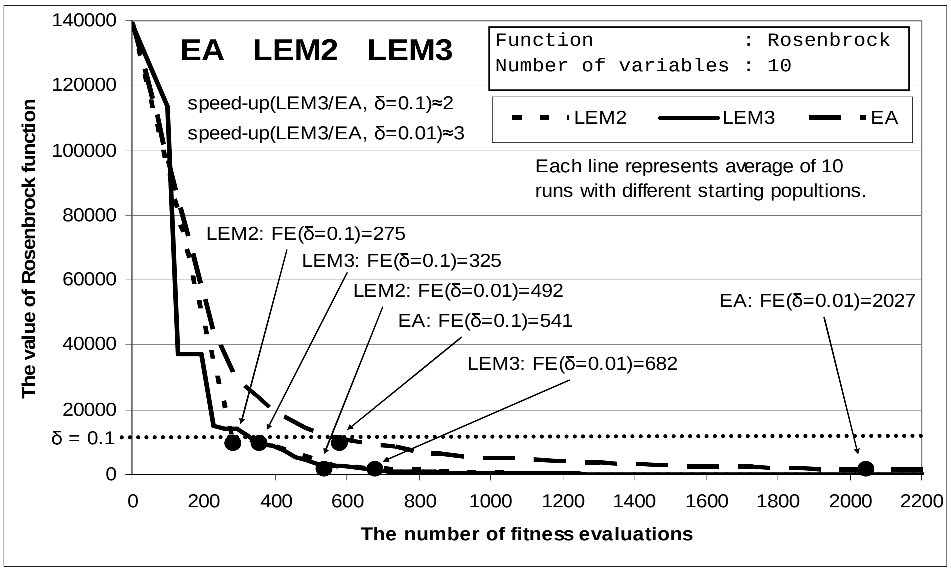


Figure 44: The LEM3, LEM2, and EA evolutionary computation in minimizing the Rosenbrock function of 10 variables.

11.3.2 Experiment 10: Optimizing the Rosenbrock Function of 100 Variables

In this experiment, LEM3 is compared with LEM2 and EA. Figure 45 presents graphs representing results from optimizing the Rosenbrock function of 100 variables by LEM3, LEM2, and EA. LEM2 and EA converged similarly at the beginning of evolution, and after about 600 fitness evaluations started slowing down. LEM3 reached a δ=0.1-close solution after 1906 fitness evaluations, a δ=0.01-close solution after 3495 fitness evaluations, and a δ=0.001-close solution after 7039 fitness evaluations. LEM2 reached a δ=0.1-close solution after 918 fitness evaluations, a δ=0.01-close solution after 2348 fitness evaluations, and a δ=0.001-close solution after 44,087 fitness evaluations. EA reached a δ=0.1-close solution after 3067 fitness evaluations, a δ=0.01-close solution after 26,944 fitness evaluations, and a δ=0.001-close solution after 151,839 fitness evaluations.

The above numbers indicate that the speedup of LEM3 over EA is about 2 times for δ=0.1, about 8 times for δ=0.01, and about 22 times for δ=0.001. All presented numbers are averaged over 10 executions of each program.

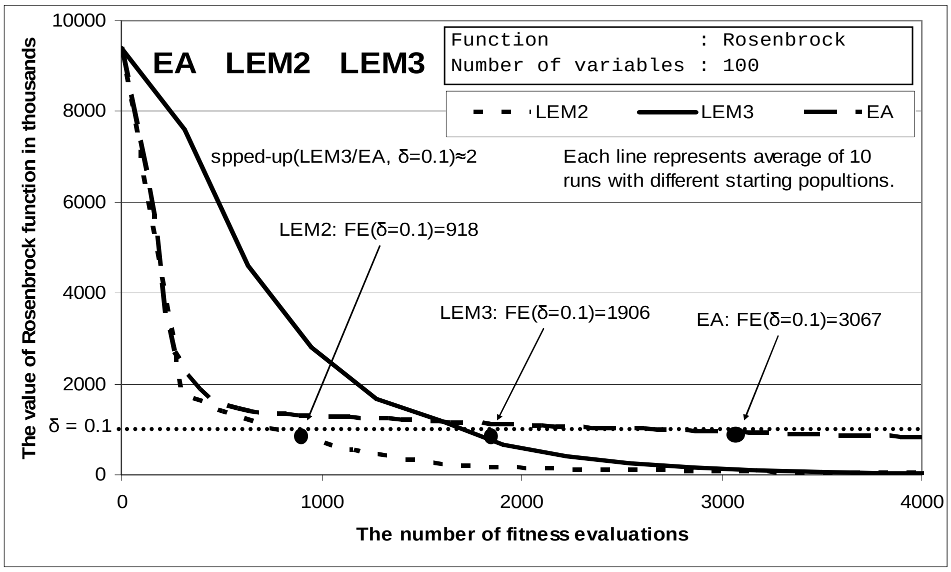


Figure 45: The LEM3, LEM2, and EA evolutionary computation in minimizing the Rosenbrock function of 100 variables.

Figure 46 shows LEM3 is more precise than LEM2 during the end-phase of evolution. LEM3 stops execution after converging to a δ=0.000075-close solution after fewer than 35,000 fitness evaluations, while LEM2 converges to a δ=0.0001-close solution after over 44,000 fitness evaluations.

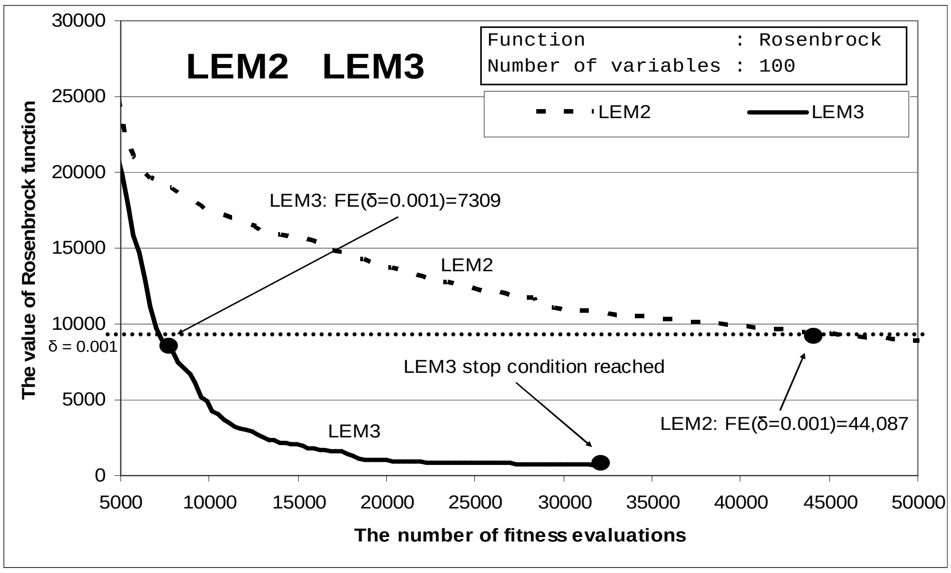


Figure 46: The end-phase of LEM3 and LEM2 evolutionary computation in minimizing the Rosenbrock function of 100 variables.

11.3.3 Experiment 11: Optimizing the Rosenbrock Function of 500 Variables

In this experiment LEM3 was compared with EA. LEM2 was not used in this experiment because of its limit on number of variables. Figure 47 presents graphs representing results from optimizing the Rosenbrock function of 500 variables by EA and LEM3 with population 100. EA converged very fast at the beginning of evolution, and after about 500 fitness evaluations significantly slowed down. LEM3 converged slower in the beginning of evolution, but after about 4000 fitness evaluations it outperformed EA. After 10,000 fitness evaluations LEM3 reached a δ=0.01-close solution and EA reached a δ=0.11-close solution, which is 11 times worse than LEM3. After about 40,000 fitness evaluations LEM3 reached a 0.001-close solution and EA reached a δ=0.068-close solution, which represents a 68-fold advantage of LEM3 over EA.

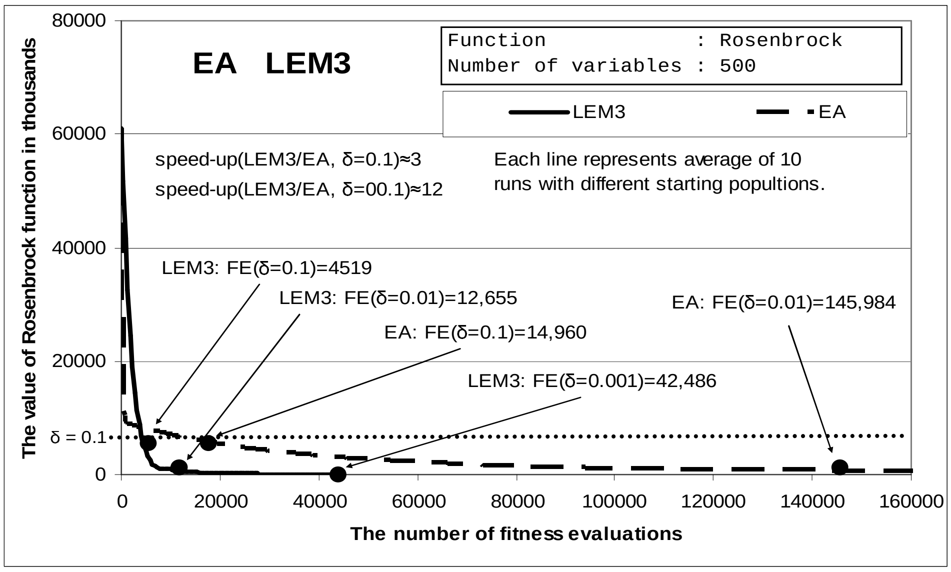


Figure 47: The LEM3 and EA evolutionary computation in minimizing the Rosenbrock function of 500 variables.

11.3.4 Experiment 12: Optimizing the Rosenbrock Function of 1000 Variables

In this experiment LEM3 was compared with EA. Figure 48 presents graphs representing results from optimizing the Rosenbrock function of 1000 variables by LEM3 and EA. Both programs converged very fast in the beginning of evolution. LEM3 reached a δ=0.1-close solution after 6851 fitness evaluations, a δ=0.01-close solution after 29,691 fitness evaluations, and a δ=0.001-close solution after 101,635 fitness evaluations. EA reached a δ=0.1-close solution after 28,468 fitness evaluations and a δ=0.01-close solution after 296,897 fitness evaluations. EA did not reach a δ=0.001-close solution; it stopped after about 380,000 fitness evaluations with δ=0074.

The above numbers indicate that speedup of LEM3 over EA for δ=0.1 is about 4 times and the speedup of LEM3 over EA for δ=0.01 is about 10 times. All presented numbers are averaged over 10 executions of each program.

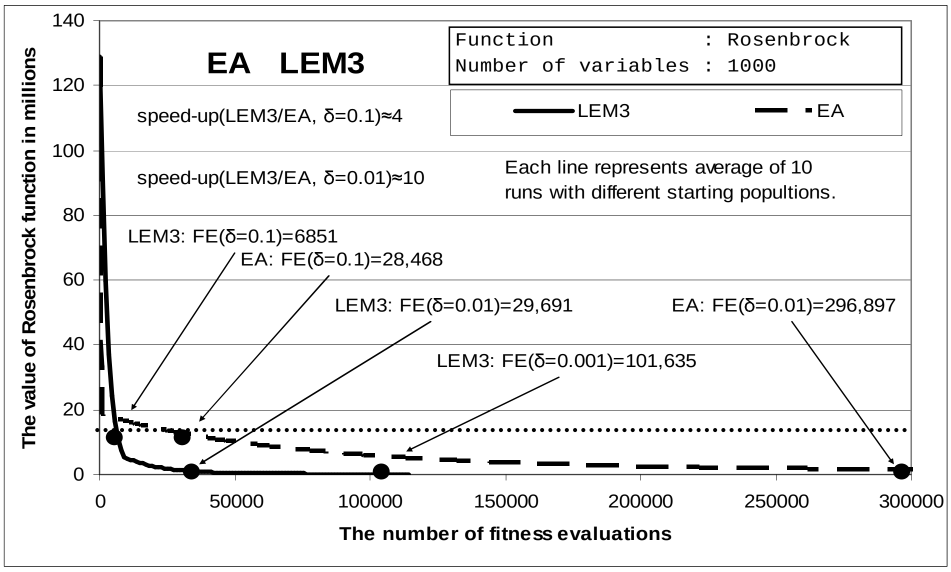


Figure 48: The LEM3 and EA evolutionary computation in minimizing the Rosenbrock function of 1000 variables.

11.3.5 Summary of experimental results with the Rosenbrock Function

Table 5 below presents a comparison of the performance of EA and LEM3 on optimizing the Rosenbrock function of 10 to 1000 variables with δ=0.1-close solutions. It can be seen that the advantage of LEM3 over EA, is small, but grows with the complexity of the problem. Finding more precise, δ=0.01-close, solutions shows a clear advantage for LEM3, up to 18 times. These results are presented in Table 6.

Table 5: LEM3 and EA evolution length and evolutionary speedup on optimizing the Rosenbrock function of different numbers of variables, δ=0.1.

Number of Variables	**Number of fitness evaluations**		**EA/LEM3 Speedup**
	EA	**LEM3**	
10	541	325	**~2**
100	3,367	1,906	**~2**
200	5,699	2,625	**~2**
300	8,547	3,518	**~2**
400	11,690	4,038	**~3**
500	14,960	4,519	**~3**
600	15,606	5,013	**~3**
700	19,448	5,491	**~4**
800	22,731	5,710	**~4**
900	25,216	6,835	**~4**
1000	28,468	6,851	**~4**

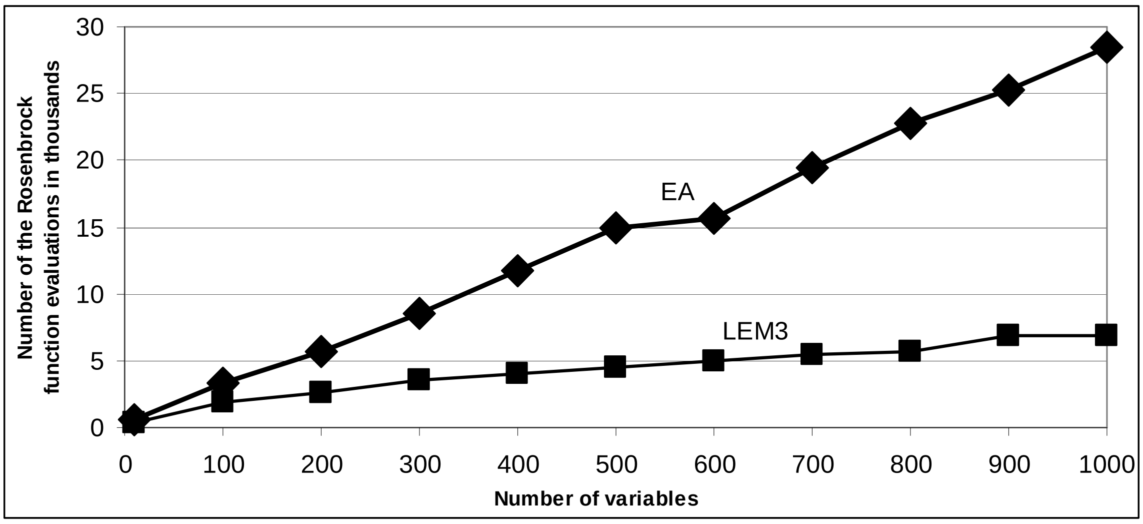


Figure 49: Evolutionary speedup on minimizing the Rosenbrock function of 10 to 1000 variables for δ =0.1.

Table 6: LEM3 and EA evolution length and evolutionary speedup on optimizing the Rosenbrock function of different numbers of variables, δ=0.01.

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

The evolution speedup of LEM3 over EA grows up to about 12 times (with one exception that is about 18 times for 300 variables) and stabilizes with a small decreasing tendency. The tendency of smaller speedup at the end of evolution is due to fact that at the end of evolution, LEM3 has to apply its mutation operator in order to increase diversity in the population and therefore starts behaving more like a Darwinian-type method. The lack of diversity makes it impossible to apply learning mode. This matter is discussed below.

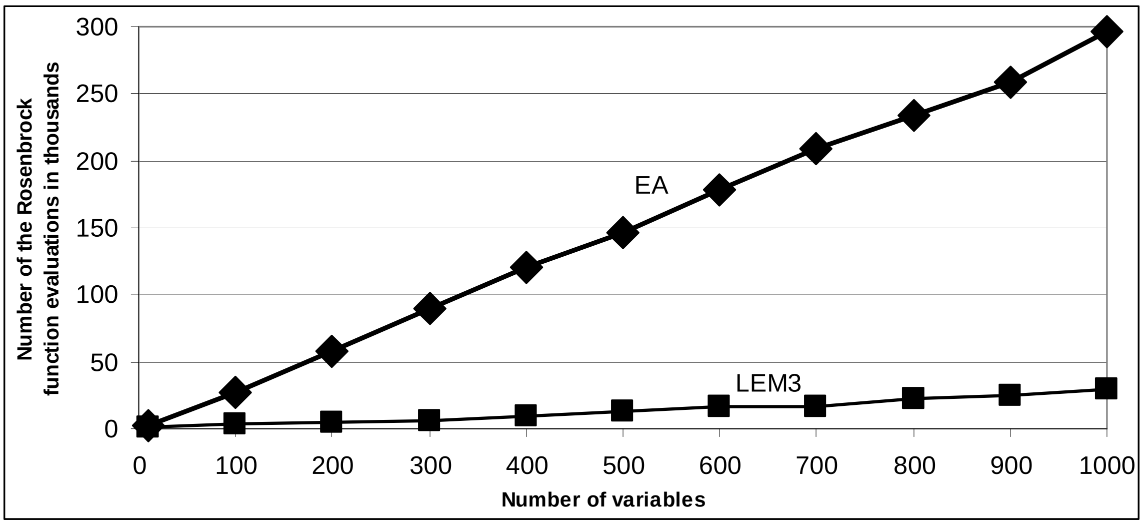


Figure 50: Evolutionary speedup on minimizing the Rosenbrock function of 10 to 1000 variables for δ =0.01.

11.4 Time of Evolution

Operators of hypothesis generation and instantiation are much more computationally complex than mutation and/or recombination operators. Consequently, the computation time needed by LEM3 to execute the hypothesis generation and instantiation operators is much longer than the time needed by EA to apply. However, whenever the evaluation of the fitness function is computationally non-trivial, LEM3 clearly wins not only in the evolution length but also evolution time.

For example, suppose that EA and LEM3 are applied to an optimization problem with complexity similar to the Rastrigin function of, say, 500 variables. To get a δ=0.1 close solution, EA requires about 128,000 fitness evaluations (optimization time is about 180 seconds) and LEM3 requires about 5,200 fitness evaluations (optimization time is about 400 seconds).

By simple calculation we find that if the programs were applied to a real world problem in which evaluation of its fitness function took only 0.1 second, which is not unusual, EA would require about 35.5 hours to find a δ=0.1 close solution, and LEM3 would require no more than about 15 minutes to find the solution. Even if the evolutionary speedup is only on the order of 5, as for example for the Griewangk function for δ=0.01, the LEM3 execution time is less than the execution time of EA.

For completeness of this discussion, it is important to mention that the AQ21 execution time grows with the number of variables. This growth, however, is compensated by the significantly smaller number of fitness function evaluations. It is also important to point out that that the time of evaluation of the fitness function also grows with the number of variables. It should also be mentioned that AQ21 was not optimized for the execution time. To speedup LEM3, one could optimize AQ21 for the execution time, for example, by implementing it in hardware or by removing its features that are not used in LEM3.

11.5 Summary of Experimental Results

In the presented experiments we compared the LEM3 system described in this paper with the EA program, representing a standard Darwinian-type evolutionary method, and LEM2, the previous implementation of the Learnable Evolution Model, on three well-known optimization problems. Because of the limitations of LEM2, it was compared only in a few cases. It is important to note that in none of the presented experiments, program parameters were fine-tuned to achieve better results; all default settings were used. In real world problems, especially in solving hard problems, there is only one run of a method due to the high complexity of fitness evaluations. Such a run may take hours of even days before optimal results are returned; thus, users cannot usually cannot change parameters and try again.

The presented results show the superiority of LEM3 over EA in terms of evolution length, measured as the number of fitness function evaluations needed to achieve a δ-close solution. LEM3 outperforms LEM2 not only in terms of the evolution length and precision of solutions, but also in terms of the limitations of LEM2, such as maximum number of variables allowed.

In all tested cases, LEM3 gave the best results, outperforming EA by up to about 46 times in case of the Rastrigin function and on average by about 16.5 times for functions of at least 100 variables. Table 7 below presents average speedups of LEM3 over EA for different numbers of variables in the experiments shown above.

Table 7: Average evolutionary speedup of LEM3 over EA for Rastrigin, Griewangk, and Rosenbrock functions of different numbers of variables.

Number of Variables	10	100	200	300	400	500	600	700	800	900	1000
~Speedup	4.3	10.7	15	16.8	17.8	17.2	16.7	19	16.6	17.2	18

In the presented experiments, the evolutionary speedup of LEM3 over the Darwinian-type method, EA, grew with the complexity of problem up to 700 variables, and then tapered of stabilized. This can be explained by the fact that with the growth of the number of variables the role of hypothesis generation operators decreases because fewer and fewer variables are needed to diffentiate between H- and L-groups, and thus more and more variables are instantiated more or less randomly, that is, the method starts behaving like a Darwinian-type algorithm.

In the presented comparison, we used a standard evolutionary computation method, to which any other method can be easily compared.

12 RELATED RESEARCH

LEM3 is the newest system, continuing the progression of LEM implementations towards more complete and advanced embodiments of the LEM methodology. Previous programs employed older learning systems from the AQ family: LEM1 (Michalski and Zhang, 1999) used AQ15c (Wnek et al., 1996) and LEM2 (Cervone, Kaufman and Michalski, 2001) used AQ18 (Kaufman and Michalski, 2000b).

An implementation of Learnable Evolution Model for Multi-objective Optimization (LEMMO; Jourdan et al., 2005) is based on rules generated from trees by the C4.5 learning program (Quinlan, 1993). LEMMO was recently applied to a water quality optimization problem (Jourdan et al., 2005). Because of its use of the C4.5 learning program, LEMMO learns not only rules that describe the H-group, but also rules that describe the L-group. Newly generated individuals are required to match the H-group descriptions and to not match the L-group description. The decision tree representation of the hypotheses is, however, significantly more limited than the attributional rule representation in LEM3, and is also more difficult to instantiate.

Most research in the field of evolutionary computation has been exploring various variants of the Darwinian-type of evolution that employs semi-random mutation and recombination operators to generate new individuals. Numerous papers, conference proceedings and books have been published in this very active field (e.g. Bäck, Fogel, Michalewicz, 2000; Michalewicz, 1992; Gen and Cheng, 2000; Schaefer, 2002; Bayer et al., 2005). Non-Darwinian evolutionary computation methods use techniques that differ from the Darwinian model. In particular, LEM employs hypothesis formation and instantiation operators.

The closest evolutionary methods in spirit to LEM are *cultural algorithms* (e.g. Reynolds, 1994; Peng and Reynolds, 2001; Reynolds and Peng, 2004) that use additional information about solutions to guide mutation and recombination operators. The cultural algorithms perform a constrained optimization process in which constraints are created during the evolutionary computation. The constraints, called beliefs, are stored in a belief space that is updated during the evolution process. Individuals that are stored in an optimization space are modified so that they satisfy the beliefs. The belief space is being built based on statistical information about individuals, which usually consists of intervals containing the fittest individuals.

Estimation of Distribution Algorithms (EDAs) use statistical inference and learning to generate distributions of high-performing individuals selected from one population (e.g. Mühlenbein and Paaß, 1996; Larrañaga and Lozano, 2002) without use of contrast set of low-performing individuals. Among the most popular methods for estimating distributions EDAs use Bayesian and Gaussian networks. The approach is significantly different from Learnable Evolution Model, which uses symbolic learning to distinguish between high- and low-performing individuals. EDAs also use values of fitness functions only for selecting individuals for learning, while LEM can effectively use the values during learning process (e.g. by learning significance-based descriptions; Wojtusiak, 2004b).

13 CONCLUSIONS AND PLANNED RESEARCH

The presented LEM3 system is the most advanced implementation of the Learnable Evolution Model. In many aspects, the algorithms implemented in LEM3 go beyond the methodology described in (Michalski, 2000). LEM3 has been shown to be a powerful optimization tool that wins in comparison with other evolutionary computation tools in terms of evolution length (number of fitness evaluations) and in terms of the versatility of methods for describing individuals in a population (due to the use of a wide range of attribute types supported by LEM3). An experimental application of LEM3 to very complex function optimization problems (with up to 1000 variables) achieved a superior performance over the EA method used in the experiments. It also showed high scalability that could not be achieved with previous implementations.

Our research also revealed a weakness of the current implementation of LEM3 for very large number of variables. As mentioned earlier, when the number of variables reached about 700, the advantage of LEM3 started to diminish, because in this case only few variables are needed to differentiate between the H- and L-groups. Consequently, the learning process has an increasingly lower influence on the evolutionary computation, because most of the variables are instantiated semi-randomly. Thus, the LEM process at that stage works like a variant of the Darwinian evolutionary computation.

Overcoming the problem of decreasing influence of learning at the end-phase of evolution for very large numbers of variables is therefore a major challenge for further research on the LEM methodology. While this problem is important to solve, it is worth mentioning that most practical problems have fewer than variables, thus the current method is applicable with a full advantage of learning.

Other research topics for investigation in LEM include handling complex constraints, self-adaptation in setting the LEM3 and AQ21 parameters during evolution process, multi-objective optimization, and automatic improvements of representation of individuals.

Our current research also focuses on investigation of theoretical aspects of Learnable Evolution Model, such as its complexity, convergence speed, and classification of optimization problems for which it is the most suitable.

Applications of LEM can be especially advantageous in areas in which standard evolutionary computation methods are too slow in terms of number of fitness function evaluations, particularly those in which fitness evaluation is time consuming or costly. Such areas include engineering design applications in which the computation of fitness function involves simulation or other computationally extensive processes.

REFERENCES

Bäck, T., Fogel, D.B., Michalewicz, Z. (eds.), *Evolutionary Computation 1 and 2*, Philadelphia Institute of Physics Publishing, 2000.

Beyer, H.G., O'Reilly, U.M., Arnold, D.V., Banzhaf, W., Blum, C., Bonabeau, E.W., Cantú Paz, E., Dasgupta, D., Deb, K., Foste r, J.A., de Jong, E.D., Lipson, H., Llora, X., Mancoridis, S., Pelikan, M., Raidl, G.R., Soule, T., Tyrrell, A., Watson, J.P., and Zitzler, E (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2005*, ACM Press, New York, 2005.

Bentley, J.B. and Corne, D.W. (eds.), *Creative Evolutionary Systems*, Morgan Kaufmann Publishers, 2002.

Cervone G., Kaufman K. and Michalski R.S., "Validating Learnable Evolution Model on Selected Optimization and Design Problems," *Reports of the Machine Learning and Inference Laboratory*, MLI 03-1, George Mason University, Fairfax, VA, June, 2003.

Coletti, M., Lash, T., Mandsager, C., Michalski, R. S. and Moustafa, R., "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), Orlando, July, 1999.

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

Gen, M. and Cheng R., *Genetic Algorithms & Engineering Optimization*, John Wiley & Sons, 2000.

Jourdan, L.; Corne, D.; Savic, D.; and Walters, G., “Preliminary Investigation of the ‘Learnable Evolution Model’ for Faster/Better Multiobjective Water Systems Design,” *Proceedings of The Third International Conference on Evolutionary Multi-Criterion Optimization*, EMO’05, 2005.

Kaufman, K. and Michalski, R.S., "ISHED1: Applying the LEM Methodology to Heat Exchanger Design," *Reports of the Machine Learning and Inference Laboratory*, MLI 00-2, George Mason University, Fairfax, VA, 2000a.

Kaufman, K. and Michalski, R. S., "The AQ18 System for Machine Learning and Data Mining System: An Implementation and User's Guide," Reports of the Machine Learning and Inference Laboratory, MLI 00-3, George Mason University, Fairfax, VA, 2000b.

Kerber, R. “Chimerge: Discretization for Numeric Attributes.” *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI ’92)*, AAAI Press, 1992, pp. 123-128.

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

Larson, J. and Michalski, R. S., "Inductive Inference of VL Decision Rules," Invited paper for the Workshop in Pattern-Directed Inference Systems, Hawaii, and published in SIGART Newsletter, ACM, No. 63, May 23-27, 1977, pp. 38-44.

Michalewicz, Z., Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, 1992.

Michalski, R. S., "A Planar Geometrical Model for Representing Multi-Dimensional Discrete Spaces and Multiple-Valued Logic Functions," *Report No. 897*, Department of Computer Science, University of Illinois, Urbana, January 1978.

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

Michalski, R. S., "Generating Alternative Hypotheses in AQ Learning," *Reports of the Machine Learning and Inference Laboratory,* MLI 04-6, George Mason University, Fairfax, VA, December, 2004b.

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

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

Michalski, R.S. and Wojtusiak, J., "Reasoning with Meta-Values in AQ Learning," Reports of the Machine Learning and Inference Laboratory, MLI 05-1, George Mason University, 2005.

Michalski, R.S. and Wojtusiak, J., "Semantic and Syntactic Attribute Types in AQ Learning," *Reports of the Machine Learning and Inference Laboratory*, George Mason University, 2005, to appear.

Michalski, R.S. and Zhang, Q., "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, May 1999.

Mühlenbein, H and Paaß, G., "From recombination of genes to the estimation of distributions I. Binary parameters," *Proceedings of The 4th International Conference on Parallel Problem Solving from Nature*, Berlin, Germany, September 22-26, 1996.

Oyama, A., "Wing Design Using Evolutionary Algorithms," Ph.D. Thesis, Department of Aeronautics and Space Engineering of Tohoku University, 2000.

Saleem, S. and Reynolds, R.G., "Function optimization with cultural algorithms in dynamic environments," *Proceedings of the Workshop on Particle Swarm Optimization 2001*. Purdue School of Engineering and Technology, Indianapolis, IN, 2001.

Simionescu, P.A., D.G. Beale and G.V. Dozier (2004) "Constrained Optimization Problem Solving Using Estimation of Distribution Algorithms," *Proceedings of 2004 IEEE Congress on Evolutionary Computation*, Portland, OR, June 20-23, 2004.

Quinlan, J. R., *C4.5: Systems for Machine Learning*, Morgan Kaufmann Publishers Inc., 1993.

Reynolds, R.G., “An Introduction to Cultural Algorithms,” *Proceedings of the Third Annual Conference on Evolutionary Programming*, 1994.

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

Schaefer, R., *Podstawy Genetycznej Optymalizacji Globalnej,* Wydawnictwo Uniwersytetu Jagiellonskiego, Krakow, 2002.

Whitley, D., Soraya, R., Dzubera, J., Mathias, K. E., “Evaluating evolutionary algorithms,” *Artificial Intelligence*, 85, 1996.

Wnek, J., Kaufman, K., Bloedorn, E. and Michalski, R. S., "Inductive Learning System AQ15c: The Method and User's Guide," *Reports of the Machine Learning and Inference Laboratory*, MLI 96-6, George Mason University, Fairfax, VA, August, 1996.

Wojtusiak, J., “AQ21 User’s Guide, ” *Reports of the Machine Learning and Inference Laboratory*, George Mason University, MLI 04-3, Fairfax, VA, 2004a.

Wojtusiak, J., “The LEM3 Implementation of Learnable Evolution Model: User’s Guide,” *Reports of the Machine Learning and Inference Laboratory*, George Mason University, MLI 04-5, Fairfax, VA, 2004b.

A publication of the *Machine Learning and Inference Laboratory*
School of Computational Sciences
George Mason University
Fairfax, VA 22030-4444 U.S.A.
http://www.mli.gmu.edu

Editor: R. S. Michalski
Assistant Editor: K. A. Kaufman

The *Machine Learning and Inference (MLI) Laboratory Reports* are an official publication of the Machine Learning and Inference Laboratory, which has been published continuously since 1971 by R.S. Michalski's research group (until 1987, while the group was at the University of Illinois, they were called Intelligent Systems Group (ISG) Reports).