

Reports

Machine Learning and Inference Laboratory

**The LEM3 Implementation of
Learnable Evolution Model
User's Guide**

Janusz Wojtusiak

MLI 04-5

P 04-7

November, 2004



School of Computational Sciences

George Mason University

THE LEM3 IMPLEMENTATION OF LEARNABLE EVOLUTION MODEL

USER'S GUIDE¹

Janusz Wojtusiak

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

jwojt@mli.gmu.edu <http://www.mli.gmu.edu/jwojt/>

also

Institute of Computer Science, Jagiellonian University, Krakow, Poland

Abstract

This report presents a User's Guide for the LEM3 program implementing Learnable Evolution Model, a machine learning-guided form of evolutionary computation. In contrast to the previous implementations, LEM3 employs the newest AQ21 machine learning program and includes several extensions and new features. The new features include a deep integration of machine learning and conventional evolution modes, capabilities for defining the fitness evaluation function outside of the main program, setting up LEM3 parameters, and deciding what information to be outputted by the program. LEM3 appears to be particularly suitable for solving complex optimization problems in which the fitness evaluation function is time-consuming. Another important feature is that it can be applied to the optimization of non-numerical problems involving multitype attributes.

Keywords: machine learning, evolutionary computation, learnable evolution model, function optimization, engineering design.

Acknowledgments

A number of people helped with LEM3 program and this manual. Work on the LEM project is constantly supervised by Professor Ryszard S. Michalski who also invented the LEM algorithm and a number of its extensions. He also provided comments and corrections to this document. Dr. Kenneth Kaufman, Dr. Bartłomiej Sniezynski, and Jarek Pietrzykowski reviewed this manual and provided useful comments about the program and LEM methodology.

Research activities in the Machine Learning and Inference Laboratory are supported in part by the National Science Foundation Grants No. IIS 9906857 and IIS 0097476, and in part by the UMBC/LUCITE #32 grant.

¹ User's Guide based on LEM3 version 2004-11-10

1	INTRODUCTION	1
2	RUNNING THE LEM3 PROGRAM	1
2.1	OPTIMIZATION PROBLEM DEFINITION.....	3
-f	<i>name of a predefined fitness function</i>	3
-att	<i>number of attributes</i>	5
-lb	<i>lower bound</i>	5
-ub	<i>upper bound</i>	5
2.2	LEM3 ALGORITHM PARAMETERS.....	6
-p	<i>population size</i>	6
-c	<i>number of children</i>	6
-g	<i>number of generations</i>	6
-h	<i>high threshold</i>	6
-l	<i>low threshold</i>	7
-lp	<i>learning probe</i>	7
-mp	<i>mutation probe</i>	7
-dp	<i>discretization probe</i>	8
-sp	<i>startover probe</i>	8
-mr	<i>mutation ratio</i>	8
-sel	<i>selection method</i>	9
-surv	<i>survival selection method</i>	9
-dist	<i>distribution</i>	9
-nodiscretize	10
-descr	<i><description type></i>	10
2.3	EXPERIMENT PARAMETERS	10
-i	<i>input file name</i>	10
-runs	<i>number of runs</i>	11
-sr	<i>number of significant runs</i>	11
-r	<i>random seed</i>	12
2.4	LEM3 OUTPUT PARAMETERS	12
-o	<i>output file name</i>	12
-aq	12
-kv	13
-pp	13
-pni	13
-ph	13
3	OUTPUT FROM LEM3	14
3.1	DEFAULT OUTPUT.....	14
3.2	OPTIONAL OUTPUT	18
4	DEFINING THE FITNESS FUNCTION	19
4.1	SIMULATOR.....	19
4.2	DEFINITION OF FITNESS FUNCTION IN THE LEM3 SOURCE CODE	20
4.2.1	<i>Class compute_significance</i>	20
4.2.2	<i>Defining Your Own Fitness Function</i>	22
5	CONCLUSIONS AND FUTURE EXTENSIONS OF LEM3	23
	REFERENCES	23

1 INTRODUCTION

The Learnable Evolution Model (LEM) is a modern approach to evolutionary computation that combines evolutionary algorithms with machine learning. Instead of applying semi-random Darwinian operators such as mutation and recombination, LEM applies a machine learning program to determine characteristic features of better-performing individuals that do not appear in worse-performing individuals. These features, learned in the form of attributional rulesets, are used to generate new individuals in an instantiation process.

LEM3 is an implementation of the Learnable Evolution Model that uses AQ21 as a learning module. A deep integration of AQ21 within the system makes LEM3 more efficient than previous implementations, and allows the user to specify a description of individuals in the population using a wide range of attribute types available in the learning program. These attribute types include *nominal*, *ordinal*, *interval*, *ratio* and *structured*. The ratio (or continuous) attributes can be used directly, or after discretization using the ANCHOR algorithm (Michalski and Cervone, 2001).

This manual serves a guide for using the program and it is not intended to provide a description of the method or algorithms involved. For the latter, the reader is referred to a LEM3 technical report (Wojtusiak, 2004b) available on the website of the Machine Learning and Inference Laboratory <http://www.mli.gmu.edu>, from which the LEM3 program can also be downloaded. Detailed information about the AQ21 machine learning program can also be found in the MLI website (Wojtusiak, 2004a).

2 RUNNING THE LEM3 PROGRAM

LEM3 is a command-based program that can be executed under Linux or Windows environments. LEM3 does not require any installation, and only the executable binary file is needed to run the program and perform experiments. Computational power required for LEM3 depends on the optimization problem. When computation of the fitness function is not time-consuming, LEM3 can be run on a personal computer with hundreds of attributes and for thousands of iterations.

For illustration LEM3 performance, Table 1 presents the execution time and maximum memory usage in experiment on optimizing the Rastrigin function with different numbers of variables. In this example, the program was not run with the optimal parameter settings, so the presented memory usage and computational time are only rough estimates. The program was run on DELL PowerEdge 4600 with Intel Xeon 2 GHz processors running the Linux Red Hat 7.3 operating system.

# of attributes	Population size	# of iterations (generations)	time of execution	Maximum memory usage
10	100	100	<1 s	1.3 MB
100	100	100	21 s	6.7 MB
200	150	200	222 s	22 MB
500	300	500	5130 s	110 MB

Table 1: Examples of execution time and memory usage of LEM3 in optimizing Rastrigin Function with 10, 100, 200, and 500 variables

In both Linux and Windows operating systems LEM3 is executed from the shell as shown below.

To run the LEM3 under Windows OS (95 or higher) open the command prompt window and type:

```
lem3 [parameters]
```

To run the LEM3 under Linux OS (Red Hat or other) type in the shell:

```
./lem3 [parameters]
```

where [parameters] stands for a list of possible options described later in this document. The following example shows command that executes LEM3 with selected parameters. Other parameters are listed later in this document.

Example:

```
./lem3 -p 100 -c 100 -g 300 -f rastrigin -att 100
      -runs 10 -o myfile -ph
```

-p indicates that the following number is the population size, so the population size is in this case 100.

-c indicates that the following number is the number of children, so the number of children is in this case 100.

-g indicates that the following number is the maximum number of generations, so the maximum number of generations is in this case 300.

-f indicates that the following name specifies the optimization fitness function to be used, so the fitness function is in this case the predefined Rastrigin function.

-att indicates that the following number is the number of attributes, so the number of attributes is in this case 100.

-runs indicates that the following number is the number of executions of program with different random seeds and the same parameters; so the number of executions is in this case 10.

-o indicates that the following name is the name of the output file to be generated; so the filename is in this case *myfile*.

-ph means that program prints hypotheses learned by AQ21 module in all iterations.

As seen in the example above parameters are passed to the program in form

-param val

where *param* is the parameter name and *val* is the (optional) value of the parameter. For instance, `-att 100` or `-pp`. The following sections of this manual describe in detail all LEM3 parameters.

2.1 Optimization Problem Definition

-f *<name of a predefined fitness function>*

Default: Rosenbrock

This parameter is used to set a fitness function from a collection of predefined functions, or to specify a user-defined fitness function. The predefined functions are *rosenbrock*, *sphere*, *rastrigin*, *step*, *gauss*, *user*, and *simulator*.

To use a new function, the user has to define it in the *sigfunct.cpp* file (see the description of user “fitness function” in Section 4.2).

It is also possible to use an external simulator to evaluate individuals (solutions). Such a simulator has to be a computer program, written in any language convenient to the user, whose executable is named *simulator* and placed in the LEM3 directory. LEM3 writes a text file named *simulator.inp* that contains the definition of one individual to be evaluated, and runs the simulator. The simulator returns an output file named *simulator.out* containing the value of the fitness function. The input file for the simulator contains a description characterizing a single individual to be evaluated, expressed in the form of a comma-separated list of values of all attributes. For details about the simulator program, file structures and simple simulator program written in C language, please refer to Section 4.1 of this guide.

Example:

```
./lem3 -p 100 -c 100 -g 200 -att 100 -f simulator
```

LEM3 waits for the computation of fitness values for each individual, and the simulator cannot be executed in parallel on different individuals.

The predefined functions, mentioned above, are well-known benchmark functions that can be used to illustrate LEM3 operation, and to compare LEM3’s performance with that of other programs on the same functions. These functions are briefly reviewed below.

A. Rosenbrock function. This is a widely-used in testing evolutionary algorithms. It presents a difficult optimization problem because its arguments (input variables) are highly correlated. The closed-form expression of the function is:

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

A plot of the function for $n=2$ is presented in Figure 1. Given a specific value of n , LEM3 seeks the maximum of the expression $1/(\text{Rosenbrock} + 1)$, which is equivalent to seeking the minimum of the Rosenbrock function.

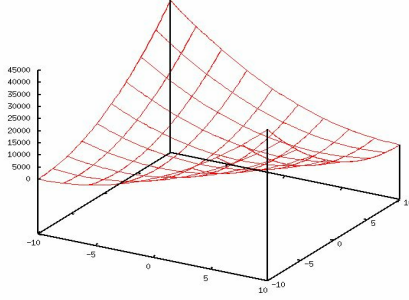


Figure 1: Plot of the Rosenbrock function

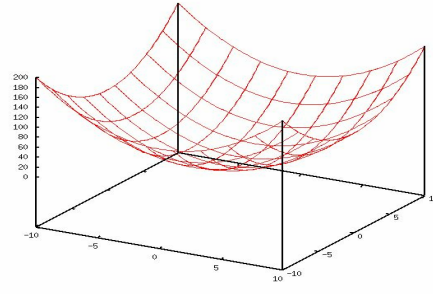


Figure 2: Plot of the Sphere function

B. Sphere function. This function presents a less complex optimization problem than Rosenbrock function, and is also widely used in testing evolutionary algorithms. A closed-form expression of the function is:

$$f(x_1, \dots, x_n) = \sum_{i=1}^n x_i^2$$

A plot of the function for $n=2$ is presented in Figure 2. LEM3 seeks for maximum of $1/(\text{sphere} + 1)$, which is equivalent to seeking the minimum of the sphere function.

C. Rastrigin function. This function is also widely used in testing evolutionary algorithms. It has a large number of local optima caused by the cosine factor in the formula shown below. A plot of the Rastrigin function with two variables is presented in Figure 3.

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

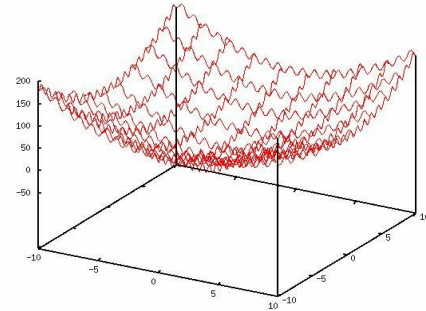


Figure 3: Plot of the Rastrigin function

D. Step function. This is the simplest example of a discontinuous function used on testing evolutionary algorithms.

$$f(x_1, \dots, x_n) = \sum_{i=1}^n \text{round}(x_i)$$

where $\text{round}(x)$ is value of attribute x rounded to the nearest integer value.

E. Gaussian quadratic function. This is an example of a fitness function whose value changes over time by a random factor. It is defined by the formula below.

$$f(x_1, \dots, x_n) = \sum_{i=1}^n i * x_i^4 + gauss()$$

where `gauss()` is function that returns a random number selected using a standard normal distribution.

-att *<number of attributes>*

Default: 10

This parameter defines the number of attributes of the optimized function. All parameters are defined in the same continuous domain whose upper and lower bounds are defined using `-ub` and `-lb` parameters.

Example:

```
./lem3 -p 100 -c 100 -f rosenbrock -att 100
```

-lb *<lower bound>*

Default: -5

Sets a lower bound for attribute domains.

Example:

```
./lem3 -p 100 -c 100 -g 200 -att 100 -lb -3.24
```

-ub *<upper bound>*

Default: 5

Sets an upper bound for attribute domains.

Example:

```
./lem3 -p 100 -c 100 -g 200 -att 100 -ub 7.11
```

Please note that in the case of the `-lb` and/or `-ub` parameters being used, domains of all attributes must be the same (all continuous with the same upper and lower bounds).

2.2 LEM3 Algorithm Parameters

-p *<population size>*

Default: 100

This parameter sets the population size – the number of individuals in one generation. In the current implementation of LEM3, this number is constant during the optimization process. The value of this parameter must be greater than zero.

Example:

```
./lem3 -p 200 -c 100 -g 300 -f rosenbrock
```

-c *<number of children>*

Default: 100

This parameter sets the number of individuals generated during the hypothesis instantiation process. Instantiation is done just after the AQ21 program creates rules characterizing high-performing individuals versus those with low fitness function values. The value of this parameter must be an integer greater than zero.

Example:

```
./lem3 -p 100 -c 200 -g 300 -f rosenbrock
```

-g *<number of generations>*

Default: 100

Number of generations is equivalent to number of iterations of the LEM3 main loop. However this is a maximal value, because the program may stop when the stop condition is met (please refer to *-lp -mp -dp -sp* parameters). The value of the parameter must be greater than zero.

Example:

```
./lem3 -p 100 -c 100 -g 300 -f rosenbrock
```

-h *<high threshold>*

Default: 0.3

-l <low threshold>**Default:** 0.3

High threshold and low threshold are parameters that control the selection of individuals into the H-group and L-group respectively. Both parameters must be between zero and one. It is also required that sum of these thresholds does not exceed 1, in order not to introduce ambiguity into the AQ21 learning module (the same individuals would be both positive and negative examples for learning). For a detailed description of the selection methods, please refer to `-sel` parameter described below.

Example:

```
./lem3 -p 100 -c 100 -g 300 -f rosenbrock -h 0.3
      -l 0.2
```

-lp <learning probe>**Default:** 10

Learning probe is a parameter that defines how many iterations the program can fail to make evolutionary progress before additional actions are taken. The actions (mutation, adjust discretization, startover) are defined in the *no-progress condition* definition, and are additionally controlled by the `-mp` `-dp`, and `-sp` parameters. The value of the `-lp` parameter must be a nonnegative integer.

Example:

```
./lem3 -p 100 -c 100 -g 300 -f rosenbrock -lp 6
```

-mp <mutation probe>**Default:** 10

This parameter defines the maximum number of consecutive mutation operations with no progress before the program switches to next operation defined in the *no-progress condition*. The value of this parameter must be a nonnegative integer (zero means that the mutation is disabled). Please note that in LEM3, the goal of mutation is not to lead the evolution process, but rather to introduce diversity into the population and provide the AQ21 learning module with a more diverse set of examples.

Example:

```
./lem3 -p 100 -c 100 -g 300 -f rosenbrock -mp 20
```

-dp <*discretization probe*>**Default:** 2

This parameter defines the maximum number of adjust discretization operations with no progress, after which the program switches to next operation defined in the no-progress condition. The value of this parameter must be a nonnegative integer (zero means that the operation is disabled).

Example:

```
./lem3 -p 100 -c 100 -g 300 -f rosenbrock -dt 4
```

-sp <*startover probe*>**Default:** 10

This parameter defines the maximum number of startover operations when there is still no progress. It is part of the no-progress condition. Before running the startover operation, the program adds the current result (the best individual in the current population) to its list of results, in order not to lose the local optima found so far. When the number of startover operations exceeds the *startover probe* parameter, the global LEM3 stop condition is met. The value of this parameter must be a nonnegative integer.

Example:

```
./lem3 -p 100 -c 100 -g 300 -f rosenbrock -sp 4
```

-mr <*mutation ratio*>**Default:** 0.1

The mutation operation changes randomly values of attributes. For each attribute, the uniform distribution is used, and the value is changed by no more than

$$\text{mutation ratio} / \text{domain range}$$

where domain range is difference between the largest and the smallest possible value in the domain, as defined by *-ub* and *-lb* parameters respectively.

Example:

```
./lem3 -p 100 -c 100 -g 300 -f rosenbrock -mr 0.2
```

-sel <selection method>**Default:** pop

This parameter defines the method used for selecting individuals as members of the H-group and L-group. There are two possible values of the parameter: *fit* and *pop*, which represent *fitness-based* and *population-based* selection respectively.

The *population-based* method selects the H-group by taking the fraction of the current population defined by *high threshold* with the best fitness. Similarly, it takes the *low threshold* fraction of the worst individuals into the L-group. It should be noted that the number of individuals selected into the H- and L-groups under a population-based method depends only on the population size and the thresholds.

The profile of the fitness function on the current population is reflected in the *fitness-based* method, which selects into H-group individuals whose fitness values are within the high threshold portion of the range of fitness values in the current population. Similarly, the L-group consists of individuals whose fitness values are in the low end of the current population's fitness range, as defined by the low threshold tolerance.

Example:

```
./lem3 -p 100 -c 100 -g 300 -f rosenbrock -sel pop
```

-surv <survival selection method>**Default:** rank

This parameter defines the survival selection method used to create a new population from the old population and the new individuals instantiated from learned hypotheses. Possible values are *rank* and *tournament* for respectively rank-based and tournament selections.

The rank-based selection selects the best individuals according to the fitness function. The tournament method randomly selects two individuals and the better of the two is copied into the new population. This operation is repeated until the new population has the desired number of individuals (as controlled by the population size parameter).

Example:

```
./lem3 -p 100 -c 100 -f rosenbrock -sel tournament
```

-dist <distribution>**Default:** uniform

This parameter defines the distribution method used in the rule instantiation process. It is used to select randomly values of an attribute within the area covered by a rule. The possible values are *uniform* and *projections* representing respectively a uniform distribution and a

distribution based on frequencies of individuals projected onto the attribute being instantiated.

Example:

```
./lem3 -p 100 -c 100 -f rosenbrock -dist projections
```

-nodiscretize

By default LEM3 uses the ANCHOR algorithm to discretize values of attributes. It is, however, possible to use a continuous representation of attributes and pass undiscretized examples into the AQ21 learning module.

Example:

```
./lem3 -p 100 -c 100 -att 100 -f gauss -o test -nodiscretize
```

-descr <description type>

Default: discr

This parameter controls the type of descriptions learned by the AQ21 program. Three values char, discr, and sim are accepted by program and stand for characteristic descriptions, discriminant descriptions, and simplicity-based descriptions, respectively.

Example:

```
./lem3 -p 100 -c 100 -att 100 -f gauss -descr char
```

2.3 Experiment Parameters

-i <input file name>

Default: no input file is used

This option is used to define the input file from which program loads an initial population. The input file consists of comma-separated values of attributes of individuals, each individual on a separate line. There should be no comma at the end of each line (after the last value for an individual). The number of individuals need not agree with the population size, as the program will generate/truncate individuals if needed. The program also permits comments in the input file. All lines starting with “#” are ignored, and all sections between “(“ and “)” are ignored (thus, comments can be more than one line in length).

Example:

```
./lem3 -att 10 -p 100 -c 100 -f rosenbrock -i input.lem
```

An example of an input file is presented below.

```
-4.5837, -0.455076, 3.34817, -1.64014, 0.654894, -4.98233, -3.1241, 4.90434,
2.50497, -1.33726
-1.48791, 0.733451, -3.67446, -4.35834, 4.50854, -3.4644, 0.846494, -2.83412,
3.06502, -3.59527
1.22059, -2.89107, -4.93422, 0.732986, 4.32664, -1.59673, 3.91081, 0.938726,
-1.07104, 3.99323
1.94994, -2.71397, 4.62459, -4.87946, -3.89043, 3.84096, -3.83244, 2.50923,
-2.03066, 1.46692
-0.734201, 0.00941012, -1.85423, -0.632157, 1.60036, 2.02546, 2.74809,
3.13267, -1.80185, 4.82327
1.79633, -3.49025, 3.76027, 1.96759, -1.22634, 0.648352, -0.250078, -2.27509,
4.39328, -2.40939
0.401839, 1.4778, 0.697754, 0.0956528, -2.96702, -4.11424, -2.19132, 2.04665,
2.59929, 1.7062
```

Note that in the shown example input file, there are seven individuals, each having ten attributes.

-runs <number of runs>

Default: 1

This parameter sets the number of experiments to be run for the same program settings, but with different random seeds. The random seeds are generated from the given (or default) random seed by adding 1 to the value. Such operations are useful when the user wants to run the program with different random seeds and compare the results (e.g., for testing purposes).

Example:

```
./lem3 -p 100 -c 100 -f rosenbrock -runs 10
```

-sr <number of significant runs>

Default: 1

This parameter defines the number of significant runs. In one of the output tables, LEM3 displays the average of the n best runs (where n is specified by this parameter).

Example:

```
./lem3 -p 100 -c 100 -att 100 -runs 10 -sr 7
```

-r **<random seed>**

Default: 1

This parameter defines a random seed that is used to initialize the random number generator.

Example:

```
./lem3 -p 100 -c 100 -f rosenbrock -r 12345
```

2.4 LEM3 Output Parameters

-o **<output file name>**

Default: no output is generated

LEM3 generates an output file, whose name is specified by this parameter. The file contains summaries that are also displayed to the screen. For details of the contents of the output file, please refer to Section 3.

Example:

```
./lem3 -p 100 -c 100 -att 100 -f gauss -o test.res
```

-aq

This parameter instructs LEM3 to generate the AQ21 input files using the current settings and the current H- and L-groups as examples. This makes it possible to repeat the learning process using the standalone AQ21 program, possibly using different learning parameters. For details of the file format please refer to the AQ21 User's Guide.

Example:

```
./lem3 -p 100 -c 100 -att 100 -f gauss -o test -aq
```

-kv

This option is used to generate input files for Knowledge Visualizer (KV) program that presents in graphical form populations and learned rules using Generalized Logic Diagrams (GLDs). The files are in AQ19 format. LEM3 will generate the files for all iterations, and their names are defined by the following key:

`_random seed_<iteration>.kv`

where *random seed* is the random seed currently being used for the program execution (see the `-r` option), and *iteration* is the number of the iteration in which the file was generated.

Example:

```
./lem3 -p 100 -c 100 -att 100 -f gauss -o test -kv
```

-pp

This option enables output of the population in all iterations of LEM3 execution. Please note that for large population sizes and large numbers of attributes, output from LEM3 under this option will be very large. For a description of the format in which the individuals are displayed, please refer to section 3.2 of this document.

Example:

```
./lem3 -p 100 -c 100 -att 100 -f gauss -pp
```

-pni

This option enables the output of all new individuals instantiated in all iterations of LEM3 execution. Please note that for large population sizes and large numbers of attributes, this will cause the output from LEM3 to be very large.

Example:

```
./lem3 -p 100 -c 10 -att 10 -f gauss -pni
```

-ph

This option enables printing of hypotheses (rules) learned by AQ21 during the optimization process. For description of the format in which rules are printed, please refer to the output section of this document (Section 3.2).

Example:

```
./lem3 -p 100 -c 100 -att 100 -f gauss -ph
```


3 OUTPUT FROM LEM3

During execution LEM3 generates output that contains results of optimization and statistical summaries. The information is printed to default output, and depending on the provided parameters may be also included in output files.

This section presents in detail the output generated by the LEM3 program.

3.1 Default Output

The standard output starts with information about parameters used to run experiment. This includes the time at which the program was started, information about the function being optimized and LEM3 algorithm parameters. Example of this part of output is presented in Figure 4. The rest of this section is based on an example of optimization of the Rosenbrock function with 10 variables.

```

Starting at: Tue Aug 24 13:44:29 2004

Function Parameters:
Function                = rosenbrock
Number of Attributes    = 10
Lower Bound             = -5
Upper Bound            = 5
Discretize              = true

Algorithm Parameters:
Population Size         = 100
Number of Children     = 100
Number of Generations  = 100
High Threshold         = 0.3
Low Threshold          = 0.3
Learning Probe         = 4
Mutation Probe         = 10
Discretization Probe   = 2
Startover Probe       = 10
Mutation Ratio         = 0.1
Selection Mode         = population-based
Survival Selection     = rank-based
Distribution            = uniform
Descriptions           = discriminant

Run parameters:
Number of Runs         = 1
Number of Significant  = 1
Random Seed           = 1

Display Parameters:
Print Population       = false
Print New Individuals  = false
Print Hypotheses      = false

```

Figure 4: LEM3 parameters displayed in the output

During the optimization process, LEM3 displays the best value of the fitness function already found and other important information, such as “NO PROGRESS DETECTED”. An example of this part of the output is presented below in Figure 5.

```

running LEM ...
Initializing discretization
test_0, iteration: 0
Best result (iteration 0 ): 0.000140449
...
test_0, iteration: 4
Best result (iteration 4 ): 0.000620347
NO PROGRESS DETECTED!
...
test_0, iteration: 16
Best result (iteration 16 ): 1
NO PROGRESS DETECTED!
Total number of births: 1848
Results found: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

```

Figure 5: Example of information displayed during the optimization process

After the optimization process is complete, LEM3 displays summaries of the performed experiments. It starts with a list of results obtained from all runs of LEM3. The displayed numbers show values of variables in the best individuals (Figure 6). There can be more than one individual output per run, for instance when the program has found a number of equivalent solutions, or the startover operation has been used). The first part of each individual consists of a list of attribute values. These are followed by three numbers internally used by AQ21 learning: *frequency*, *value of fitness* (in AQ21 called *significance*), and *key* of the individual. The *frequency* and *key* are not used by LEM3 and should be ignored (they are always equal to 1 in LEM3).

```

Solutions for 0 run:
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

Solutions for 1 run:
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0.1, 1

Solutions for 2 run:
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

```

Figure 6: List of the best solutions found

In the example above, the program found correct solution in runs 0 and 2, and in run 1 it stopped optimization in point 0, ..., 0 (for the Rosenbrock function, the optimal solution is 1, ..., 1).

After the list of solutions, LEM3 displays a list of the best fitness values achieved by program for all iterations and for all runs. The example presented in Figure 7 shows this part of the LEM3 output.

1	0.0001404	0.0001084	0.0002072
2	0.0001404	0.0002313	0.0005498
3	0.0006203	0.0002313	0.0005498
4	0.0006203	0.0003667	0.0005498
5	0.0006203	0.0007616	0.001238
6	0.001653	0.001086	0.004926
7	0.002451	0.001408	0.004926
8	0.003289	0.002445	0.004926
9	0.009709	0.003226	0.004926
10	0.009709	0.009174	0.00495
11	0.009804	0.009615	0.009804
12	0.1	0.1	1
13	0.1	0.1	1
14	0.1	0.1	1
15	0.1	0.1	1
16	0.1	0.1	1
17	1	0.1	1

Figure 7: Values of fitness function for each iteration

The next three tables displayed in the program output show the fitness value of the best run, the average fitness for all runs, and values from the n best runs (where n is controlled by the `-sr` parameter). For each run value of the fitness for the best individual is used. The first column in the tables represents the number of fitness evaluations (averaged over all runs), and the associated best fitness value is shown in the second column.

# births	best fitness
0	0.0002072
92	0.0002072
184	0.0002072
277	0.0005498
369	0.0006203
462	0.0006203
554	0.001238
646	0.004926
739	0.004926
831	0.004926
924	0.009709
1016	0.009709
1108	0.009804
1201	1
1293	1
1386	1
1478	1
1570	1
1663	1
1755	1
1848	1
# births	average fitness
0	0.000152
92	0.000152
184	0.000152
277	0.0003072
369	0.0004671
462	0.0005123

554	0.0008732
646	0.002555
739	0.002929
831	0.003554
924	0.005954
1016	0.007945
1108	0.009741
1201	0.4
1293	0.4
1386	0.4
1478	0.4
1570	0.4
1663	0.4
1755	0.7
1848	0.7
# births	n of m
0	0.0001738
87	0.0001738
175	0.0001738
262	0.0003451
350	0.0005851
438	0.0005851
525	0.000929
613	0.00329
701	0.003689
788	0.003689
876	0.004108
964	0.007317
1051	0.00733
1139	0.009804
1227	0.55
1314	0.55
1402	0.55
1490	0.55
1577	0.55
1665	1
1753	1

Figure 8: Summary of results displayed by program

At the end of the standard output, LEM3 displays the time of the end of execution and the total execution time, as shown in Figure 9.

```
Finished at: Tue Jun 29 18:16:13 2004
Total time : 3 seconds
BYE!
```

Figure 9: End of LEM3 output

3.2 Optional Output

Optional output from LEM3 is selected through parameters specified by user. As described in Section 2.4, the user can define what is printed into LEM3 output. This includes populations, newly created individuals and hypotheses learned by AQ21. The population is a set of individuals, so it is printed in the same format as the results described in Section 3.1. The format in which rules generated by AQ21 program are output in LEM3 (when *-ph* parameter is used) is described below.

The AQ21 program learns attributional decision rules as presented in example in Figure 10.

```
Rules:
  <-- [x3=-0.500000..0.499900 : 17,10]
      [x7=-0.500000..0.499900 : 26,19]
      [x8=-0.500000..0.499900 : 26,20]
      : p=17,np=17,u=15,cx=21,c=1,s=1.52 # 14031
  <-- [x0=0.500000..1.499900 : 11,6]
      [x8=-0.500000..0.499900 : 26,20]
      [x9=-0.500000..0.499900 : 26,17]
      : p=11,np=9,u=9,cx=21,c=1,s=0.104 # 14032
  <-- [x0=-0.500000..0.499900 : 19,18]
      [x2=0.500000..1.499900 : 15,21]
      [x9=0.500000..1.499900 : 4,12]
      : p=4,np=4,u=4,cx=21,c=1,s=0.0391 # 14030
```

Figure 10: Examples of rules learned by AQ21 during the optimization process preformed by LEM3

The consequent part of learned rules can be ignored because it is always the same in LEM3, specifically, [group=H], denoting that the rules describe high-performing individuals (i.e., the H-group). Because of that, each rule starts with “<--“ sign followed by a conjunction of attributional conditions (a.k.a. selectors). For instance, the first rule in the example above is a conjunction of three selectors. The first condition covers 17 positive and 10 negative examples (individuals) and it says that x_3 is between -0.5 (inclusive) and 0.5 (exclusive). The summary after each rule consists of the following information: p – number of positive examples covered, np – number of positive examples newly covered, u – number of positive examples uniquely covered (covered by this rule and no others), cx – complexity, s – significance of the rule. Although the information is printed, the user can ignore it since it is mostly irrelevant to the evolution process. A detailed explanation of all parameters printed with hypotheses and details about AQ learning can be found in the AQ21 User’s Guide (Wojtusiak, 2004a).

4 DEFINING THE FITNESS FUNCTION

This section describes two different ways that LEM3 users can define their own fitness functions. Both methods require basic programming skills, and one of them requires an installed C++ compiler. The first method uses an external program to compute the fitness function and communicates with LEM3 through text files. In the second method, the function that computes the fitness function is linked with LEM3 into one executable file.

4.1 Simulator

LEM3 allows an external program to evaluate individuals. The external program, called *simulator*, reads one individual at a time and returns its fitness value. Communication between LEM3 and the simulator is done through text files. The simulator input file is called “simulator.inp” and contains a comma-separated list of attribute values (Figure 11).

```
3.12, -4.01, -2.0004, -4.243, -4, 2, 0.12
```

Figure 11: Example of the simulator.inp file

Output from the simulator consists of the value of the fitness function, and is written into “simulator.out” file. An example of this output is presented in Figure 12.

```
2.345607
```

Figure 12: Example of the simulator.out file

Executable simulator, input, and output files are stored in the same directory as the LEM3 executable. Figure 13 presents an example of a simulator written in C++. This simulator computes the value of the Rastrigin function.

```

#include <stdio.h>
#include <math.h>

main()
{
    // open the input file
    FILE *fin = NULL;
    fin = fopen( "simulator.inp", "r" );
    if( fin == NULL ) // input file not opened
        return 1;

    float val;
    float re = 0; // returned value
    int n = 0;

    // read the input file
    while( feof( fin ) == 0 )
    {
        val = 0;
        fscanf( fin, "%f,", &val );
        re += val*val - 10*cos( 2*3.14*val);
        n++;
    }

    re += 10*n;
    fclose( fin );

    FILE *fout = NULL;
    fout = fopen( "simulator.out", "w" );
    fprintf( fout, "%f", re );
    fclose( fout );
}

```

Figure 13: Source code of the simulator program that computes the Rastrigin function

4.2 Definition of Fitness Function in the LEM3 Source Code

Using the simulator, as described in the previous section, has limitations due to the inefficiency of the input/output operations. It may therefore be desirable to link the function that computes fitness with LEM3 into one executable file. Such a function should be written in the `sigfunct.cpp` file that can be linked with LEM3. Data structures used for the definition are described below. LEM3 was compiled and tested using the GNU C++ 2.96 compiler under Linux OS and using Borland C++ Builder 5.0 under Windows OS.

4.2.1 Class `compute_significance`

Every fitness function is represented in LEM3 by a class that is derived from class `compute_significance`. The definition of the class is presented in Figure 14.

```

class compute_significance
{
public:
    virtual double compute( Event const & );
};

```

Figure 14: Definition of class `compute_significance` in the LEM3 source code

The class contains only one virtual function that computes the significance function.

The following code presents examples of significance functions that are defined in the system.

```
Example1 - sphere

class sphere_significance : public compute_significance
{
    double compute( Event const & );
};

double sphere_significance::compute( Event const & event )
{
    int i;
    double re;
    double val;

    re = 0;
    for( i = 0; i < event.positions()->size(); i++ )
        if( event.check_value( i, val ) == true )
            {
                re += (val) * (val);
            }
        else
            cout << "error computing significance" << endl;

    return 1 / ( 1 + re);
};
```

Figure 15: Definition of the Sphere function in LEM3 source code

```
Example 2 - Rosenbrock's function

class rosenbrock_significance : public compute_significance
{
    double compute( Event const & );
};

double rosenbrock_significance::compute( Event const & event )
{
    int i;
    double re;
    double val;
    double vall;
    re = 0;
    event.check_value( 0, vall );
    for( i = 0; i < event.positions()->size() - 1; i++ )
        if( event.check_value( i + 1, val ) == true )
            {
                re += 100 * ( val - vall ) * ( val - vall )
+ ( vall - 1 ) * ( vall - 1 );
                vall = val;
            }
        else
            cout << "error computing significance" << endl;

    return 1 / ( re + 1 );
};
```

Figure 16: Definition of the Rosenbrock function in LEM3 source code

Example 3 - step function

```

class step_significance : public compute_significance
{
    double compute( Event const & );
};

double step_significance::compute( Event const & event )
{
    int i;
    double re;
    double val;

    re = 0;
    for( i = 0; i < event.positions()->size(); i++ )
        if( event.check_value( i, val ) == true )
            {
                re += (int)val;
            }
        else
            cout << "error computing significance" << endl;

    return re;
};

```

Figure 17: Definition of the Step function in LEM3 source code

4.2.2 Defining Your Own Fitness Function

To define a fitness function, the user needs to complete the *compute* function in class *user_significance* located in *sigfunct.cpp* file analogically to the definitions of presented functions. Compiling the file and linking with LEM3 are done automatically after typing the “make” command (all makefiles are prepared) when using the GNU compiler, or by selecting option “make” from the Project menu in Borland C++ Builder.

In the AQ21 data structures used in LEM3, an individual is represented by “event” whose most important function is *check_value* as shown in Figure 17. The first parameter of the function is an integer representing the number of the attribute, and the second parameter is a double, to which the value of the attribute is assigned. The function returns *true* if the operation is successful.

5 CONCLUSIONS AND FUTURE EXTENSIONS OF LEM3

The LEM3 system is a powerful optimization tool that offers a wide variety of parameters to control the Learnable Evolution, performs experiments and displays the results of those experiments. It makes it possible for the user to define his or her own fitness function, in order to apply LEM3 to real world problems.

LEM3 has several limitations, due mostly to the fact that all parameters must be specified in the command line. This makes it impossible to use different types and domains of attributes used in the problem specification, even though such operation is fully supported by the algorithm. Only the interface limitations make it unavailable.

Future work on LEM will include enhancements of the interface by allowing the user to specify all LEM3 options in input files, instead of using command line parameters. It will also allow the user to specify simple fitness functions as mathematical formulas that will be parsed in the input files.

A second proposed extension to the LEM system involves the improvement of the program's evolution algorithm, in particular the AQ21 learning module.

REFERENCES

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

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

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

Wojtusiak J., "LEM3 System for Non-Darwinian Evolution," *Reports of the Machine Learning and Inference Laboratory*, George Mason University, Fairfax, VA, 2004b (to appear).

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 ISG (Intelligent Systems Group) Reports, or were part of the Department of Computer Science Reports).

Copyright © 2004 by the Machine Learning and Inference Laboratory.