LEM2
Theory and Implementation of the Learnable Evolution Model

G. Cervone

P 99-6
MLI 99-6

LEM2
Theory and Implementation of the Learnable Evolution Model

Guido Cervone

Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA 22030
gcervone@gmu.edu

P 99-6
MLI 99-6

June 1999

Abstract

LEM stands for Learnable Evolution Model and it is an Evolutionary Computation procedure which uses a rule learner program as main operator. The rule learner can work in combination with a Darwinian Evolutionary Algorithm (duoLEM) or by itself (uniLEM).

LEM1 was a preliminary implementation of the methodology [Michalski and Zhang 1999].

LEM2 is an expansion and an improved implementation of the methodology that is used as function optimizer. However the domain of applications of LEM2 can go far beyond what is presented in this paper, and it can possibly be applied wherever a Darwinian type (traditional non-deterministic) Evolutionary algorithms is used.

This paper gives a overview of the methodology, and presents the results of the experiments both using best-so-far curves and KV (GLD) graphs. It also includes an analysis of the methodology and suggestions for future development.

Keywords: Learnable Evolution Model, Evolutionary Algorithms, Quantization of variables,

Acknowledgements

This research has been conducted in collaboration with Dr. Ryszard Michalski, who has constantly supervised the experiments and the progress of the work, as well as given ideas on how to extend the original LEM methodology to overcome obstacles and emphasize strengths discovered during the implementation. The figure of how LEM partitions the space is taken from [Michalski 1999]. Immense help was received from Dr. Ken De Jong by explaining how Evolutionary Algorithms work, by always giving correct and accurate references to published papers and by releasing the code for EV3 and chapters from his not yet published book. Dr. Ken Kaufman helped in solving AQ-related problems, and suggested how to solve the problem with the quantization of variables.

Dr. M. Sebag sent important references through e-mail that helped in many ways to expand the methodology and improving the LEM2 implementation.

Crucial to the outcome of this research have been all the many conversations I had with other Computer Science and Biology students in the corridors, in the caffetteria, and in the MLI laboratory. Among all the people I wish to thank are: Mark Coletti for patiently listening every day to a report on the progress and for always suggesting cases where the implementation would fail, Hiren Vyas and Alaaeldin El-Nattar for giving ideas on how to improve the methodology and exchanging information on their research, Salem Ryan for helping in writing the parser for the AQ output file, Giovanni Vincenti for his comprehensive review of biology, Domenico Napoletani for helping in plotting the graphs of the functions, Qi Zhang for the original implementation of LEM, Hoang Lam, Saritha Mankala, Senthilkumar Thiyagarajan Janani Panchapakesan for showing me how to improve the diagrams and figures of this paper, Rida Mustafa for suggesting an interesting formula to optimize and test the implementation.

A special thank to my parents and grandparents who partially supported this research.

A special thank to Janejira Kalsmith, who patiently waited for the end of each experiment during her brief visits in Washington D.C., and for always giving support throughout the research.

I. Introduction

The term Evolutionary Computation was coined in 1991 as an effort to combine the different approaches of simulating evolution [Fogel 1997][De Jong 1999]. The methodologies and algorithms that follow under this denomination are numerous, however most of them share one fundamental similarity: they use semi-random operators such as mutation and recombination to achieve improvements of the fitness. These operators are semi-blind and the evolution is not guided by knowledge learned in the past generations, but it is a form of search process executed in parallel [Michalski 1999]. The Learnable Evolution Model uses a Machine Learning program as the main operator to guide the process of Evolution. The individuals of a population are engineered according to the rules found by the rule learner.

This paper describes the implementation of a subset of the LEM methodology, called LEM2. It differs from the first early implementation LEM1 in a number of features, of which the most important are the implementation of the *uniLEM* mode and of the *startover*, the dynamically adjusted quantization level, and the *fitness-based* selection method. All these terms are explained in sections IV and V.

LEM1 was a preliminary implementation of the methodology that was tested first to find the maximum of the functions of the De Jong's test suite [Michalski and Zhang 1999] and later was applied to design digital filters [Coletti et al]. In both cases it proved to be an effective evolutionary procedure.

This research has been conducted in collaboration with Dr. Ryszard Michalski, who developed the original idea of the Learnable Evolution Model.

II. Terminology

As LEM builds a bridge across two fields, some of the terms are those usually used in Evolutionary Computation, while other terms are those usually adopted in Machine Learning. For example in this paper the biological terms gene, alleles and values, are most of the time referred respectively as Vector of variables, variables and feature of variables. This is especially true when the execution of the rule learner program is described, as it is not a custom to say that a gene is passed to a symbolic learning program.

In Evolutionary Computation a population is a group of individuals. Each individual has a fitness value and a vector of variables (gene). Each variable (allele) corresponds to a dimension in the problem, and there is no restriction on it representation. A variable may be a binary string, a floating point number, a structure or a discrete variable. If an Evolutionary Algorithm is applied to a find the maximum of function $f(x_1\ x_2) = (x_1^2 + x_2^2)$, a population of individuals with two variables (alleles) will be created.

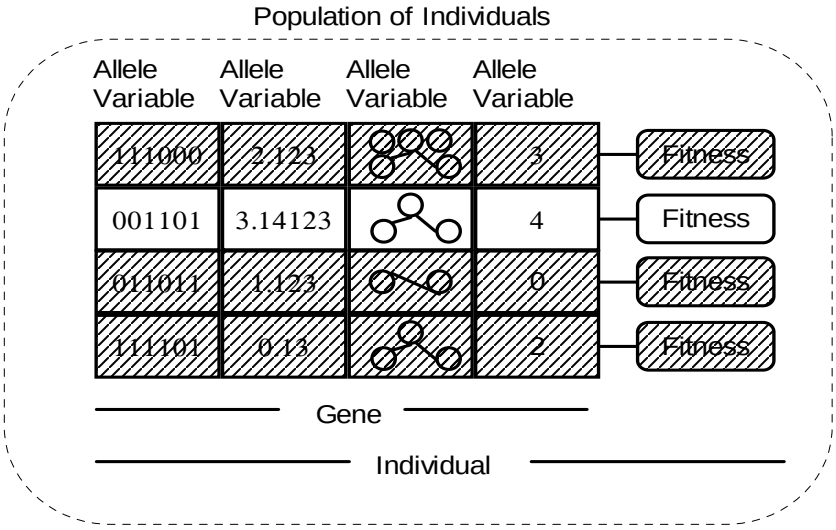


Figure 1: Description of Population, Individual, Gene, Alleles

III. LEM Vs. Darwinian EAs

Evolutionary procedures are stochastic techniques used to perform a parallel search in a space of possible solutions. They simulate evolution by creating a population of individuals and evolving it until an ending condition is met. It is possible to summarize the behavior of an Evolutionary Algorithm in three main steps:

- Select one or a group of individuals from a population
- Use birth operators over the selected individuals to create new individuals
- Select the individuals that will be part of the new population.

LEM follows this simplified scheme, and its main difference from Darwinian type Evolutionary Algorithms is in the technique used to generate new individuals.

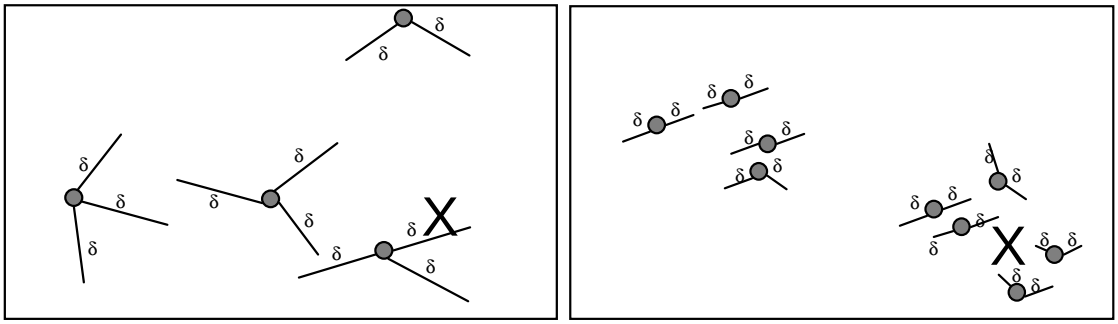


Figure 2: Self adjusting mutation in a 2D solution space (X represents the target)

Darwinian Evolutionary Algorithms use birth operators to create new individuals such as mutation and recombination to guide the process of evolution. These operators are semi-blind because they randomly change an existing individual by mutating the values of its alleles. It is possible to control such operators, like for example the step increment δ (delta) in mutation or the crossover rate in uniform recombination, in order to boost the performance and avoiding creating bogus individuals. However, mutation can still occur in such a way that the result moves further away from the solution rather than towards it. This can be visualized in the following figures, where a two dimensional space is represented, in which the mutation operator implements a dynamic way to adjust the step increment. It is possible to notice that as the individuals localize on the adjacent peaks, they also decrease their delta. This method minimizes the chances of loosing the peak, but they do not prevent individuals from mutating away from the target.

In LEM, the individuals are *genetically* engineered, in the sense that the values of the alleles are not randomly assigned, but they are set according to the rules found by the Machine Learning program. This process is more expensive in terms of CPU usage, but eliminated the factor of randomness that could be considered one of the weaknesses of the Evolutionary Algorithms. As for the Evolutionary Algorithms, there is not a clear distinction between *exploration* and *exploitation*. In the beginning of the evolution, the rule learner program is set to generalize maximally the rules, in order to explore further the space, and to ensure that no areas where maxima may reside go unchecked. Later in the evolution, the rule learning program is set to specialize as much as possible the rules found. The search conducted in Machine Learning mode can be viewed as a progressive partitioning of the solution space. [Michalski 1999]

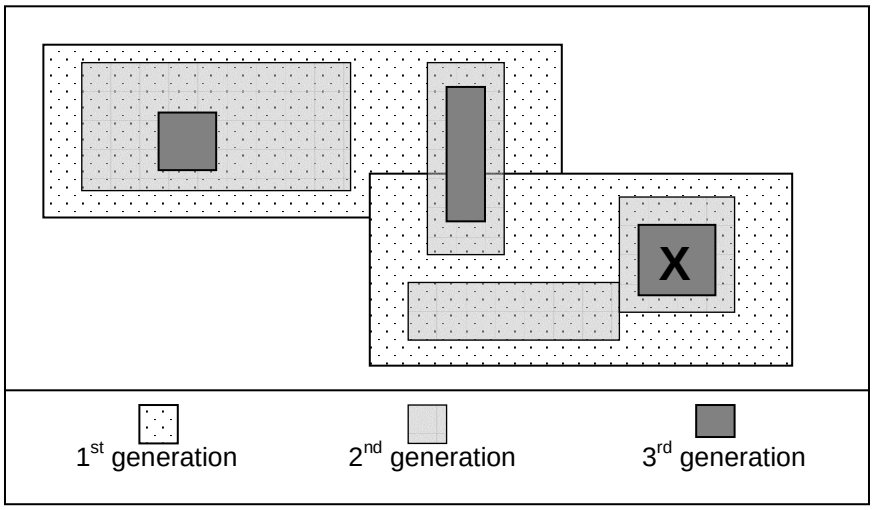


Figure 3: Progressive partitioning of a 2D solution space

Each time it is run, the rule learner defines an area of the space that is most fit to the landscape. New examples are instantiated from the rules found, are they will be classified as High or Low, depending on their fitness value. The descriptions for the High groups are only hypotheses of where maxima may be, because they are generated through induction from examples [Michalski 1999].

For example the simple function $f(x_1\ x_2) = (x_1^2 + x_2^2)$ describes four peaks that, depending on the bounds of the variables, may have different heights. Let's imagine that the bounds are $-1 < x_1 < 1$ and $-1 < x_2 < 2$ and that random points are generated. An H describe a High individual, and a L a low individual. The rule learner program will generate three rules, from which examples for the new population will be instantiated. The darker the color, the higher is the fitness of the elements in that region of the space, the more examples will be generated from that rule.

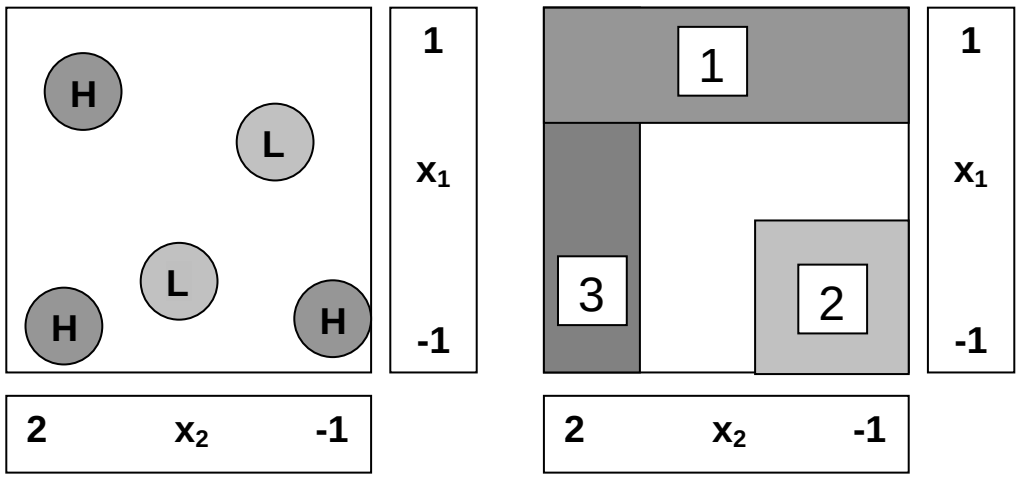


Figure 4 : Random examples, and generated rules

IV. LEM2 – Methodology

The main difference of the LEM methodology with respect to Darwinian Evolutionary Algorithms is the fact that the main operator is a Machine Learning program. The Rule Learner discovers why some individuals in a population have higher fitness than others, and it generates rules to describe the genetic information of the High examples. New individuals are born according to the rules, and they will compete with the existing individuals be part of the new population.

Following is a brief description of each step of the LEM. For a comprehensive description of the methodology refer to [Michalski 1999]. Figure 1 is a flowchart for both uniLEM and duoLEM. It shows the execution sequence of the methodology. Detailed information on how each step is implemented is given later in the Implementation section.

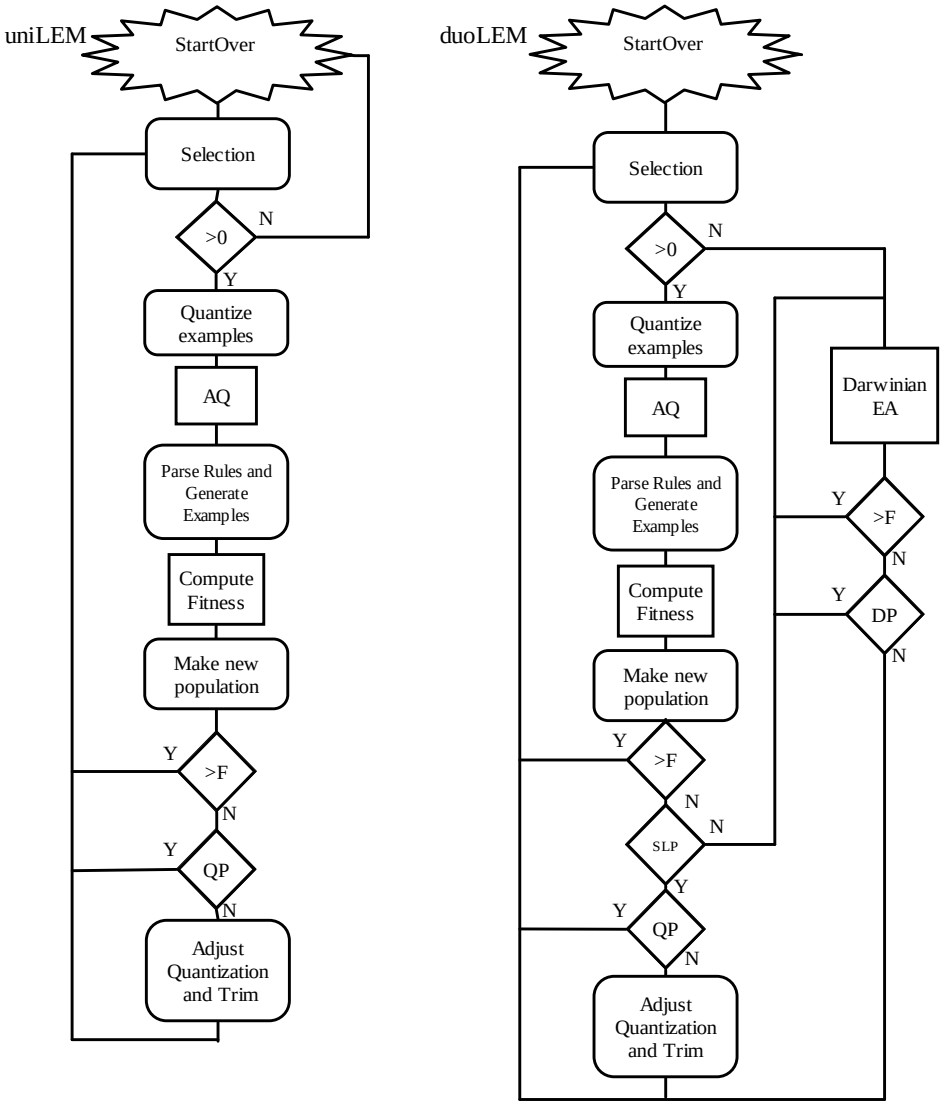


Figure 5 : Flowchart for uniLEM and duoLEM

Steps relative to both uniLEM and duoLEM:

StartOver – This operator is used to initiate a population, or a set of alleles, randomly or according to some constraints. It is used primarily to initiate the population at the beginning of the evolution, but also when no further improvements can be achieved. The latter is of particular interest because it overcomes a problem that may arise if LEM settles on a pick that is not a global maximum. If the entire population converges to a single point, it is impossible to classify examples as high or low since they all have the same fitness value, and consequently the symbolic learning program cannot generate any rules. In this situation an entire new population is generated, randomly or according to some rules that may implicitly exclude the previous maximum in order to minimize the convergence to the same point which may not be the global maximum. Alternatively only some alleles may be reassigned, in order to obtain individuals with different fitness, and classify them as high or low.

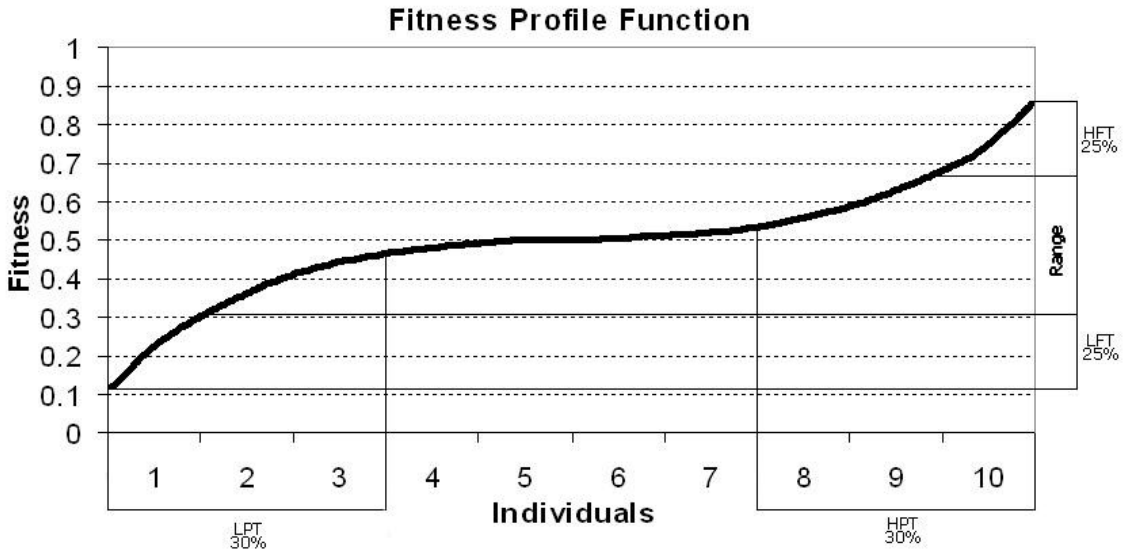


Figure 6: Fitness Profile Function and examples selected using Fitness-based Selection and Population-based Selection

1. *Selection* – The selection mechanism in LEM can be done in two different ways: Fitness-Based Selection (FBS), or Population-Based Selection (PBS). The difference between the two is that in the first case the number of High and Low examples vary depending on the shape of the Fitness Profile Function, while in the latter it is constant throughout the evolution. The High Fitness Threshold (HFT) and Low Fitness Threshold (LFT) divide the range in three areas. The Individuals whose fitness is included in the upper part of the range will be the High examples while the Individuals whose fitness is included in the lower part of the range will be the Low examples. The High Population Threshold (HPT) and Low Population Threshold (LPT) divide the population in three areas. The individuals that are included in the leftmost area are the representatives of the Low group, and the individuals that are included in the rightmost area are the representatives of the High group. In both cases the central area, which may occasionally be empty, represents neutral individuals, that are not used to generate rules. These two selection methods and their thresholds are shown in figure 2.

2. *>0?* – After running either of the selection algorithms it is necessary to check whether the examples are valid or not. A problem occurs when the Fitness Profile Function is flat or almost flat. The FPS will generate empty groups for both High and Low, while the PPS will generate groups of identical individuals. Both these situation will not generate two invalid groups of High and Low Individuals, and consequently AQ will not be able to generate any rules. To overcome this problem, in uniLEM mode the *StartOver* operator is invoked, while in

duoLEM the execution is switched to the Darwinian Evolutionary Algorithm. These two steps will increase the diversity in the landscape, so that eventually the next time *Selection* is invoked, it will be able to correctly characterize the two groups with valid individuals.

3. *Quantize Examples* – AQ18, the implementation of AQ used in LEM2, accepts only discrete variables as input. If for example LEM is used to optimize a function of real-value parameters, it is necessary to quantize these parameters. On the other hand, if the domain of the variables is discrete, LEM will create the input for AQ using their exact values. Quantization is a crucial step for the correct execution of LEM, and different problems may require different types of quantizations.

4. *AQ* – AQ is the rule learner program that is the main engine of the evolution. Even though this methodology was designed and implemented with the intention of using AQ, different rule learners may be used for this purpose. AQ constructs rules to characterize the High Examples with respect to the Low examples and optionally also rules to characterize the Low examples with respect to the High examples. [Michalski 1983] It also gives information on the coverage and quality of each rule. Figure 3 and 4 show a simple AQ output.

```
parameters
run   mode    echo   verbose
 1     ic     pve       1

parameters
run  ambig  trim  wts  maxstar  criteria  noise
 1    empty spec  cpx    1      default    no


variables
  #   type    size    cost    name
  1   lin      15     1.00    x1.x1
  2   lin      15     1.00    x2.x2
  3   lin      15     1.00    x3.x3
  4   lin      15     1.00    x4.x4


highcell-events
  #   x1   x2   x3   x4   Weight
  1   7    7    7    8    2
  2   7    6    6    7    2
  3   7    6    7    8    1
  4   6    6    7    8    3
  5   6    7    6    8    2
  6   6    7    8    8    1
  7   7    7    8    8    1
  8   6    6    7    7    1
  9   6    6    6    8    1
 10   6    6    8    8    1


lowcell-events
  #   x1   x2   x3   x4   Weight
  1   6    6    14   8    9
  2   6    7    14   8    4
  3   7    7    14   7    1
```

Figure 7: Parameters and Examples

```
goodcell-outhypo
   #   rule
   1   [x1=6..7] [x2=6..7] [x3=6..8] [x4=7..8]   (t:15, u:15,
q:1)


  badcell-outhypo
   #   rule
   1   [x1=6..7] [x2=6..7] [x3=14] [x4=7..8]   (t:14, u:14,
q:1)


This learning used:
  System time:      0.000 seconds
  User time:        0.00  seconds
```

Figure 8: Rules and Elapsed Time

AQ is given a set of parameters that characterize its behavior, a description of the variables, and examples for the High and Low groups. The output will repeat the input information and will append rules express in VL language. LEM2 uses AQ18, the latest implementation of the AQ methodology but without constructive induction [Blodeorn 199?].

5. *Parse Rules and Generate Examples* – The output returned by AQ has to be interpreted, and new examples generated from the rules found. The first step consists in scanning and parsing the output to extract the rules and thei attributes (coverage, quality). The second step consists in generating new individuals which will later compete to be inserted in the population. A rule may not include all the variables, and it means that only some of them are essential for describing the High individuals. When such a case occurs, a new individual has: for the alleles included in the rule values generated according to the rules; for the alleles not included in the rule, either values chosen from the corresponding alleles of other individuals in the population, or values randomly chosen in the range for which the allele is defined. The quality and coverage of the rule determine how many individuals will be generated using that specific rule. To explain this concept let rule 1 cover 50% of the High examples, and let rule 2 cover 5% of the High examples. If the average fitness of rule 1 is less than the average fitness of rule 2, than more examples will be generated using rule 2, rather than rule 1.

6. *Compute Fitness* – After generating new individuals, their fitness is computed. This operation may be very costly if the fitness is not given by a function, but by a different application, such as a simulation. It is important not to generate too many examples in order not to slow down the execution, but on the other hand, it is also necessary to generate enough number of examples to use all the rules found by AQ. The *selection* mechanisms described below can be used to find the right number of individuals to generate.

7. *Make New Population* – This step involves creating a new population using examples from the previous population and/or the new individuals generated according to the rules found. Different types of *survivors selections* can be used to determine the combination of old and new individuals to be used. Each selection method can be divided into *intergenerational* and *generational*. The first implies that both the old individuals and the new individuals compete to be inserted into the new population, while the latter implies that only the new individuals compete to be part of the new population. This second method requires a sufficient number of examples to be created. More in particular the number of new individuals to be generated must be equal or higher than the population size. These following are only some of the many selection mechanisms that may be implemented, and they should be chosen depending on the landscape and on how many individuals ought to be selected.

    - *Proportional Selection* – Proportional Selection assigns to each individual a survival probability that is proportional to the individual's fitness. This method first maps an individual to its fitness, then it creates a probability distribution proportional to the fitness, and last it draws samples from this distribution. [Grefenstette 1997]

    - *Tournament Selection* – In Tournament selection a group of Individuals is randomly chosen from the population. The chosen individuals take part in a tournament, and the those with higher fitness survive [Blickle 1997]

    - *Rank-based Selection* – Rank Based selection assigns a reproductive or survival probability to each individual that depends only on the random oredering of the individuals in the current population. [Grefenstette 1997]

- *Soft Brood Selection* – Soft Brood Selection holds a tournament between members of a *brood* of two parents. The winner of the tournament is considered to be the offspring contributed by the mating. [Foegel 1997]

- *Disruptive Selection* – Disruptive Selection can be used to select against individuals with moderate values. An Individual fitness increases with its distance from the mean of all current solutions. This is done in order to distribute more search effort to both the extremely good, and extremely bad solutions. [Foegel 1997]

- *Stabilizing Selection* – Stabilizing Selection selects individuals with extreme values. Both Individuals with very high fitness and very low fitness are chosen, and those in between will never be picked. [Foegel 1997]

- *Variable Lifespan* – This method of selection is a compromise between the G*enerational model* and the *Intergenerational model.* A parent is allowed to survive a fix number of generations, before being permanently removed from the population. [Back 1994]

8. *>F?* – This condition checks whether there has been an improvement in the global maximum. If there is an improvement, LEM will repeat the last cycle, starting again from selection (uniLEM), or by running once more the Darwinian Evolutionary Algorithm (duoLEM).

9. *QP? (Quantization Probe)* - This condition checks if there has not been an improvement in the last generation (uniLEM and duoLEM) and if the number of times AQ run is lower than the *Symbolic Learning Threshold* (duoLEM), the quantization level is increased, given that the number of executions with the current level is less than the *Quantization Probe.* An alternative solution not yet implemented would be to change the representation space to be a smaller space where the high and low examples are included.

10. *Adjust Quantization and Trim* – This step is very important for the correct execution of LEM. The quantization needs to be expanded to better represent the variables. The Trim is the AQ Generalization Level. It can be set to **gen**, **min**, and **spec**. Usually it is good to start the evolution with **gen** or **mini**, because AQ will generalize more the rules, and so it will explore more the space. Later in the evolution using **spec** AQ will refine the previous rules. This can be visualized using GLD graph.

Steps relative to duoLEM only:

Darwinian EA – If LEM cannot improve the global maximum by itself, the execution is passed to the Darwinian Evolutionary Algorithm. There is no restriction on the kind and characteristics of the algorithm, as long as it can share the same fitness function used in LEM.

SLP? (Symbolic Learning Probe) - This parameter defines how many times LEM is allowed to run without achieving an improvement in the global maximum fitness. If this condition is true LEM will be run again, if it is false the execution will switch to the Darwinian Evolutionary Algorithm.

DP? (Darwinian Probe) - This parameter defines how many times the Darwinian Evolutionary Algorithm is allowed to run without achieving an improvement in the global maximum fitness. If this condition is true the Darwinian Evolutionary Algorithm will be run again, if it is false the execution will switch to LEM.

V. Implementation

LEM2 was originally intended to work in combination with EV3, an evolutionary program developed by Dr. Ken De Jong for his students, in duoLEM mode. Therefore, the implementation of LEM2 is highly tightened with EV3 [De Jong 1999]. EV3 is a simple Evolutionary Algorithm which uses a real value representation of the alleles, a fix mutation operator, and different selection mechanisms for choosing the parents, and for determining which offspring survives to the next generation. Later in the research, the idea of uniLEM became clear, and so the development of the program followed a different path. The original code of EV3 is used to initialize the landscape, to store the representation of the variables and to compute the fitness of an individual. This code is almost untouched from its original implementation, especially the part relative to the data structure which has not been modifies at all.

The main implementation of the LEM methodology happens in two files: lem.h and lem.c which are respectively the definitions of the functions, and their implementations.

LEM2 is written in ANSI C, and it is compiled on a SUN SPARCstation 20 running Solaris 2.7. All of the experiments in this report are run on the same machine. The interaction with AQ is done through file IO, which significantly slows the execution of the program. It is planned however, to improve the communication between AQ and the rest of the modules, by passing all the parameters in memory, and by also reading the learned rules off memory. The LEM code is divided into five main steps:

1. *Select the examples for the High and Low groups.*
2. *Quantize the variables and construct the AQ input file*
3. *Run AQ*
4. *Parse the AQ output file, and unquantize the variables*
5. *Generate examples, and select representative for the next generation*

Step 1 is carried out in sort_individuals(), and select_examples(), step 2 in generate_examples(), step 3 in run_aq(), and steps 4 and 5 in generate_individuals()

Those names are taken from the first implementation of LEM. In fact it has been tried to keep the implementation of LEM2 as similar as possible to the implementation of LEM1, to take advantage of the previous experience reported by Qi Zhang, and to easily compare the two implementation. This approach was useful to determine more weaknesses of the previous implementation. Most important was the inability to correctly manipulate the individuals and the population, because EV3 operators were designed specifically to operate over single individuals, while LEM operates mostly on populations. To overcome this problem it will be necessary to completely separate LEM from the implementation of EV3, and to develop a well structured set of libraries that will allow very easy and efficient manipulation of a population of etherogeneous individuals.

Such a library could be EC++ [Cervone, Coletti, Latimer] developed specifically keeping in mind the characteristics which would be important in a future development of LEM.

It would also be very important to allow a population to be made of individuals that hold different kind of variables. For examples a variable may be continuous, another may be discrete and yet another a structure. This will allow LEM to be applied more effectively than Darwinian type Evolutionary Algorithms, because they usually achieve good performance in a continuous domain or in a discrete domain, but not in both.

Function specific implementation:

Select the examples for the High and Low groups: As specified in the methodology there are two distinct ways to select examples: Fitness-Based method, and Population Based Method. The user specify first which method to use, and then gives the values for the high threshold and low threshold. Instructions on how to create a configuration file are given later in the paper.

Quantize the variables and construct the AQ input file: The quantization method used in LEM2 is χ^2. It divides the range for which a variable is defined into equal parts. The user specifies the starting domain size (in how many parts the interval is divided), which can be very small (e.g 5) and the program will automatically increase it if necessary. For example, if all the individuals of a population have the same values after being quantized, the quantization domain size will be increased by five units. The number five was chosen as being the best candidate over a set of examples. However it will be appropriate to change the quantization technique from □□ to an Adaptive Anchored Quantization (AAQ) [Michalski-Cervone 1999]. When the quantization level is increased, the aq trim is also switched to *aqStartTrim* for *aqTrimProbe* number of runs. This is to ensure that the exampled do not localize immediatelly when the level is increased, but are nicely distributed.

The AQ input file is created in a file called aqin. It includes a set of parameters for AQ, and the set of quantized examples for the high and low groups. The AQ parameters are both specified by the user, and automatically adjusted. The adjustable settings modify the AQ mode, AQ Trim (Generalization Level), and AQ ambiguity. A comprehensive description of these parameters and how to use them can be found in [Michalski-Kaufman 1999].

Run AQ: The AQ program is run by making a system call. The input file is passed using the redirection operator '<' and the output is written to a file using the redirection operator '>'.

Parse the AQ output file, and unquantize the variables: The rules generated by AQ are parsed. This is done by hand written parser, and it is apporpriate to change it in the future using LEX and YACC. A rule may look like the following: [X2 = 3..5,7..9] which means that the value for X2 should be included in the range 3..5 and 7..9. These rules are then unquantized, and each value is mapped to the value in the original interval that it represents. The unquantized variables are stored in the data structure struct VL posRulesp[a][b] where the first column represent the X and the second the rule to which it is associated.

Generate examples, and select representative for the next generation: New examples are created according to the rules found. If one variable is not included in the rule, its value is chosen from other variables in the population. It could be possible to choose its value from the range from which it is defined, but this did not achieve good results in the early experiments. The number of examples generated is at least the population size minus the positive examples, however it may happen that the number of examples generated is higher than the population size. In this case it is necessary to choose the survivors. In LEM2 the number of examples generated is equal to the population size minus the high examples. The new examples simply replace the low and the neutral ones.

Notes on EV3:

EV3 is a simple but robust implementation of three canonical Evolutionary paradigm: EV, ES, EP. Exhaustive information on the three methodologies and on how EV3 works, can be found [De Jong 1999]. EV3 uses a real-value representation of the parameters, and a uniform mutation operator.

When used in EV mode, the program selects randomly a parent from a population, and it creates as many children as specified in the configuration file by mutating the original parent. Then each children compete for survival by competing with a randomly selected parent of the original population.

The ES mode introduces the idea of brood size, in which each parent produces K children. Each children is then mutated and appended to the original population. All the individuals are then sorted, and truncated, such as only the right amount of individuals with the higher fitness survives.

In EP mode each parent is mutated, and then the children compete for survival with a random selected parents. The survival selection procedure is the same as defined in EV.

A brief comparisons of LEM2 and LEM1:

Some of the major improvements of LEM2 with reguard to LEM1 are:

- The implementation of the uniLEM mode and of the *StartOver* operation.
- The dynamically adjustable Quantization Domain Size.
- The ability of using more than one rule to instantiate new individuals.
- The implementation of the Fitness-Based Selection method
- The possibility of running experiments on highly dimensional spaces.
- The semi-integration with KV.
- The ability of creating input files for Microsoft Excel.

Configuration files:

The settings for LEM2 are specified in the lem.cfg file, and in the ev3.cfg file. In the lem.cfg file, each line that starts with a # identifies that the value for that parameters will be specified in the next line. The valid # options are:

#selectMethod
it can have value **fitness** or **population.**

#high
it is the percentage expressed in decimal of the HFT or HPT, depending on what #selectionMethod specifies.

#low
It is the percentage expressed in decimal of the LFT or LPT, depending on what #selectionMethod specifies.

#maxrules
It specifies the maximum number of rules that AQ finds that are used to generate examples.

#levels
This specifies the initial domain size.

#aqMode
This specifies the AQ learning mode. It can have values **vl, ic** and **dc**.

#aqTrim
This specifies the AQ trim (generalization level) through out the evolution. It can have values **gen, spec** and **mini**.

#aqStartTrim
This specifies the AQ trim (generalization level) at the beginning evolution. It can have values **gen, spec** and **mini**.

#aqTrimProbe
This specifies how many times AQ is invoked with the generalization level specified in aqStartTrim, before switching to aqTrim

#aqambig
This specifies the AQ ambiguity. It can have values **empty, pos** and **neg**.

#maxstar
This specifies the AQ maxstar parameter.

#bias
It specifies whether there will be a bias when generating examples from the rule. This parameter has not yet been fully implemented, and it is related to determine the distance from the negative. [Michalski-Zhang 1999]

#elitism
It specifies wheter to add always the global max to the current population. It can have value **yes, no**

#kvFile

It specifies whether to create a KV input file to analyze the evolution process. It can have value **0, 1, 2**. 0 specifies that no file is created, 1 a file with only the positive and negative examples, and the rules found, and 2 that also the neutral examples are included.

For detailed information on the EV3 configuration file, refer to [De Jong 1999]. Following is a brief summary of the EV3 parameters specified in the configuration file.

The EV3 configuration file in is named ev3.cfg.The first line represent the type of algorithm to be runValid choices are: **aq, aqev, aqes, aqep, ev, es, ep**. Next there is the landscape ID, which is formed by a two digit number. The first digit can take value 1, 2, 3, 4 or 5. It represents the landscape type (ie. 1 for De Jong's function 1, 2 for De Jong's function 2 etc). The second digit can take value 0 or 1. In the first case the program is trying to find the maximum of the function, in the latter, the minimum. The next field specifies how many variables are involved. This is done in order to make it easier to test the scalability of the algorithm.

Following fields will represent the bounds of the variables, a population size, and a children population size, the number of births to be made before stopping the execution, and the random seed.

The last field is the report type. An extra report type, **excel**, is included to easily plot BSF curves using Microsoft Excel.

Setting the parameters:

Nevertheless the number of parameters in LEM2 may look conspicuous, it is fairly easy to find a good combination of parameters that will maximize the results.

The **selectionMethod** should be chosen accordingly to the fitness profile function. If the range is very small, which implies the individuals in the population have similar fitness, it is better to use a Population-Based selection. The **high** threshold and **low** threshold will determine the number of examples that are passed to AQ. The values should never exceed .50, and never be smaller than .1. The more negative examples are passed, the more specialized AQ rules will be. The speed at which AQ will run, is highly correlated to the number of negative examples that are passed.

The best aqMode in most of the cases is **vl**. It is also the faster, because it will not generate a rule for the negative examples. The **ic** mode seems to lead to the same results of **vl**, but in a slower way, since it will also find rules to describe the low examples with respect to the high examples. This feature may be useful in the future for implementing *constrained startover* [Michalski 1999].

The **aqStartTrim** should be **gen** and the **aqTrimProbe** depends on the size of the search space relative to the population size. When this parameter is set to **gen** or **mini**, the rules are more general, and consequently the examples generated from these rules further explore the space. The **aqTrimProbe** specifies how many runs AQ has to run before swithching to the mode that will be used for the rest of the evolution (**aqTrim**). In fact, once the areas of the space with higher fitness are identified, it is necessary to change the trim to **spec**, to specialize the rules, and correctly identify the point that achieve the best fitness.

Elitism should always be set to **yes**.

The parameters that are not described here are yet to be experimented, and it is appropriate to run LEM with the defaults.

VI. Experiments are Results

LEM2 has been tested over three different functions, first proposed in [De Jong 1997] for testing genetic algorithms. The number of dimensions has been increased to test the scalability of the method. The results for each experiment are reported in the form of Best-So-Far curves and/or GLD graphs. A discussion of the results is also included, in combination of a summary of the parameters used for both EV3 and LEM2. Whenever a setting for a parameter is not specified, it means that the default is used.

Experiments 1..4

Experiments 1.. 4 are relative to De Jong's function 2. The function is defined and plotted as following:

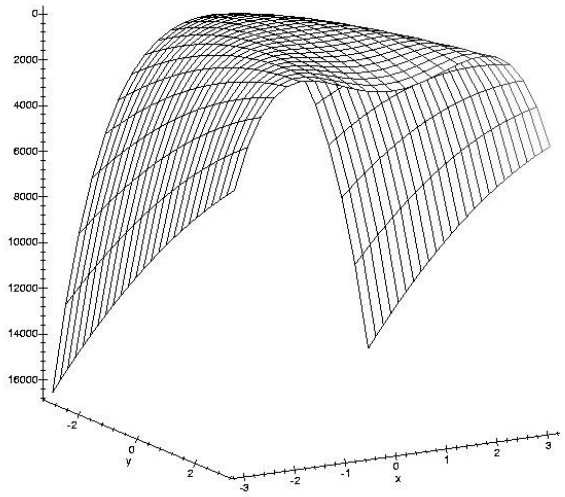


Figure 9: De Jong's function 2 (ROSENBROCK)

$$f(x) = \sum_{i=0}^{n} (100 \cdot (x_{i+1} - x_i)\hat{}2 + (x_i - 1)\hat{}2)$$

The ROSENBROCK function is a very complex minimization problem. It has a very narrow ridge. The tip of the ridge is very sharp, and it runs around a parabola. Algorithms that are not able to discover good directions underperform in this problem [Yuret 1997]. The function is plotted here on a very small interval. The function looks symmetric but it is not. The minimum is found when all the variables are equal to 1. It grows very rapidly as the variables move away from 1, and it is very difficult to show the asymmetry.

Experiment 1: - Analyze the behavior of LEM2 using GLD graphs over a simple problem.

The goal of this experiment is to observe the progressive partitioning of a multi-dimensional space. This is accomplished by analyzing a set of GLD graphs, generated using the KV program [Zhang 1997], that represents different stages of the evolution process.

KV is specifically designed to visualize various aspects of concept learning. These visualized aspects include: representation space (or instance space), qualitative and quantitative distribution of training or testing examples, relationships between examples and rules characterizing concepts, qualitative and quantitative visual display of concept rules. [Zhang 1997].

Two GLD graphs are included for each cycle of LEM (a cycle is defined in the flowchart from the previous section). The first graph shows the high and low examples selected from the population, and the second shows the rules generated with the covered examples, from which new individuals will be instantiated. The neutral examples are removed for emphasizing the areas of low and high individuals.

Following is a summary of all the parameters for this experiment.

Algorithm:	**UniLEM**
Function: Random seed:	F2 – Minimum 12345
Number of Variables	4
Bounds	-3 < x1 < 3 -3 < x2 < 3 -3 < x3 < 3 -3 < x4 < 3
Selection Method	Fitness-Based
High Threshold	20%
Low Threshold	20%
AQ mode	VL
AQ trim (Gen Level)	Spec
AQ Start Trim	Gen
AQ Trim Probe	1
AQ Ambig	Neg
Population size	50
Initial Quantization Levels	5
Ending Condition	All the variables are equal to 1

Figure 10: Experiment 1 Settings

Population after births: 50

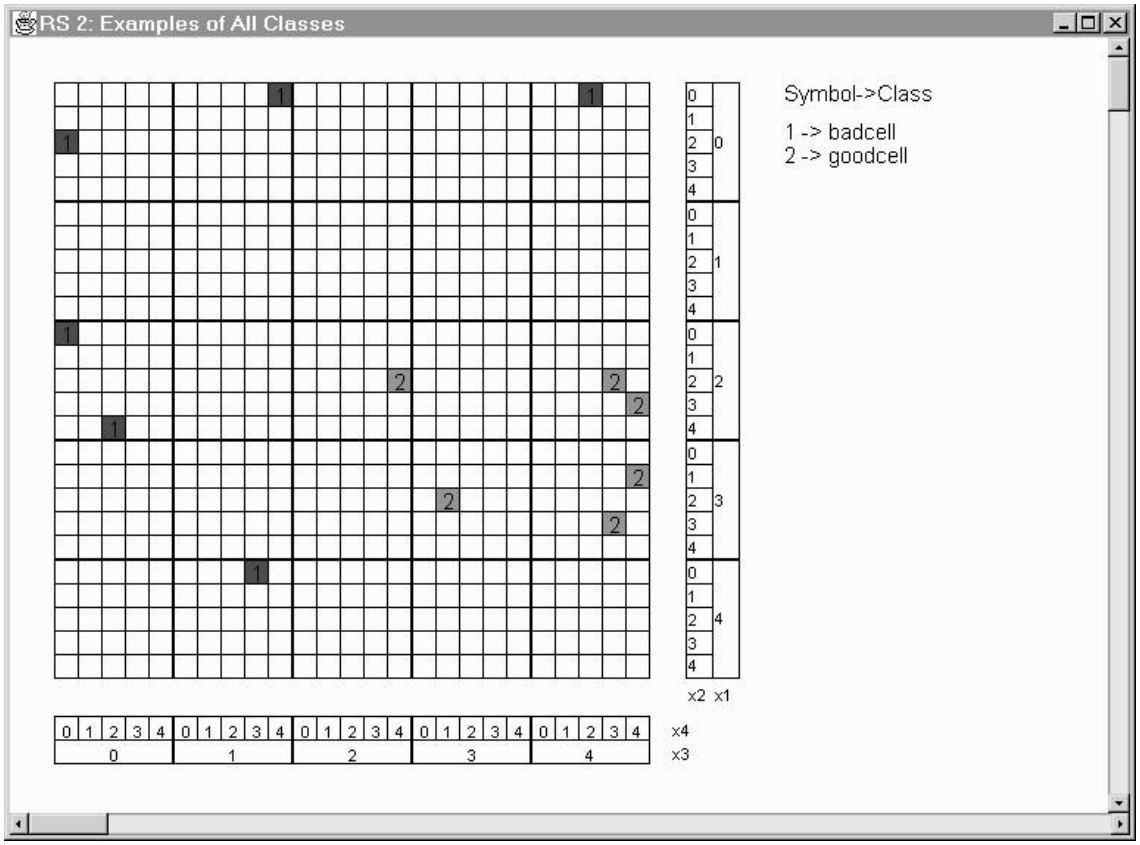


Figure 11: High and Low examples

This first graph represents the high and low examples selected from the initial population generated randomly. The high individuals (individuals with higher fitness) are shown in green and with number 2, the low individuals (individuals with lower fitness) are shown in red and with number 1. The target (not shown) is at position $x_1 = x_2 = x_3 = x_4 = 3$, and not 1, because the graph shows *quantized* values for the variables. The following table shows how the quantized values are mapped into the original range.

Quantized Value	Represents the interval
Value 0	–2 .. -1.2
Value 1	–1.2 .. -.4
Value 2	–.4 .. -.4
Value 3	.4 .. 1.2
Value 4	1.2 .. 2

Figure 12: Quantize values, and their correspondence to theoriginal range

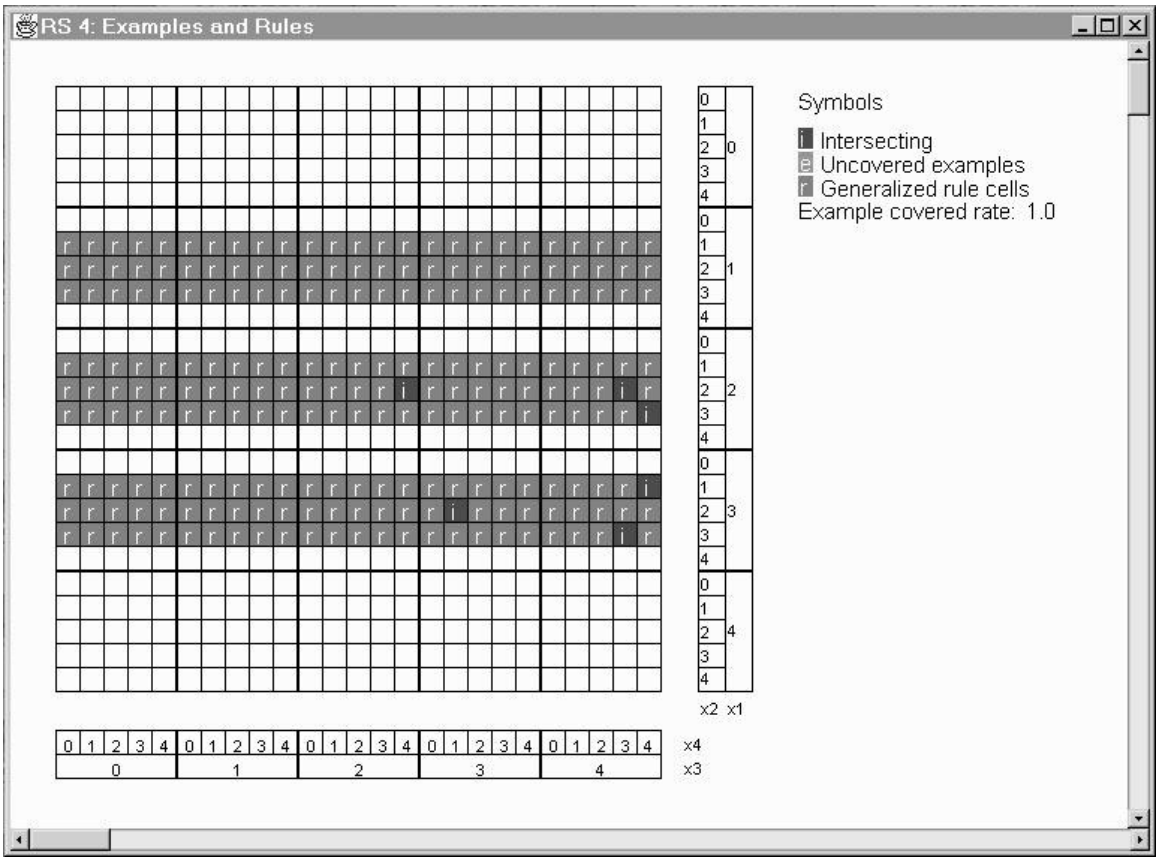


Figure 11: Rules for the High class, and intersecting High individuals

This graph shows the rules found to describe the high class, and also how they intersect with the high examples. The r represents a cell that is covered by the rule because of the generalization AQ , while the i represents a cell where an example for the high class was present.

It is possible to notice that the rules are very general. In fact, they cover a large area of the space where no high examples are present. This result is obtained because AQ run with **gen** trim (Max generalization level). In fact, it was specified in the configuration file to generalize as much as possible (aqStartTrim = gen) the first run (aqTrimProbe = 1).

This behavior is desired because it is possible that the initial population, generated randomly, is not a good representative of all the areas of the space. It is then appropriate to further explore the space by maximally generalizing the rules. This will cause the next individuals that will be instantiated from these rules, to be distributed over a larger space.

Population after births: 94

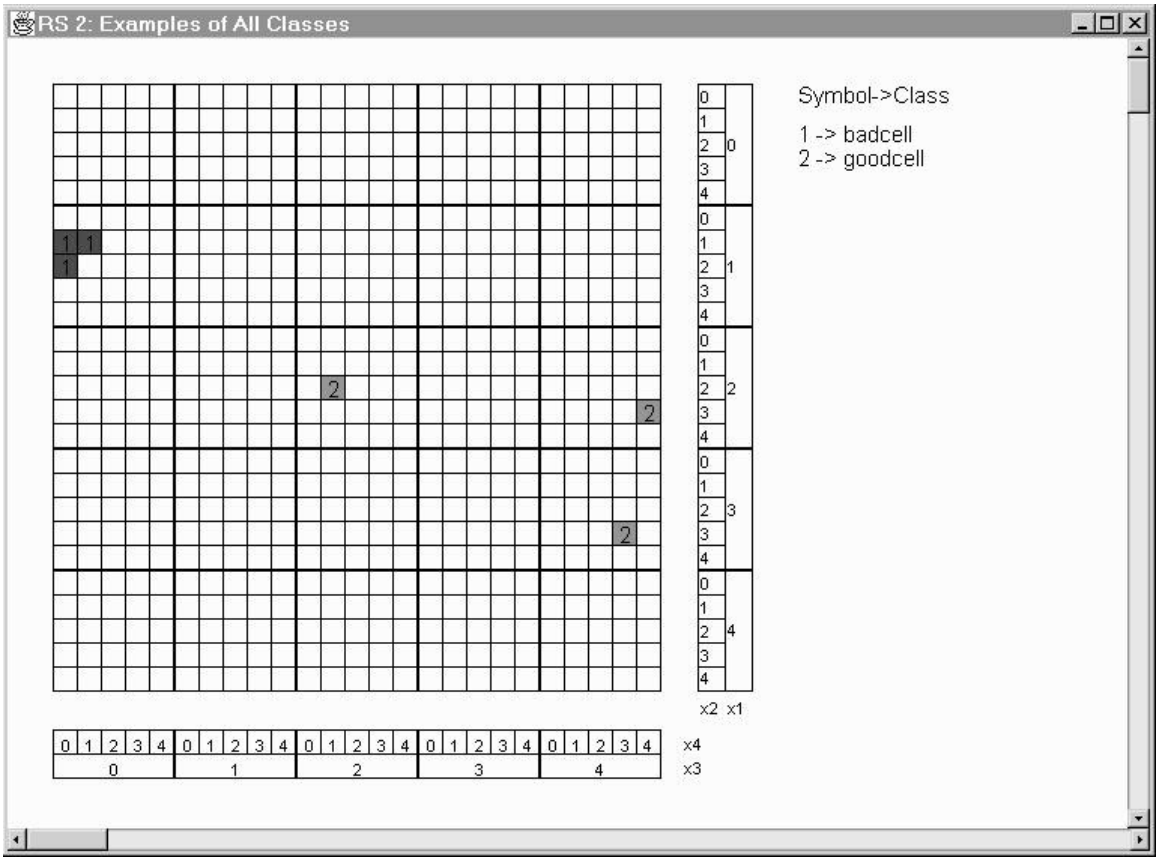


Figure 14: High and Low examples

This graph shows the high and low individuals that are instantiated from the rules found in the previous run, as shown in the figure 13. It is possible to notice that the high examples are distributed in the lower right quadrant, and the low individuals towards the upper left quadrant. By comparing these examples with the previous, it is possible to identify that the low examples are indeed distributed in the upper-left part of the space, while the high examples in the lower-right part of the space.

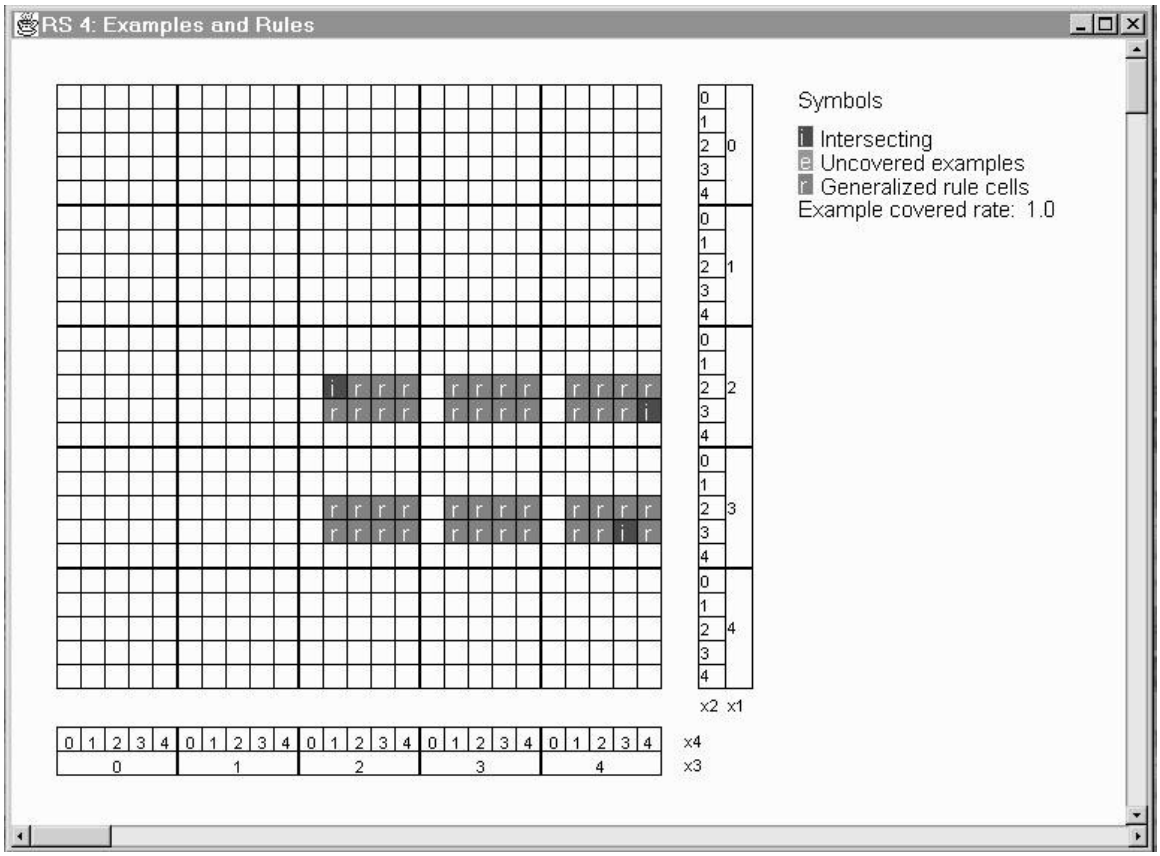


Figure 15: Rules for the High class, and intersecting High individuals

The rules that describe the space are now very specific. The space is partitioned in a much smaller area, however it is still not clear where the minimum is. New examples will now be instantiated in this smaller partition of the space.

Population after births: 191

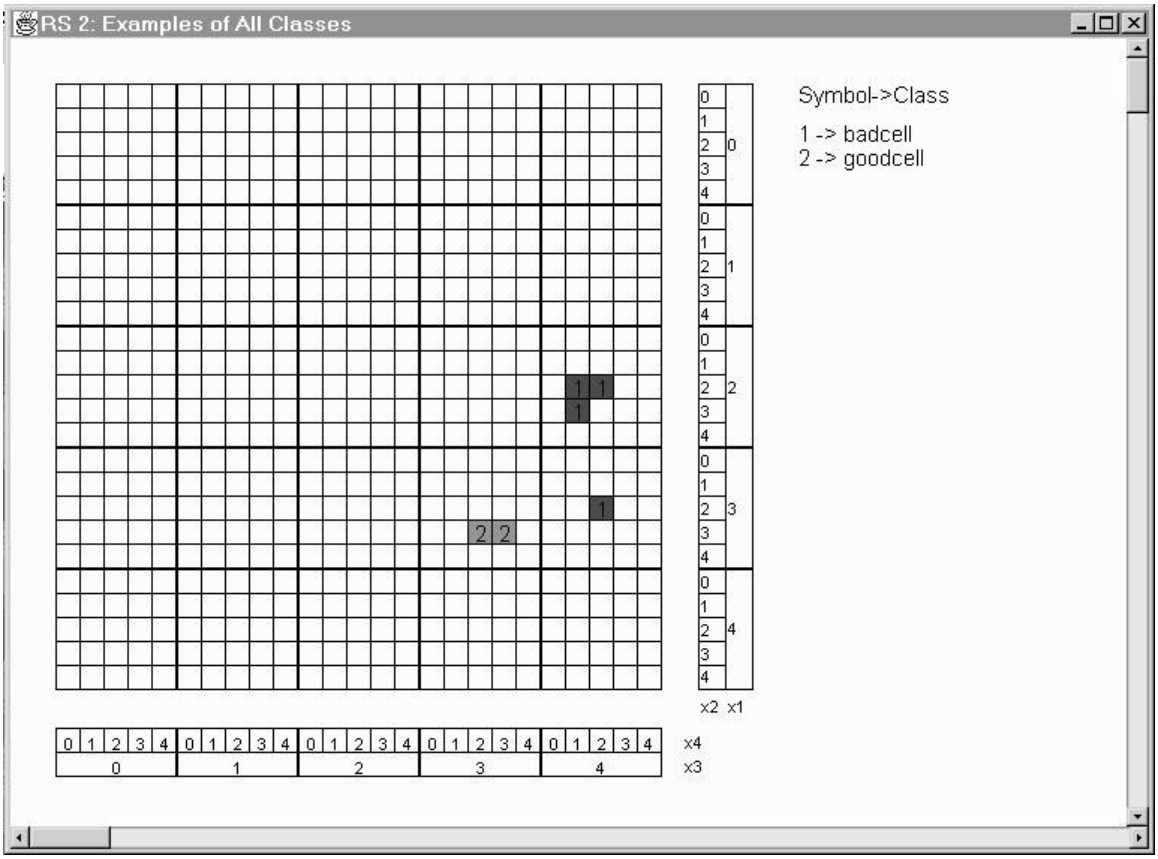


Figure 16: High and Low examples

The High individuals instantiated from the rules as shown in figure 15 are all localized in one specific area of the space. It is now clear where the minimum will be, and most in particular, the target is not covered. In fact a positive example was generated with quantized values $x_1 = x_2 = x_3 = x_4 = 3$, which represents perfectly the area in which the solution is included .

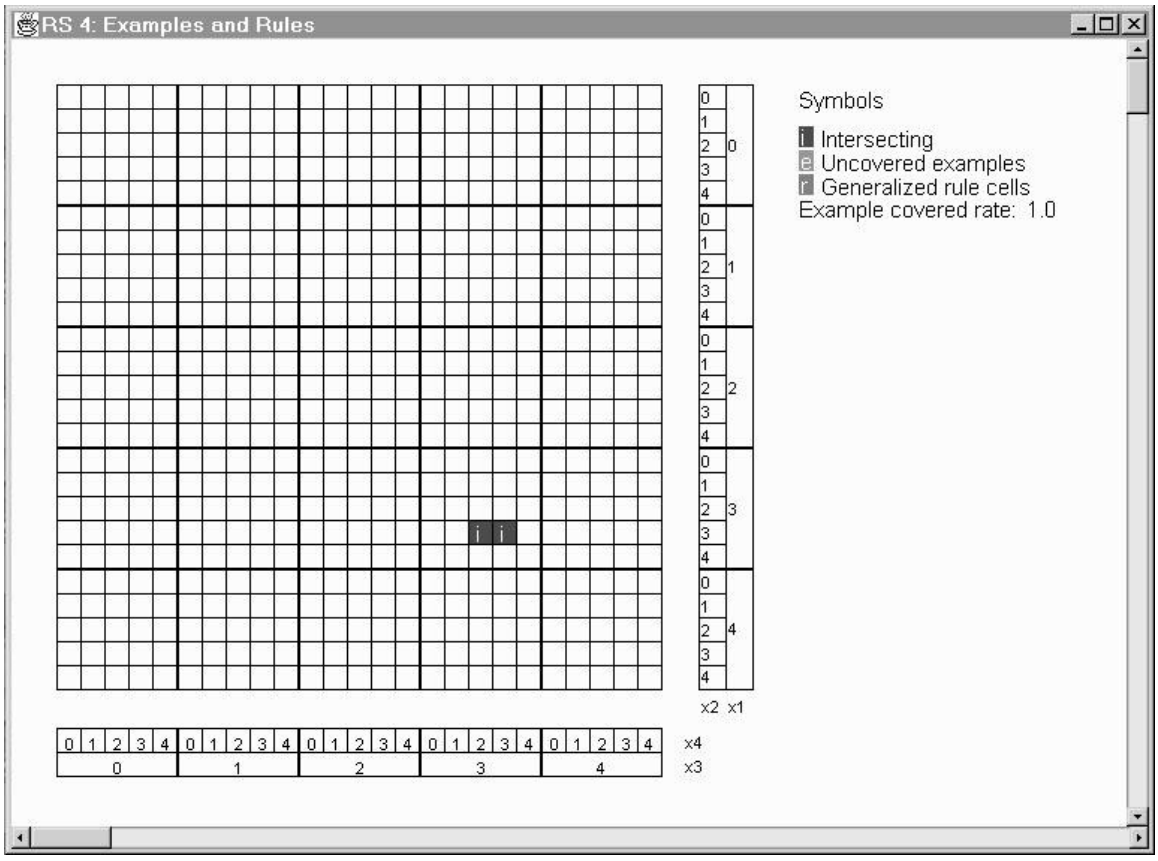


Figure 16b: Rules for the High class, and intersecting High individuals

The search space has been reduced drastically. The rules for the high individuals cover a very small region of the space. The examples instantiated from this rule will clearly define which of the two cells contains the optimal solution.

Population after births: 240

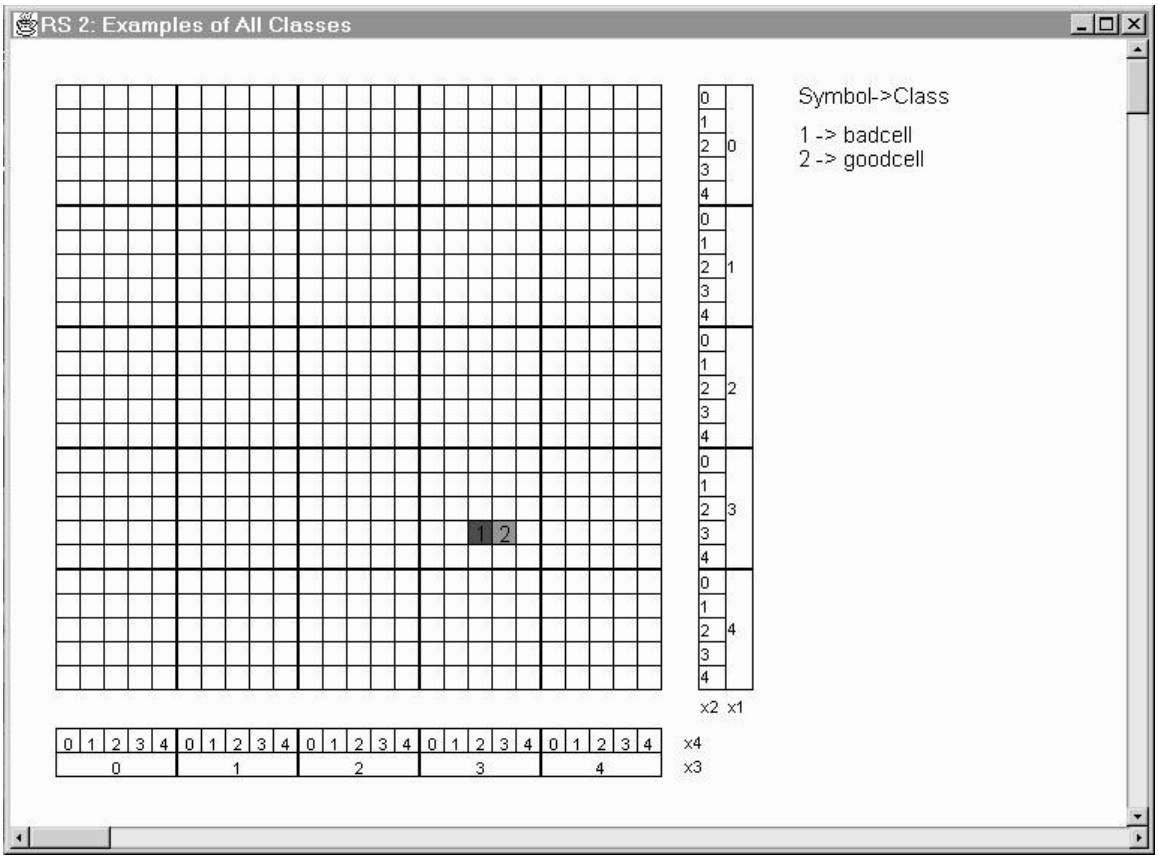


Figure 18: High and Low examples

After 240 the entire population is now concentrated in two cells. This is a very important result, because it means that the program was able to correctly identify the specific area of the space in which the optimal solution is contained.

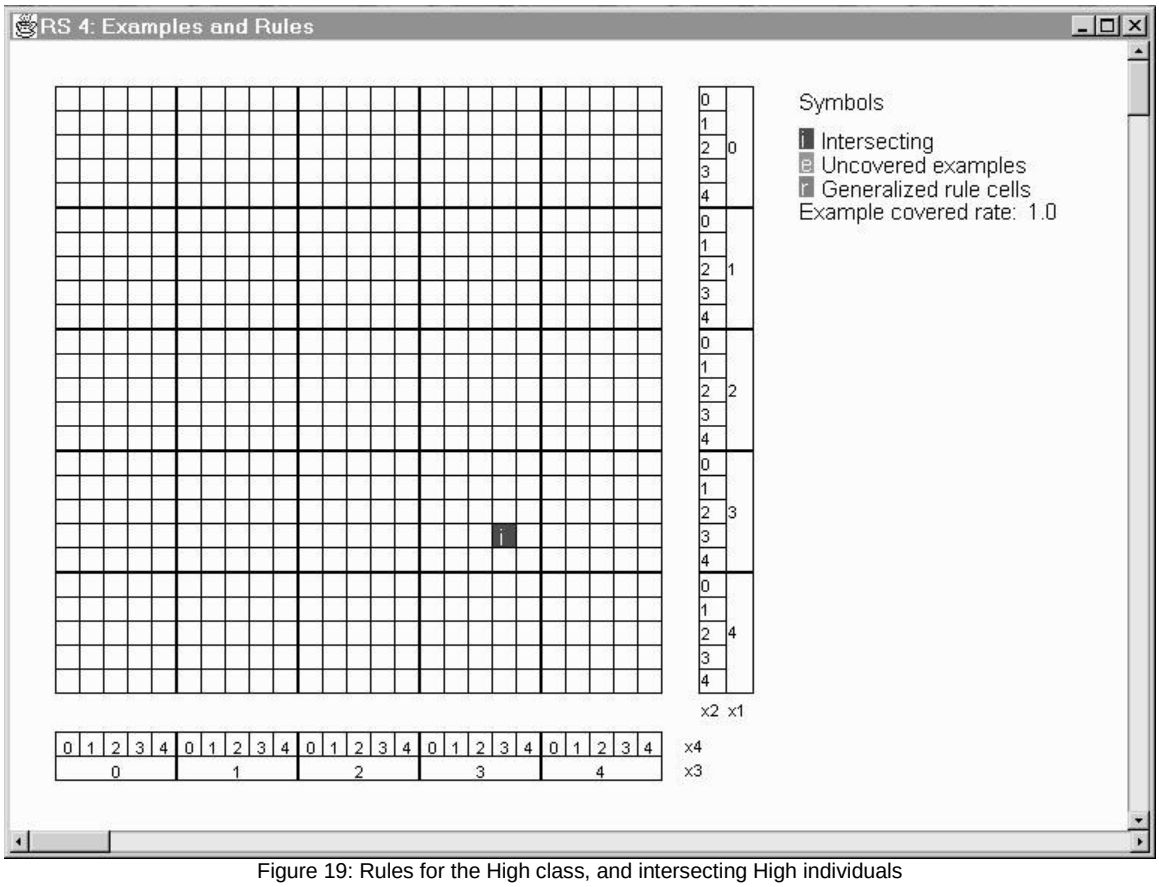


Figure 19: Rules for the High class, and intersecting High individuals

This rule clearly identify that the optimal solution has quantized value three for each of the four variables. However quantized value three represents the interval that goes from .4 to 1.2. A series of examples will be now generated with x values included in the range .4 .. 1.2. However this process may not be very efficient, or in the worst case as inefficient as a random walk, because it may be necessary to generate several individuals before findingthe right combination of variables (1, 1, 1, 1). To overcome this obstacle, the quantization domain size is increased, to represent with more resolution the cell where all the examples are confined. An alternative solution would be to keep the quantization domain size constant, and make the new representation space be the cell in which the best solution resides. For example, in the latter case, the representation space would not be bound between –2 and 2, but between .4 and 1.2.

In this experiment was not necessary to do any of the above, since the best solution was found after 283 births.

Conclusion Experiment 1:

With this experiment, it was shown that LEM2 correctly partition the space into smaller subspaces. Together with the findings of experiment 2, they were the starting point to develop the theories of increasing the quantization domain size or to change the representation space, once all the population is confined to a single cell.

Understanding how to dynamically set the quantization domain size is particularly important, because it relives the user from having to set a crucial parameter, and allow LEM2 to achieve much better results.

In order to perform this experiment in a fast and good manner, LEM2 was equipped with the option of automatically generating KV input file.

Experiment 2: - Try to determine the importance of the Quantization Domain Size, and of the AQ mode.

The goal of this experiment is to observe the behavior of the LEM algorithm using different combinations of quantization domain sizes, and AQ modes. These two parameters play a major role in the LEM evolutionary procedure, and it is important to analyze how to correctly set them to obtain best results.

The results of experiment 2 are relative to only one set of runs. Several sets of runs have been done using different random seeds, AQ parameters and different landscapes, and they all lead to the same conclusions. The results shown here are the best representatives for this type of experiment.

Following is a summary of all the parameters for this experiment.

Algorithm:	**UniLEM**
Function:	F2 – Minimum
Number of Runs	6
Random seed	12345
Number of Variables	20
Bounds	-5.28 < x1 < 5.28 -5.28 < x2 < 5.28 -5.28 < x3 < 5.28 -5.28 < x4 < 5.28
Selection Method	Fitness-Based
High Threshold	30%
Low Threshold	30%
AQ mode	ic - dc
AQ trim (Gen Level)	Spec
AQ Start Trim	Gen
AQ Trim Probe	2
AQ Ambig	Neg
Population size	100
Initial Quantization Levels	5, 10, 50, 100
Ending Condition	220 generations (22000 births)

Figure 20: Experiment 2 Settings

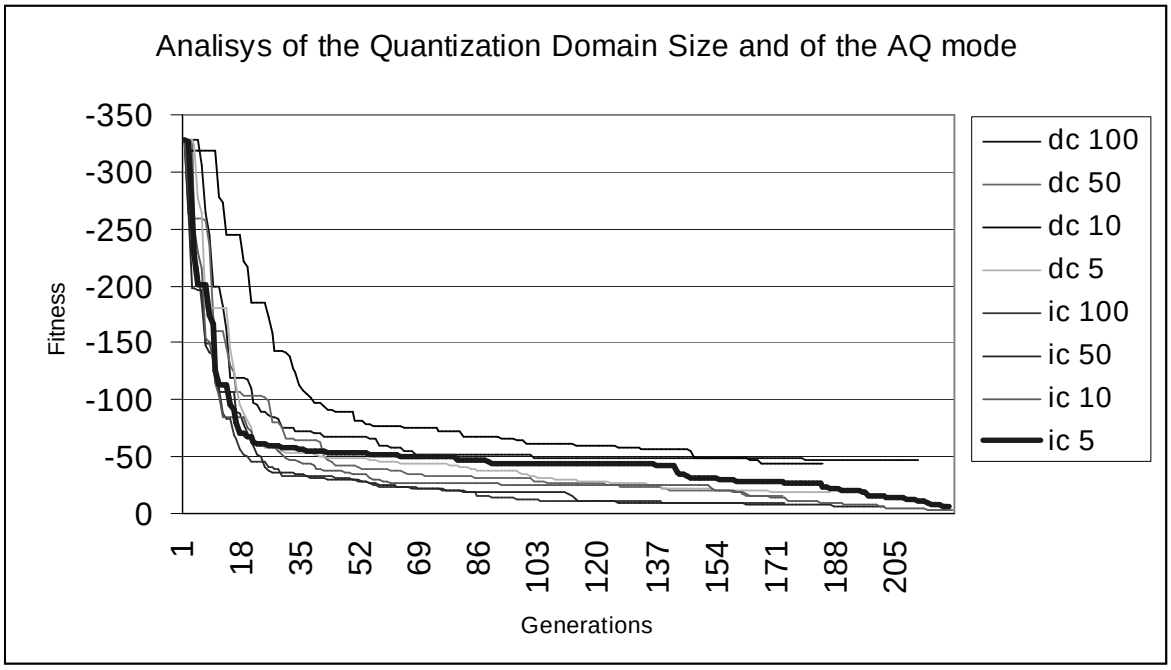


Figure 21: BSF curves for different types of quantization levels and AQ modes

Figure 21 shows the effect that different combinations of quantization domain sizes and of AQ modes have on the evolution process. It is possible to note that the quantization domain sizeplays a major role as it was expected, however these results lead to surprising conclusions. It was expected that using a higher quantization domain size, much better results would be achieved. The surprise was that the lowest fitness was obtained both using very high and very low domain sizes. This is an important finding, because as shown in figure 22, the execution time drastically decreases as the quantization level decreases.

This finding is also important because it shows the characteristics of abstraction. This finding lead to the major problem discussed in Experiment 1, of how to dynamically adjust the quantization domain size. It is not possible to keep the quantization level constant throughout the evolution for the reasons given in the previous experiment.

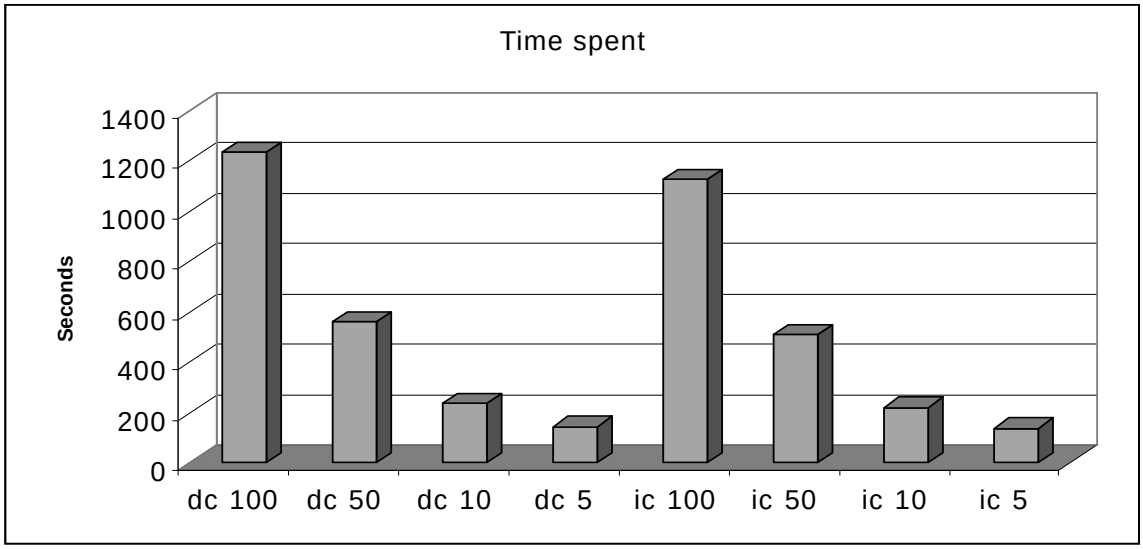


Figure 22: Time spent in seconds to complete 220 generations in the different modes

This graph shows that there was a significant difference in execution time, depending on which quantization domain size was used. On the other hand, there was minimal difference on the type of AQ mode used. The dc mode performed relatively slower, and worst than the ic mode.

Conclusion Experiment 2:

This experiment pointed out several important facts that have considerably affected the development of both the methodology and the implementation. First of all discovering that a low quantization domain size performs nearly as good as a higher one, boosted the speed of the experiments by approximately eight times.

Together with the findings from experiment 1, it emphasized the necessity of dynamically setting the domain size.

Experiment 3: - Compare duoLEM with EV

The goal of this experiment is to compare how a combination of LEM and EV performs over EV alone.

Following is a summary of all the parameters for this experiment.

Algorithm:	**duoLEM, EV**	**EV3 - EV**
Function:	F2 – Minimum	
Number of Runs	23	9
Random seed	random	
Number of Variables	20	
Bounds	-5.28 .. 5.28 for every X	
Selection Method	Population-Based	Uniform Selection
High Threshold	20%	N/A
Low Threshold	20%	N/A
AQ mode	Ic	N/A
AQ trim (Gen Level)	Spec	N/A
AQ Start Trim	Gen	N/A
AQ Trim Probe	1	N/A
AQ Ambig	Empty	N/A
Population size	100	
Children Population Size	N/A	1, 10
Symbolic Learning Learn Probe	3	N/A
Darwinian Learn Probe	N/A	2
Initial Quantization Levels	5	N/A
Ending Condition	1000 generations (100,000 births)	

Figure 23: Experiment 3 Settings

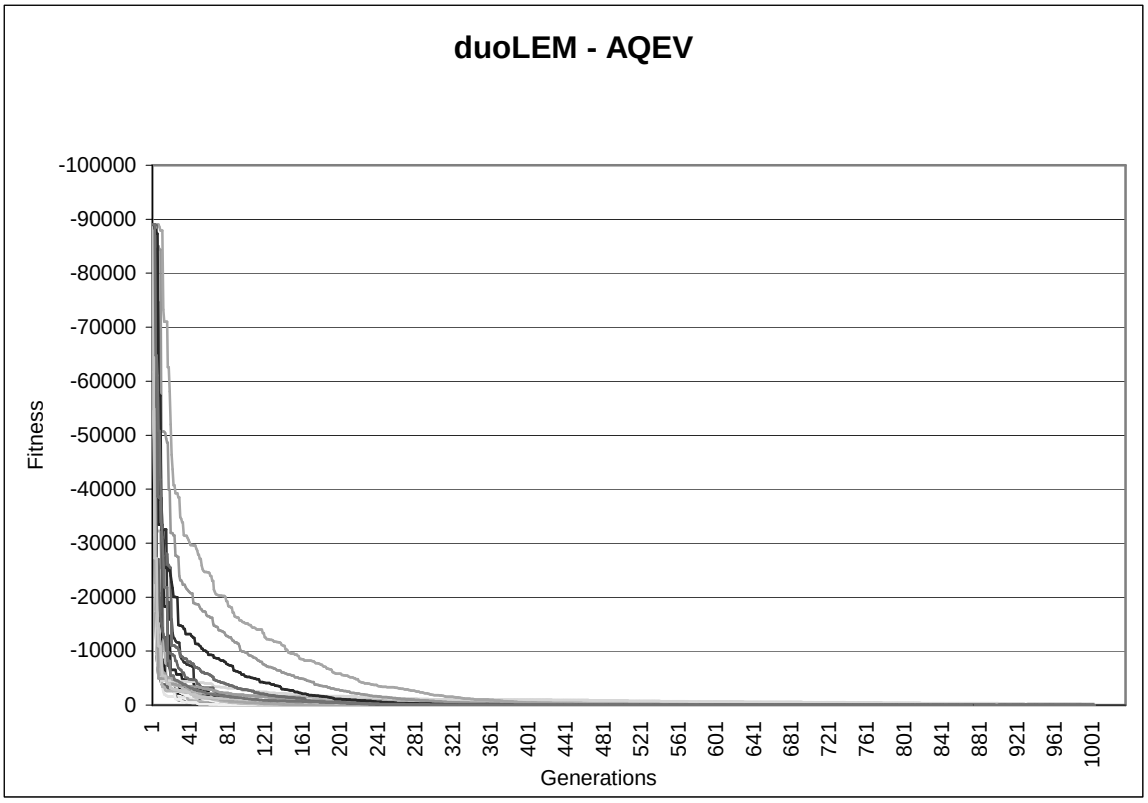


Figure 24: duoLEM AQEV

This graph shows 25 different runs of duoLEM. In this experiment, AQ was run in combination with EV. It is possible to notice that the except for three exceptions, all the runs follow almost the same path. This implies a small variance in the algorithm, that means that for this particular problem, random seed, and population dynamics are not important. Ideally it would be important to prove that the Learnable Evolution Model is not affected as much as the Darwinian Evolutionary Algorithms by the changes of the parameters. All the runs find the minimum of the function.

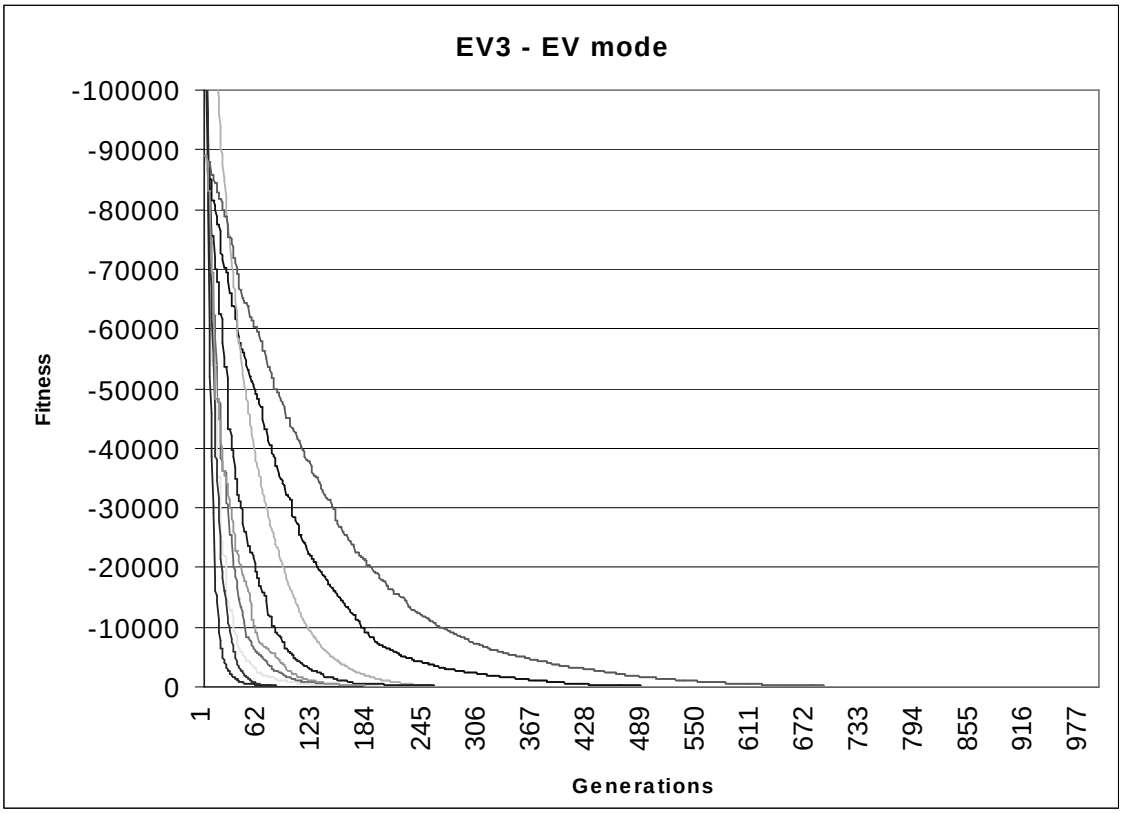


Figure 25: EV3 - EV

This graph shows 9 different runs of EV3 run in EV mode. It is possible to notice that all the runs follow almost the same path. It is possible to notice that the variance is higher than for AQEV, and that the lowest fitness is achieved in a higher number of generations. All the runs find the minimum of the function.

Conclusion Experiment 3:

This experiment pointed out that a combination of AQ + EV performs better than EV alone. This does not imply that duoLEM will always performs better than the Darwinian Evolutionary Algorithm, as more experiments must be run to ensure the correctness of the methodology.

It is interesting to notice that the variance remains lower as the parameters are changed, which would imply that the LEM methodology is not prone to parameters tuning as much as Darwinian Algorithms are. This is important because correctly setting the parameters is very difficult, and not clear solutions have been found yet.

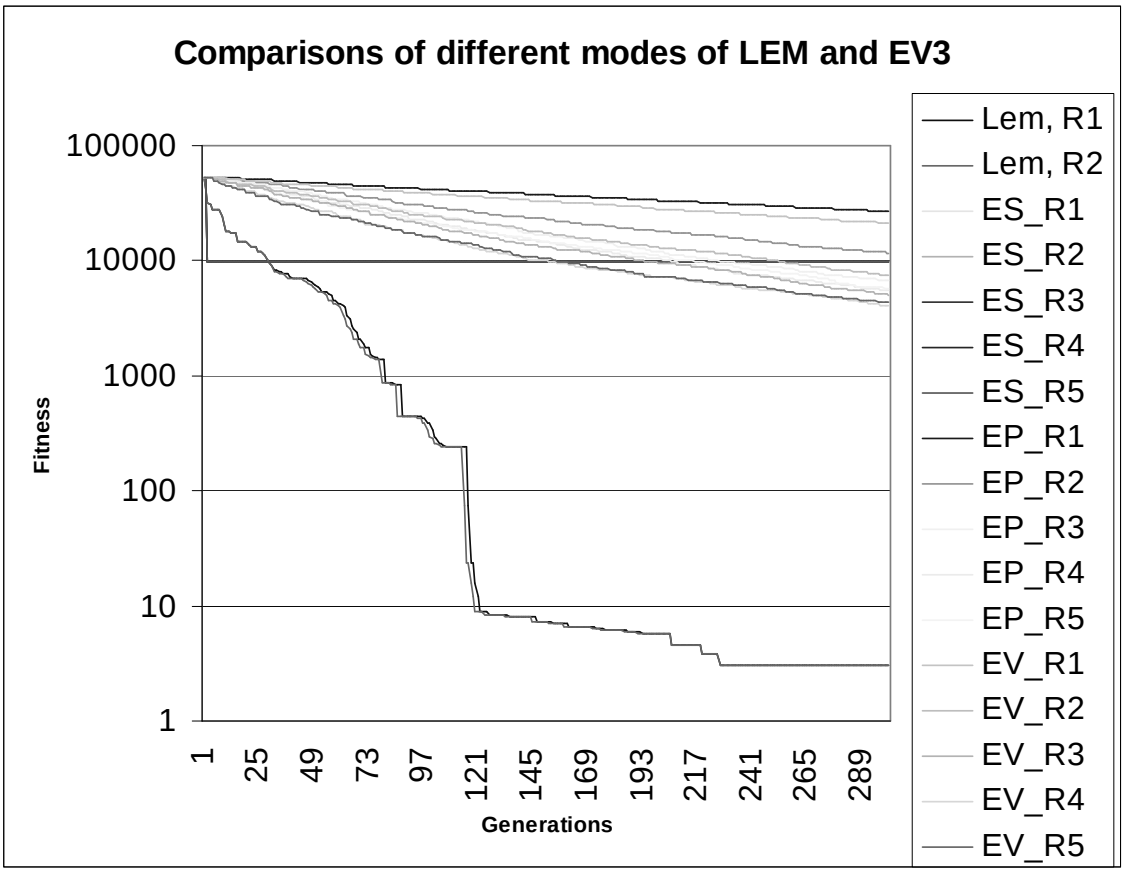
Experiment 4: - Compare LEM with EV on a high dimensional problem, to test the performance and the scalability

The goal of this experiment is to compare both uniLEM with all the three modes of EV3 over a high dimensional optimization problem

Following is a summary of all the parameters for this experiment.

Algorithm:	**UniLEM**	**EV ES EP**
Function:	F2 – Minimum	
Number of Runs	2	15
Random seed	Random	
Number of Variables	200	
Bounds	-5.28 .. 5.28 for every X	
Selection Method	Population-Based	Uniform Selection
High Threshold	20%	N/A
Low Threshold	20%	N/A
AQ mode	Ic	N/A
AQ trim (Gen Level)	Spec	N/A
AQ Start Trim	Gen	N/A
AQ Trim Probe	1	N/A
AQ Ambig	Empty	N/A
Population size	200	
Children Population Size	N/A	1, 10
Mutation Rate	N/A	.1, .3, .5, .7, .9
Symbolic Learning Learn Probe	3	N/A
Darwinian Learn Probe	N/A	2
Initial Quantization Levels	5	N/A
Ending Condition	1000 generations (100,000 births)	

Figure 26: Experiment 4 Settings



	1 – uniLEM HFT 30% LFT 30%		2 – uniLEM HFT 20% LFT 20%	
3 – ES Mutation Rate .1	4 – ES Mutation Rate .3	5 – ES Mutation Rate .5	6 – ES Mutation Rate .7	7 – ES Mutation Rate .9
8 – EV Mutation Rate .1	9 – EV Mutation Rate .3	10 – EV Mutation Rate .5	11 – EV Mutation Rate .7	12 – EV Mutation Rate .9
13 – EP Mutation Rate .1	14 – EP Mutation Rate .3	15 – EP Mutation Rate .5	16 – EP Mutation Rate .7	17 – EP Mutation Rate .9

Figure 27: Comparisons of LEM and EV3 in EV, ES, and EP over a high dimensions problems

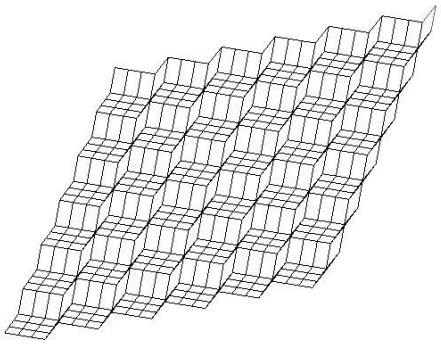
This graph is of particular interests because it shows the ability of LEM to optimize a higly dimensional problem. 250 variables are considered simultaneously, and even though the minimum is not met in the target number of generations, its proximity to the ideal solution is remarkable, especially if compared to the different modes of EV.

Conclusion Experiment 4:

This experiment shows the remarkable results that LEM achieved. It shows that the scalability of the algorithm works very well, but that can still be improved. The incapacity of the Algorithm to achieve the lowest fitness is because of a problem that arises with the quantization of the variable. AQ actually finds the area of the space where the optimal solution is included, however LEM2 is not able to represent the examples. This problem is being solved by adapting an Adaptive Anchor Quantization model. [Michalski-Cervone 1999]

Experiment 5

Experiment 5 is relative to De Jong's function 3. The function is defined and plotted as following:



$$f_3(x_i) = \sum_{i=1}^{n} \text{floor}(x_i), \qquad -4.12 \le x_i \le 4.12$$

This STEP function is the representative of the problem of flat surfaces. Flat surfaces are obstacles for optimization algorithms, because they do not give any information as to which direction is favorable. Unless an algorithm has variable step sizes, it can get stuck on one of the flat plateaus.
The minimum of the function is found when all the variables are equal to –5.

Figure 28: De Jong's function 3 (STEP)

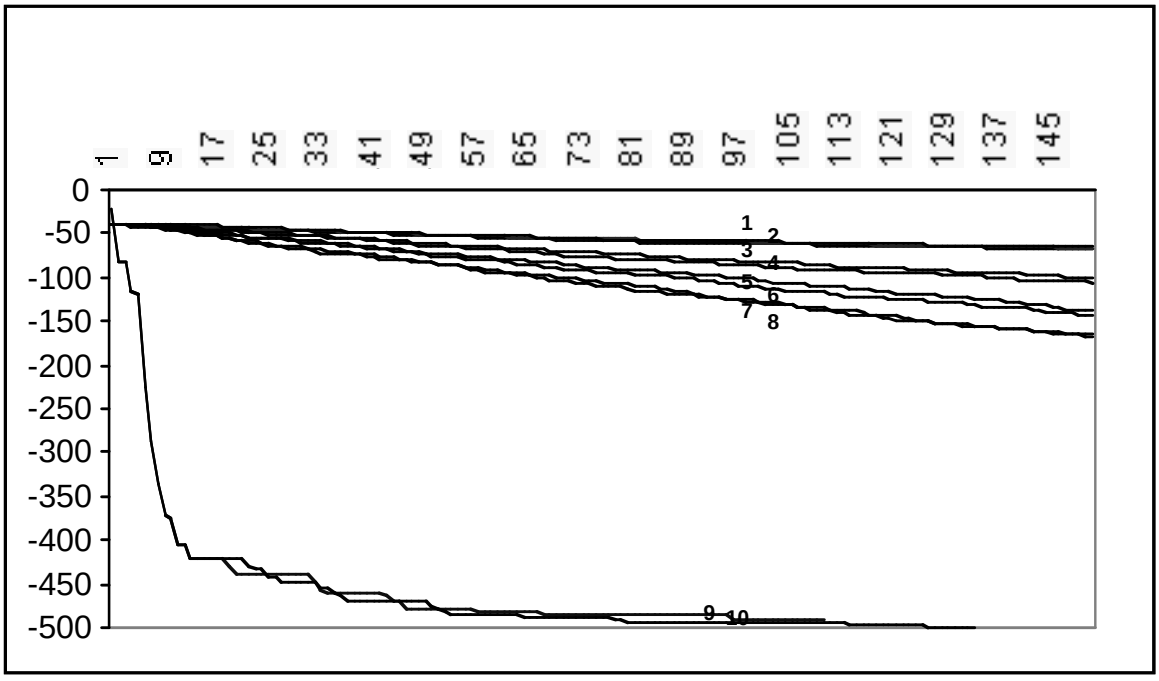
Experiment - 5: Compare LEM with EV on a high dimensional problem on a function that represents flat surfaces.

The goal of this experiment is to compare both uniLEM and duoLEM with all the three modes of EV3 over a high dimensional optimization problem. This function is of particular interest because Evolutionary Algorithms are usually not able to perform well over such a problem. On the other hand LEM has a clear advantage because it does not need to receive constant feedback. Once the original points are plotted in the space, LEM will calculate the qualitative differential [Michalski 1999], and evolve in the direction that is considered to be favorable.

Following is a summary of all the parameters for this experiment.

Algorithm:	**UniLEM, duoLEM**	**EV ES EP**
Function:	F3 – Minimum	
Number of Runs	2	8
Random seed	12345	
Number of Variables	100	
Bounds	-4.12 .. 4.12 for every X	
Selection Method	Population-Based	Uniform Selection
High Threshold	30%	N/A
Low Threshold	30%	N/A
AQ mode	Ic	N/A
AQ trim (Gen Level)	Spec	N/A
AQ Start Trim	Gen	N/A
AQ Trim Probe	5	N/A
AQ Ambig	Empty	N/A
Population size	100	
Children Population Size	N/A	1, 10
Mutation Rate	.5	.1, .3, .5, .7, .9
Symbolic Learning Learn Probe	3	N/A
Darwinian Learn Probe	N/A	2
Initial Quantization Levels	5	N/A
Ending Condition	145 generations (1,450 births)	

Figure 26: Experiment 5 Settings



1 – EP 100, 10, 0.1;	2 – EV 100, 10, 0.1;	3 – EP 100, 10, 0.5
4 - ES 100, 0.5;	5 – EP 100, 10, 0.9;	6 – ES 100, 0.9
7 – EV 100, 10, 0.5;	8 - EV 100, 10, 0.9;	
9 - uniLEM with HFT 30% LFT 30%	10 - duoLEM (AQ18 + EV 100,10, 0.5)	

Figure 27. An evolutionary process for finding the minimum of f_3 with 100 variables using EP, EV and ES evolutionary computation methods and LEM-2 in uniLEM and duoLEM versions

Figure 27 shows one of the most interesting of the results found. As it was predicted LEM performs considerably better than the Darwinian Evolutionary Algorithms, by getting very close the the minimum in uniLEM mode, and by finding the exact solution in duoLEM mode.

The reason why uniLEM is not able to achieve the maximum fitness is because at the end of the evolution there is not enough diversity in the population, and so the qualitative differential is not calculated correctly. To solve this problem it would be enough to detect such a situation, and increase the generalization level, so that more areas of the space are covered by individuals.

To obtained better performances the aqTrimProbe was set to 5, so that the spaces is further explored at the beginning of the evolution, and each time the quantization level is increased.

Conclusion Experiment 5:

Experiment five confirmed the expectations that LEM2 would run well over such a particular landscape. The high number of dimensions of the space made this problem particularly difficult and interesting.

Experiment 6

Experiment 6 is relative to De Jong's function 4. The function is defined and plotted as following:

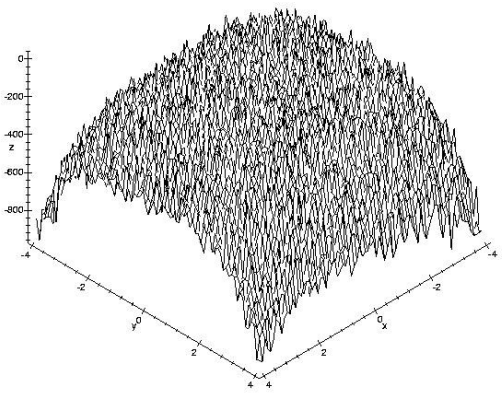


Figure 28: De Jong's function 4 (QUADRATIC)

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

The QUARTIC function is a simple unimodal function padded with noise. The gaussian noise makes sure that the algorithm never gets the same value on the same point. Algorithms that do not do well on this test function will do poorly on noisy data. [Yuret 1997]

Experiment 6: - Compare LEM with EV on a high dimensional problem on a function that simulates noisy data using the defaults parameters.

The goal of this experiment is to compare uniLEM with all the EV mode of EV3 over a high dimensional optimization problem, using for both problems the default parameters. This function is of particular interest because the gaussian error introduces makes sure that a point never gets the same fitness. However the main shape is a simple quadratic function.

One of the major problem with Evolutionary Algorithms is that the settings for the population dynamics, random seed, mutation rate etc play a major role in the outcome of this experiment.

In this experiment LEM is run five times with different random seeds using the default parameters, while EV is run twenty times, using the same random seeds used for LEM, and a different set of mutation rates.

Following is a summary of all the parameters for this experiment.

Algorithm:	**UniLEM, duoLEM**	**EV ES EP**
Function:	F4 – Minimum	
Number of Runs	5	20
Random seed	random	
Number of Variables	100	
Bounds	-5.28 .. 5.28 for every X	
Selection Method	Fitness-Based	Uniform Selection
High Threshold	30%	N/A
Low Threshold	30%	N/A
AQ mode	Ic	N/A
AQ trim (Gen Level)	Spec	N/A
AQ Start Trim	Gen	N/A
AQ Trim Probe	5	N/A
AQ Ambig	Empty	N/A
Population size	100	
Children Population Size	N/A	1, 10
Mutation Rate	.5	.1, .3, .5, .7, .9
Symbolic Learning Learn Probe	3	N/A
Darwinian Learn Probe	N/A	2
Initial Quantization Levels	5	N/A
Ending Condition	150 generations (15,000 births)	

Figure 28: Experiment 6 Settings

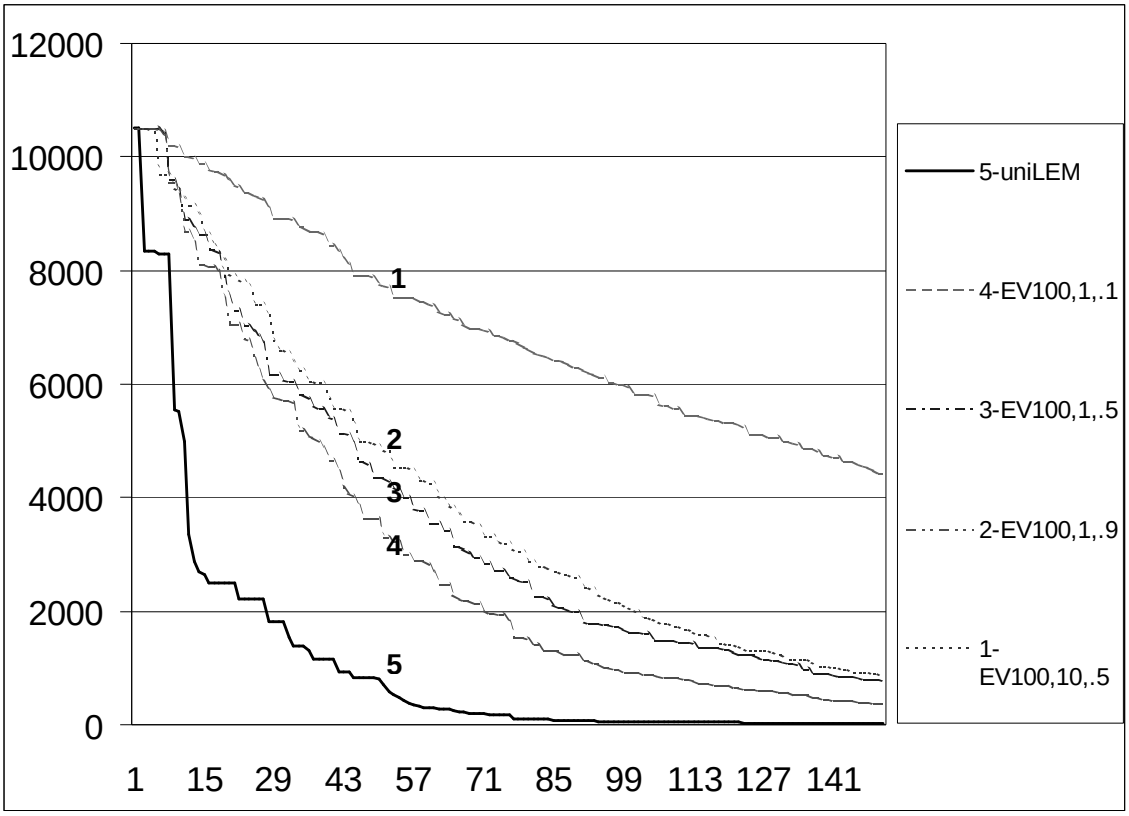


Figure 29. An evolutionary process for finding the minimum of f_4 with 100 variables using uniLEM and EV

This graph shows that LEM performed well on a noisy landscape. Each line is the best representative of 10 runs. Unfortunately it was not possible to plot the variance for each line, which would give a better description of the behavior of the algorithm.

It is interesting to note the crucial role that the mutation rate plays in EV mode. This emphasizes one of the strengths of the LEM methodology, which is the ability of correctly tuning the parameters.

Conclusions experiment 6:

This experiment was important to determine the ability of LEM to perform well over a highly dimensional landscape which contains noisy data.

The most important finding is that LEM performed very well with the default parameters, and together with the findings of the previous experiments, it leads to the conclusion that the LEM methodology is not highly dependent on parameters tuning. This makes the methodology very robust, and shows the correctness of both the implementation and of the theory.

VII. Known Issues

Several issues were learned from this implementation. The most important findings, which will lead to a further expansion of this project are:

- The quantization of the variables plays a major role. The χ^2 technique that is used does not work well, because it makes it very difficult to correctly represent integer values. It is absolutely necessary to adopy an Adaptive Anchored Quantization (AAQ) [Michalski-Cervone 1999].

- The Quantization Domain Size does not have to be very big. In the old Implementation of LEM it was fixed to 200, and this made experiments very slow. Furthermore, it was very underperforming if the variables were defined over a very large interval.

- The LEM methodology seems to be effective even without the use of the Darwinian Evolutionary Algorithm. In all the he experiments in which duoLEM outperformed uniLEM, it was observed that LEM2 was not able to correctly instantiate individuals from the rules found by AQ. This is a problem related to the χ^2 method of quantization.

- AQ fails to correctly extract rules on some valid input files. This is true especially when it is run in **dc** mode with **spec** generalization level. The problem is not crucial because the dc mode seemed to perform worst than **ic** (or **vl**) mode.

- AQ creates a rule to describe low group when used in **vl** mode in combination with **spec** generalization level. This may confuses the parser, since the rule is bogus. In **vl** mode AQ should only create rules to characterize the High group, and it does so when it is used in combination of **gen** and **mini** generalization level.

- The newer version of AQ generates an extra column which represent the weight of a particular individual. Unfortunately the KV parser was not designed to take into consideration this extra column, and fails to read the input file. It is then necessary to post-peocess the AQ output file, and eliminating the Weight column.

VIII. Future Work

Several issues arise from the implementation that will further extend the methodology in the future. The problems and solutions discussed below are derived from the behaviors observed in the experiments described above, and in many others not listed.

1. First of all the inability to have full control over a population may limit the effectiveness of the rules found, and consequently will not achieve best results. This problem is crucial and it requires to reimplement LEM independently from other already existing Evolutionary Algorithms, since the contest is completely different. This task is undergoing by developing complete and powerful libraries for Evolutionary Computations, that not only will allow the implementation of traditional Darwinian Evolutionary Algorithm, but also the easy implementation of the LEM methodology.

2. The quantization of the variables presents a crucial and difficult problem for the correct behavior of the methodology. LEM2 uses X2 to quantize the variables, and this technique fails to correctly describe the integer boundaries of the variables. Sometimes LEM2 fails to achieve the best value, because it is unable to correctly represent the rules generated by the rule learner. To overcome this problem it is suggested to use a *dynamic anchor quantization*. This problem is now being solved in collaboration with Dr. Michalski.

3. LEM2 passes an entire population to the rule learner, divided into high examples and low examples. It would be interesting to assign to each individual a unique indentifier (a label), which represents the rule from which the individual was generated. AQ will analyze populations of homogeneous individuals with respect to the identifier, so that it will specialize each rule separately.

4. Quantize the fitness into ten intervals using X2, and use it to define the weight of the examples. This allows to generate examples depending on the quality of the rules found, their total weight, and unique weight.

5. Implement and experiment different survivors selection mechanisms, and analyze the behavior of the program.

6. Improve the interaction with the KV program. LEM2 will generate an input file for the KV program, but it will have to be called separately. In order to automate this process, it will be necessary to modify the code for KV. Another improvement could be to visualize each variable with a different colour, so to monitor the rate of convergence of the program. [De Jong 1993]

7. Test how LEM performs in a Dynamic Landscape. First examining the progress made in Evolutionary Compuation with cope with this problem, and also analyzing the idea of Concept Drifting.

IX. References

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

Bloedorn, E., Kaufman, K., Michalski, R.S., and Zhang, Q., *An Implementation and User's Guide of the AQ18 Learning and Data Mining Environment*, Reports of the Machine Learning and Inference Laboratory, George Mason University, 1999 (to appear).

Bloedorn, E. and Michalski, R.S., *Data-Driven Constructive Induction: A Methodology and Its Applications*, Special issue on Feature Transformation and Subset Selection, *IEEE Intelligent Systems* Huan Liu and Hiroshi Motoda (Eds.), March-April 1998.

Cervone, G. and Michalski, R.S., *Design and Experiments with LEM-2 Implementation of the Learnable Evolution Model*, Reports of Machine Learning and Inference Laboratory, 1999 (to appear).

Clark, P. and Niblett, R., *The CN2 Induction Algorithm*, *Machine Learning*, No.3, 1989.

Coletti M., Lash, T., Mandsager, C., Michalski, R.S., and Moustafa, R., Comparing the Learnable Evolution Model with Genetic Algorithms in the Area of Digital Filter Design, *Reports of Machine Learning and Inference Laboratory* MLI 99-4, George Mason University, Fairfax, VA, May, 1999.,

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

de Garis, Hugo, "CAM-BRAIN: The Evolutionary Engineering of a Billion Neuron Artificial Brain by 2001 Which Grows/Evolve at Electronic Speeds Inside a Cellular Automata Machine (CAM)," *Lecture Notes in Computer Science* - Towards Evolvable Hardware, Vol. 1062, pp 76-98, Springer-Verlag, 1996.

de Garis, H., Korkin, M., Gers, F., Nawa, E., Hough, M., "Building an Artificial Brain Using an FPGA Based CAM-Brain Machine", *Applied Mathematics and Computation Journal*, Special Issue on "Artificial Life and Robotics, Artificial Brain, Brain Computing and Brainware", North Holland. AMC Journal (Invited by Editor, to appear 1999).

De Jong, K.A., *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. thesis, Department of Computer and Communication Sciences, University of Michigan, An Arbor, 1975.

De Jong, K.A., *Evolutionary Computation: Theory and Practice*, MIT Press, 1999 (to appear).

De Jong, K.A., *Learning with Genetic Algorithms: An Overview*, Machine Learning, Vol. 3, pp. 121-138, 1988.

De Jong, K.A. and Schultz, A.C., *Using Experience-Based Learning in Game Playing*, Proceedings of the Fifth International Conference on Machine Learning, Ann Arbor, MI, Oxford: Clarendon Press, pp. 284-290, June 1988.

Dietterich, T.G., *Machine-Learning Research: Four Current Directions, AI Magazine, Vol*. 18, No.4, 1997.

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

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

Hinton. G.E., and Nowlan, S.J. *How learning can guide evolution*, Complex Systems 1: 495-502, 1987.

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

Kaufman, K. and Michalski, R.S., AQ18: *An Implementation and User's Guide*, Reports of Machine Learning and Inference Laboratory, George Mason University, 1999 (to appear).

Koza, J.R., *Genetic Programming II: Automatic Discovery of Reusable Programs*, The MIT Press, 1994.

Maloof, M. and Michalski, R.S., *AQ-PM: A System for Partial Memory Learning,* Proceedings of Intelligent Information Systems VIII, Ustron, Poland, 14–18 June 1999.

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

Michalski, R.S., *Learnable Evolution Process: Evolutionary Process Guided by Machine Learning*, Accepted for Machine Learning , Special Issue on Multistrategy Learning, 1999

Michalski, R.S., *On the Quasi-Minimal Solution of the General Covering Problem*, Proceedings of the V International Symposium on Information Processing (FCIP 69), Vol. A3 (Switching Circuits), pp. 125-128., Yugoslavia, Bled, October 8-11, 1969.

Michalski, R.S., *Discovering Classification Rules Using Variable-Valued Logic System VL1*, *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford, CA, pp. 162-172, 1973.

Michalski, R.S., *A Theory and Methodology of Inductive Learning, Artificial Intelligence*, Vol 20, No. 2, pp. 111-161, 1983.

Michalski, R.S., *Inferential Theory of Learning: Developing Foundations for Multistrategy Learning*, in Machine Learning: A Multistrategy Approach, Vol. IV, R.S. Michalski and G. Tecuci (Eds.), Morgan Kaufmann, San Mateo, CA, 1994.

Michalski, R.S., *Learnable Evolution: Combining Symbolic and Evolutionary Learning*, *Proceedings of the 4th International Workshop on Multistrategy Learning*, Decenzano del Garda, Italy, June 11-14, 1998.

Michalski, R.S., "Natural Induction: Theory, Methodology and Its Application to Machine Learning and Data Mining," *Reports of the Machine Learning and Inference Laboratory*, George Mason University, 1999 (to appear).

Michalski, R.S., Bratko, I. and Kubat, M., *Machine Learning and Data Mining: Methods and Applications*, John Wiley and Sons, 1998.

Michalski, R.S. and McCormick, B.H., *Interval Generalization of Switching Theory*, Proceedings of the Third Annual Houston Conference on Computer and System Science, Houston, Texas, April 26-27, 1971.

Michalski, R.S., Mozetic, I., Hong, J., N. Lavrac, *The AQ15 Inductive Learning System: An Overview and Experiments*, Reports of the Intelligent Systems Group, No. 86-20, UIUCDCS-R-86-1260, Department of Computer Science, University of Illinois, Urbana, 1986.

Michalski. R.S. and Zhang, Q., *Initial Experiments with the Learnable Evolution Model*, Reports of Machine Learning and Inference Laboratory, George Mason University, 1999 (to appear).

Mitchell, M., *An Introduction to Genetic Algorithms*, Cambridge, MA, MIT Press, 1996.

Mitchell, T. M., *Does Machine Learning Really Work*, *AI Magazine*, Vol. 18, No.3, 1997.

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

Sebag, M. and Schoenauer, M., *Controlling Crossover Through Inductive Learning* In Y. Davidor, H.P. Schwefel, and R. Manner (eds*.), Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, Springer-Verlag, LNVS 866, pp. 209-218, 1994.

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

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

Spears, W.M. and De Jong, K.A., *Using Genetic Algorithms for Supervised Concept Learning*, Proceedings of the Tools for AI Conference, Reston, VA, November 1990.

Turney, P.D., Cost-sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm, *Journal of Artificial Intelligence Research,* vol. 2. pp. 369-409, 1995.

Yuret, Denits, *Mater Thesis,* Department of Computer Science, M.I.T. 1997, http://alpha-bits.ai.mit.edu/people/deniz/html/aitr1569/node2.html

Zhang, Q., *Knowledge Visualizer: A Software System for Visualizing Data, Patterns and Their Relationships* Reports of the Machine Learning and Inference Laboratory , MLI 97-14, George Mason University, Fairfax, VA, September, 1997.