Non-Darwinian Evolutionary Computation: Guiding Evolution by Machine Learning

Ryszard S. Michalski

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

Web: www.mli.gmu.edu/michalski
MLI Laboratory: www.mli.gmu.edu

Plan

- Introduction
- Learnable Evolution Model (LEM)
- Experiments
  - Function Optimization
  - Heat Exchanger Design
- Conclusion
- Demo of the iAQ Natural Induction Program

INTRODUCTION

Since its beginning, about 35 years ago, research in Machine Learning has centered on problems of inducing general descriptions of specific facts (training examples).

A question that I will explore today is whether methods for inducing such descriptions can be useful for an opposite task, for determining specific facts?

An answer to this question will be provided by considering evolutionary computation as a methodology for generating specific facts (individuals, problem solutions, etc.)

Darwinian Evolution

In his prodigious treatise

“On the Origin of Species by Means of Natural Selection,”

published in 1859, Darwin conceived the idea that the evolution of species is governed by:

“one general law, leading to the advancement
of all organic beings, namely, multiply, vary,
let the strongest live and the weakest die”

A century later, computer scientists adopted Darwinian evolution as a model of computation, which has proven to be applicable to a wide range of problems, and constitutes a foundation of the contemporary field of evolutionary computation.

Assumptions Behind Darwinian Model

- Individuals are holders and transmitters of their genetic material to the offspring; their life experience is not passed to the next generation
- Evolution proceeds through semi-random modifications of genotypes by mutations (asexual reproduction) and/or recombinations (sexual reproduction); it not guided by any "external mind"
- The evolution progresses to more advanced forms according to the "survival of the fittest ."

Lamarckian Evolution and Baldwin Effect

- Before Darwin formulated his theory, Jean-Baptiste Lamarck, a French naturalist (1744-1829), advanced the idea that traits learned during the lifetime of an individual can be transmitted to the offspring's genome
- Many scientists accepted, however, the idea of another mechanism through which learned traits might influence evolution, namely, the so-called *Baldwin effect* (Baldwin, 1896).

Darwinian Evolutionary Computation

The Darwinian-type evolutionary computation can be viewed as a general-purpose parallel search process:

1. It starts with a population of solutions
2. The individuals in the population are modified by *change operators*, which are typically various forms of mutations and/or recombinations
3. New solutions are evaluated using a fitness function, and the “best” ones are chosen, through a *selection method*, for the next generation
4. The process is repeated until a *termination condition* is met

A General Schema of Evolutionary Computation

(1) Initialization

t := 0

Create an initial population P(t) and evaluate fitness of its individuals.

(2) Selection

t := t+1

Select a new population from P(t) based on the fitness of individuals

P(t) := Select(P(t-1)

(3) Modification

Generate new individuals by *change operators* to : P(t) := Modify(P(t))

(4) Evaluation

Evaluate fitness of individuals in P(t)

(5) Termination

If P(t) satisfies the *termination condition*, END; otherwise go to **2.**

Advantages of Darwinian-type Evolutionary Computation

- The change operators are typically certain forms of mutations and recombinations, which are easy to implement and easy to execute using all kinds of data structures
- These operators can be applied without knowledge of the problem domain
- Consequently, Dawinian evolutionary computation has found a very wide range of applications, including many kinds of optimization and search problems, automatic programming, engineering design, evolvable hardware, game playing, machine learning, and many others.

Limitations of the Darwinian-type Evolutionary Computation

- The Darwinian-type evolutionary computation is a semi-blind process executed in parallel:
  - the mutation is a random, typically small, modification of a single solution
  - the recombination (crossover) is a semi-random combination of two or more solutions
  - selection is a form of parallel hill climbing
- The generation of new individuals is not guided by any principles learned from past generations, but is a trial and error process
- Consequently, Darwinian evolutionary computation tends to be not very efficient.

Modeling Nature May not Lead to the Best Technological Solutions

- The field of evolutionary computation has followed a long-practiced tradition of looking to nature for technological solutions
- The imitation of bird flying by mythological Icarus and Daedalus, or German aeronautical engineer Otto Lilienthal who build flying models employing flapping, bird-like, wings are early examples of such efforts
- In seeking technological solutions, the “imitate-the-nature” approach frequently does not lead to the best engineering results
- Modern examples of successful solutions not imitating nature include automobiles, airplanes, rockets, computers, etc.

Intellectual Evolution

- The evolution of technology and other human creations does not follow the Darwinian model of biological evolution
- Such an evolution is guided by humans who analyze advantages and disadvantages of previous solutions, and then use the findings to develop new solutions
- Due to such *intellectual evolution*, the process of evolving the automobile, airplane or computer from their primitive prototypes to modern forms was astonishingly rapid, taking just a few human generations.

Modeling Intellectual Evolution: LEM

- An attempt to implement the intellectual evolution has been undertaken in *Learnable Evolution Model* (*LEM*), which employs machine learning to guide the evolutionary process

- The central idea is to "genetically engineer" new individuals (solutions) by hypothesis formation and instantiation operators, rather than by random mutations and/or recombinations

- The initial idea of LEM was introduced at the Multistrategy Learning Workshop (MSL '98) in Desenzano del Garda, Italy.

Basic Idea of LEM

- The fundamental difference between LEM and Darwinian evolutionary algorithms is in the way it generates new individuals
- Specifically, at each step of evolution, LEM selects two groups of individuals from a population: the *H-group* and the *L-group* that consist of high performing and low performing individuals, respectively
- A machine learning program determines a general hypothesis discriminating between the H-group and L-group
- The hypothesis is then instantiated in various ways to generate new individuals.
- This process is repeated until a termination criterion is satisfied.

Two Forms of LEM

- The general form of LEM encompasses two versions:

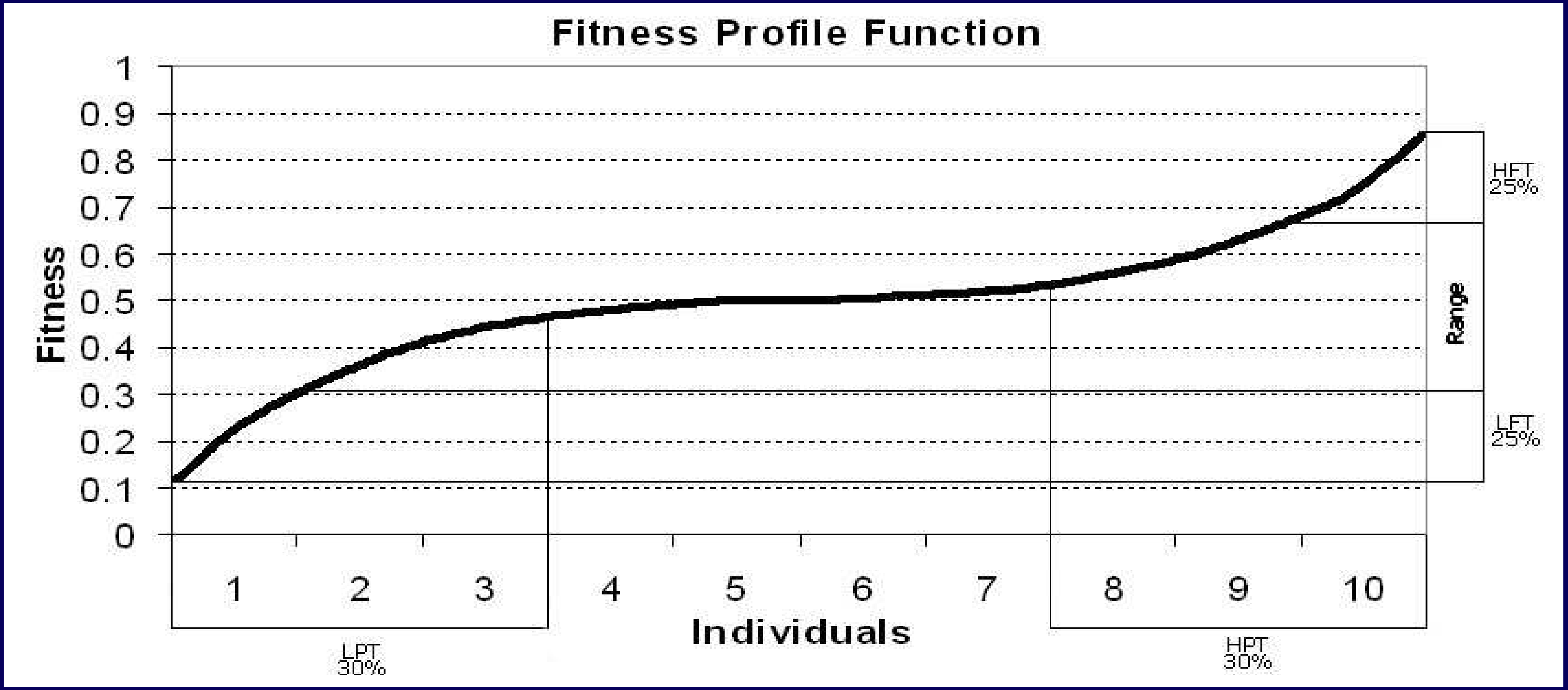
  uniLEM   and   duoLEM

- The uniLEM version generates new individuals for each new generation only by hypothesis formation and instantiation; it stops when *LEM Termination Condition* is met
- The duoLEM generates new individuals either by hypothesis formation and instantiation (Machine Learning mode), or by mutation and recombination (Darwinian Evolution). It toggles between the two modes, switching to another mode when *Mode Termination Condition* is met.

Major Issues in Implementing LEM

1. How to select H-group and L-group?
2. What machine learning method to use?
3. How to use hypotheses to generate new solutions?
4. Which method to use in DE mode (in duoLEM)?
5. How to define the LEM stopping criterion?
6. How to cope with continuous variables (when using a symbolic learning method)?

Selection of H- and L-group



- Population-based method
- Fitness-based method

Learning Method in LEM:

The AQ Natural Induction Program

- In principle, any learning method can be used in LEM
- In our first implementations, we employed AQ learning that has several features particularly useful for LEM:
  - ➢ Generates descriptions that are easy to instantiate
  - ➢ Descriptions are compact and easy to understand

    (due to attributional calculus (AC) that has higher representation power than conventional decision rules or trees, and facilitates natural induction)
  - ➢ Small syntactic changes in descriptions create to small semantic changes
  - ➢ The degree of description generality can be easily controlled
  - ➢ It has an easily modifiable multi-criterion description quality measure
  - ➢ Learned descriptions can be matched flexibly against examples

An Illustration of Input and AQ-generated Ouput At One Step of LEM Evolution

Parameters

run	ambig	trim	mode	maxstar
1	empty	spec	ic	1

Variables

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

H-group

#	x1	x2	x3	x4	Weight
1	7	7	7	8	12
2	7	6	6	7	9
3	7	6	7	8	7
4	6	6	7	8	5
5	6	7	6	8	5

L-group

#	x1	x2	x3	x4	Weight
1	6	6	9	8	5
2	6	7	10	8	3
3	7	7	11	7	1

A hypothesis generalizing the H-group:

[x1=6..7] & [x2=6..7] & [x3=7..8] & [x4 > 6]

(t= 38; u = 38; q= 0.8)

Note:

The values in the conditions of the rule above are symbols representing *ranges* of original values of these variables, not the values themselves.

The ranges were determined by the *adaptive anchoring discretization method, called* ANCHOR(Michalski & Cervone, 2000).

The ANCHOR Method:
Adaptive Anchoring Discretization

- When LEM is applied to problems of optimizing a function of a large number of continuous variables, a problem arises as to how to discretize these variables.

- To avoid the problem of insufficient precision or over-precision, the method of ANCHOR was developed that dynamically adapts the precision of the variables to the needs of the problem.

- It produces consecutively more precise discrete values that are rounded to the nearest whole numbers (“anchors”)

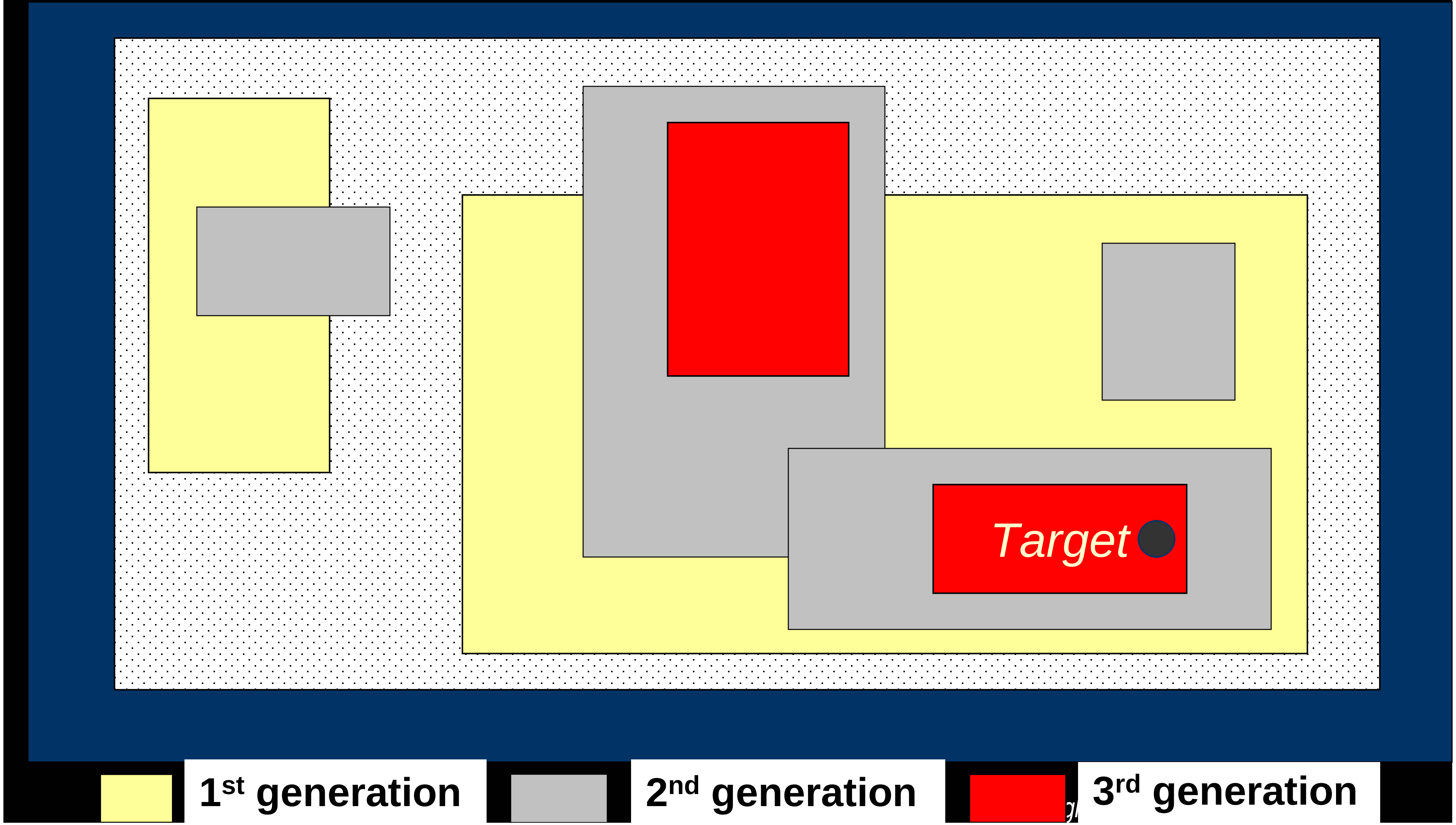
Generating New Individuals:
Proportional Instantiation

- New individuals are generated by instantiating rules created by the learning program in various ways

- The instantiation of variables that do not occur in the hypothesis is done by a random selection of values present in the training examples

- The number of individuals generated from a rule is proportional to the *rule fitness*, defined as the sum of fitness values of individuals covered by the rule.

LEM's Power is Due to a Progressive Reduction of the Search Space



🟨 1st generation ⬜ 2nd generation 🟥 3rd generation

Related Research

- Cultural algorithms (Reynolds, 1994; Rychtyckyj and Reynolds, 1999, 2000; Saleem and Reynolds, 2000, Rychtyckyj and Reynolds, 2001)

  They use high performing individuals to develop beliefs constraining the way in which individuals are modified by genetic operators.

- Population-based Incremental Learning or PBIL (e.g., Baluja, 1995; Baluja and Caruana, 1995).

  PBIL creates a real-valued probability vector characterizing high fitness solutions. Experiments have shown that PBIL may outperform standard genetic algorithms on some problems, and under-perform on some others.

Related Research (cont)

- Muhlenbein and Paas (1996) estimate the probability density functions of binary variables in their chromosomes by the product of the individual probability density functions.

- Pelikan and Goldberg (1999) developed an algorithm "BOA" (Bayesian Optimization Algorithm) that extended above ideas by using Bayesian Networks to model the chromosomes of superior fitness.

- Similar work has also been performed by Larranaga and Lozano, 2002 (Spanish group), who has given the term "EDA" (Estimation of Distribution Algorithms) to the statistical estimation approach to EC.

EXPERIMENTAL STUDIES

- ***Study 1: Function optimization***
  1A Using LEM1: Q. Zhang, RSM
  1B Using LEM2: G. Cervone, RSM
  1C Using LEM3: J. Wojtusiak, RSM (in progress)

- ***Study 2: Non-linear filter design***
  Using LEM1: M. Coletti, T. Lash, C. Mandsager,
  R.S. Michalski and R. Moustafa

- ***Study 3: Engineering design***
  Using ISHED1 for optimizing heat exchangers
  K. A. Kaufman and R. S. Michalski in collaboration with NIST

ES: Evolution Strategy

A Darwinian Evolution Program Used in the Experiments

- Data points are vectors of real values
- New individuals are created by:
  - selecting variables for mutation with the probability 1/*L*, where *L* is the vector length
  - mutating attribute values according to a Gaussian distribution, in which the mean is the value being mutated, and the standard deviation is a controllable parameter
- Using binary tournament selection method

(Individuals in the parent population are selected randomly and their fitness is compared with the fitness of randomly chosen new individuals. The winning individual becomes a member of the new population and the loosing individual is deleted. The process lasts until the list of new individuals is empty.)

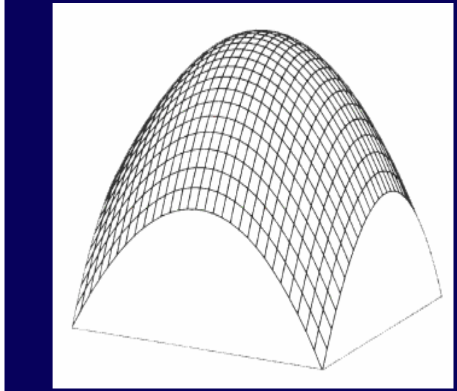
Study 1B: Function Optimization

- LEM2 was applied to a wide range of function optimization problems and its performance was compared to that of ES (Evolution Strategy program) on the same problems

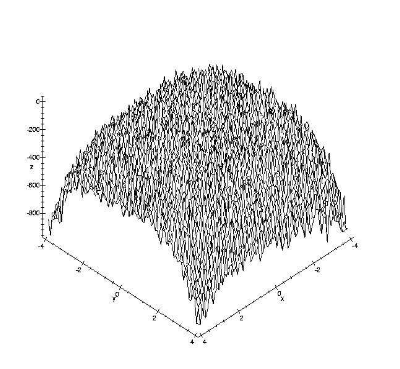
- LEM2 was also applied to problems for which the best known results were published on the web

- What is shown in the next viewgraphs is a small sample of fairly typical results

Functions Used in the Experiments

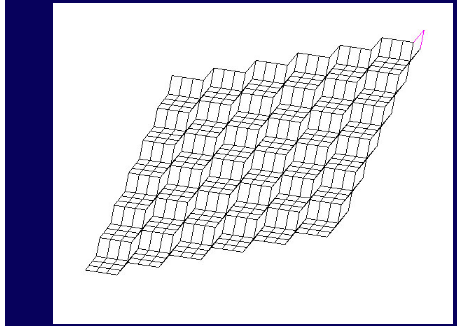


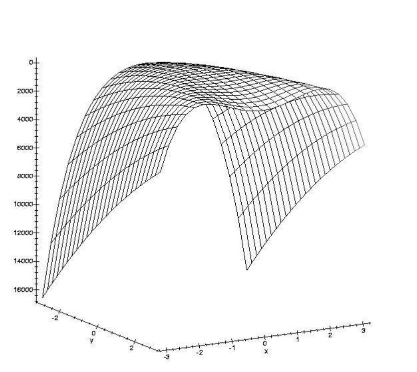
Sphere



Guassian Quartic

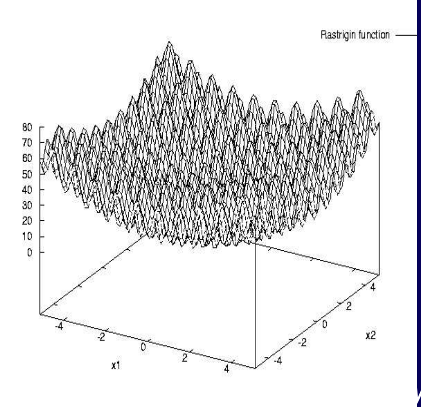
Step

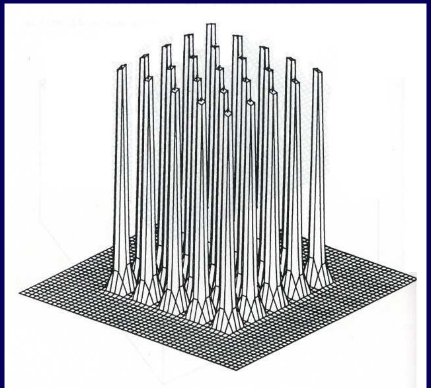




Rosenbrock

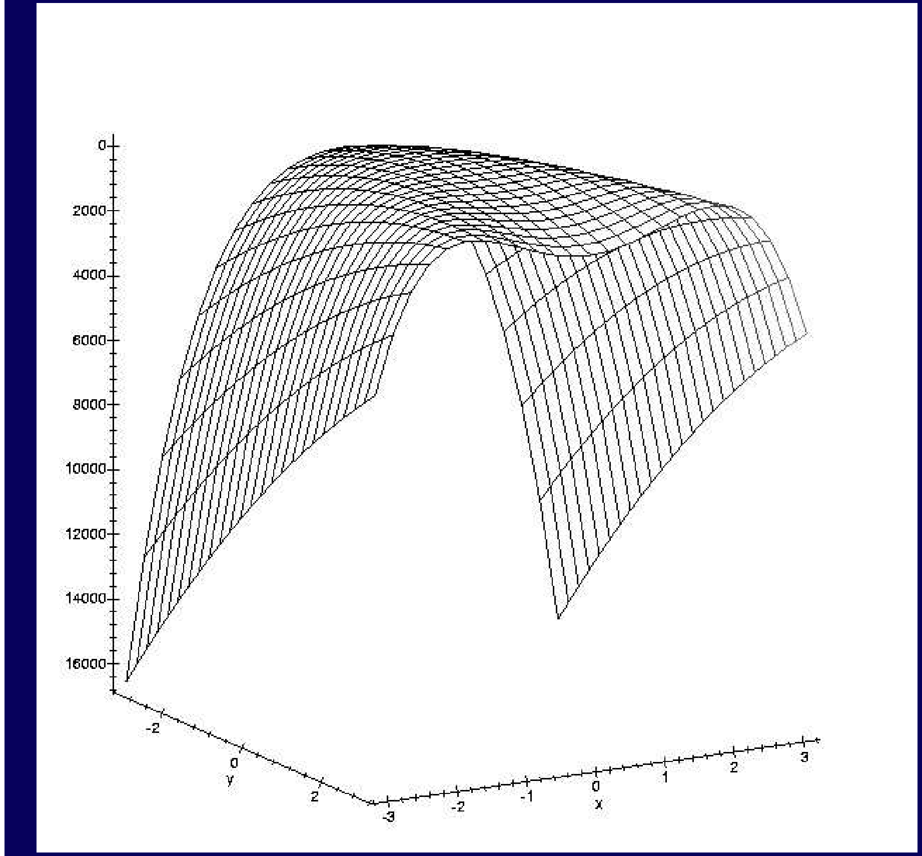
Rastrigin





Shekel's foxholes

Rosenbrock Function

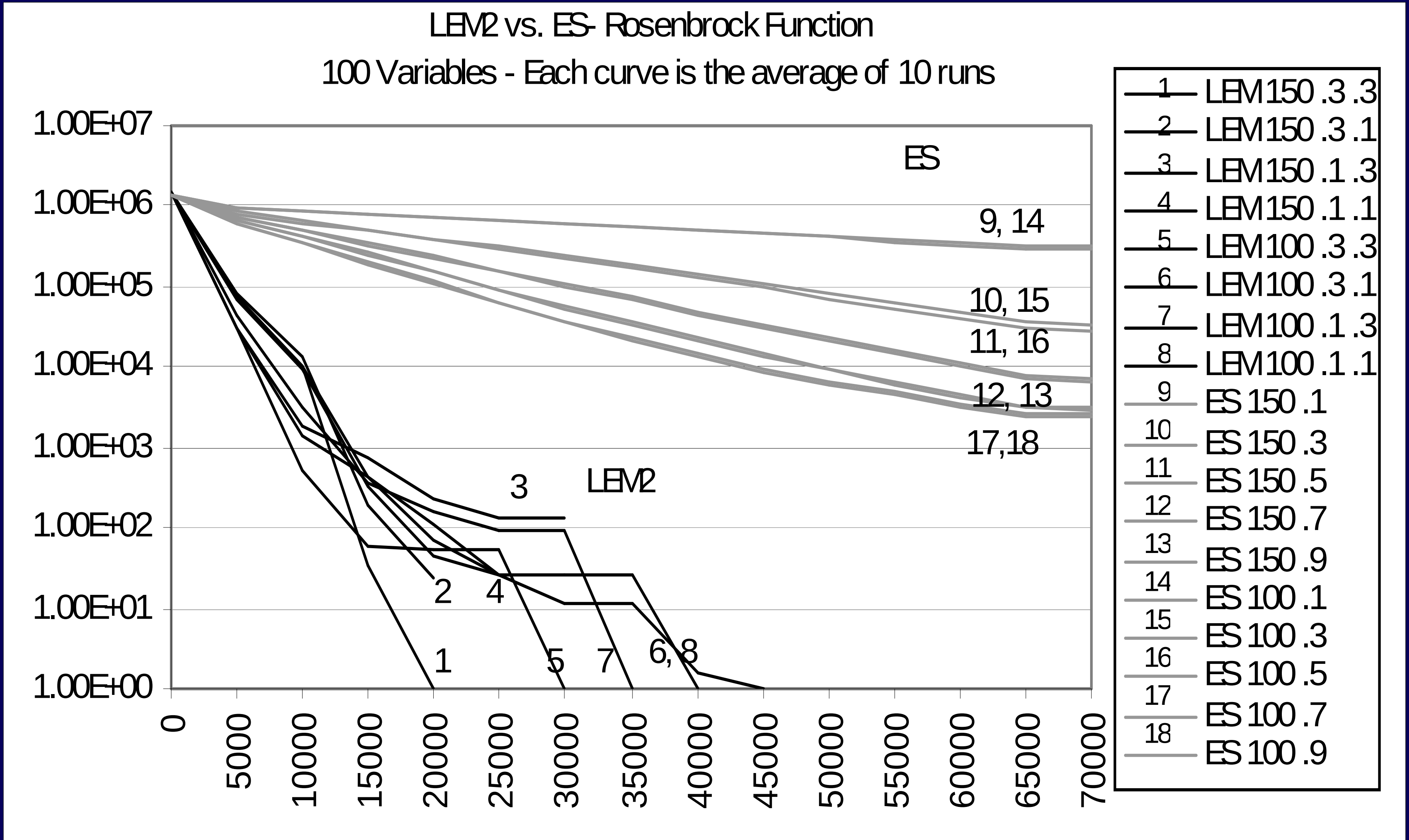


From a collection of functions by Dr De Jong

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

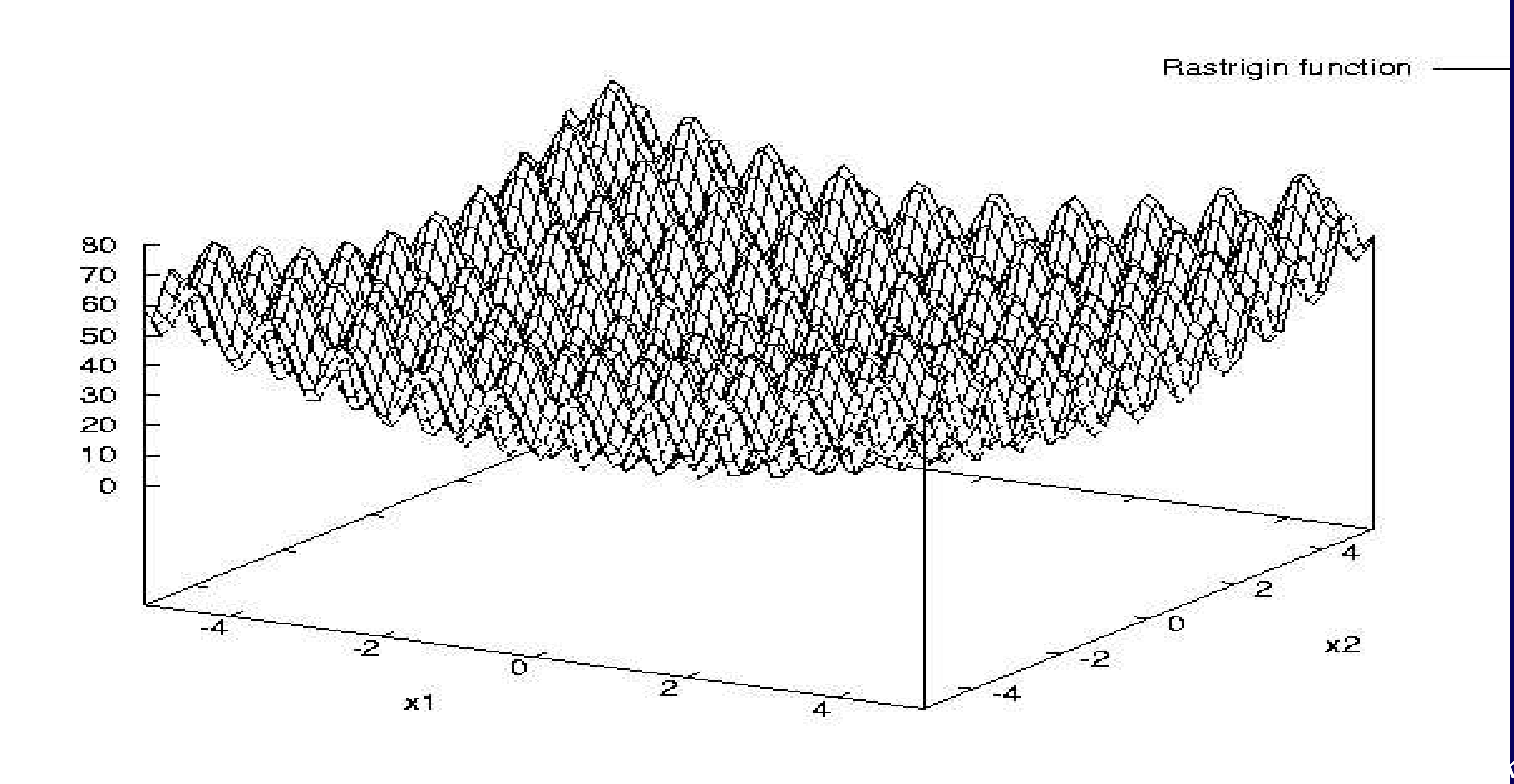
This function represents a very complex minimization problem. It has a very narrow ridge. The tip of the ridge is very sharp, and it runs around a parabola. Algorithms that are not able to discover good directions under perform in this problem. The function is plotted here on a very small interval. The 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.

Results from LEM2 and ES in Optimizing the Rosenbrock Function of 100 Variables

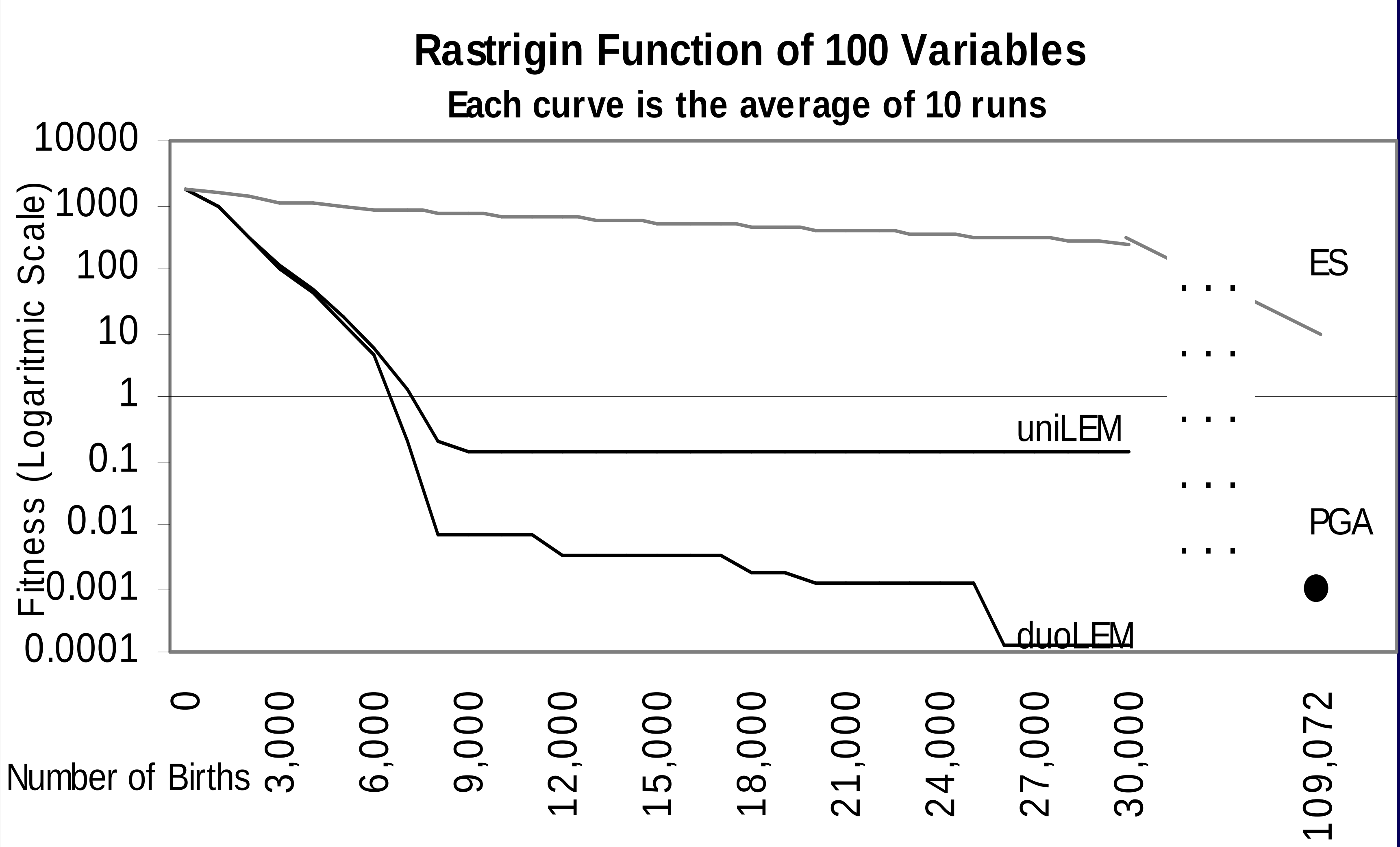


The Rastrigin Function

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



Minimizing the Rastrigin Function of 100 Variables by ES, PGA and LEM2



LEM2's Results vs. the Best Results Published on the Web

- In this experiment, LEM2 was applied to problems for which best known solutions were published by the members of the Evolutionary Computation community on the website under URL:

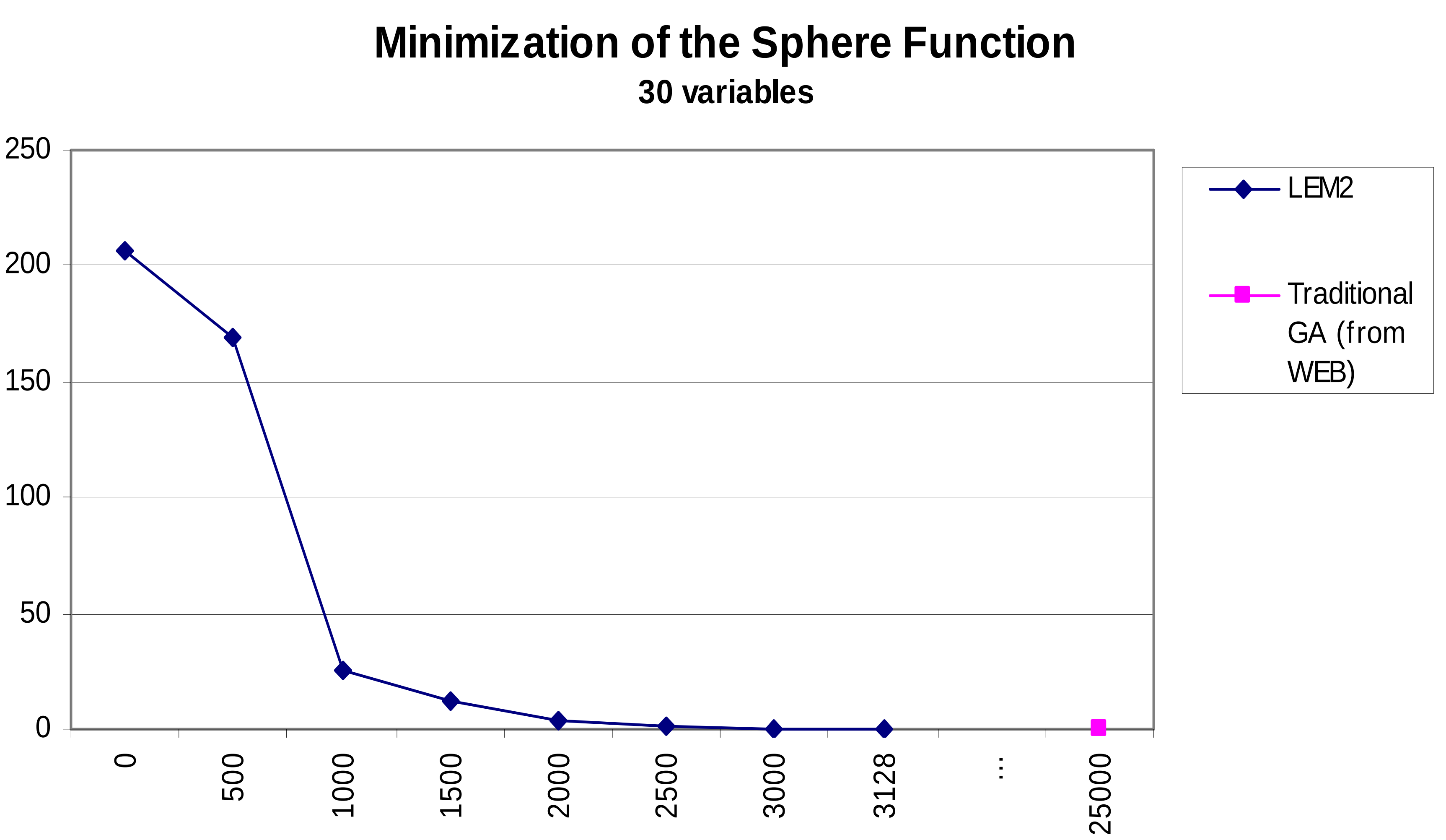
  http://www.maths.adelaide.edu.au/Applied/llazausk/alife/realfopt.html

- The website is maintained by Leo Lazauskas from the Department of Applied Mathematics, University of Adelaide, South Australia

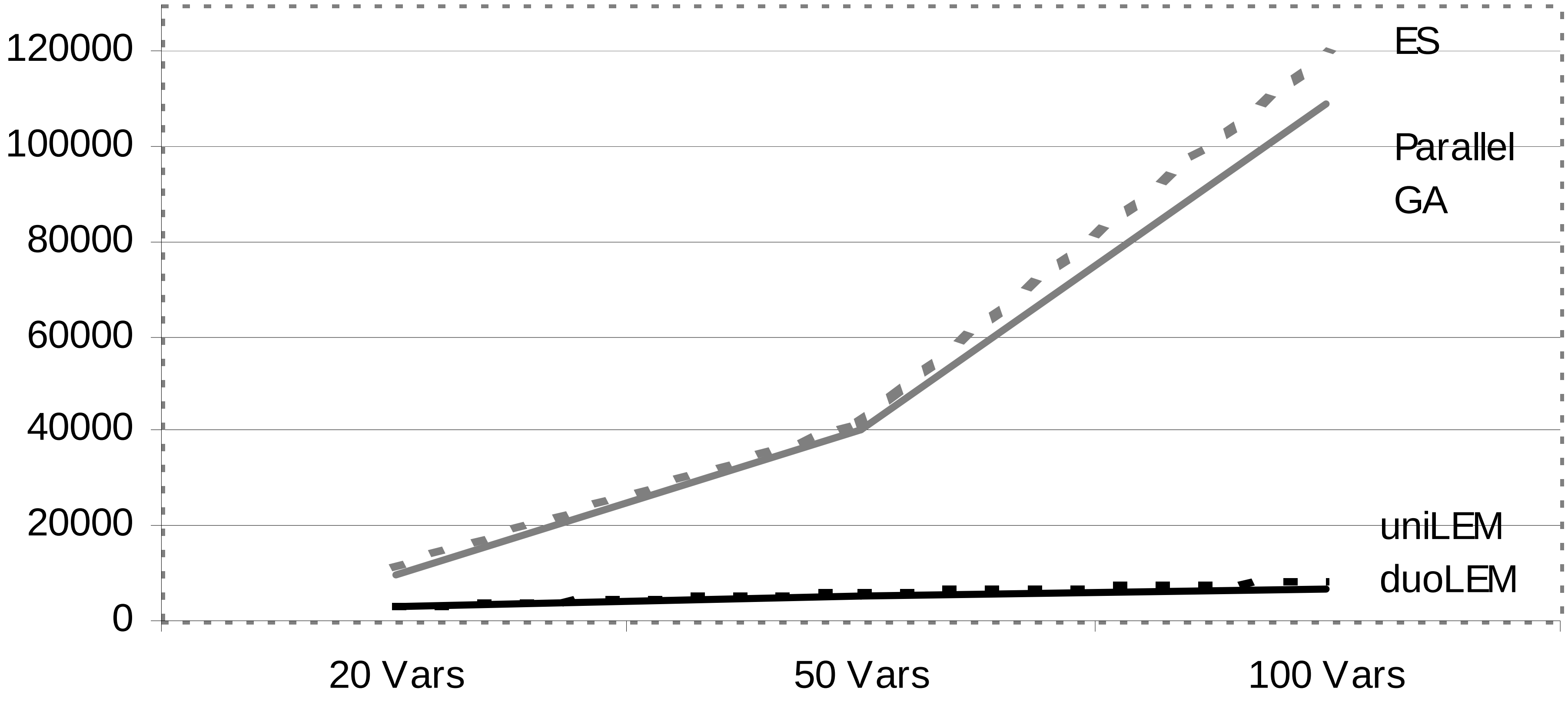
llazausk@maths.adelaide.edu.au

LEM2 vs. GA

Minimizing the Sphere Function of 30 Variables



An Important Result:
Advantage of LEM Grows with the Complexity of Problems



Evolution Speedup vs.Execution Speedup

- In all experiments, LEM2 used fewer evaluations (births), but generating individuals in LEM is more computationally costly than in Darwinian evolutionary algorithms (DEA)
- To evaluate the tradeoff, let's define the following concepts:
  - *Evolution length*, EL, is the total number of births in the evolutionary process
  - *Evolution time*, ET, is the total computation time of evolutionary processes
  - *Evolution speedup of LEM/DEA* – the ratio of evolution lengths of DEA and LEM
  - *Execution speedup of LEM/DEA* – the ratio of evolution times of DEA and LEM
- Evolution time, ET_A of algorithm A, can be approximated by:

$$ET_A = (T_{gA} + T_e)\, EL_A \qquad (1)$$

where T_{gA} is the average time of generating an individual during an evolutionary process executed by the algorithm A, T_e is the average evaluation time of an individual, and EL_A is the evolution length of the algorithm A. It can be assumed that $T_{gd} << T_{gl}$, where d (in T_{gd}) stands for a DEA and l (in T_{gl}), stands for LEM.

Execution Speedup LEM/DEA

- Experiments have shown that $EL_l << EL_d$. The *execution speedup*, ES, of LEM over DEA can be expressed as:

  $$ES = ((T_{gd} + T_e) EL_d)) / ((T_{gl} + T_e) EL_l) \qquad (2)$$

  where T_{gd} and EL_d are the generation time and the evolutionary length of DEA, and T_{gl} and EL_l are the same quantities for LEM.

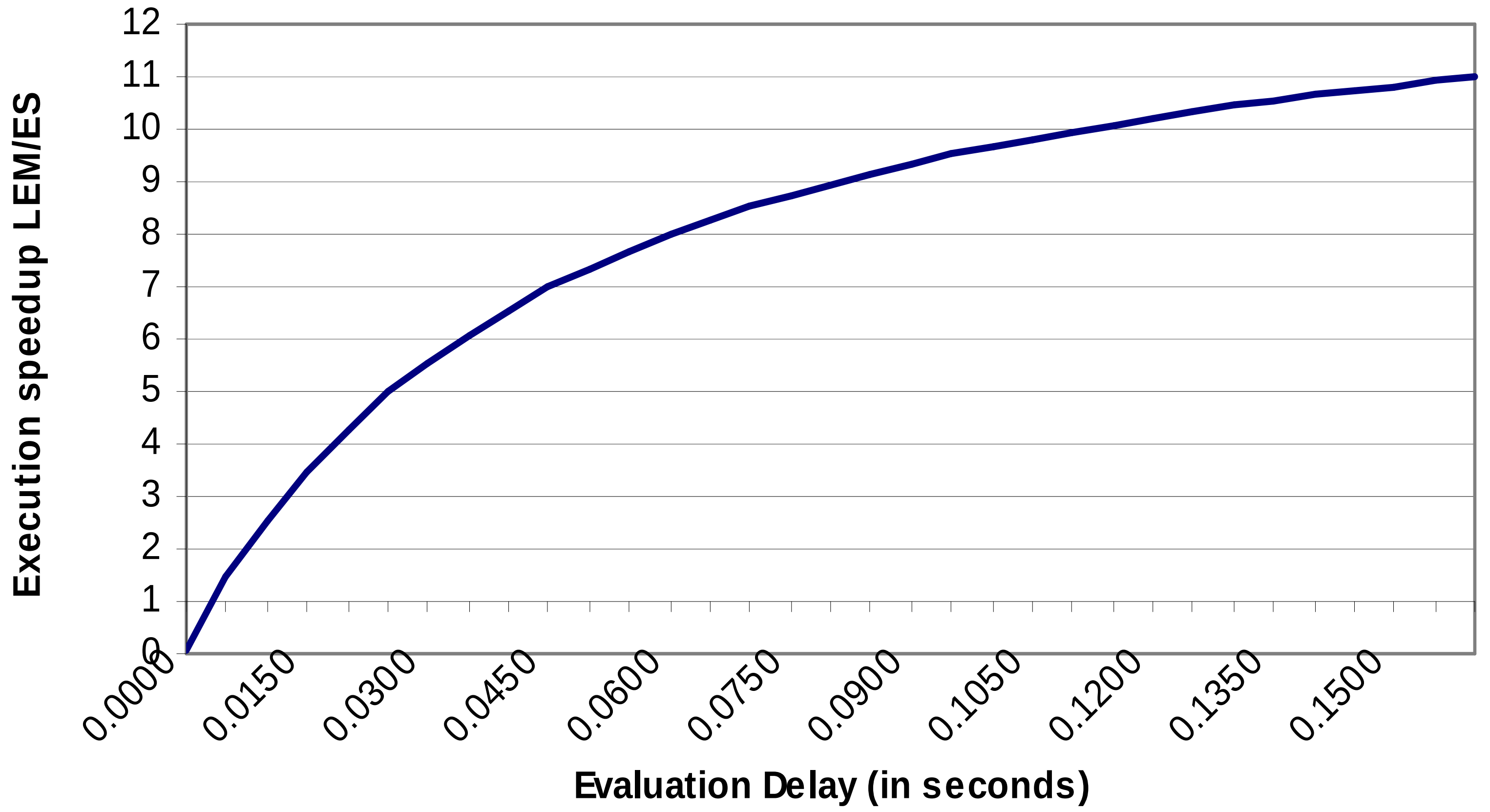
- After making appropriate transformations, the following approximate condition is derived for T_e to achieve the execution speedup of LEM/DEA greater than 1:

  $$T_e > T_{gl}/k - T_{gd} \qquad (3)$$

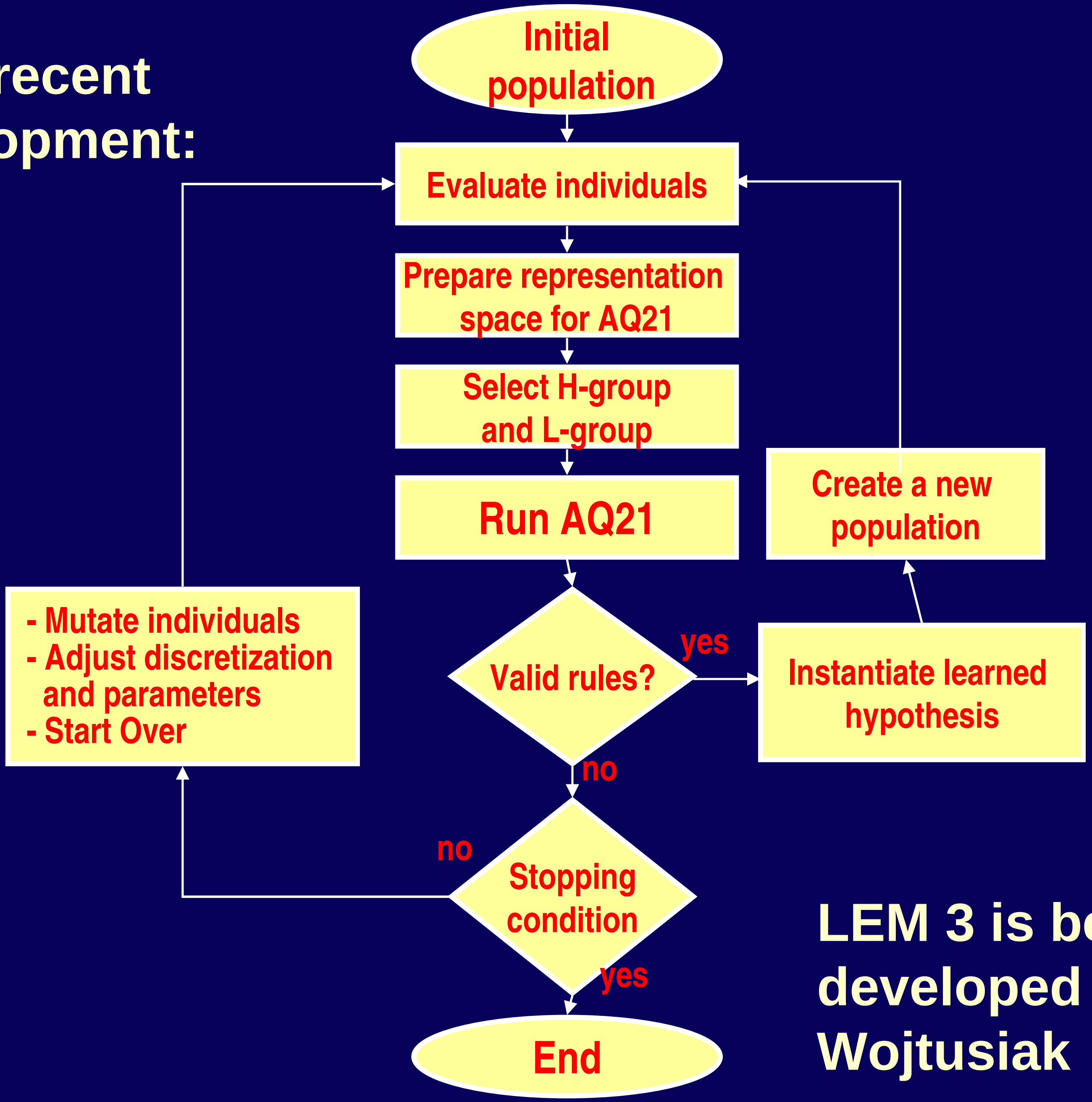
  where k denotes the evolutionary speedup of LEM over DEA.

- According to (3), if T_e is negligible, the execution speedup of LEM/DEA will be greater than 1, if $T_{gd} > T_{gl}/k$, that is, if LEM's generation time divided by the evolutionary speedup is smaller than the DEA generation time. If $T_e >> T_{gd}, T_{gl}$ (the fitness evaluation time is significantly larger than both DEA and LEM generation times), then, according to (2), the execution speedup will converge to the evolutionary speedup: $ES \approx k = EL_d/El_l$

Execution and Evolution Speedup, LEM/ES, in Optimizing Rastrigin Function of 100 Continuous Variables



Most recent development: LEM3



LEM 3 is being developed by Janusz Wojtusiak

For Those Interested

- For publications on the LEM methodology and results on function optimization, see www.mli.gmu.edu*, click on "Papers," and look for papers with Learnable Evolution Model or Non-Darwinian Evolution in the title.*

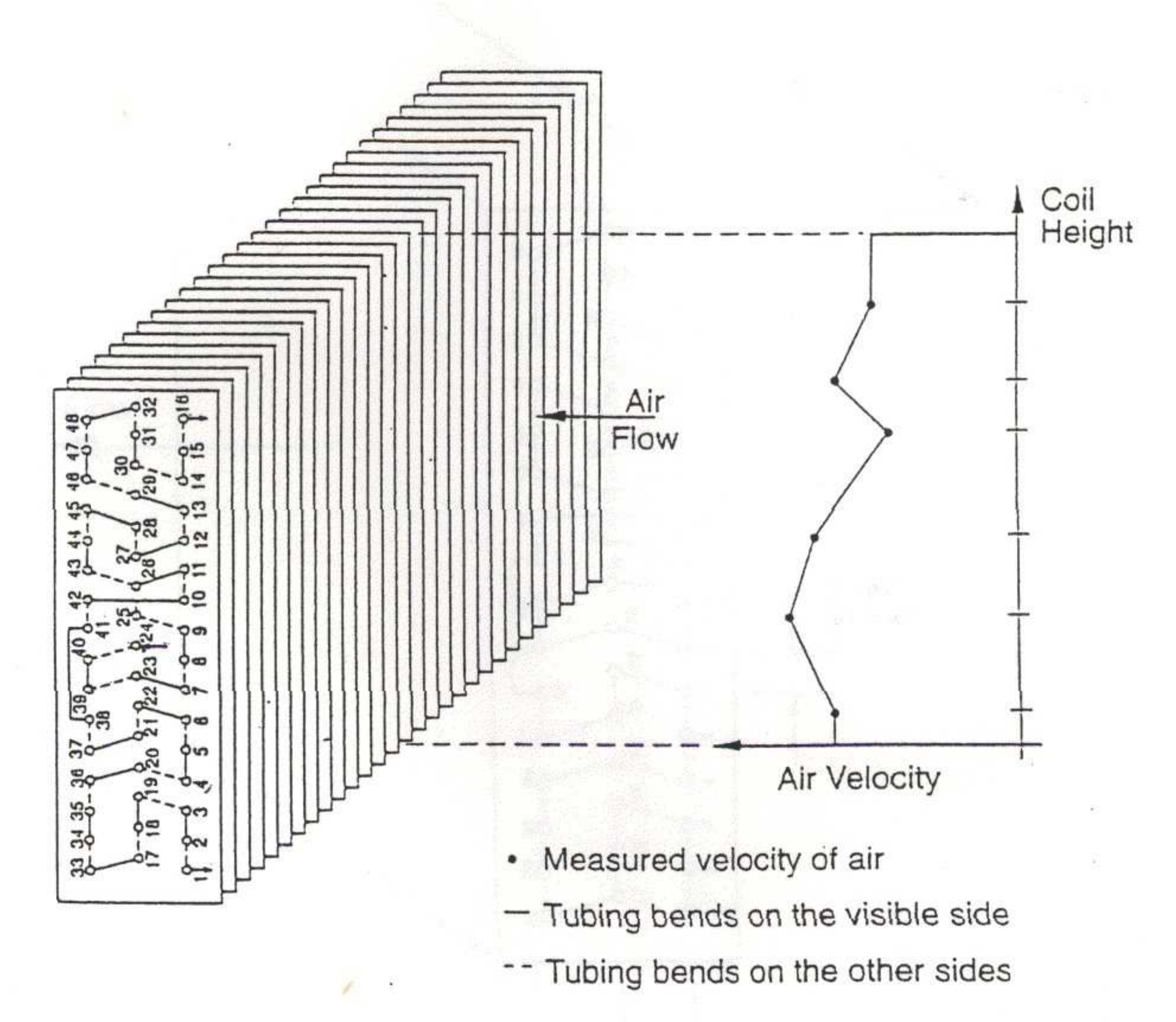
- The U.S. patent on LEM No. 6,518,988 was issued on February 11, 2003.

An Exploratory Application to Heat Exchanger Design

- To test LEM on a real-world problem, we applied it to a complex engineering design problems, specifically, to optimizing heat exchangers
- The problem is to design heat exchangers with maximal capacity under given technical and environmental constraints (such as the size of the exchanger--the number of rows of tubes and the number of tubes per row in the exchanger, the refrigerant used, outside air temperature and humidity, the flow of air through the heat exchanger, etc.)
- This research was conducted in collaboration with the National Institute of Standards and Technology (NIST)

Heat Exchanger

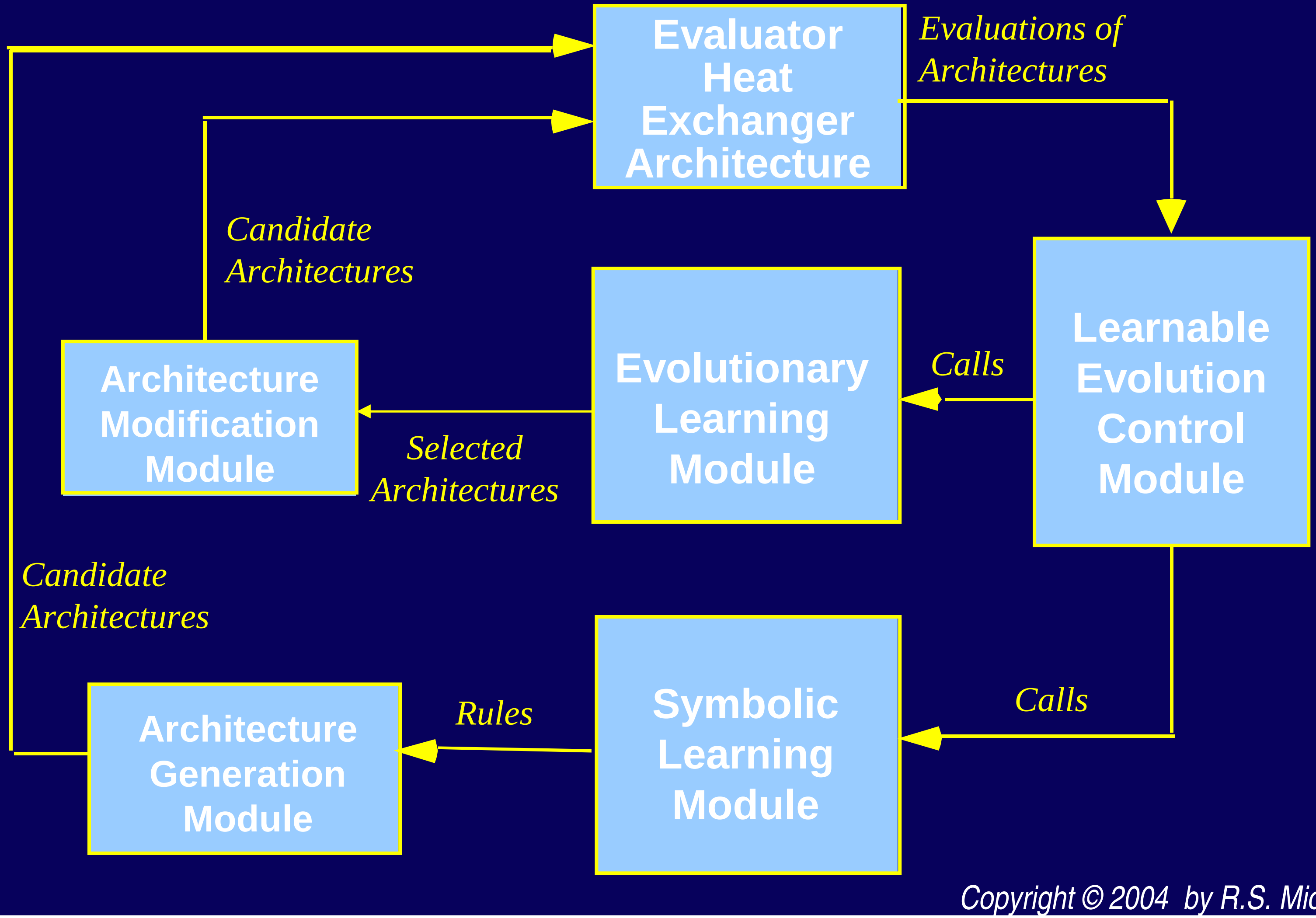
The way the refrigerant flows through the evaporator's tubes strongly affects the unit's cooling capacity


Coil Height
Air Flow
Air Velocity
• Measured velocity of air
— Tubing bends on the visible side
-- Tubing bends on the other sides

Problem Description

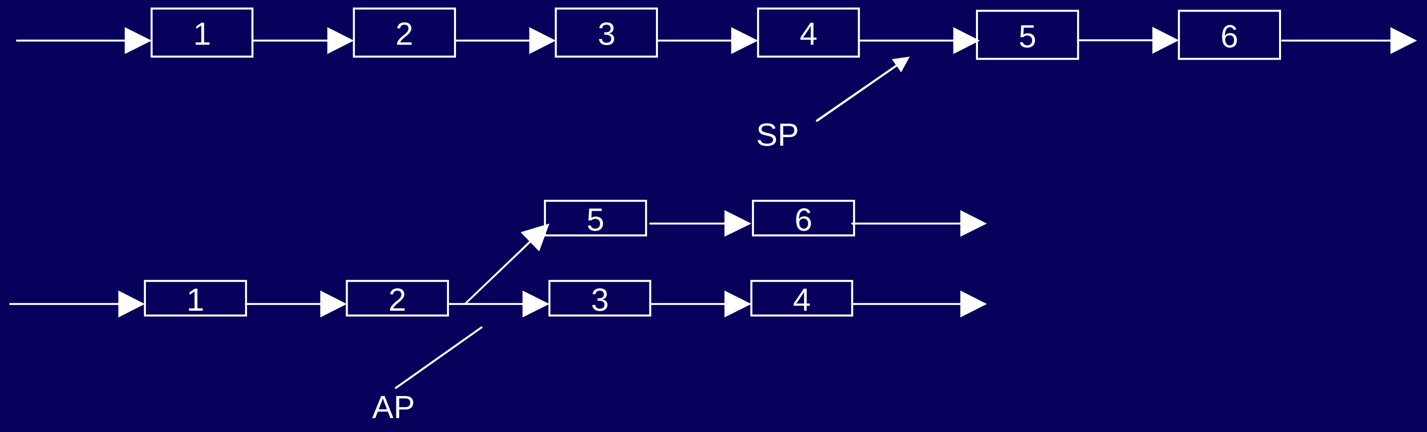
- Changing the order of the tubes affects the unit's cooling efficiency, as it will affect the temperature of the air passing over the evaporator and of the refrigerant drawing out the air's heat
- The search space for such an optimization task can contain well over 10^{50} designs, many of which are infeasible
- Designs are evaluated by a numerical simulator
- Changes in the environmental conditions may drastically affect the efficiency of a given evaporator configuration

ISHED1 Architecture


Evaluator Heat Exchanger Architecture
Evaluations of Architectures
Candidate Architectures
Learnable Evolution Control Module
Calls
Evolutionary Learning Module
Architecture Modification Module
Selected Architectures
Candidate Architectures
Symbolic Learning Module
Calls
Rules
Architecture Generation Module

Approach

- **Eight structure modification operators were designed in collaboration with a domain expert.**
- **Example: SPLIT(SP, AP) , where SP is split point, and AP is the application point**



- A machine learning program creates hypotheses for distinguishing better-performing designs from worse ones, and then uses these hypotheses to suggest new designs.

An Example of ISHED1's Run

Heat Exchanger Size: 16 x 3

Population Size: 15 Generations: 40

Operator Persistence: 5 (# of unsuccessful trials of a SM operator)

Mode Persistence: Dar-probe=2 and Learn=probe=1

Generation 1: Initial Population

Structure #1.3:

17 1 20 3 4 22 6 24 8 26 10 28 27 15 16 32 33 2 18 19 5 38 7 40 9 42 11 4 13 45 30 48 34 35 36 l 21 37 23 39 25 41 27 43 46 29 31 47: **Capacity=5.2093**

(15 structures total)

Selected Members: 1.2, 1.3, 1.6, 1.7, 1.9, 1.20, 1.26, 1.30, 1.33, 1.35, 1.36

Operators: NS(23, 39), SWAP(8), SWAP(28), SWAP(19), SWAP(1), SWAP(27), SWAP(40), SWAP(43), SWAP(15), SWAP(25), SWAP(7), SWAP(36), SWAP(29), SWAP(25), SWAP(1)

Next generations

Generation 2: Conventional Evolution mode

Structure #1.13:

17 1 20 3 4 22 6 24 8 26 10 28 27 15 16 32 33 2 18 19 5 38 7 40 9 42 11 4 13 45 30 48 34 35 36 I 21 37 23 39 25 41 27 43 46 29 31 47: **Capacity=5.2093**

(15 structures total)

Selected Members: 6, 15, 11, 3, 13, 1, 10, 6, 12, 10, 5, 4, 13, 1, 3

……

Generation 5: Machine Learning mode

A learned rule:

[x4.x5.x6.x7.x8.x9.x11.x12.x13.x14.x15.x17.x18.x19.x20.x21.x22.x23.x24.x25.x26.x27.x28.x29.x30.x31.x32.x33.x34.x35.x36.x37.x38.x39.x40.x41.x42.x43.x44.x45.x46.x47.x48=regular] & [x10=outlet] & [x16=inlet]
(t:7, u:7,q:1)

An example of generated structure:

Structure #5.1: 17 1 2 3 4 5 6 7 8 9 12 29 45 30 31 I 18 33 20 36 22 38 24 40 26 42 11 27 13 15 47 48 34 35 19 37 21 39 23 41 25 43 44 28 46 14 32 16: **Capacity=5.5377**

Final Step

Generation 21:

Structure #21.15 2 18 4 1 6 3 5 7 8 9 12 13 45 15 31 I 33 17 35 36 22 39 24 40 42 25 11 44 30 46 32 47 34 19 20 37 21 23 38 41 26 43 28 27 29 14 48 16: Capacity=5.5387

Finally, ISHED1 achieves:

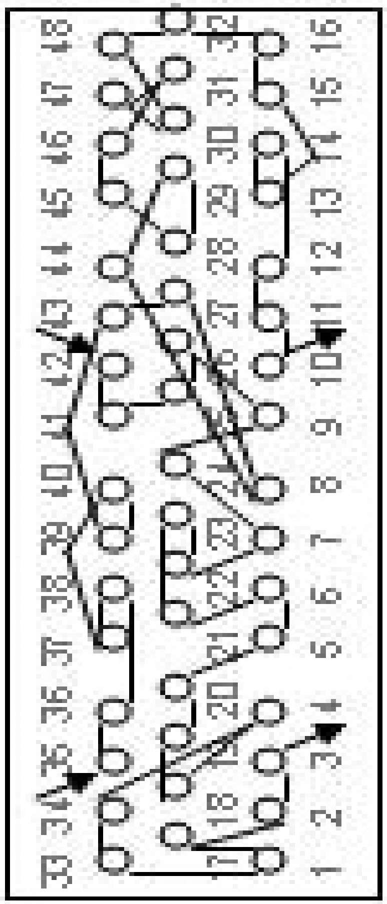
Generation 40: *:*

Structure #40.15: 33 17 2 41 4 5 6 9 7 8 12 29 46 45 47 I 1 34 20 36 22 38 24 3 42 43 44 27 13 15 32 16 18 11 19 37 21 32 23 25 40 26 28 35 30 14 48 31: Capacity=6.3686

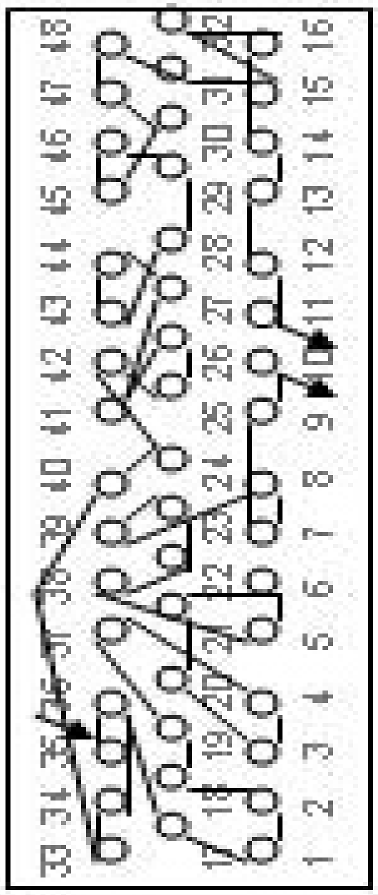
(The highest capacity in this run:

~13% improvement)

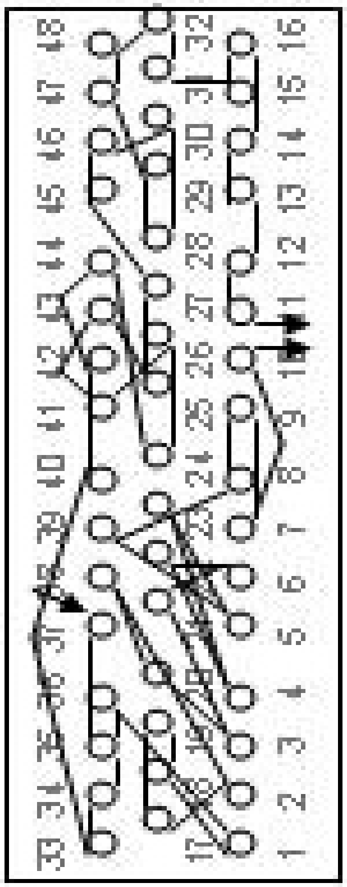
Longer Experiments Produced Better Designs



Capacity = 6.5973



Capacity = 6.6219



Capacity = 6.6555

Summary of Results from ISHED1

- ISHED1 was tested with different sizes and different airflow patterns in heat exchangers
- In the case of uniform airflow, ISHED1 designs were comparable to best expert designs
- In non-uniform airflow, ISHED1's designs were evaluated by the collaborating expert from NIST as better than commercial designs
- NIST gave these results a high evaluation is now promoting ISHED1 in industry.
- Forthcoming paper on this work:

Piotr A. Domanski, David Yashar, Kenneth A Kaufman, Ryszard S Michalski, "An Optimized Design of Finned-Tube Evaporators Using the Learnable Evolution Model", will appear in the *International Journal of Heating, Ventilating, Air-Conditioning and Refrigerating Research, 2004*.

An Evolution of Evolutionary Computation: An Analogy to the Development of AI

Phase 1: TABULA RASA

The development of universal, domain independent methods that were closely related to the principles Darwinian evolution

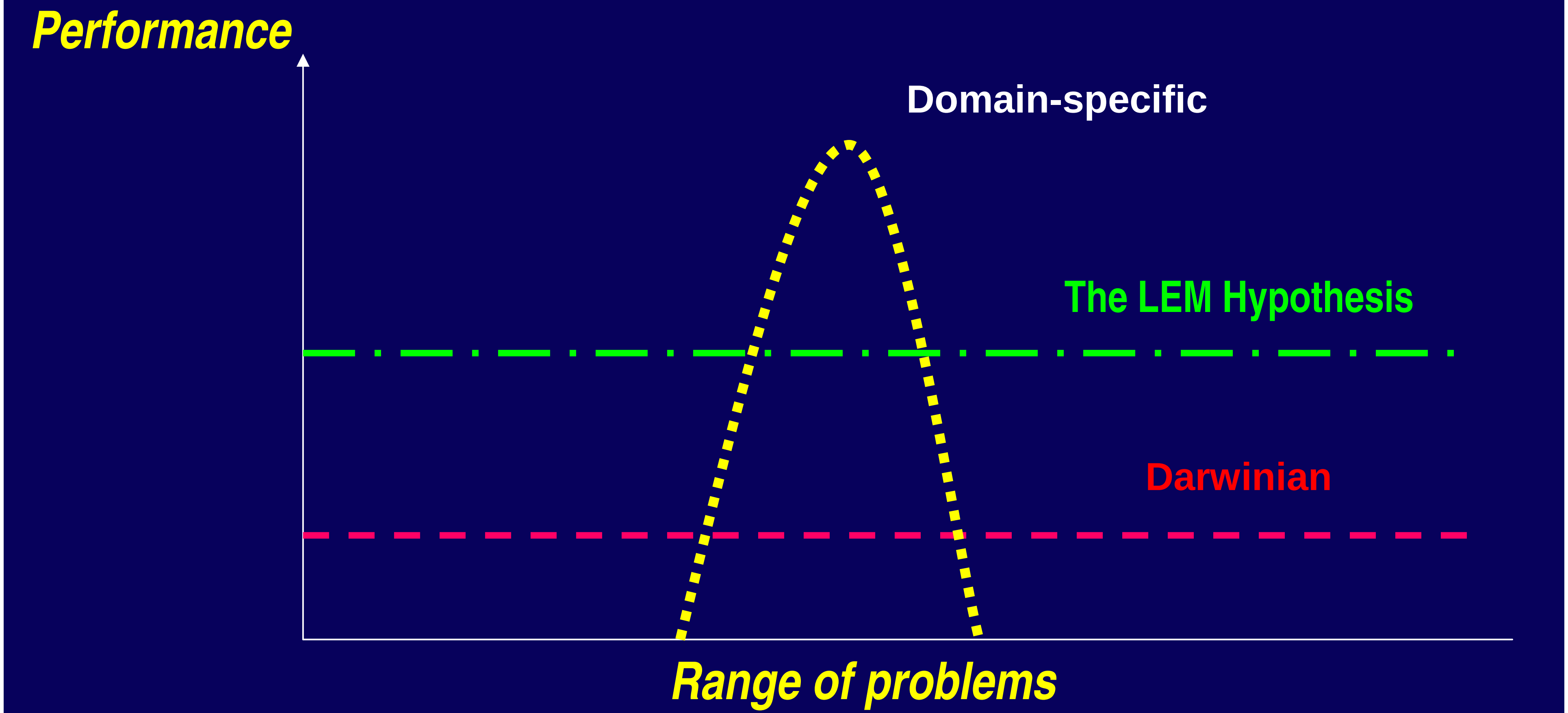
Phase 2: NEED FOR DOMAIN KNOWLEDGE

Introduction of domain knowledge in the form of problem-oriented representations and operators that modify these representations

Phase 3: EVOLUTION SHOULD LEARN

Introduction of learning methods that allow the evolutionary process to learn from past successes and failures

Comparing Darwinian and LEM Approaches with Domain-specific Problem Solutions



Both, the Darwinian-type and LEM evolution can be viewed as general-purpose search methods

Summary

- LEM attempts to model intellectual evolution rather then biological evolution
- It generates theories about the “right” direction of evolution, and uses these them to guide evolution
- The effect of Machine Learning mode is frequently seen by quantum leaps of the fitness function
- In every experiment conducted so far, LEM has outperformed the tested evolutionary computation algorithms in terms of the evolution length (frequently achieving evolution speedups of two or more orders of magnitude).
- The price for these advantages is a higher complexity of hypothesis generation and instantiation operators
- LEM appears be particularly useful in application domains in which fitness evaluation is time-consuming or costly, such as very complex optimization problems, engineering design, fluid dynamics, drug design, evolvable hardware, etc.

Planned Work

- Completion of the new implementation, LEM3
- Development a methodology for handling complex constraints
- Development of constructive induction capabilities in LEM
- Integrating LEM as an operator for design and optimization within the VINLEN system for knowledge mining and decision support .

For more information see: **www.mli.gmu.edu,** or contact:

Ryszard S. Michalski
michalski@mli.gmu.edu
www.mli.gmu.edu/michalski

LEM2 and iAQ can be downloaded from

www.mli.gmu, ***select*** **software**

LEM3 will be downloadable when completed.

For assistance on LEM programs, contact

Janusz Wojtusiak: wojtusiak@mli.gmu.edu

For assistance on iAQ, contact

Jarek Pietrzykowski: jpietrzykowski@mli.gmu.edu

Acknowledgement

This research has been supported in part by the National Science Foundation under Grants No. IIS-9906858 and IIS-0097476.

iAQ

A Demo of Natural Induction

To download:

http://www.mli.gmu.edu/mlisoftware.html