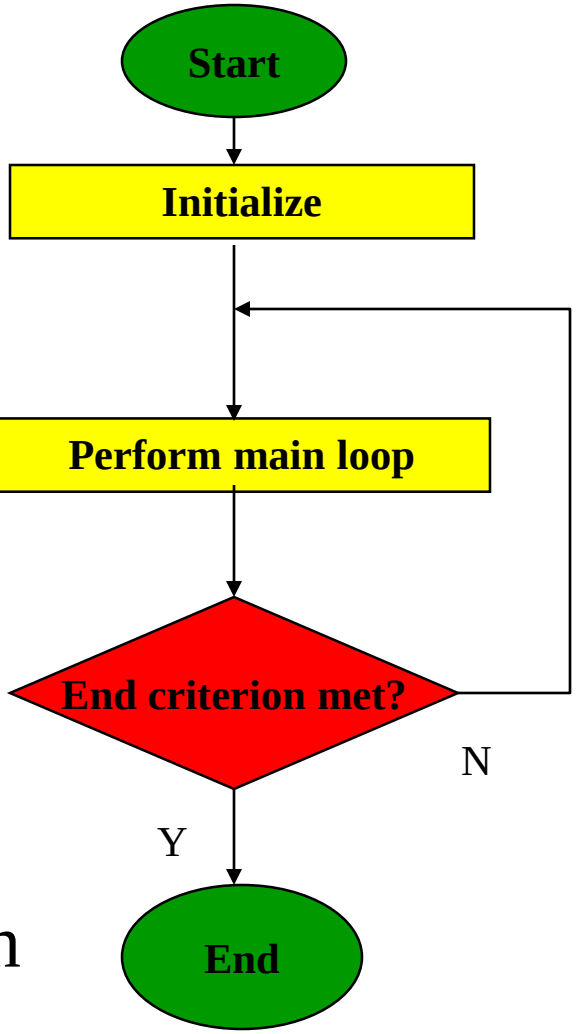
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Distributed Intelligent Systems – W10

An Introduction to Particle Swarm Optimization and its Application to Benchmark Functions and Single-Robot Systems

Outline

- Machine-learning-based methods
  - Rationale for embedded systems
  - Terminology
- Particle Swarm Optimization (PSO)
- Comparison between PSO and GA
- Application to single-robot systems
  - Examples in control design and optimization (obstacle avoidance, homing)
  - Examples in system design and optimization (locomotion)
- An introduction to expensive optimization problems and noise resistance



Rationale and Classification

Why Machine-Learning?

- Complementary to a model-based/engineering approaches: when low-level details matter (optimization) and/or good models do not exist (design)!
- When the design/optimization space is too large (infinite)/too computationally expensive (e.g., NP-hard) to be systematically searched
- Automatic design and optimization techniques
- Role of the engineer refocused to performance specification, problem encoding, and customization of algorithmic parameters (and perhaps operators)

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Disal

Why Machine-Learning?

- There are design and optimization techniques robust to noise, nonlinearities, discontinuities
- Possible search spaces: parameters (continuous and discrete spaces), rules, SW or HW structures/architectures
- Individual real-time adaptation to new or unpredictable environmental/system conditions

ML Techniques: Classification Axis 1

- Supervised learning: off-line, a teacher is available
- Unsupervised learning: off-line, teacher not available
- Reinforcement-based (or evaluative) learning: on-line, no pre-established training and evaluation data sets

Supervised Learning

- Off-line
- Training and test data are separated, a teacher is available
- Typical scenario: a set of input-output examples is provided to the system, performance error given by difference between system output and true/teacher-defined output, error fed to the system using optimization algorithm so that performance is increased over trial
- The generality of the system after training is tested on examples not previously presented to the system (i.e. a “test set” exclusive from the “training set”)

Unsupervised Learning

- Off-line
- No teacher available, no distinction between training and test data sets
- Goal: structure extraction from the data set
- Examples: data clustering, Principal Component Analysis (PCA) and Independent Component analysis (ICA)

Reinforcement-based (or Evaluative) Learning

- On-line
- No pre-established training or test data sets
- The system judges its performance according to a given metric (e.g., fitness function, objective function, performance, reinforcement) to be optimized
- The metric does not refer to any specific input-to-output mapping
- The system tries out possible design solutions, does mistakes, and tries to learn from its mistakes

ML Techniques: Classification Axis 2

- In simulation: reproduces the real scenario in simulation and applies there machine-learning techniques; the learned solutions are then downloaded onto real hardware when certain criteria are met

- Hybrid: most of the time in simulation (e.g., 90%), last period (e.g., 10%) of the learning process on real hardware

- Hardware-in-the-loop: from the beginning on real hardware (no simulation). Depending on the algorithm more or less rapid

ML Techniques: Classification Axis 3

ML algorithms require often fairly important computational resources (in particular for population-based algorithms), therefore a further classification is:

- On-board: machine-learning algorithm run on the system to be learned (no external unit)
- Off-board: the machine-learning algorithm runs off-board and the system to be learned just serves as embodied implementation of a candidate solution

ML Techniques: Classification Axis 4

– Population-based (“multi-agent”): a population of candidate solutions is maintained by the algorithm, for instance Genetic Algorithms (individuals), Particle Swarm Optimization (particles), Particle Filter (particles), Ant Colony Optimization (ants/tours); agents can represent directly the candidate solution in the search space (e.g., GA, PSO, PF) or might be the entity that through their behavior generate the candidate solution (e.g., ACO)

– Hill-climbing (“single-agent”): the machine-learning algorithm work on a single candidate solution and try to improve on it; for instance, (stochastic) gradient-based methods, reinforcement learning algorithms

Selected Evaluative Machine-Learning Techniques

- Evolutionary Computation
  - Genetic Algorithms (GA) **population-based, W10**
  - Genetic Programming (GP) **population-based**
  - Evolutionary Strategies (ES) **population-based**
- Swarm Intelligence
  - Ant Colony Optimization (ACO) **population-based, W1-2**
  - Particle Swarm Optimization (PSO) **population-based, W10**
- Learning
  - In-Line Learning (variable thresholds) **hill-climbing, W6**
  - In-Line Adaptive Learning **hill-climbing, W9**
  - Reinforcement Learning (RL) **hill-climbing**
- Particle Filters (PF) **population-based, W4**

Particle Swarm Optimization

Why PSO?

- Comes also from the Swarm Intelligence community such as ACO
- Competitive metaheuristic especially on continuous optimization problems
- Appears to deliver competitive results with population sizes smaller than those of other metaheuristic methods
- Well suited to distributed implementation thanks to the native concept of neighborhood
- Early formal results on convergence

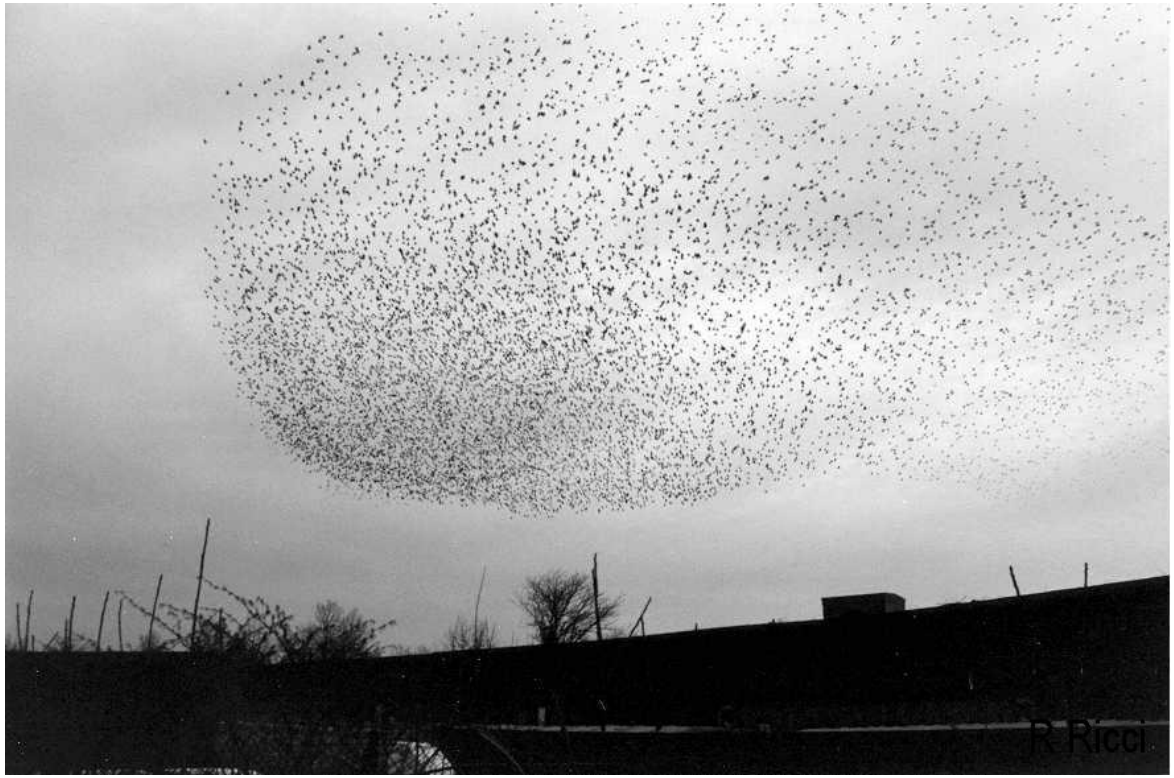
Inspiration and Principles

Principles:

- Imitate
- Evaluate
- Compare

Inspiration:

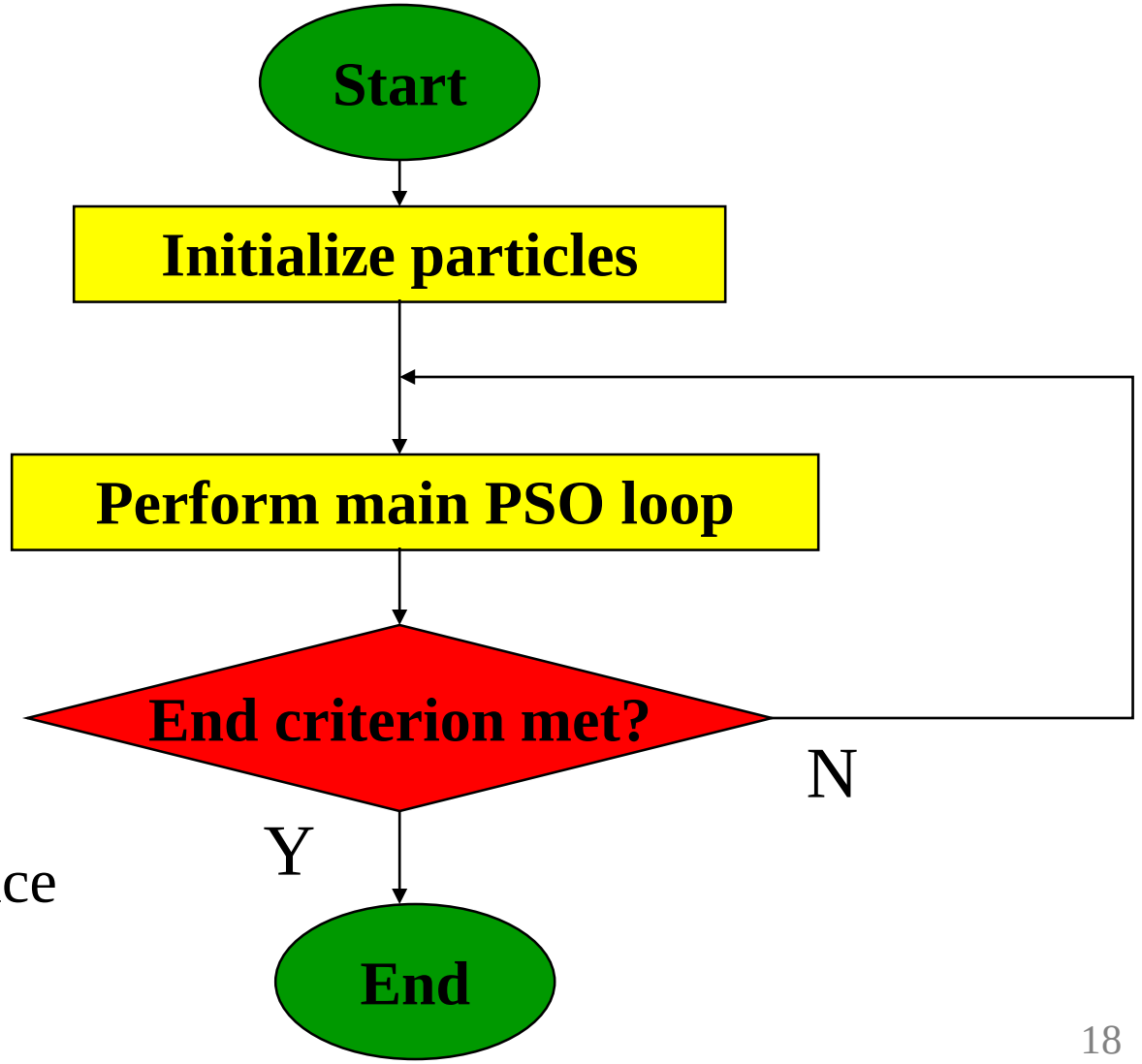
- Bird flock searching for food (see also Week 5)
- Socio-psychological model of social influence and social learning



PSO: Terminology

- Swarm: pool of candidate solutions tested in one time step, consists of m particles (typical # of particles: 20 to 50).
- Particle: represents a candidate solution; it is characterized by a velocity vector ***v*** and a position vector ***x*** in the hyperspace of dimension D.
- Neighborhood: set of particles with which a given particle share performance information
- Iteration: at the end of an iteration, a new pool of candidate solutions after the metaheuristic operators have been applied is available (typical # of iterations: 50, 100, 1000).
- Fitness function: measurement of efficacy of a given candidate solution during the evaluation span.
- Evaluation span: evaluation period of each candidate solution during a single time step (algorithm iteration); the evaluation span might take more or less time depending on the experimental scenario.
- Life span: number of iterations a candidate solution is present in the pool.
- Swarm manager: update velocities and position for each particle according to the main PSO loop.

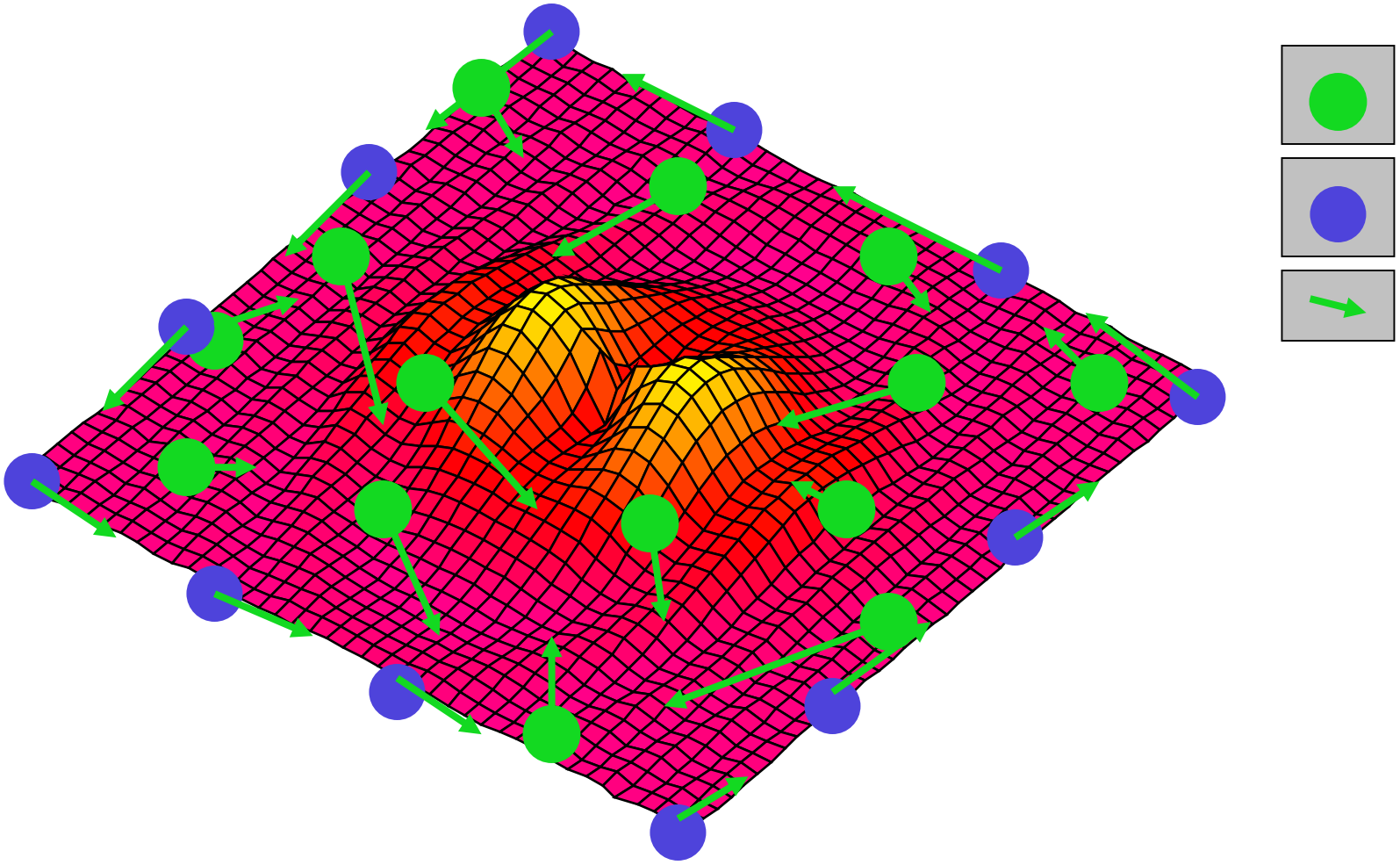
Algorithm Flowchart



Ex. of end criteria:

- # of time steps
- best solution performance
- …

Initialization: Positions and Velocities



The Main PSO Loop – Parameters and Variables

- Functions
  - rand ()= uniformly distributed random number in [0,1]
- Parameters
  - w: velocity inertia (positive scalar)
  - c_p: personal best coefficient/weight (positive scalar)
  - c_n: neighborhood best coefficient/weight (positive scalar)
- Variables
  - $x_{ij}(t)$: position of particle i in the j-th dimension at iteration t (j = [1,D])
  - $v_{ij}(t)$: velocity of particle i in the j-th dimension at iteration t
  - $x^*_{ij}(t)$: position of particle i in the j-th dimension with maximal fitness up to iteration t (personal best)
  - $x^*_{i'j}(t)$: position of particle i' in the j-th dimension having achieved the maximal fitness up to iteration t in the neighborhood of particle i (neighborhood best)

The Main PSO Loop
(Eberhart, Kennedy, and Shi, 1995, 1998)

At each time step t

for each particle i

for each component j

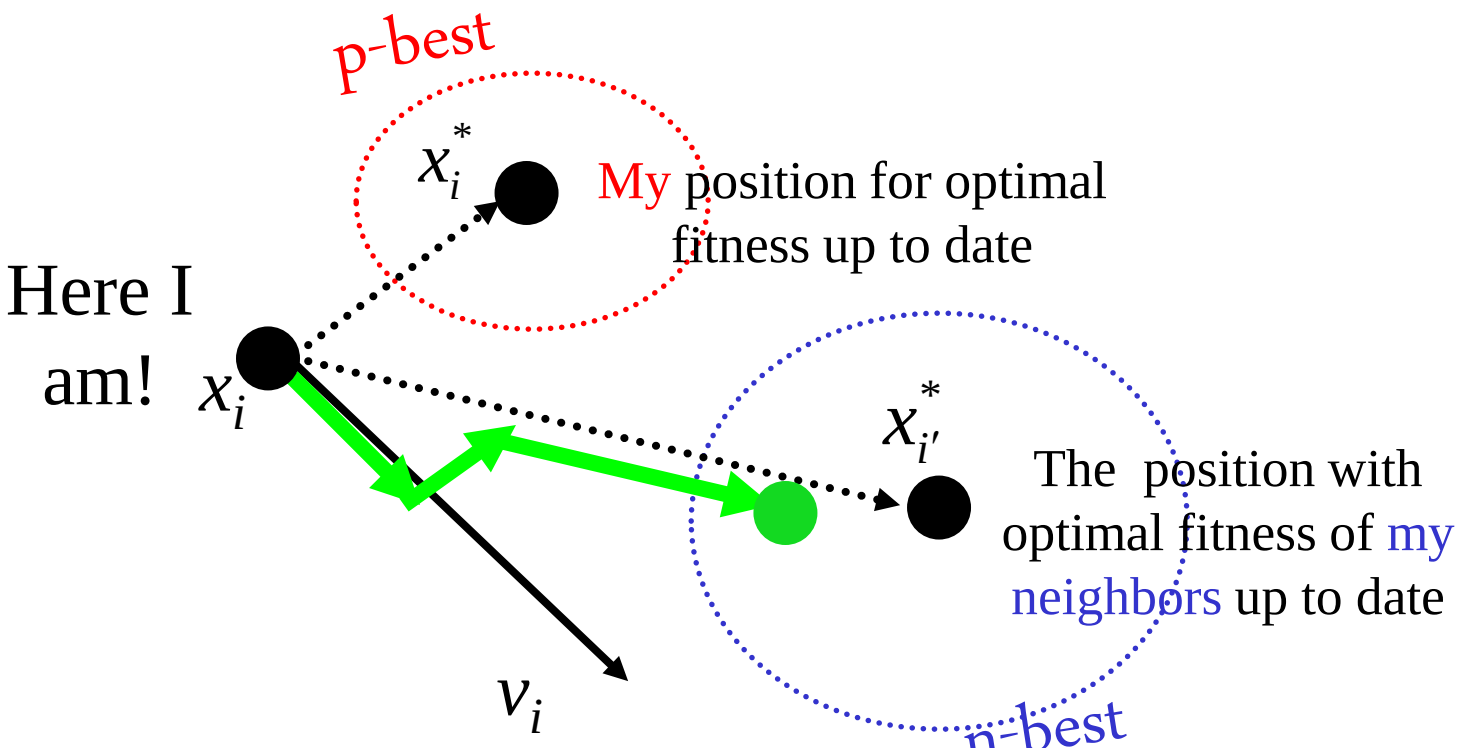
update the velocity

$$v_{ij}(t+1) = wv_{ij}(t) + c_p rand()(x^*_{ij} - x_{ij}) + c_n rand()(x^*_{i'j} - x_{ij})$$

then move

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1)$$

The main PSO Loop - Vector Visualization



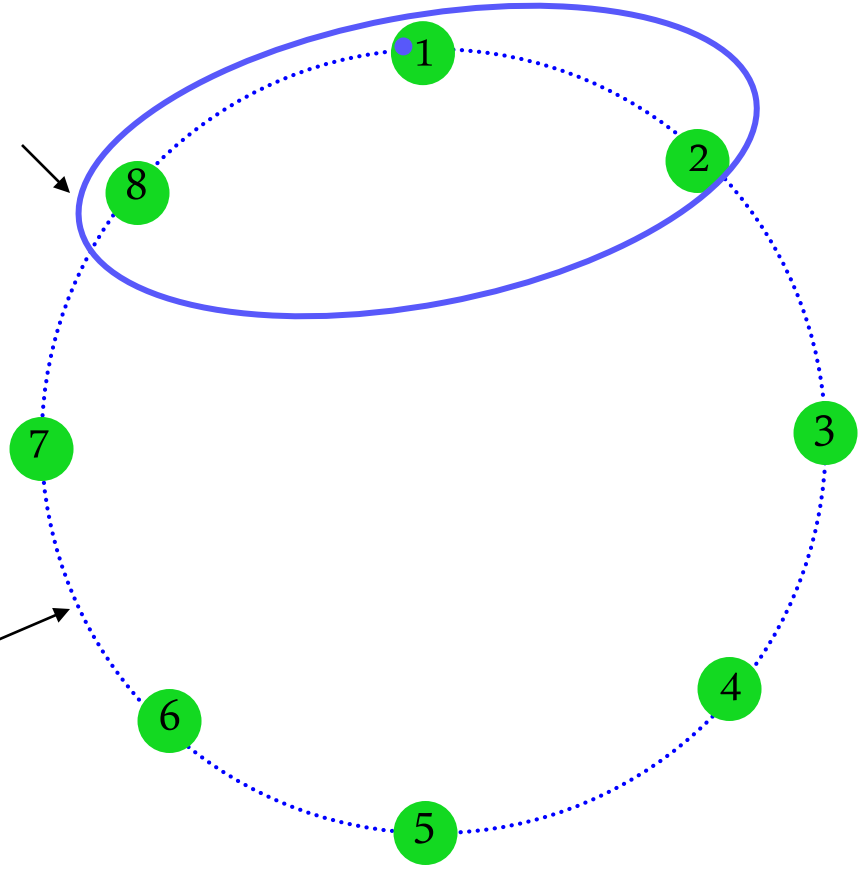
Neighborhood Types

- Size:
  - Neighborhood index considers also the particle itself in the counting
  - Local: only k neighbors considered over m particles in the population ($1 < k < m$); $k=1$ means no information from other particles used in velocity update
  - Global: m neighbors
- Topology:
  - Indexed
  - Geographical
  - Social
  - Random
  - …

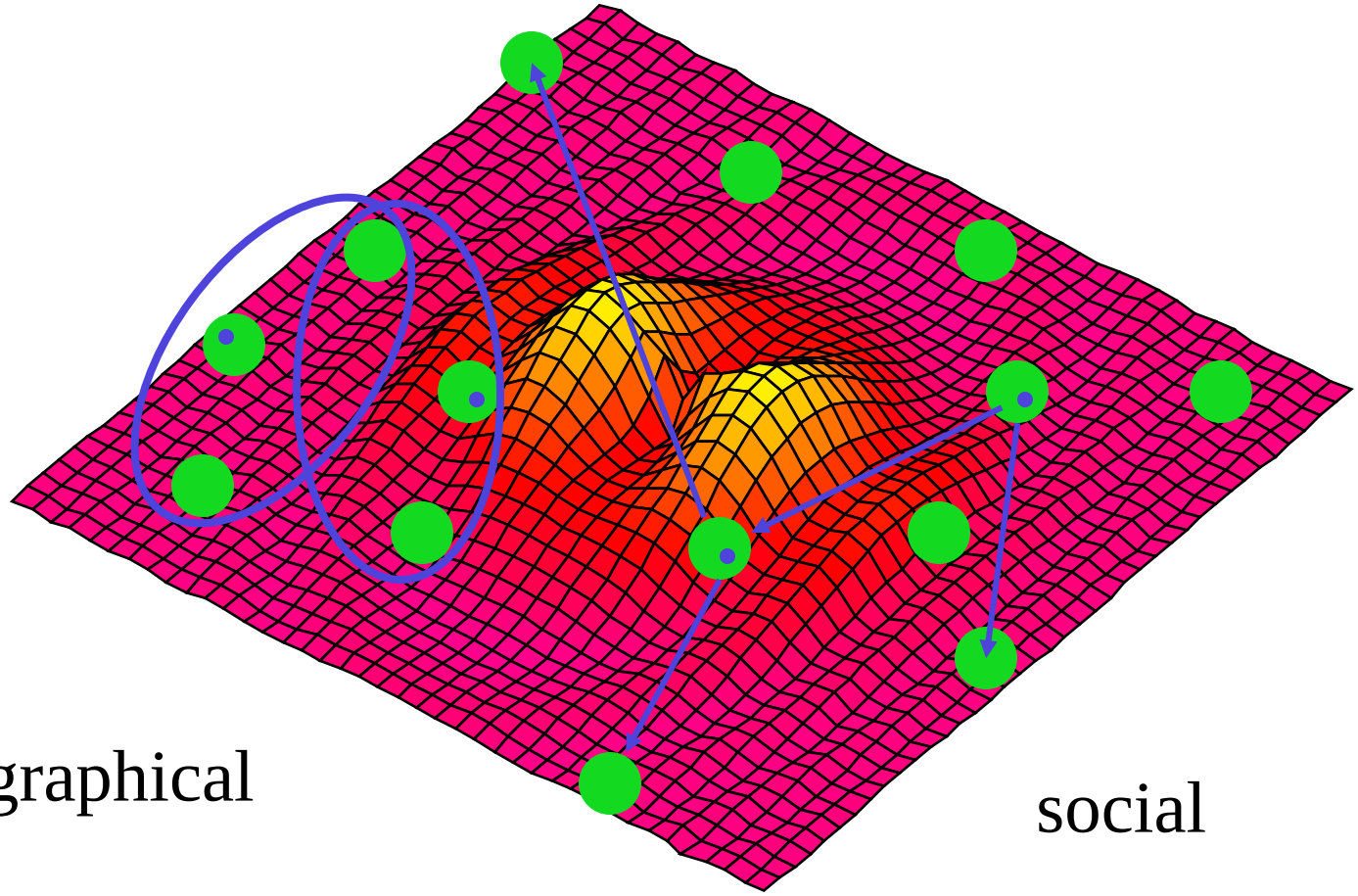
Neighborhood Example: Indexed and Circular (lbest)

Particle 1's 3-neighborhood

Virtual circle



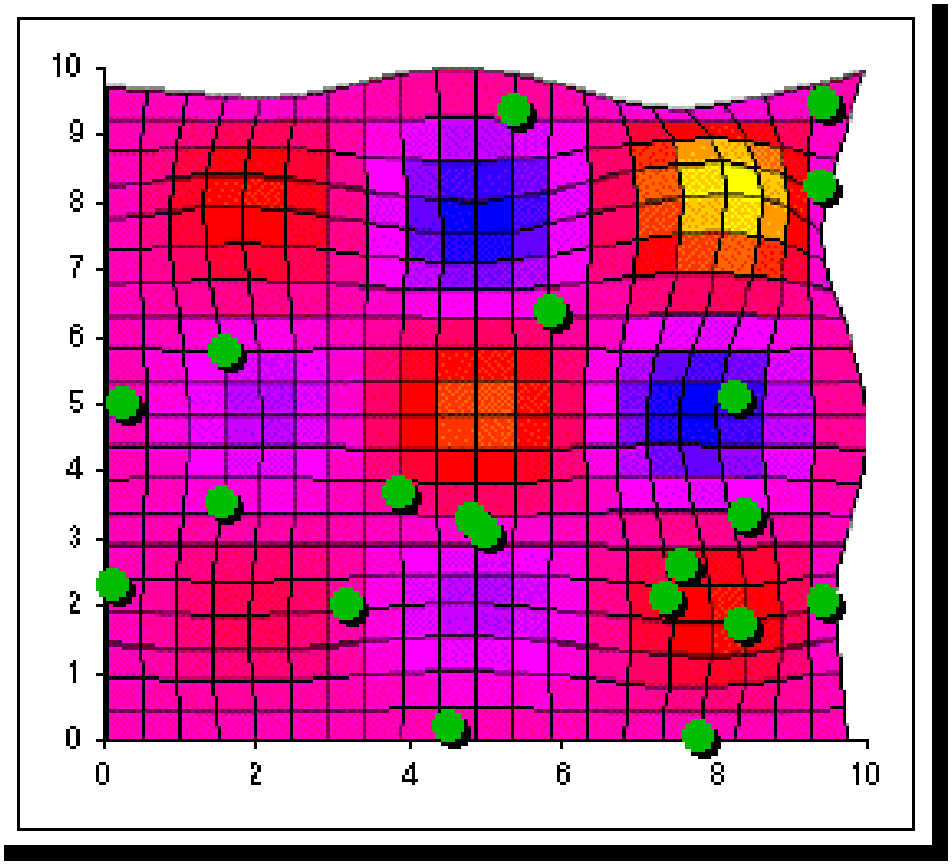
Neighborhood Examples: Geographical vs. Social



geographical

social

A Simple PSO Animation



Genetic Algorithms

Why GA?

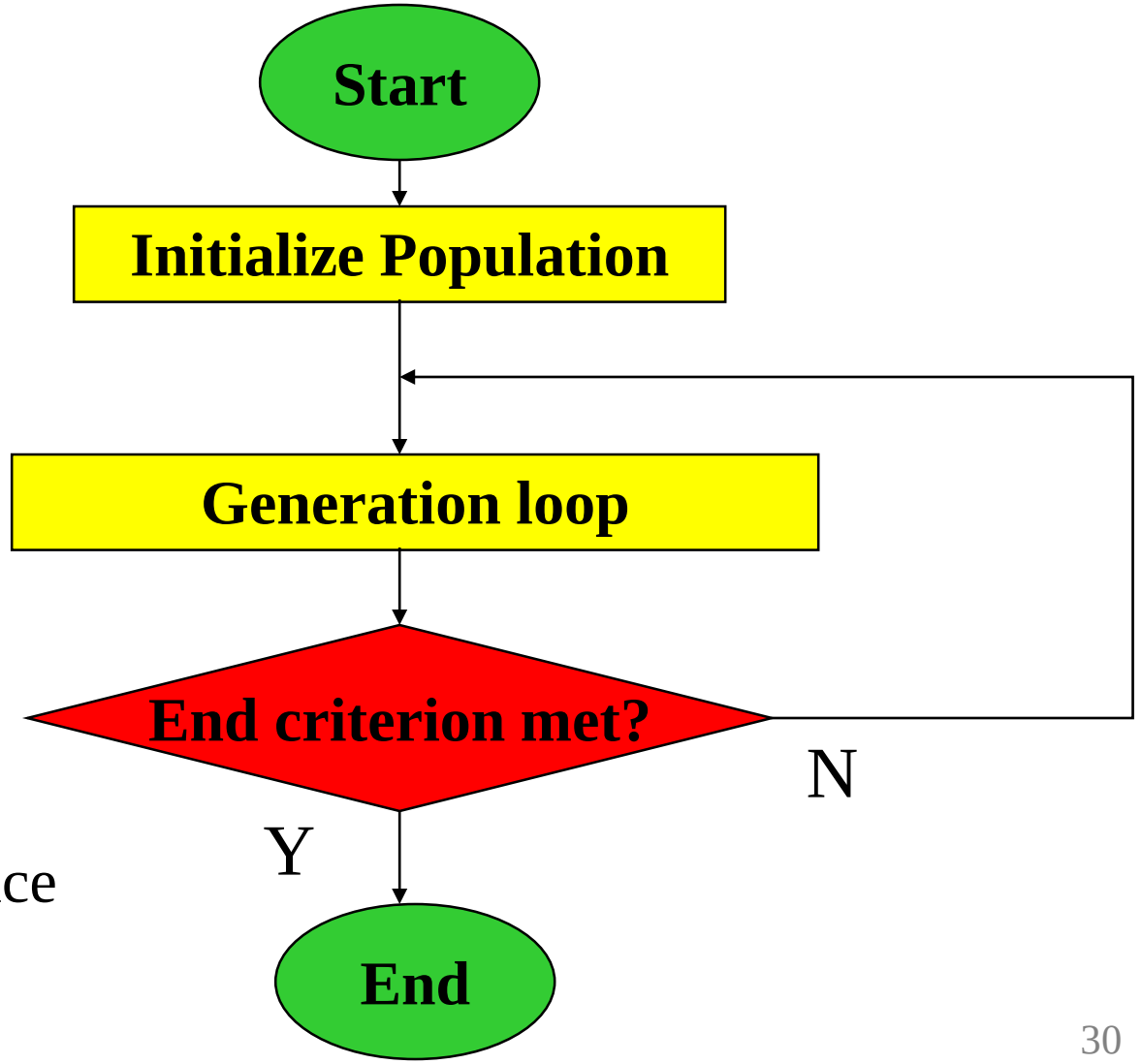
- Reference as often compared with PSO
- Older and widely spread metaheuristic technique, pioneering work applied to robotics used GA
- Originally designed for discrete optimization problems
- In this course: highly summarized description

Note: PSO is NOT an evolutionary algorithm!
See [Engelbrecht, IEEE SIS 2006]

Genetic Algorithms Inspiration

- In natural evolution, organisms adapt to their environments – better able to survive over time
- Aspects of evolution:
  - Survival of the fittest
  - Genetic combination in reproduction
  - Mutation
- Genetic Algorithms use evolutionary techniques to achieve parameter optimization and solution design

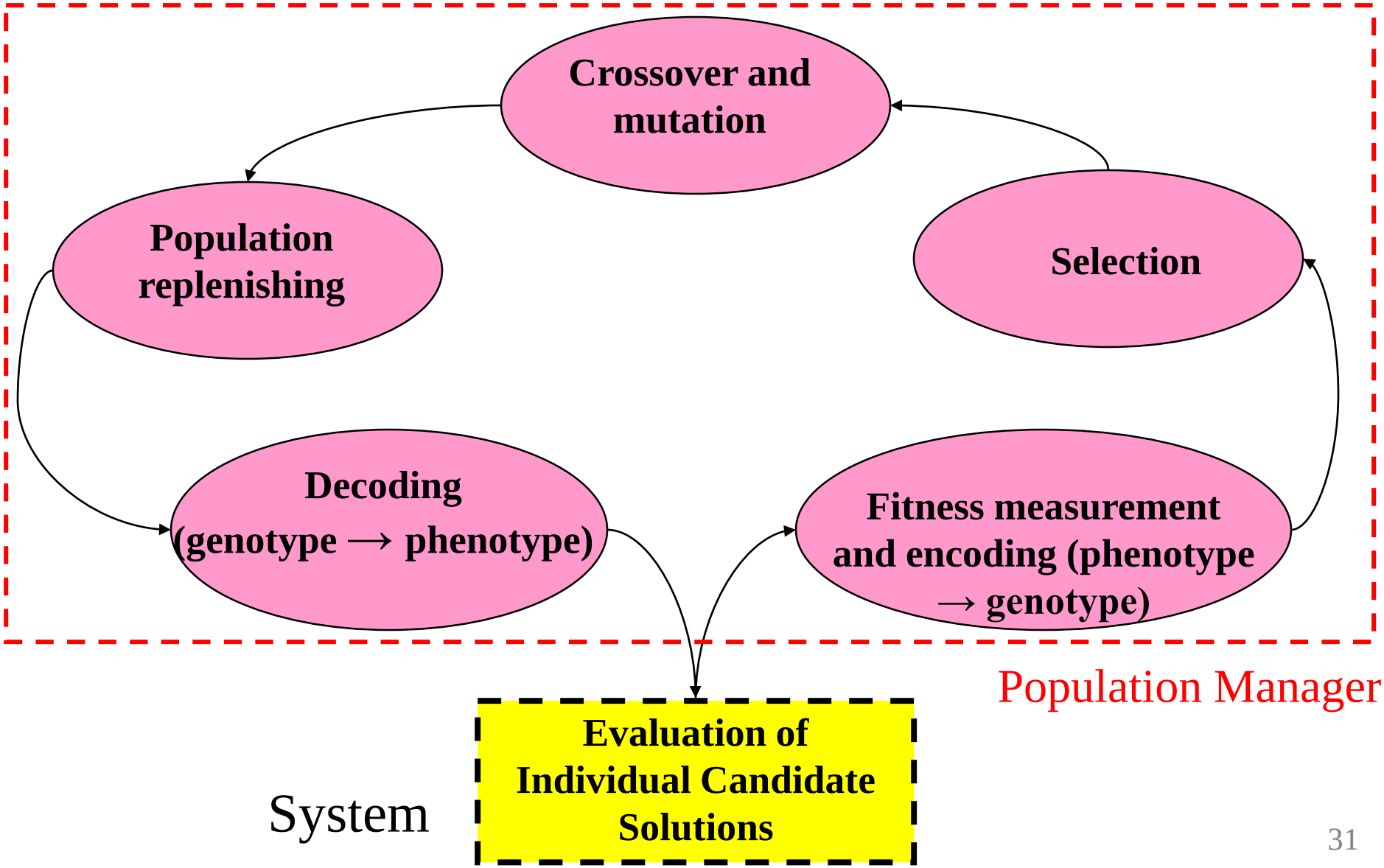
Algorithm Flowchart



Ex. of end criteria:

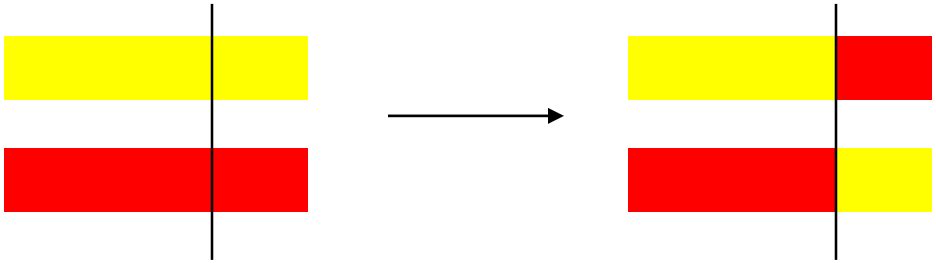
- # of generations
- best solution performance
- …

Generation Loop

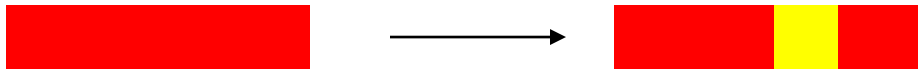


GA: Basic Operators

- Selection: *roulette wheel* (selection probability determined by normalized fitness), *ranked selection* (selection probability determined by fitness order), *elitist selection* (highest fitness individuals always selected)
- Crossover (e.g.,1 point, $p_{crossover} = 0.2$)



- Mutation (e.g., $p_{mutation} = 0.05$)



G_k G_k’

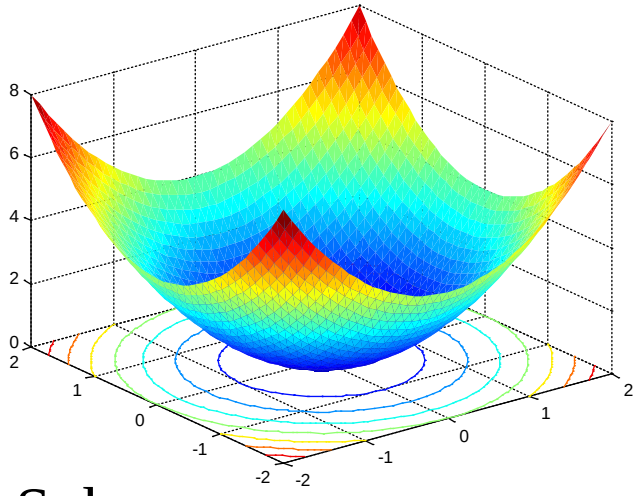
GA vs. PSO
on Benchmark Functions

Benchmark Functions

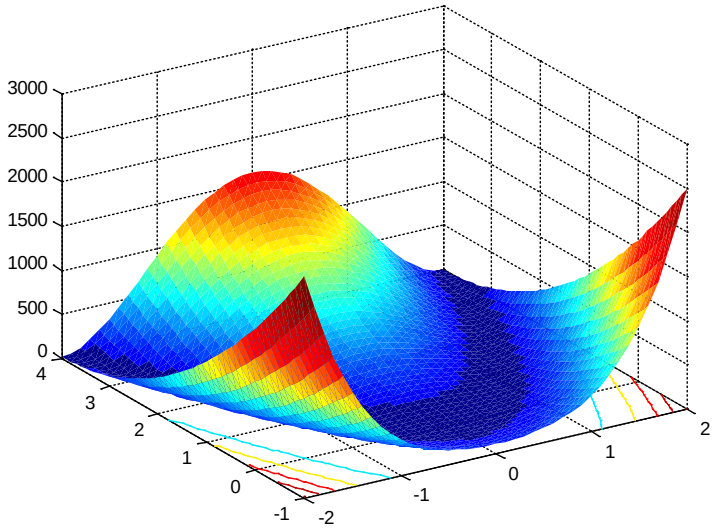
- Goal: minimization of a given f(**x**)
- Standard benchmark functions with thirty terms (n = 30) and a fixed number of iterations
- All x_i constrained to [-5.12, 5.12]
- [Pugh et al, IEEE SIS 2005]

Function Name	Function	Iterations
Sphere	$f_1(\bar{x}) = \sum_{i=1}^{n} x_i^2$	400
Generalized Rosenbrock	$f_2(\bar{x}) = \sum_{i=1}^{n-1} \left[100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2\right]$	20000
Rastrigin	$f_3(\bar{x}) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	400
Griewank	$f_4(\bar{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}})$	400

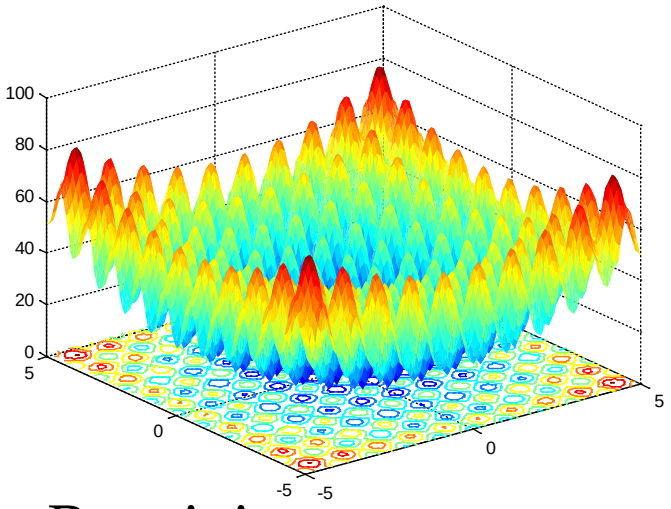
Benchmark functions



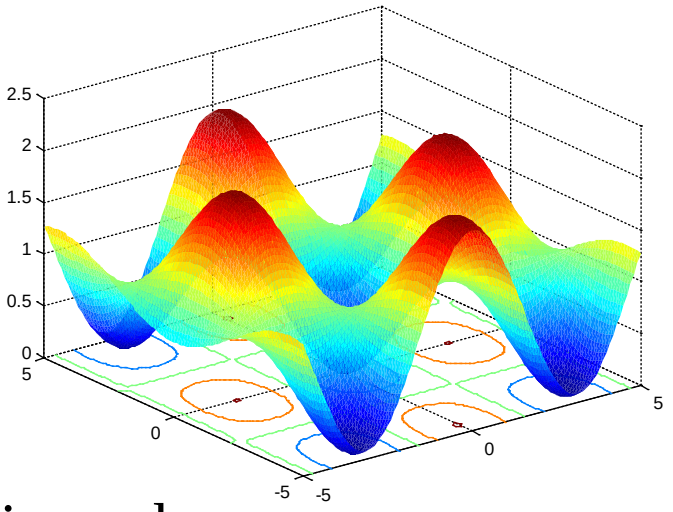
Sphere



Generalized Rosenbrock



Rastrigin



Griewank

Algorithmic Parameters

- GA: Roulette Wheel for selection, mutation applies numerical adjustment to gene
- PSO: lbest ring topology with neighborhood of size 3
- Algorithm parameters used (not optimized in general):

GA		PSO	
Population Size	20*, 30	Swarm Size	20*,30
Crossover Probability	0.6	Personal Best Attraction	2.0
Mutation Probability	0.05	Neighborhood Best Attraction	2.0
Mutation Range	[-0.5, 0.5]	Inertia Factor	0.6

* [Pugh et al, SIS 2005]: distributed handout but biased results (small population, limited numbers of runs)

GA vs. PSO – 20 candidate solutions

Bold: best results; 20 runs; no noise on the performance function

Function	GA (mean ± std dev)	PSO (mean ± std dev)
Sphere	0.02 ± 0.01	**0.00 ± 0.00**
Generalized Rosenbrock	34.6 ± 18.9	**7.38 ± 3.27**
Rastrigin	157 ±21.8	**48.3 ± 14.4**
Griewank	**0.01 ± 0.01**	0.01 ± 0.03

GA vs. PSO – 30 candidate solutions

Bold: best results; 30 runs; no noise on the performance function

Function	GA (mean ± std dev)	PSO (mean ± std dev)
Sphere	**0.01 ± 0.00**	0.64 ± 0.22
Generalized Rosenbrock	37.7 ± 23.05	**9.70 ± 7.23**
Rastrigin	116.35 ± 26.0	**105.15 ± 13.2**
Griewank	**0.00 ± 0.01**	0.04 ± 0.02

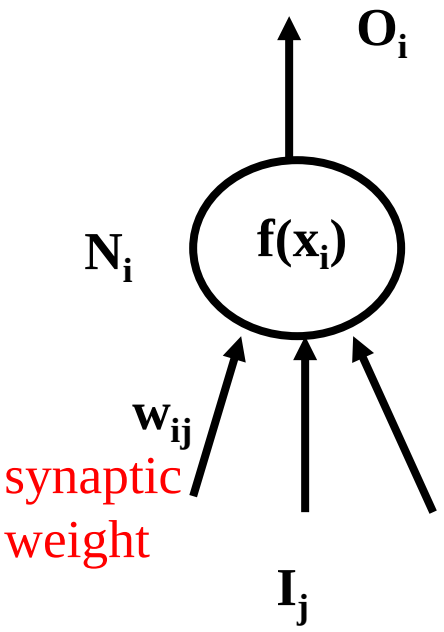
Note: data different from [Pugh et al., SIS 2005]; experiments by E. Di Mario

GA vs. PSO – Overview

- According to most recent research, PSO outperforms GA on most (but not all!) continuous optimization problems
- GA still much more widely used in general research community and robust to continuous and discrete optimization problems
- Because of random aspects, very difficult to analyze either metaheuristic or make guarantees about performance

Design and Optimization of Obstacle Avoidance Behavior using Genetic Algorithms

Evolving a Neural Controller

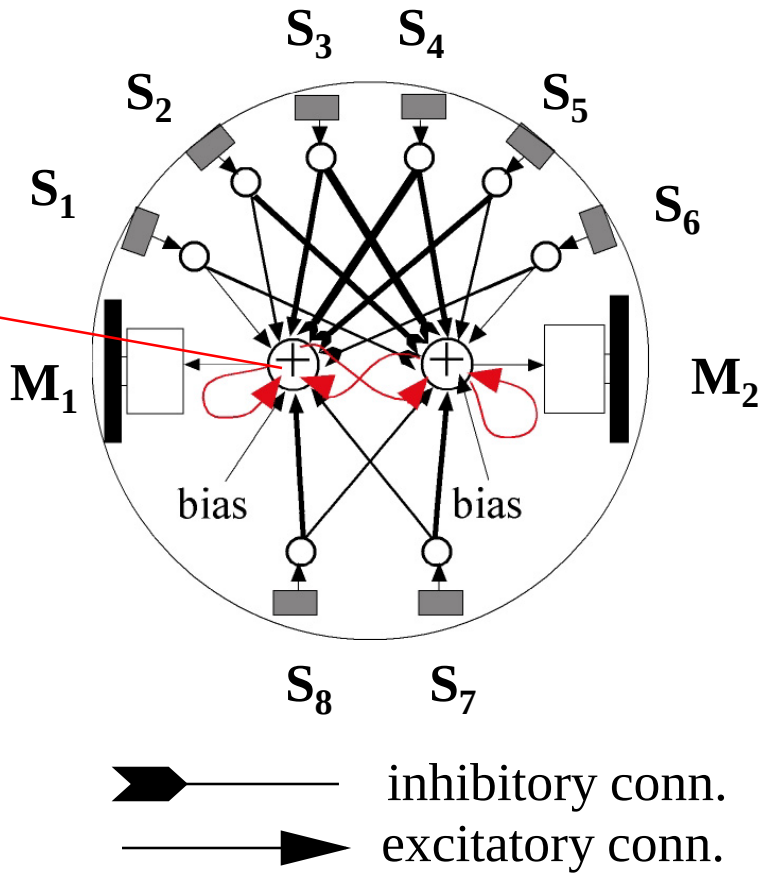


neuron N with sigmoid transfer function f(x)

$$O_i = f(x_i)$$

$$f(x) = \frac{2}{1+e^{-x}} - 1$$

$$x_i = \sum_{j=1}^{m} w_{ij} I_j + I_0$$



Note: In our case we evolve synaptic weigths but Hebbian rules for dynamic change of the weights, transfer function parameters, … can also be evolved (see Floreano's course)

Evolving Obstacle Avoidance (Floreano and Mondada 1996)

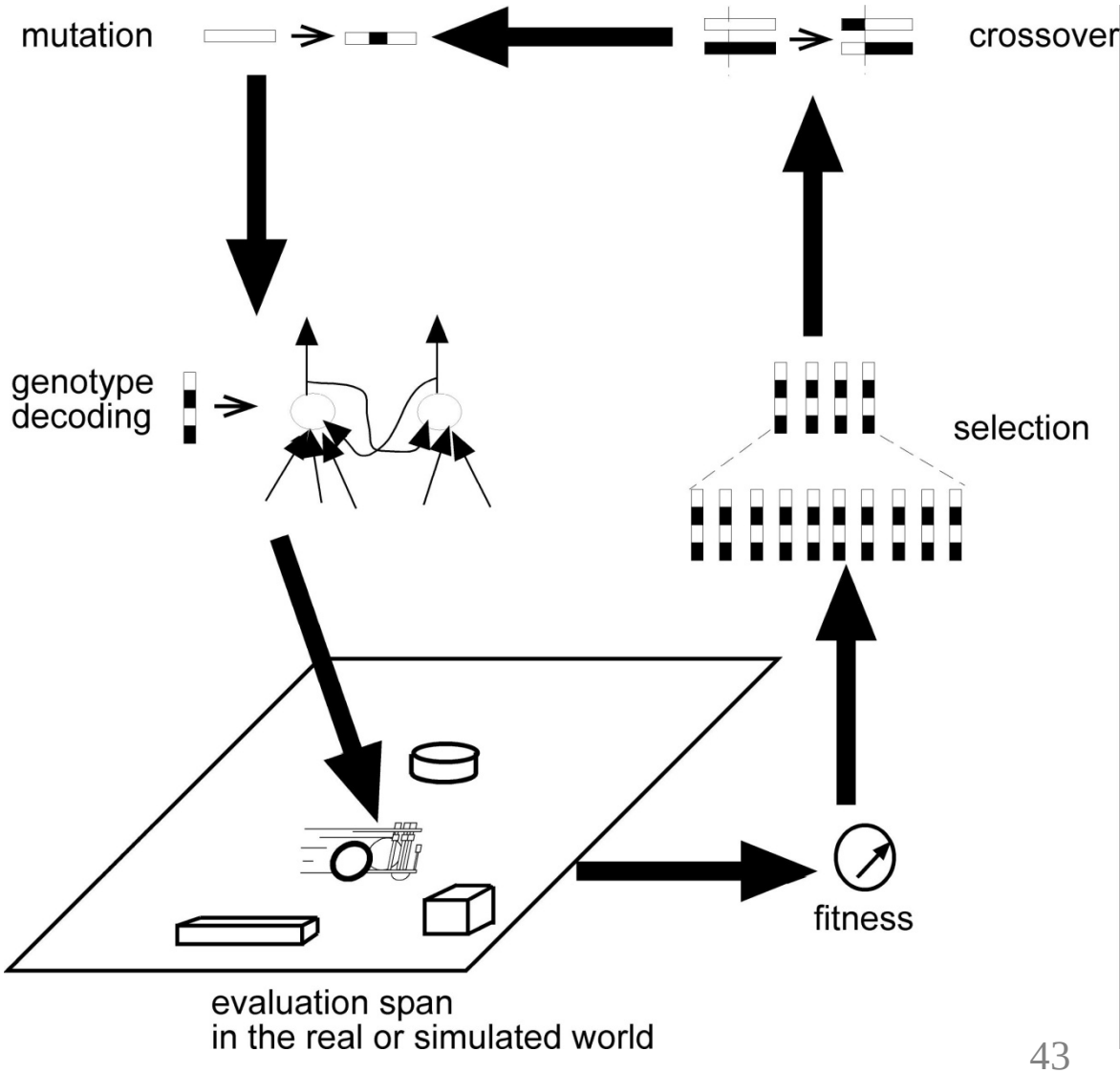
Defining performance (fitness function):

$$\Phi = V(1-\sqrt{\Delta V})(1-i)$$

- **V** = mean speed of wheels, $0 \leq V \leq 1$
- **Δv** = absolute algebraic difference between wheel speeds, $0 \leq \Delta v \leq 1$
- **i** = activation value of the sensor with the highest activity, $0 \leq i \leq 1$

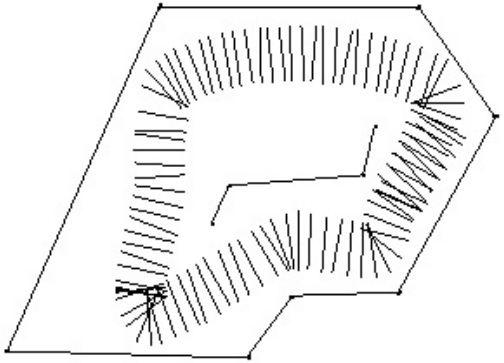
Note: Fitness accumulated during evaluation span, normalized over number of control loops (actions).

Shaping Robot Controllers

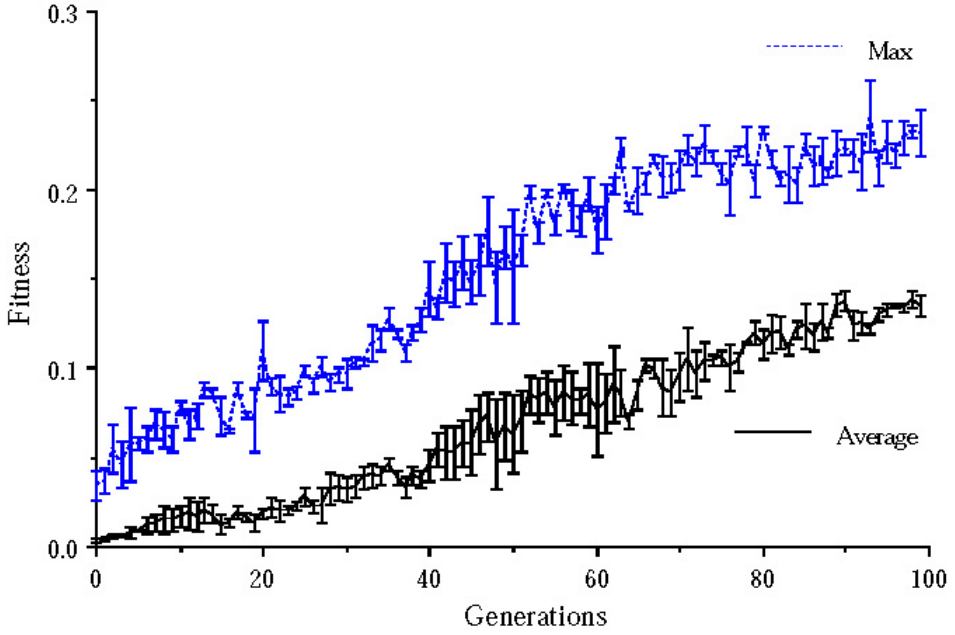


Note:
Controller architecture can be of any type but worth using GA/PSO if the number of parameters to be tuned is important

Evolving Obstacle Avoidance



Evolved path



Fitness evolution

Evolved Obstacle Avoidance Behavior

Generation 100

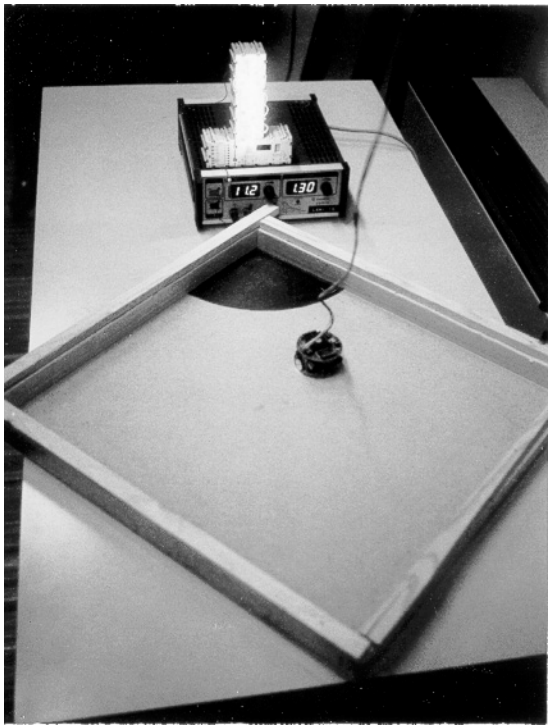
- on-line
- off-board (PC-hosted)
- hardware-in-the-loop
- population-based

Note: Direction of motion NOT encoded in the fitness function: GA automatically discovers asymmetry in the sensory system configuration (6 proximity sensors in the front and 2 in the back)

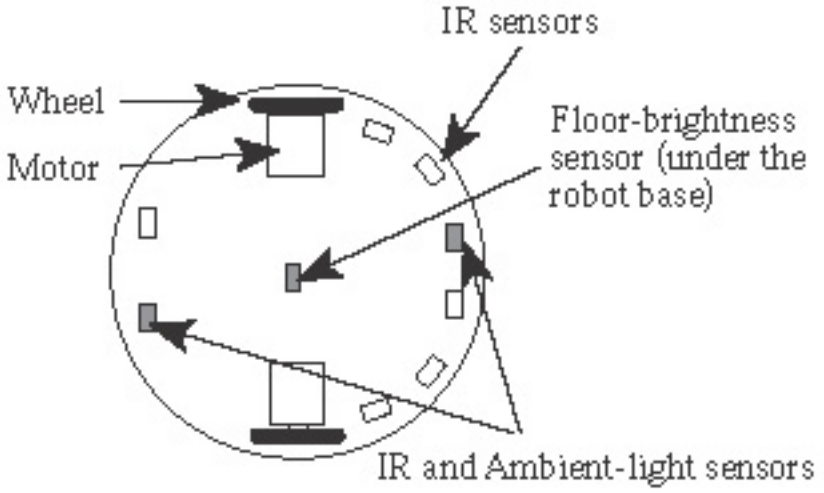
Not only Obstacle Avoidance: Evolving More Complex Behaviors

Evolving Homing Behavior

(Floreano and Mondada 1996)



Set-up



Robot's sensors

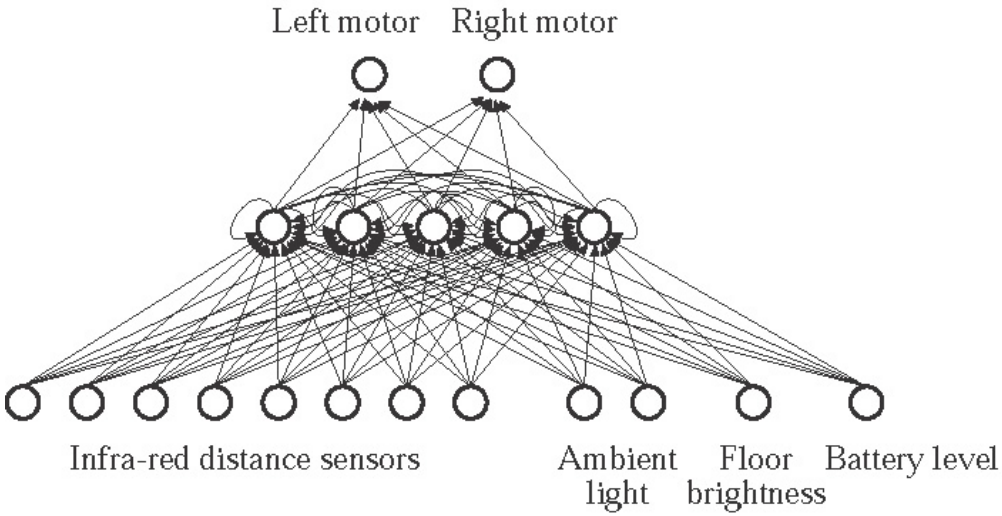
Evolving Homing Behavior

- **Fitness function:**

$$\Phi = V(1-i)$$

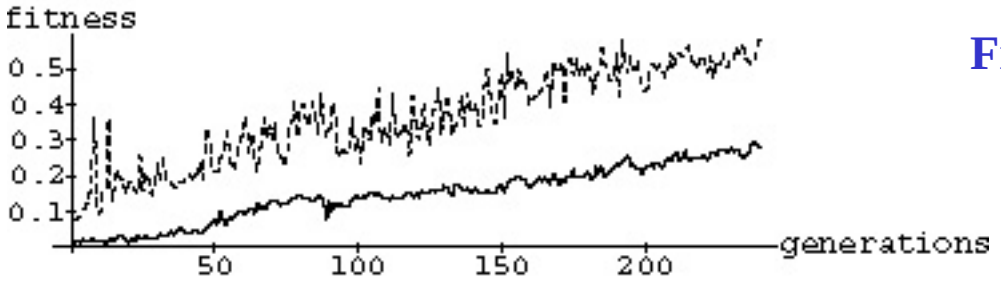
- **V** = mean speed of wheels, $0 \leq V \leq 1$
- **i** = activation value of the sensor with the highest activity, $0 \leq i \leq 1$

Controller

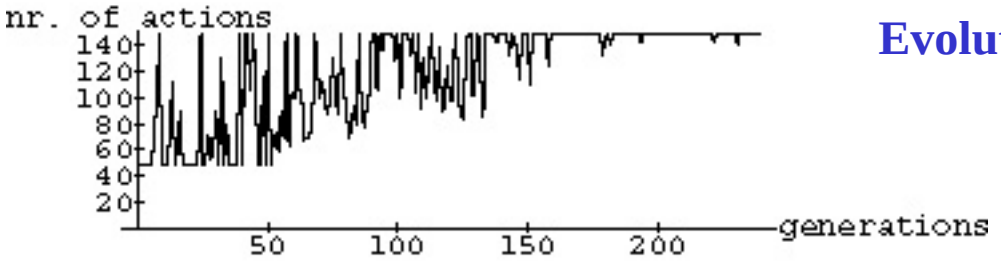


- Fitness accumulated during life span, normalized over maximal number (150) of control loops (actions).
- No explicit expression of battery level/duration in the fitness function (implicit).
- Chromosome length: 102 parameters (real-to-real encoding).
- Generations: 240, 10 days hardware-in-the-loop evolution

Evolving Homing Behavior

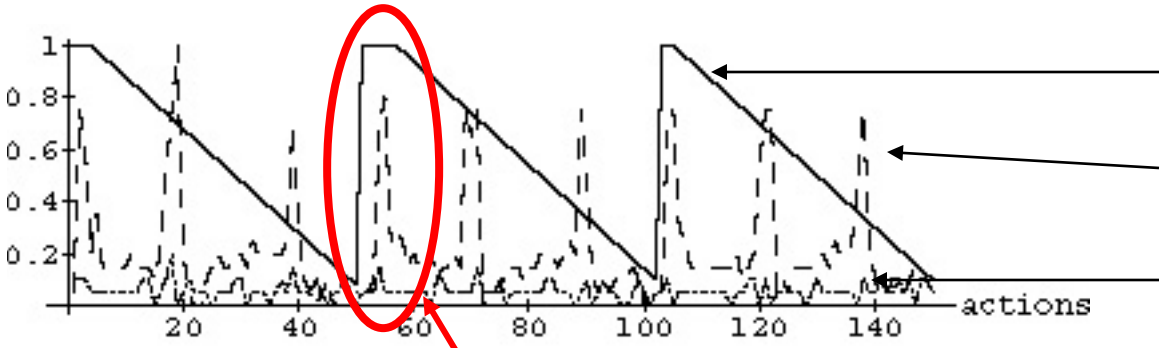


Fitness evolution



Evolution of # control loops per evaluation span

Battery recharging vs. motion patterns



Battery energy

Left wheel activation

Right wheel activation

Reach the nest -> battery recharging -> turn on spot -> out of the nest

Evolved Homing Behavior

Not only Control Shaping: Automatic Hardware-Software Co-Design and Optimization in Simulation and Validation with Real Robots

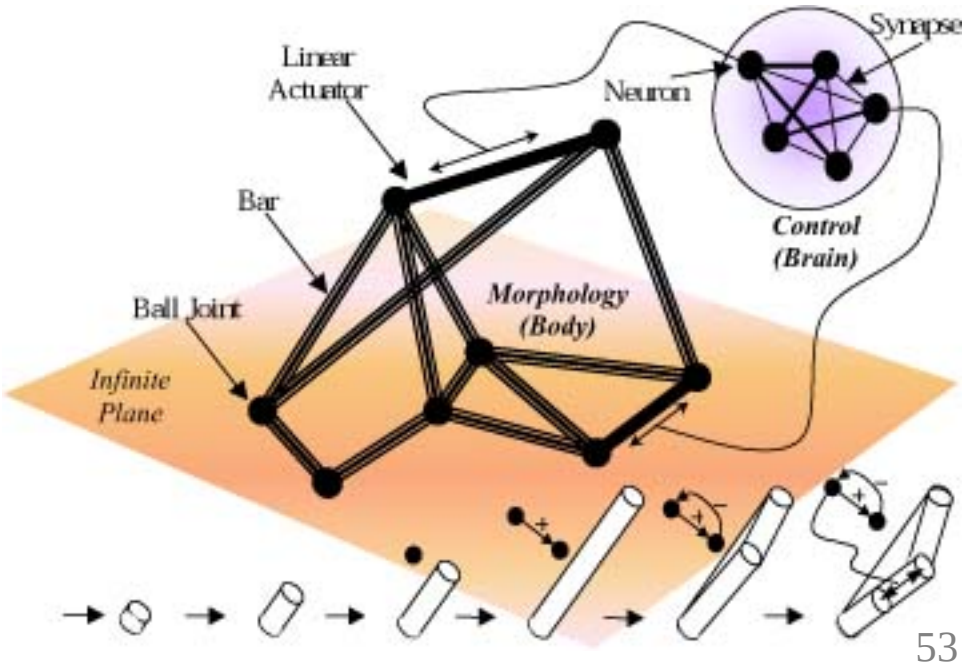
Moving Beyond Controller-Only Evolution

- Evidence: Nature evolve HW and SW at the same time …
- Faithful realistic simulators enable to explore design solution which encompasses co-evolution (co-design) of control and morphological characteristics (body shape, number of sensors, placement of sensors, etc.)
- GA (PSO?) are powerful enough for this job and the methodology remain the same; only encoding changes

Evolving Control and Robot Morphology
(Lipson and Pollack, 2000)

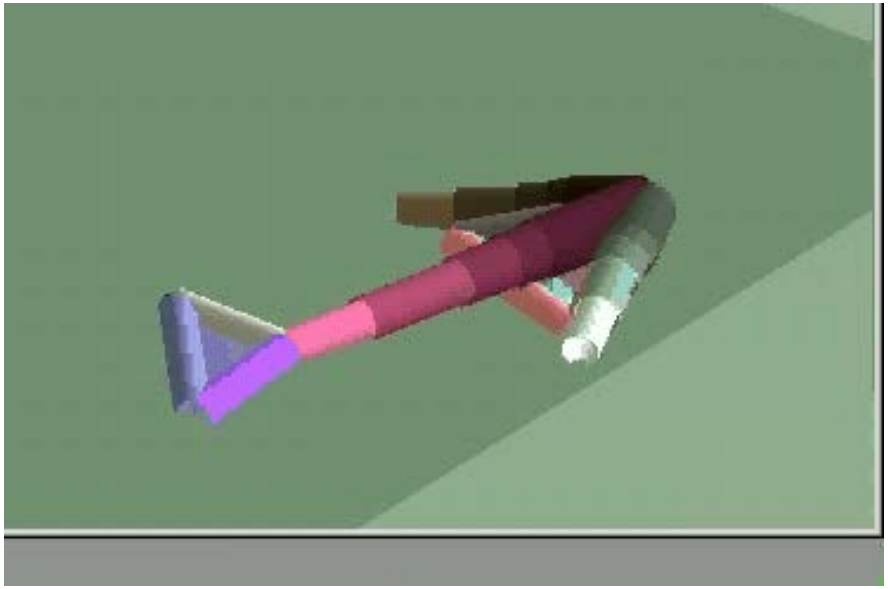
http://www.mae.cornell.edu/ccsl/research/golem/index.html

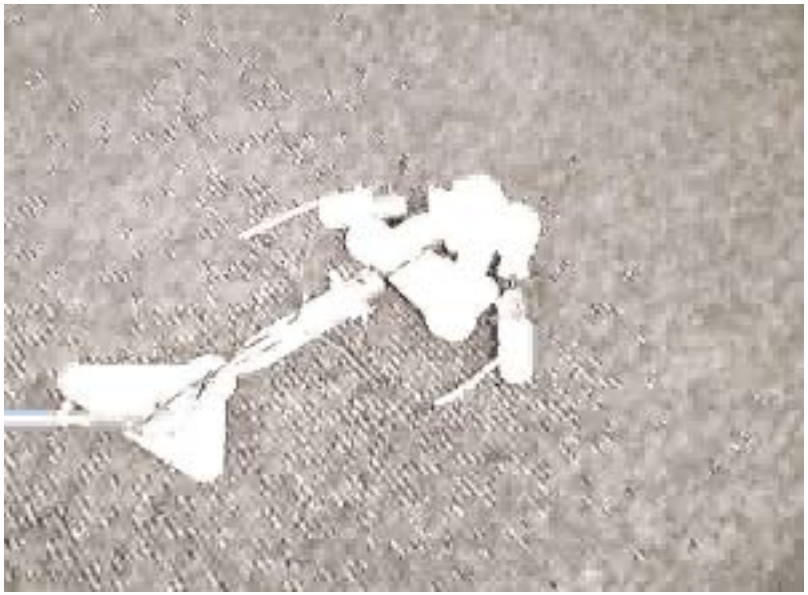
- Arbitrary recurrent ANN
- Passive and active (linear actuators) links
- Fitness function: net distance traveled by the centre of mass in a fixed duration



Example of evolutionary sequence:

Examples of Evolved Machines





Problem: simulator not enough realistic (performance higher in simulation because of not good enough simulated friction; e.g., for the arrow configuration 59.6 cm vs. 22.5 cm)

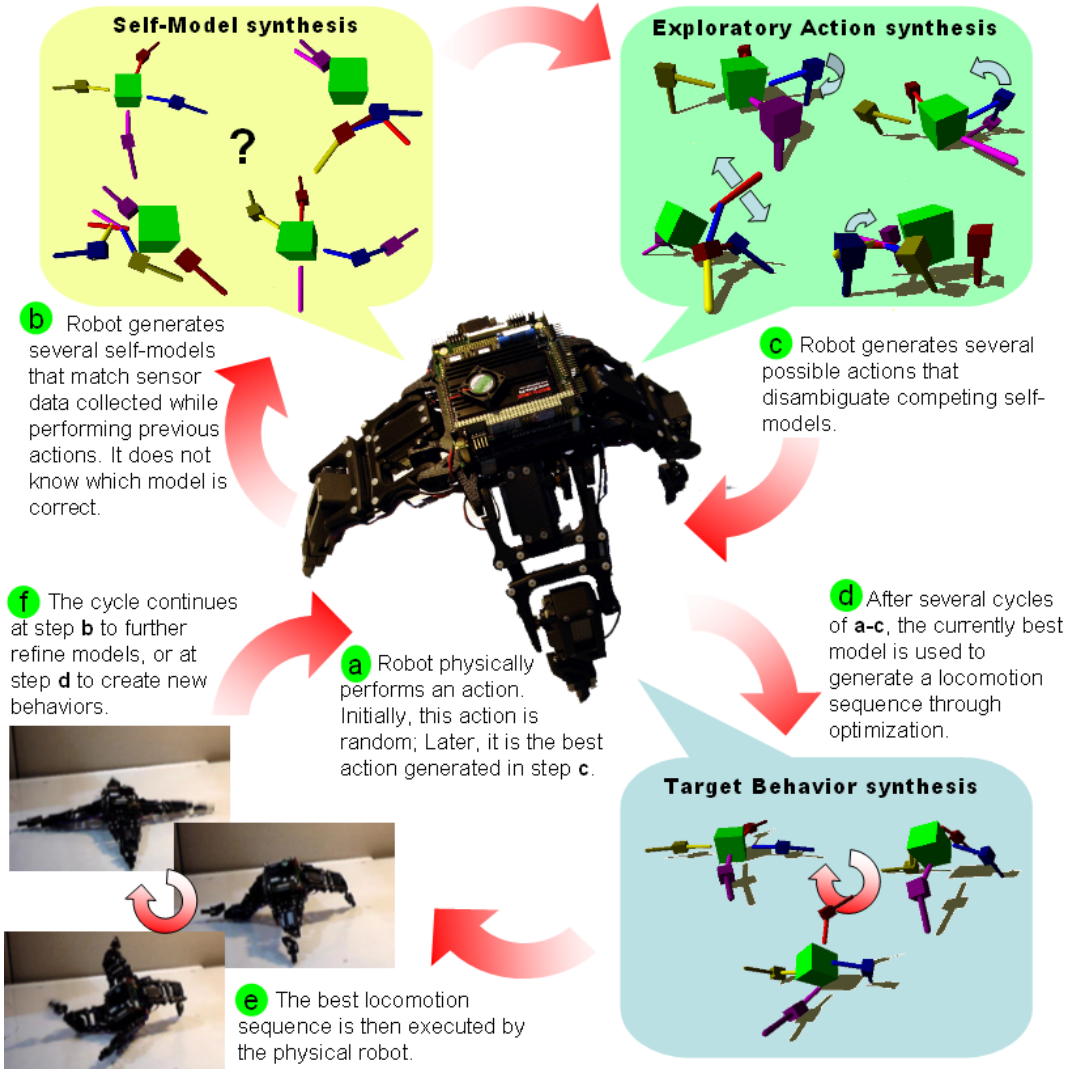
Issues for Evolving Real Systems by Exploiting Simulation

Goal: speeding up evolution

Traditional recipes (see Axis 2 classification):

1. Evolve in simulation, download on real HW
   - Issue: Bridge the simulation-reality gap
2. Evolve with real HW in the loop
   - Issue: evaluation span too time consuming (and fragile)

Co-Evolution of Simulation and Reality



[Bongard, Zykov, and Lipson, 2006]

http://ccsl.mae.cornell.edu/emergent_self_models

Morphological Estimation and Damage Recovery

[Bongard, Zykov, and Lipson, 2006]

Co-Evolution of Models and Reality

- "Analysis by synthesis" method
- Powerful system identification method (not only model parameters but also model structure)
- Limitation: implies knowledge about the underlying phenomena, shrinking the system rules to a possible alphabet (gray box as opposed to black box identification)

Expensive Optimization and Noise Resistance

Expensive Optimization Problems

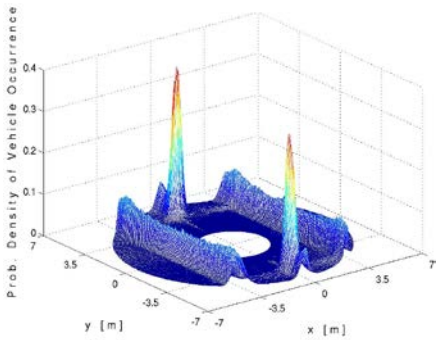
Two fundamental reasons making robot control design and optimization expensive in terms of time:

1. Time for evaluation of candidate solutions (e.g., tens of seconds) >> time for application of metaheuristic operators (e.g., milliseconds)
2. Noisy performance evaluations disrupt the adaptation process and require multiple evaluations for actual performance

Reducing Evaluation Time

General recipe: exploit more abstracted, calibrated representations (models and simulation tools)

See also multi-level modeling lectures (W8 and W9)

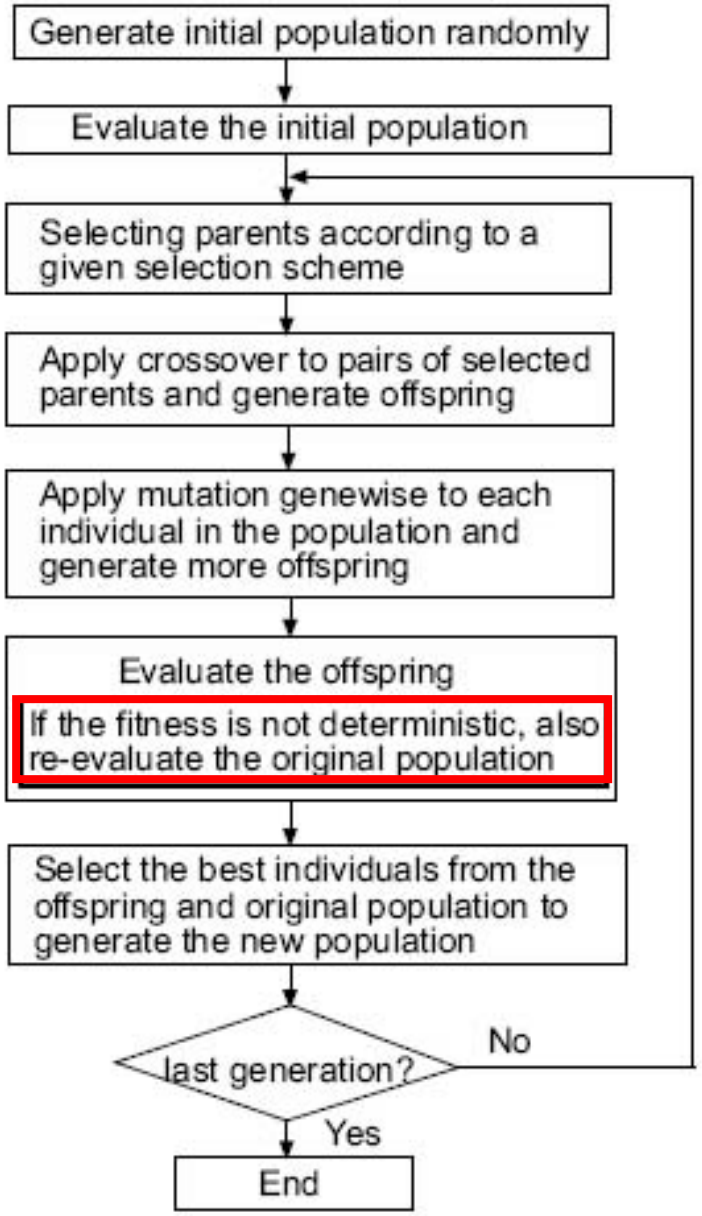


Dealing with Noisy Evaluations

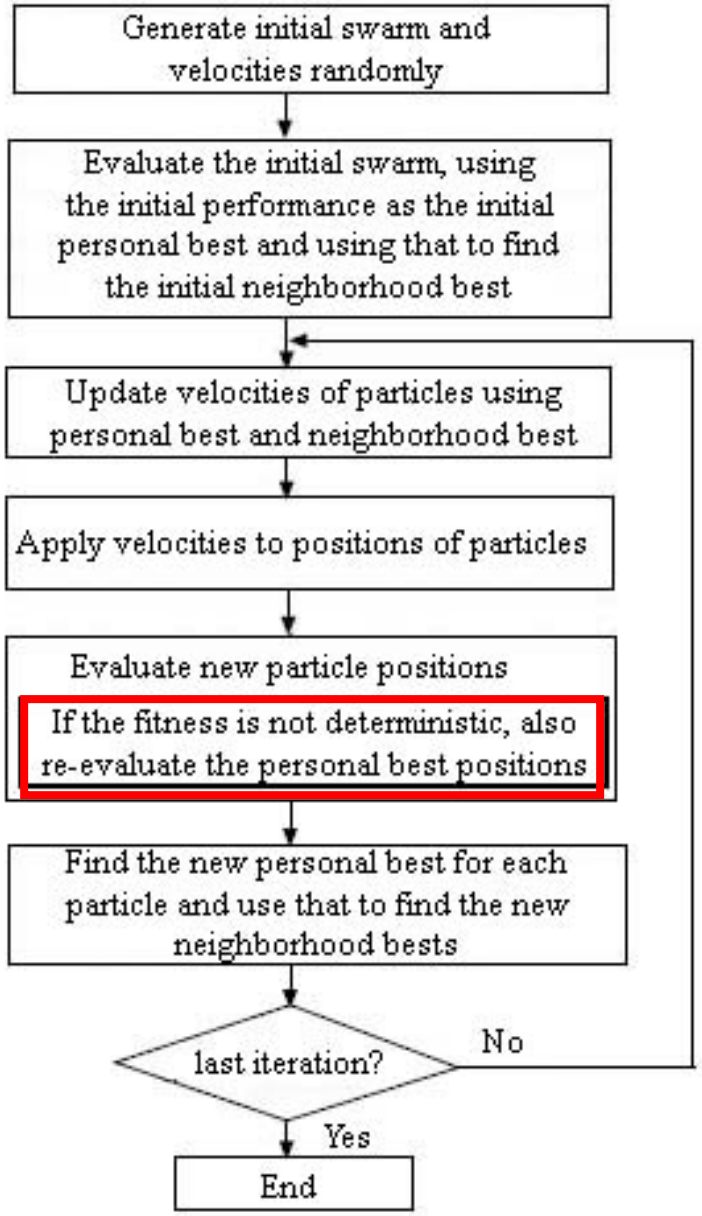
- Better information about candidate solution can be obtained by combining multiple noisy evaluations
- We could evaluate systematically each candidate solution for a fixed number of times → not efficient from a computational perspective
- We want to dedicate more computational time to evaluate promising solutions and eliminate as quickly as possible the "lucky" ones
- Idea: re-evaluate and aggregate → each candidate solution might have been evaluated a different number of times → compare the aggregated value
- For instance, in GA good and robust candidate solutions survive over generations; in PSO they survive in the individual memory
- Use dedicated functions for aggregating multiple evaluations: e.g., minimum and average or more generalized aggregation functions (e.g., quasi-linear weighted means), perhaps combined with a statistical test for comparing resulting aggregated performances (see also W11 lecture)

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

GA



PSO



Testing Noise-Resistant on Benchmarks

- Benchmark 1 : Sphere and Generalized Rosenbrock functions
  - 30 real parameters [Pugh et al., SIS 2005]
  - Minimize objective function
  - Expensive only because of noise

- Benchmark 2: obstacle avoidance on a robot
  - 24 real parameters
  - Maximize objective function
  - Expensive because of noise and evaluation time

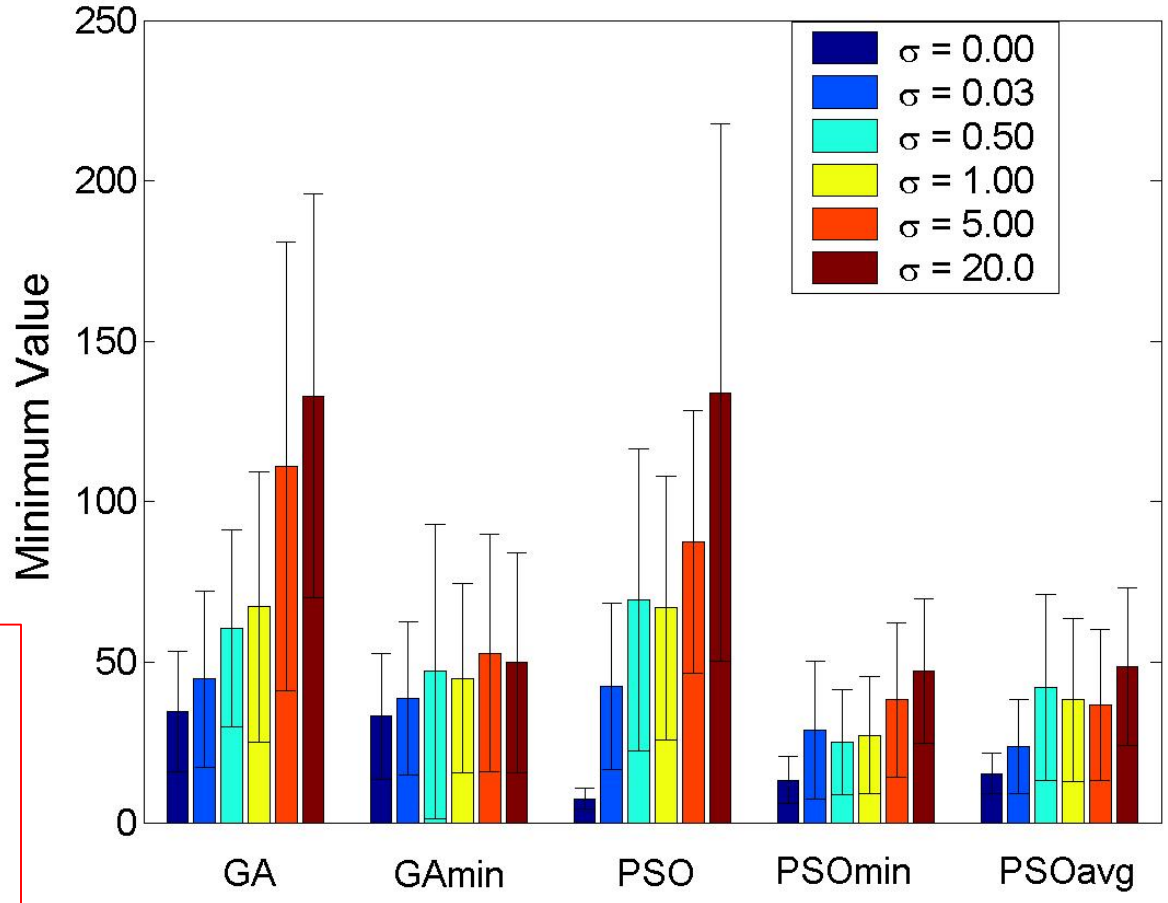
Benchmark 1: Gaussian Additive Noise on Generalized Rosenbrock

$$f'_j(\bar{x}) = \mathcal{N}(0, \sigma^2) + f_j(\bar{x})$$

[Pugh et al., SIS 2005]

Fair test: **same** number of evaluations candidate solutions for all algorithms

(i.e. N generations/ iterations of standard versions compared with N/2 of the noise-resistant ones)

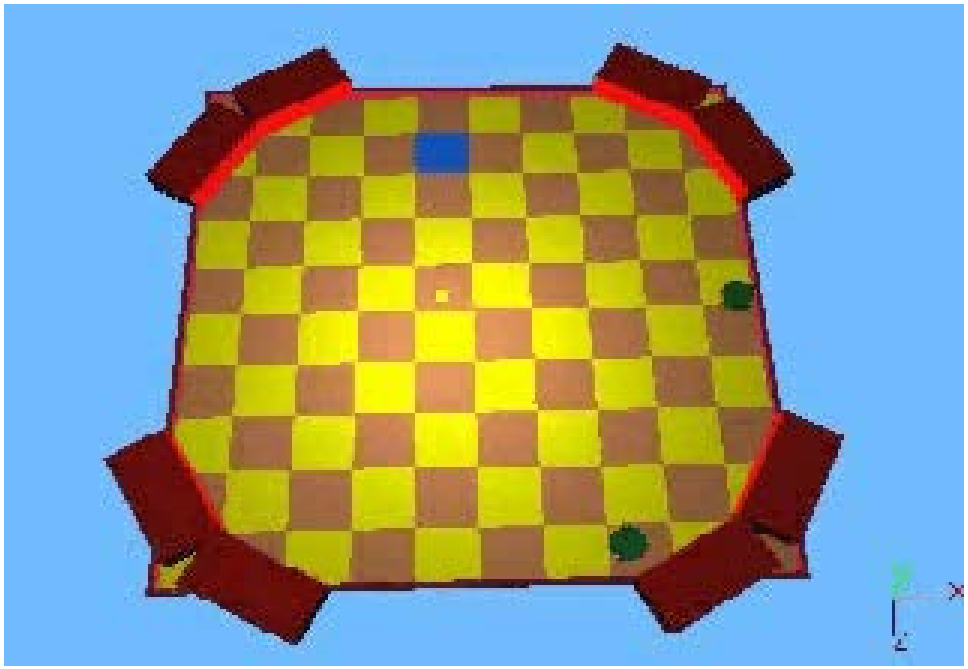


Biased results: low number of runs (20) and population size (20) < search dimension (30)!

Increasing Noise Level – Set-Up

- **Scenario 1**: One robot learning obstacle avoidance
- **Scenario 2**: One robot learning obstacle avoidance, one robot running pre-evolved obstacle avoidance
- **Scenario 3**: Two robots co-learning obstacle avoidance

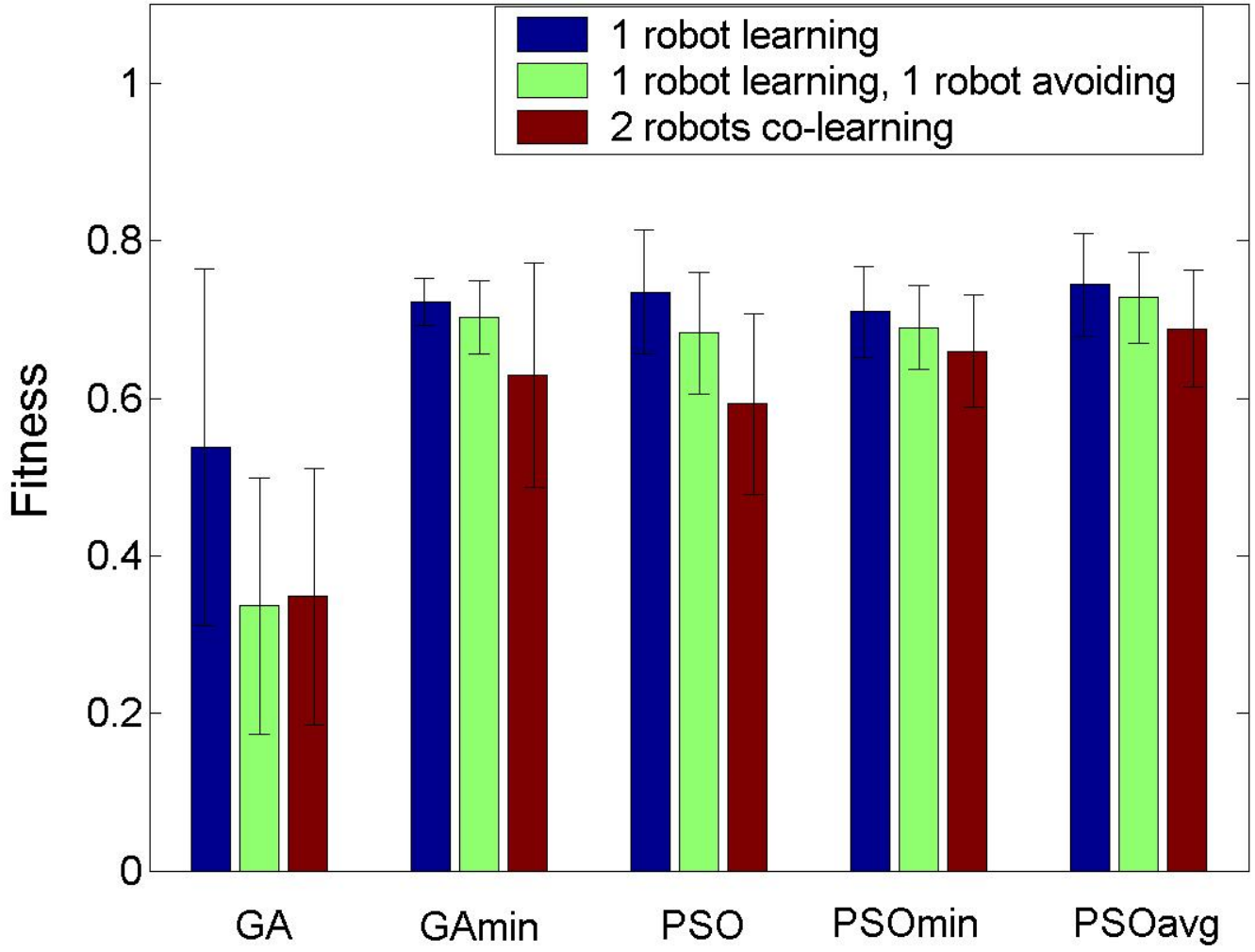
1x1 m arena, PSO, 50th iteration, scenario 3



Idea: more robots more noise (as perceived from an individual robot) because there is no explicit communication between the robots (in scenario three even more noisy because both robots are learning independently).

[Pugh et al, SIS 2005]

Increasing Noise Level – Sample Results



[Pugh et al, SIS 2005]

Conclusion

Take Home Messages

- Machine-learning algorithms can be classified as supervised, unsupervised, and reinforcement-based (evaluative)
- Evaluative techniques are key for robotic learning
- Two robust population-based metaheuristics are PSO and GA
- PSO is a younger technique than GA but extremely promising
- Evaluative techniques can be used for design and optimization of behaviors whose complexity goes beyond obstacle avoidance
- Machine-learning techniques can be also used for design and optimization of hardware features with the help of simulation tools
- Automatic design and optimization problems in robotics are computationally expensive because of long and noisy evaluation of candidate solutions

Additional Literature – Week 10

Books

- Mitchell M., “An Introduction to Genetic Algorithms”. MIT Press, 1996.
- Kennedy J. and Eberhart R. C. with Y. Shi, “Swarm Intelligence”. Morgan Kaufmann Publisher, 2001.
- Clerc M., “Particle Swarm Optimization”. ISTE Ltd., London, UK, 2006.
- Engelbrecht A. P., “Fundamentals of Computational Swarm Intelligence”. John Wiley & Sons, 2006.
- Nolfi S. and Floreano D., “Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines”. MIT Press, 2004.
- Sutton R. S. and Barto A. G., “Reinforcement Learning: An Introduction”. The MIT Press, Cambridge, MA, 1998.