Advanced Control

January 24, 2014

École Centrale Paris, Châtenay-Malabry, France

Anne Auger
INRIA Saclay – Ile-de-France

INVENTORS FOR THE DIGITAL WORLD

Dimo Brockhoff
INRIA Lille – Nord Europe

Course Overview

Date		Topic
Fri, 10.1.2014	DB	Introduction to Control, Examples of Advanced Control
Fri, 17.1.2014	DB	Introduction to Fuzzy Logic
Fri, 24.1.2014	DB	Introduction to Artificial Neural Networks, Bio-inspired Optimization, discrete search spaces
Fri, 31.1.2014	AA	Continuous Optimization I
Fri, 7.2.2014	AA	Continuous Optimization II
break		
Fri, 28.2.2014	AA	The Traveling Salesperson Problem
Fr, 7.3.2014	DB	Controlling a Pole Cart
Fr, 14.3.2014		written exam (paper and computer)

all classes + exam at **8h00-11h15** (incl. a 15min break around 9h30) here in CTI-B3

Remark to last lecture

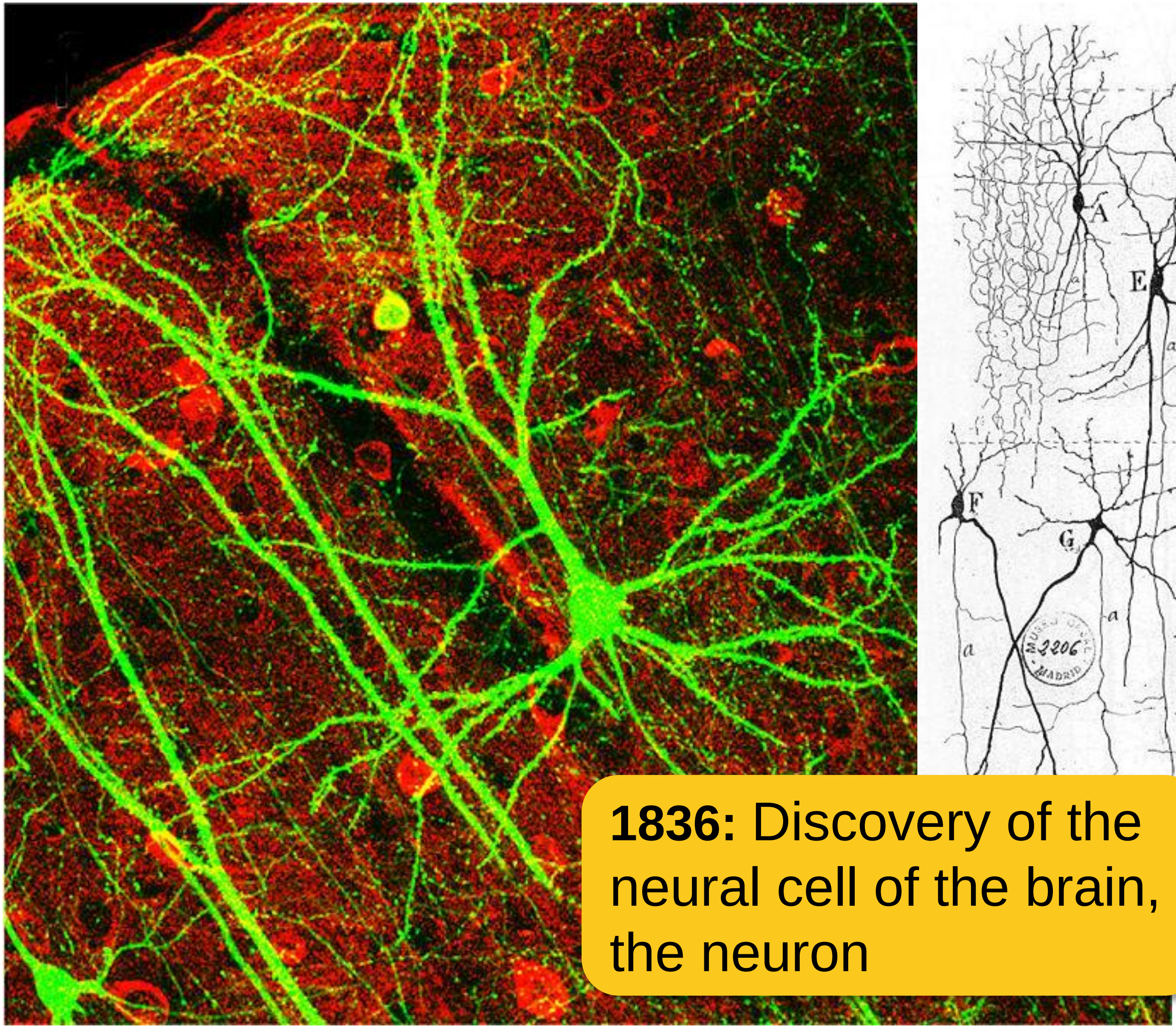
All information also available at

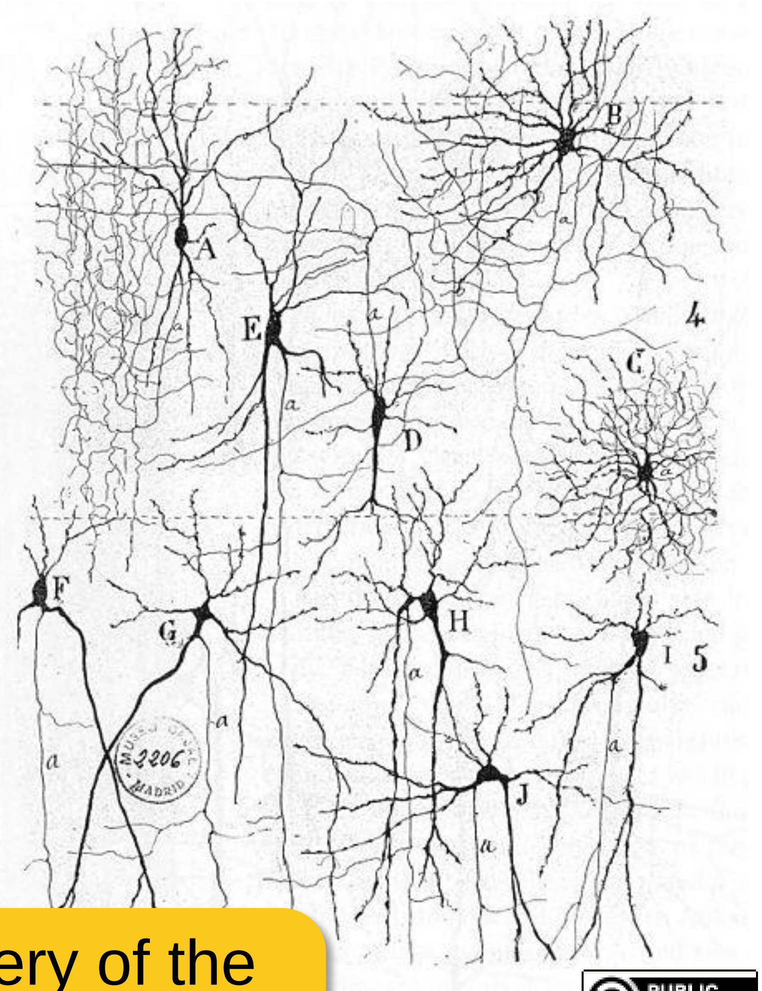
`http://researchers.lille.inria.fr/~brockhof/advancedcontrol/`

(exercise sheets, lecture slides, additional information, links, ...)

Artificial Neural Networks

The Biological Neuron

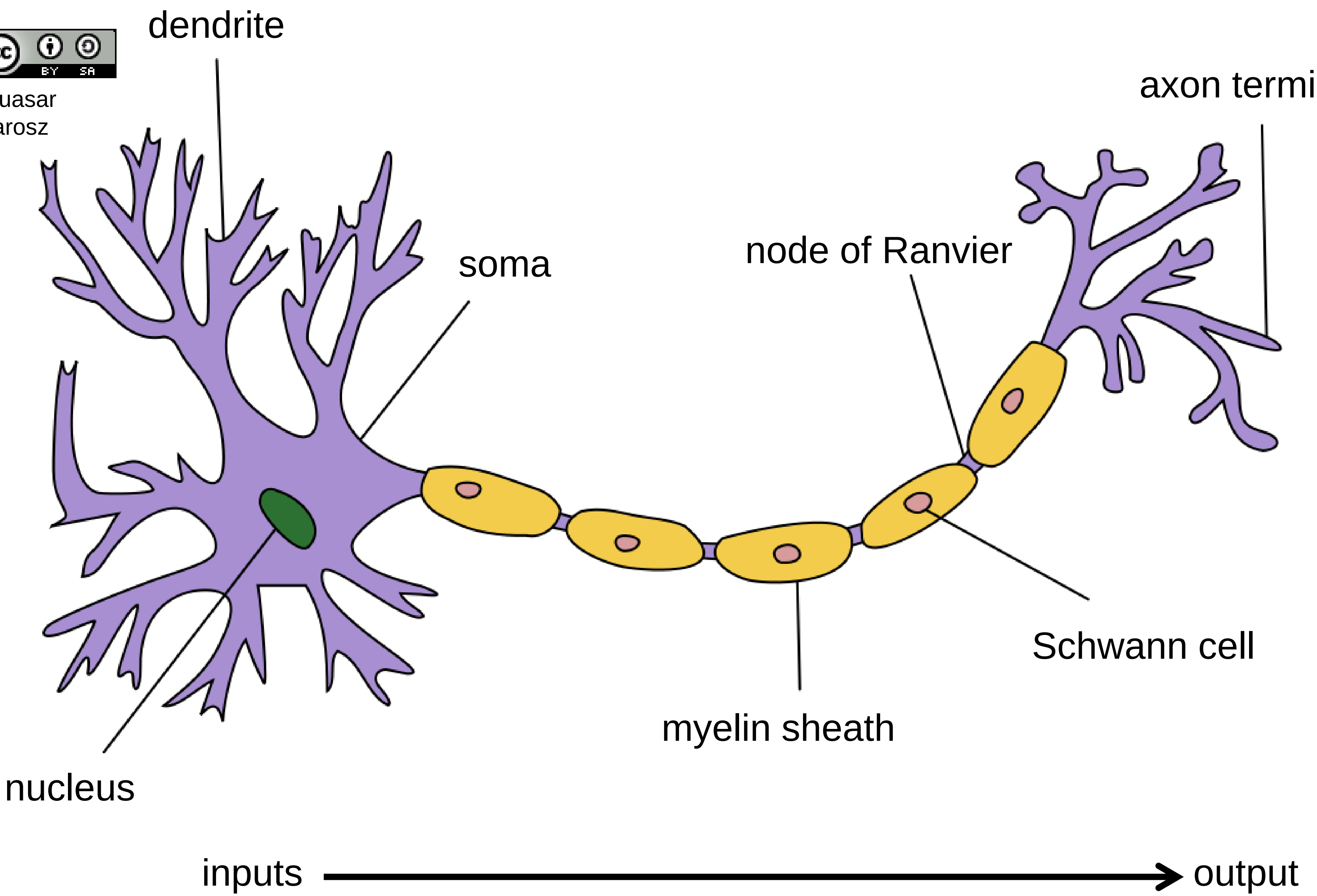




1836: Discovery of the neural cell of the brain, the neuron

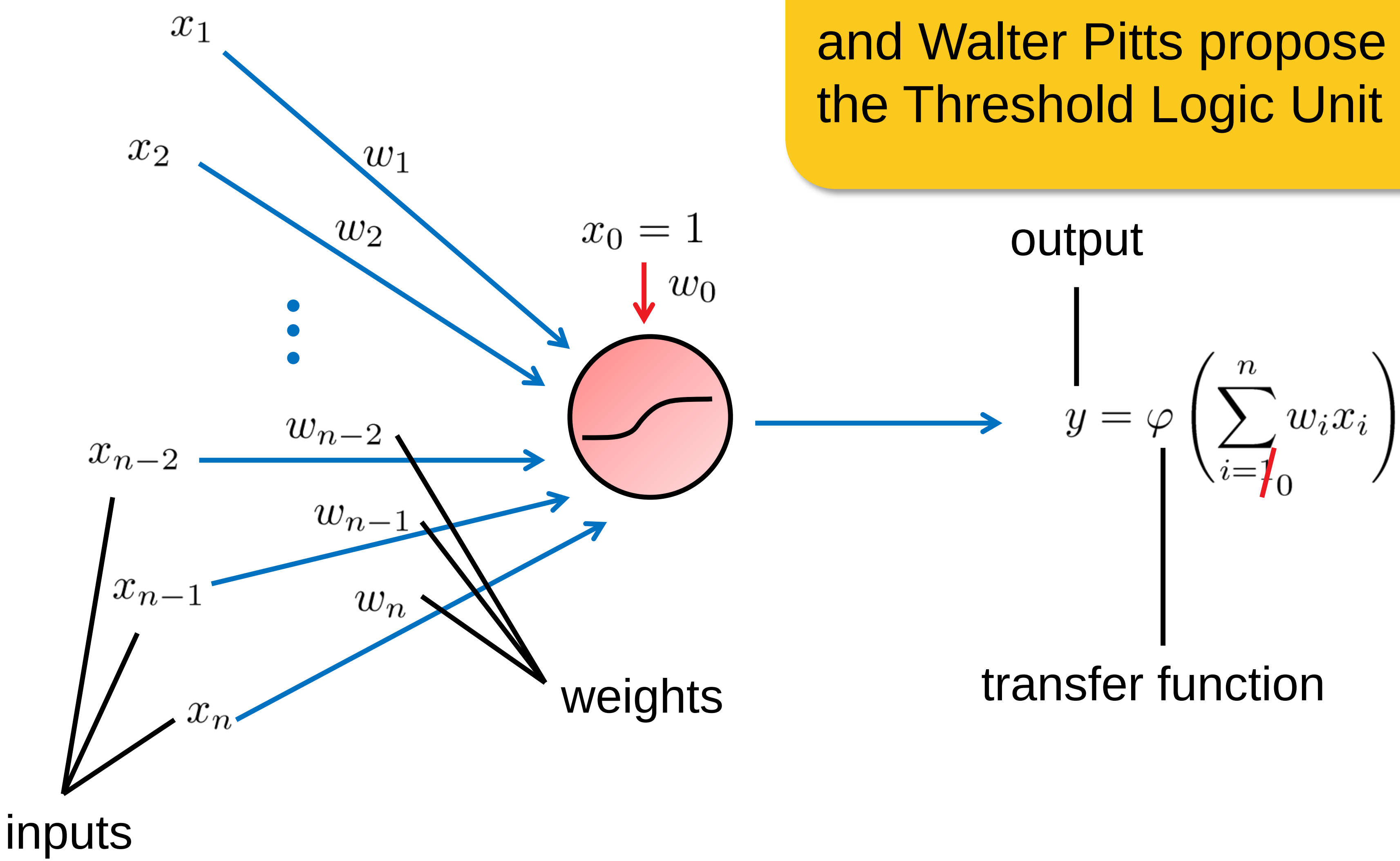
 W.-C. A. Lee, H. Huang, G. Feng, J. R. Sanes, E. N. Brown, P. T. So, E. Nedivi

The Biological Neuron



An Artificial Neuron

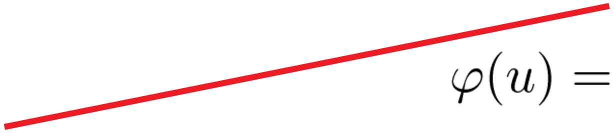
1943: Warren McCulloch and Walter Pitts propose the Threshold Logic Unit



Types of Transfer Functions

$$y = \varphi\Big(\underbrace{\sum_{i=1}^{n} w_i x_i}_{u}\Big)$$

linear

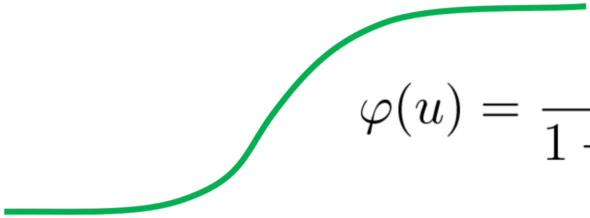


$$\varphi(u) = u + x_0$$

step

$$\varphi(u) = \begin{cases} 1 & \text{if } u \geq \theta \\ 0 & \text{if } u < \theta \end{cases}$$

sigmoidal

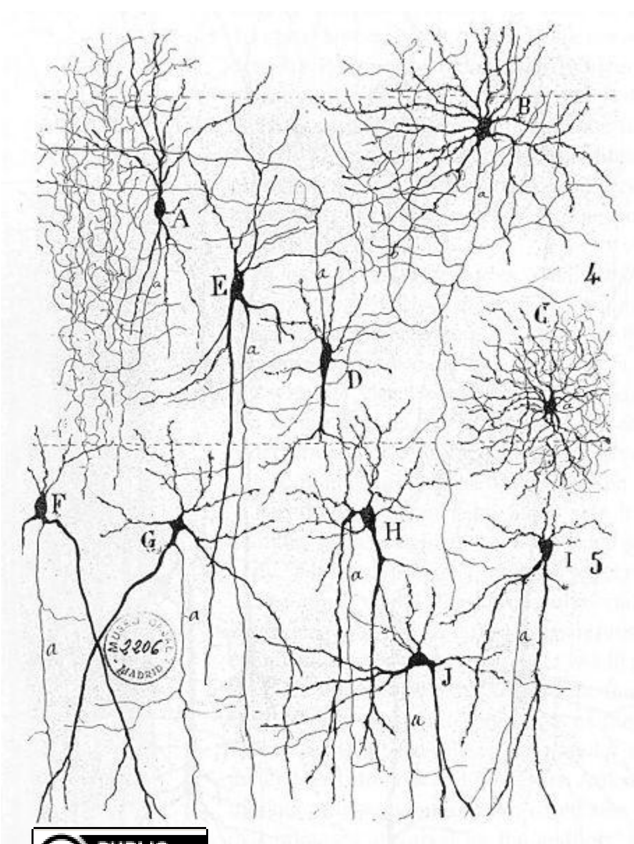


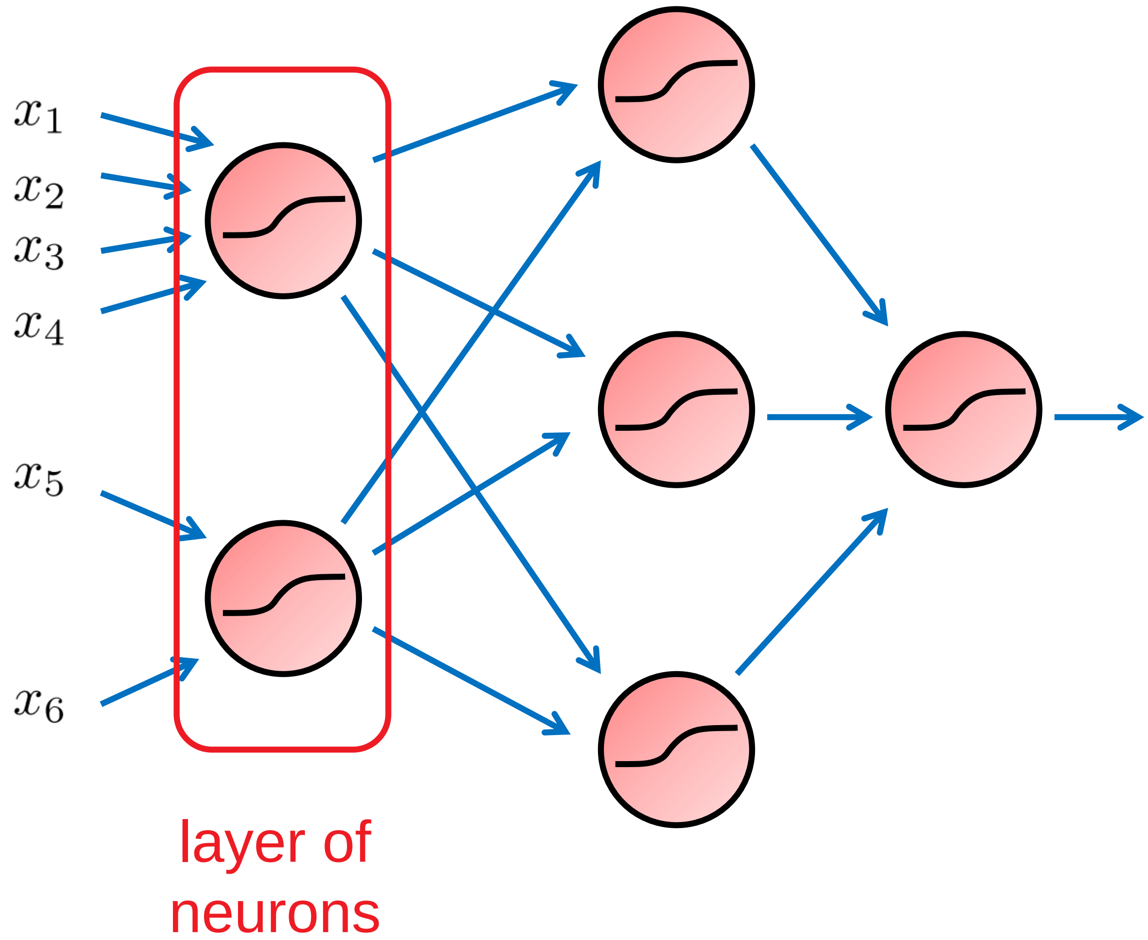
$$\varphi(u) = \frac{1}{1 + e^{-cu}}$$

advantage: differentiable

Combining Artificial Neurons

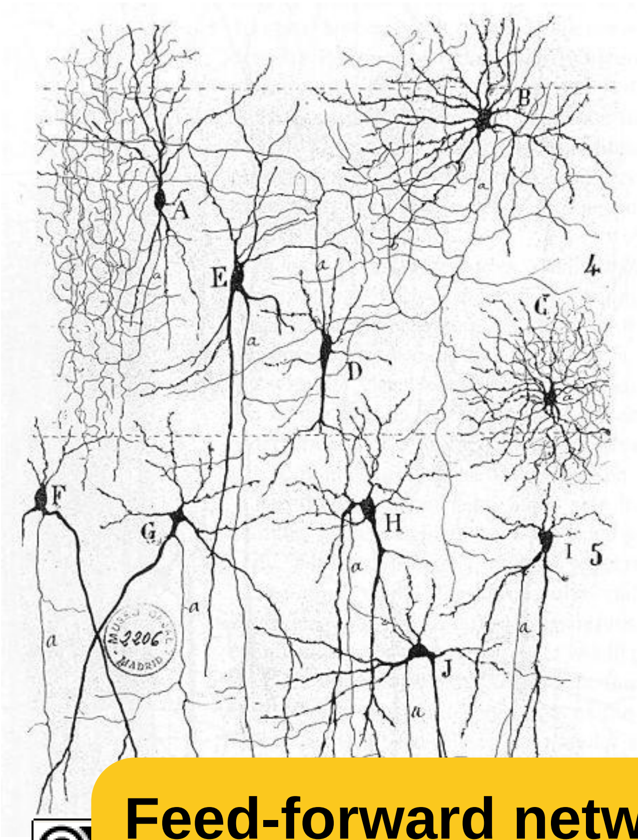
Artificial Neural Networks (ANNs) = a network of artificial neurons

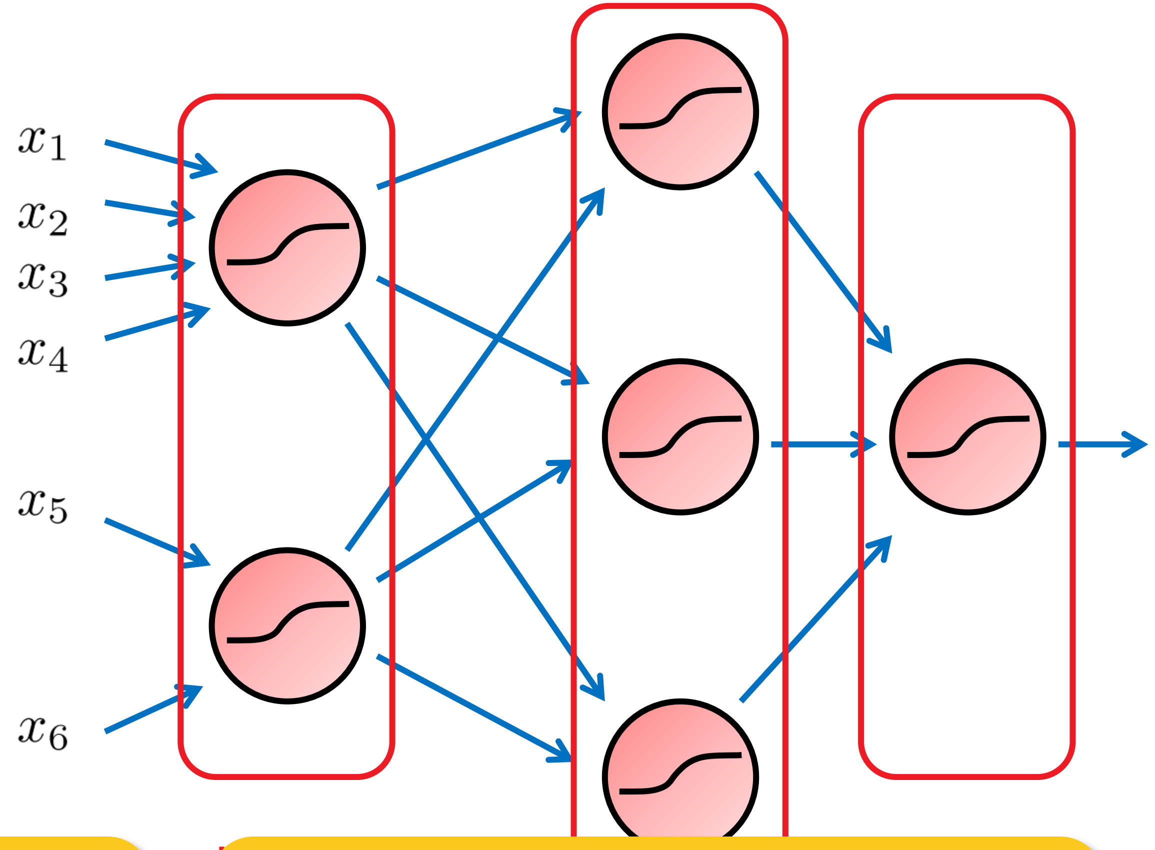




Combining Artificial Neurons

Artificial Neural Networks (ANNs) = a network of artificial neurons





Feed-forward network: no “backwards” flow of information

Linear transfer functions: multi-layer networks can be simulated by a single-layer ANN

Optimizing Weights in Order to Optimize Output

Supervised learning scenario:

- neural network with n inputs and m outputs
- given a set of training data $(\vec{x}_1, \vec{d}_1), \ldots, (\vec{x}_p, \vec{d}_p)$
- what are “optimal” weights such that

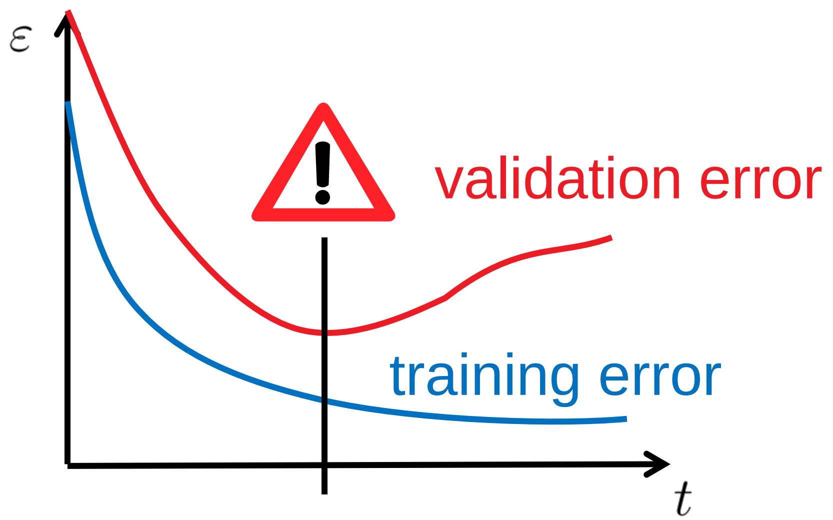
$$E(\vec{w}) = \sum_{j=1}^{p} \sum_{k=1}^{m} ||y_{j,k} - d_{j,k}||^2 = \sum_{j=1}^{p} \sum_{k=1}^{m} ||\varphi_k(\vec{x}_j, \vec{w}) - d_{j,k}||^2$$

is minimal?

Testing and Training in Supervised Learning

training data set vs. testing data set

training error vs. validation error



Generalization vs. Overfitting

- generalization behaviour desired
- overfitting especially when not much training data available

Gradient Descent to Optimize

Optimization: $$\min_{x \in \mathbb{R}^n} f(x)$$

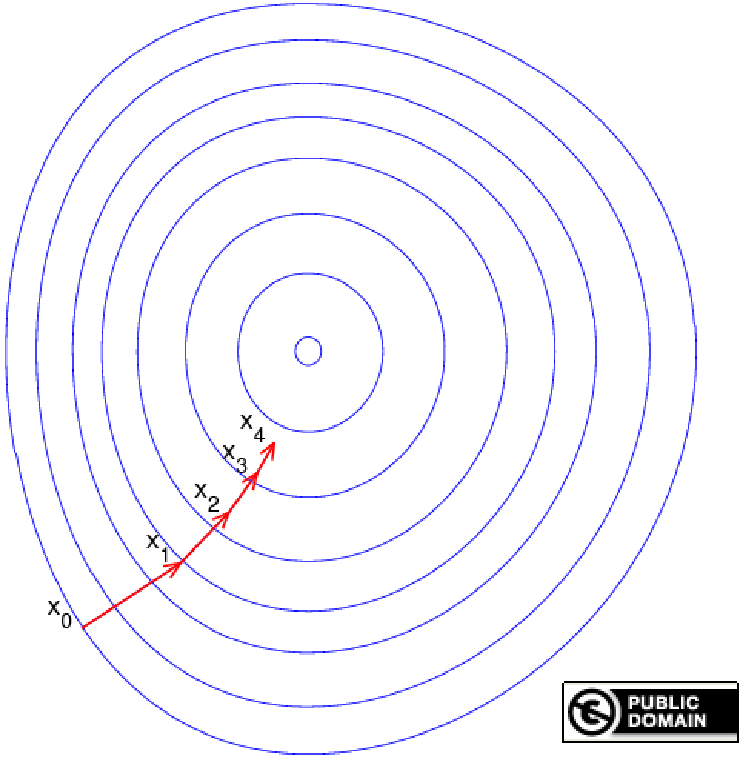
Gradient Descent Algorithm

initialize $x_0 \in \mathbb{R}^n$

At each iteration t :

- compute gradient ∇f
- $x_{t+1} = x_t - \gamma \nabla f(x_t)$

learning rate



Gradient Descent to Optimize

Optimization: $\min_{x \in \mathbb{R}^n} f(x)$

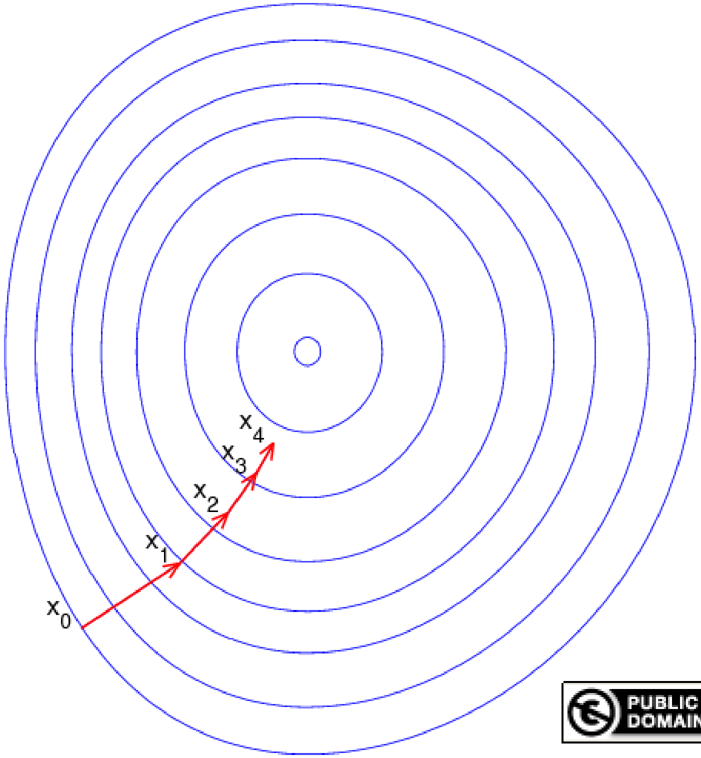
Gradient Descent Algorithm

initialize $x_0 \in \mathbb{R}^n$

At each iteration t :

- compute gradient ∇f
- $x_{t+1} = x_t - \gamma \nabla f(x_t)$

learning rate

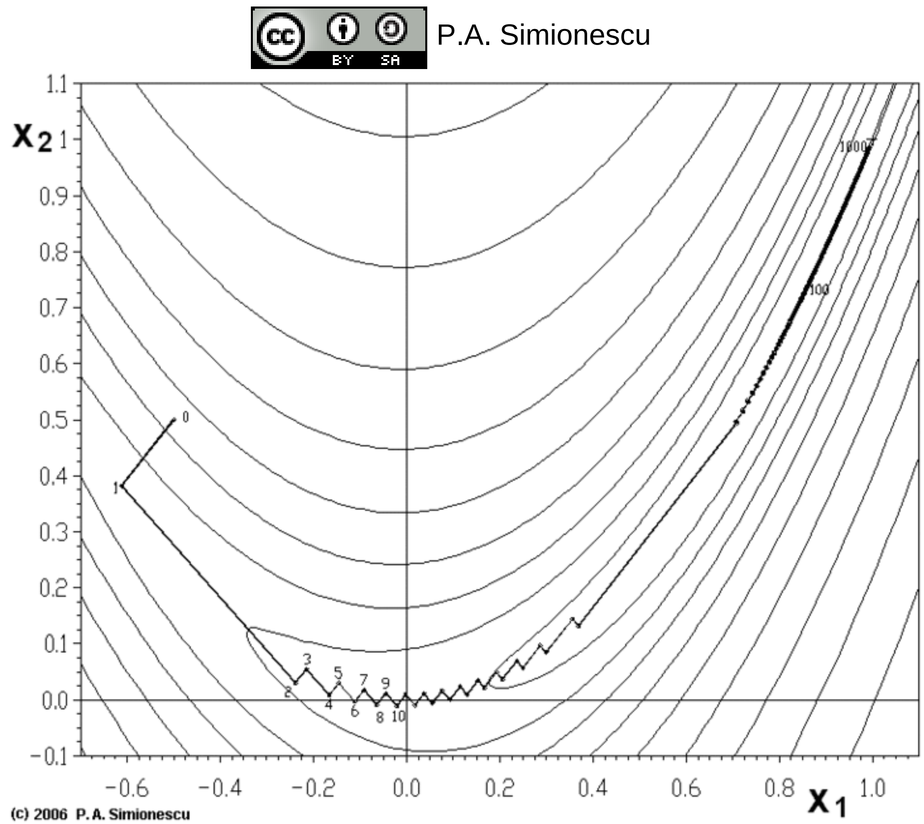


! can be slow close to optimum

other algorithms might be favorable

Gradient Descent to Optimize

Optimization: $\min_{x \in \mathbb{R}^n} f(x)$



Gradient Descent Algorithm

initialize $x_0 \in \mathbb{R}^n$

At each iteration t :

- compute gradient ∇f
- $x_{t+1} = x_t - \gamma \nabla f(x_t)$

learning rate

! can be slow close to optimum

other algorithms might be favorable

(keyword: natural gradient)

Example:
Rosenbrock function

Optimizing Weights in a Layered Network

How to choose the weights in a multi-layered ANN?

Why not optimize weights directly?

$$E(\vec{w}) = \sum_{j=1}^{p}\sum_{k=1}^{m} ||y_{j,k} - d_{j,k}||^2 = \sum_{j=1}^{p}\sum_{k=1}^{m} ||\varphi(\vec{x}_j, \vec{w}) - d_{j,k}||^2$$

$$\vec{w} = \vec{w} - \nabla \left(\sum_{j=1}^{p}\sum_{k=1}^{m} ||\varphi(\vec{x}_p, \vec{w}) - d_{j,k}||^2 \right)$$

since complicated*, better:

- gradient descent after each training sample

  = stochastic gradient descent (SGD, online gradient descent)

$$w = w - \nabla \left(\sum_{k=1}^{m} ||\varphi(\vec{x}_j, \vec{w}) - d_{j,k}||^2 \right)$$

- descent steps can be performed multiple times over the training set (e.g. with random shuffling)

* complicated: difficult analytically, numerically expensive

Optimizing Weights in a Layered Network

The Backpropagation Algorithm

- introduced around 1970, it gave rise to a renaissance of ANNs
- “backwards propagation of errors”
- mainly useful for feed-forward networks
- all transfer functions must be differentiable

Main Idea:

for each training sample:

- compute output of ANN, given the current weights
- compute gradient wrt. weight *on each node* from the output layer *backwards* to the input layer
- update the weights according to gradient descent

→ an efficient stochastic gradient descent by updating all weights at once in a smart way

Optimizing Weights in a Layered Network

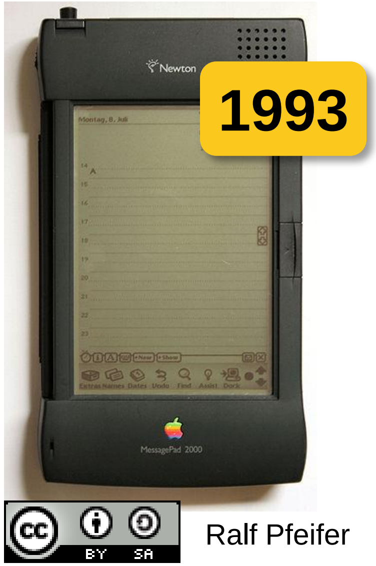
Notes:

- stochastic gradient descent converges to local minimum
- random initial values, restarts
- more about optimization algorithms within the next weeks

Applications of Neural Networks

Many application areas: e.g.

- identification problems
  - face recognition
  - medical diagnoses
  - character recognition in mobile devices
- predictions/forecasting
  - stock market
  - electronic nose
- control



At the end of the course:
exercise using ANNs for
the pole balancing problem

Introduction to
Bio-inspired Optimization
and Genetic Algorithms in particular

General Context Optimization

Given:

set of possible solutions — *Search space*

quality criterion — *Objective / Fitness function*

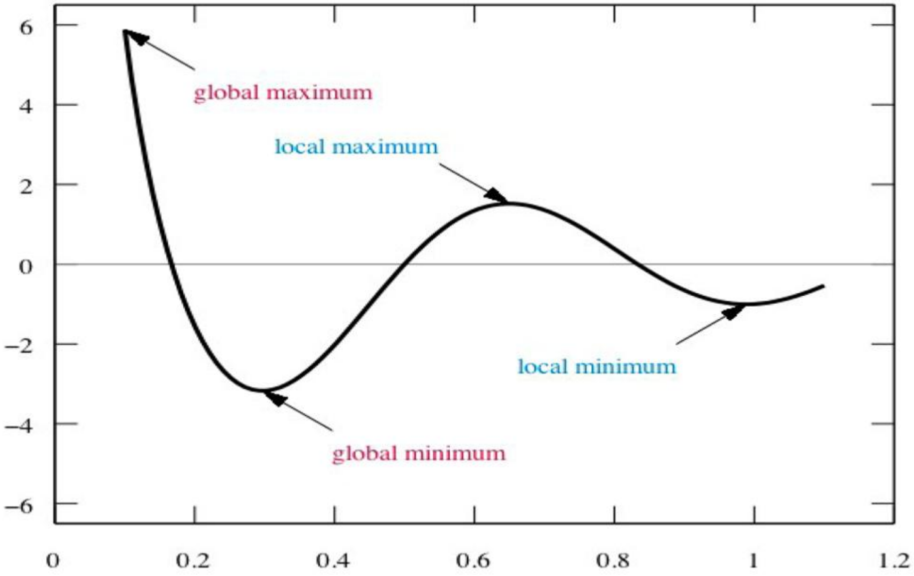
Objective:

Find the best possible solution for the given criterion

Formally:

Maximize or minimize

$$\mathcal{F} : \Omega \mapsto \mathbb{R},$$
$$x \mapsto \mathcal{F}(x)$$



Black Box Scenario

$$x \in \Omega \quad \longrightarrow \quad \boxed{\textbf{black box}} \quad \longrightarrow \quad \mathcal{F}(x) \in \mathbb{R}$$

Why are we interested in a black box scenario?

objective function F often noisy, non-differentiable, or sometimes not even understood or available

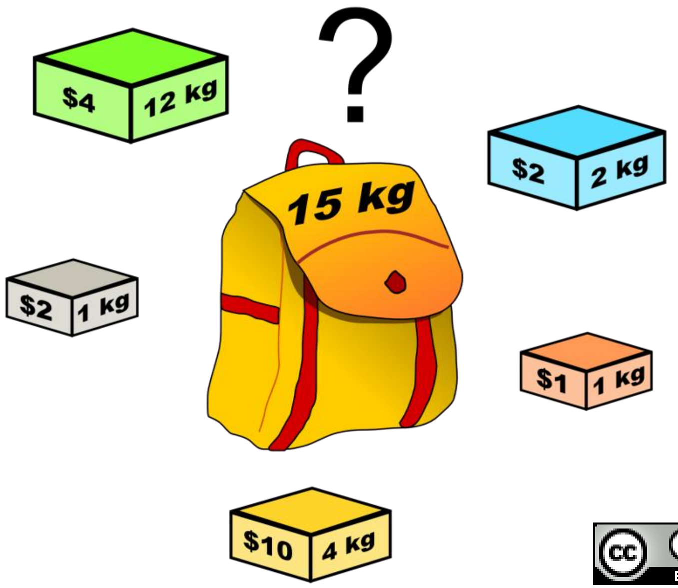
Objective: find x with small F(x) with as few function evaluations as possible

assumption: internal calculations of algo irrelevant

Example 1: Combinatorial Optimization

Knapsack Problem

- Given a set of objects with a given weight and value (profit)
- Find a subset of objects whose overall mass is below a certain limit and maximizing the total value of the objects



[Problem of ressource allocation with financial constraints]

$$\max. \sum_{j=1}^{n} p_j x_j \text{ with } x_j \in \{0,1\}$$

$$\text{s.t.} \sum_{j=1}^{n} w_j x_j \leq W$$

$$\Omega = \{0,1\}^n$$

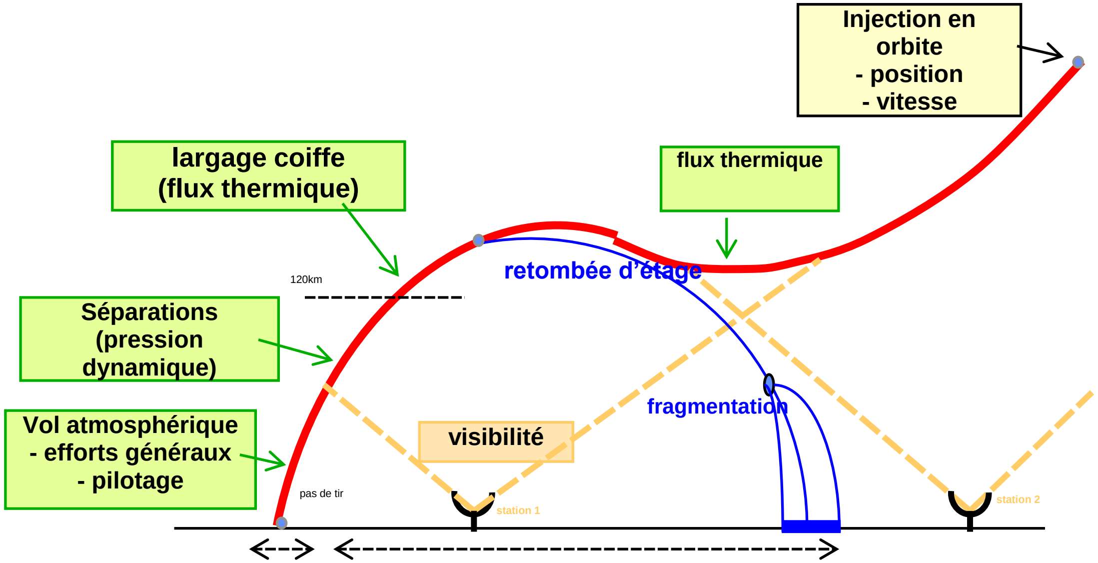
Example 2: Combinatorial Optimization

Travelling Salesperson Problem (TSP)

- Given a set of cities and their distances
- Find the shortest path going through all cities

$$\Omega = S_n \text{ (set of all permutations)}$$

Design of a Launcher



copyright by Astrium

- Scenario: multi-stage launcher brings a satellite into orbit
- Minimize the overall cost of a launch
- Parameters: propellant mass of each stage / diameter of each stage / flux of each engine / parameters of the command law

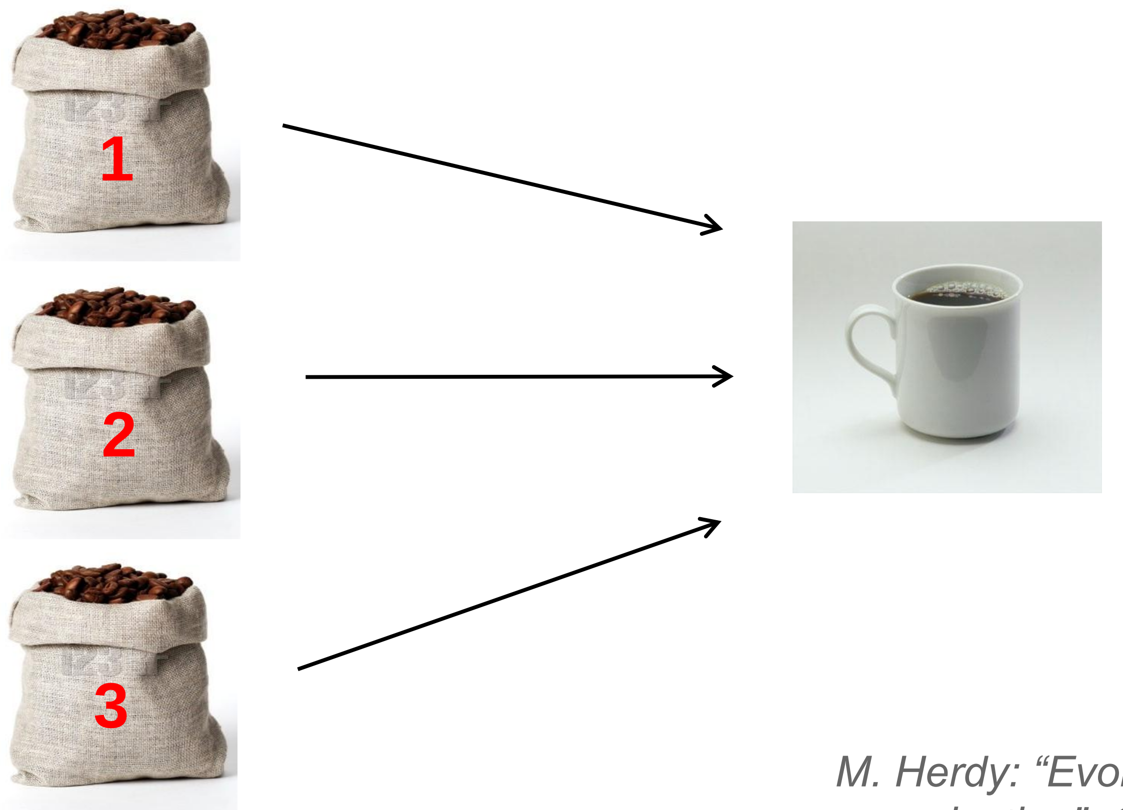
23 continuous parameters to optimize

+ constraints

$$\Omega = \mathbb{R}^{23}$$

Coffee Tasting Problem

- Find a mixture of coffee in order to keep the coffee taste from one year to another
- Objective function = opinion of one expert



M. Herdy: "Evolution Strategies with subjective selection", 1996

What makes an optimization problem difficult?

Why using (bio-inspired) search heuristics?

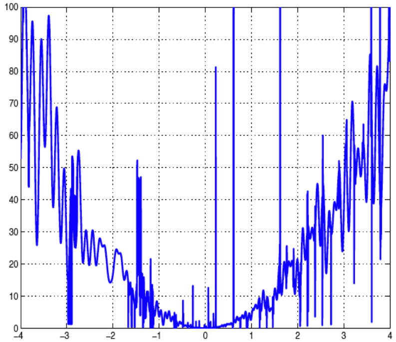
- Search space too large

exhaustive search impossible

- Non conventional objective function or search space

mixed space, function that cannot be computed

- Complex objective function

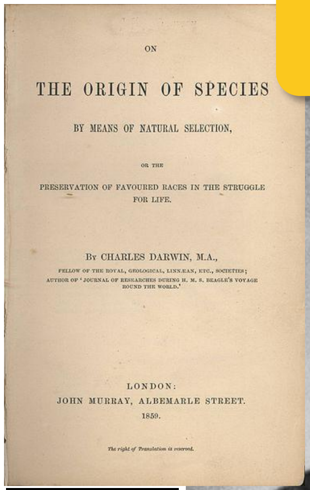


non-smooth, non differentiable, Noisy, ...

Basic Algorithms

One class of bio-inspired stochastic optimization algorithms: Evolutionary Algorithms (EAs)

1859


ON

THE ORIGIN OF SPECIES

BY MEANS OF NATURAL SELECTION,

OR THE

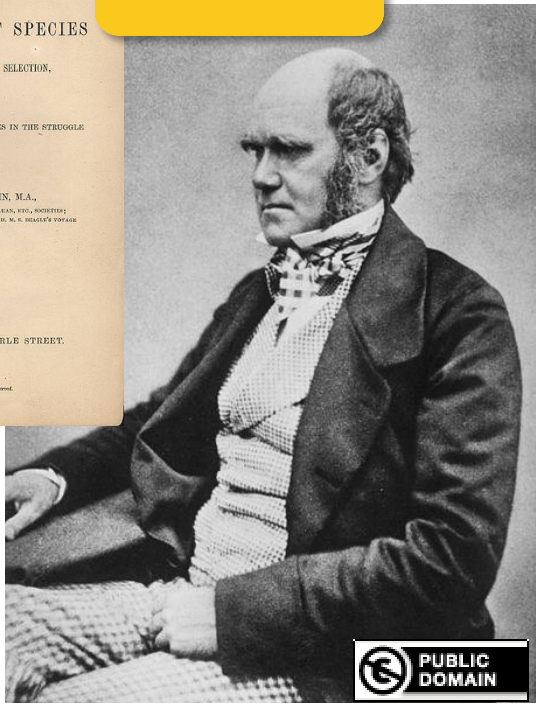
PRESERVATION OF FAVOURED RACES IN THE STRUGGLE FOR LIFE.

BY CHARLES DARWIN, M.A.,

FELLOW OF THE ROYAL, GEOLOGICAL, LINNÆAN, ETC., SOCIETIES;
AUTHOR OF 'JOURNAL OF RESEARCHES DURING H. M. S. BEAGLE'S VOYAGE ROUND THE WORLD.'

LONDON:
JOHN MURRAY, ALBEMARLE STREET.
1859.

The right of Translation is reserved.

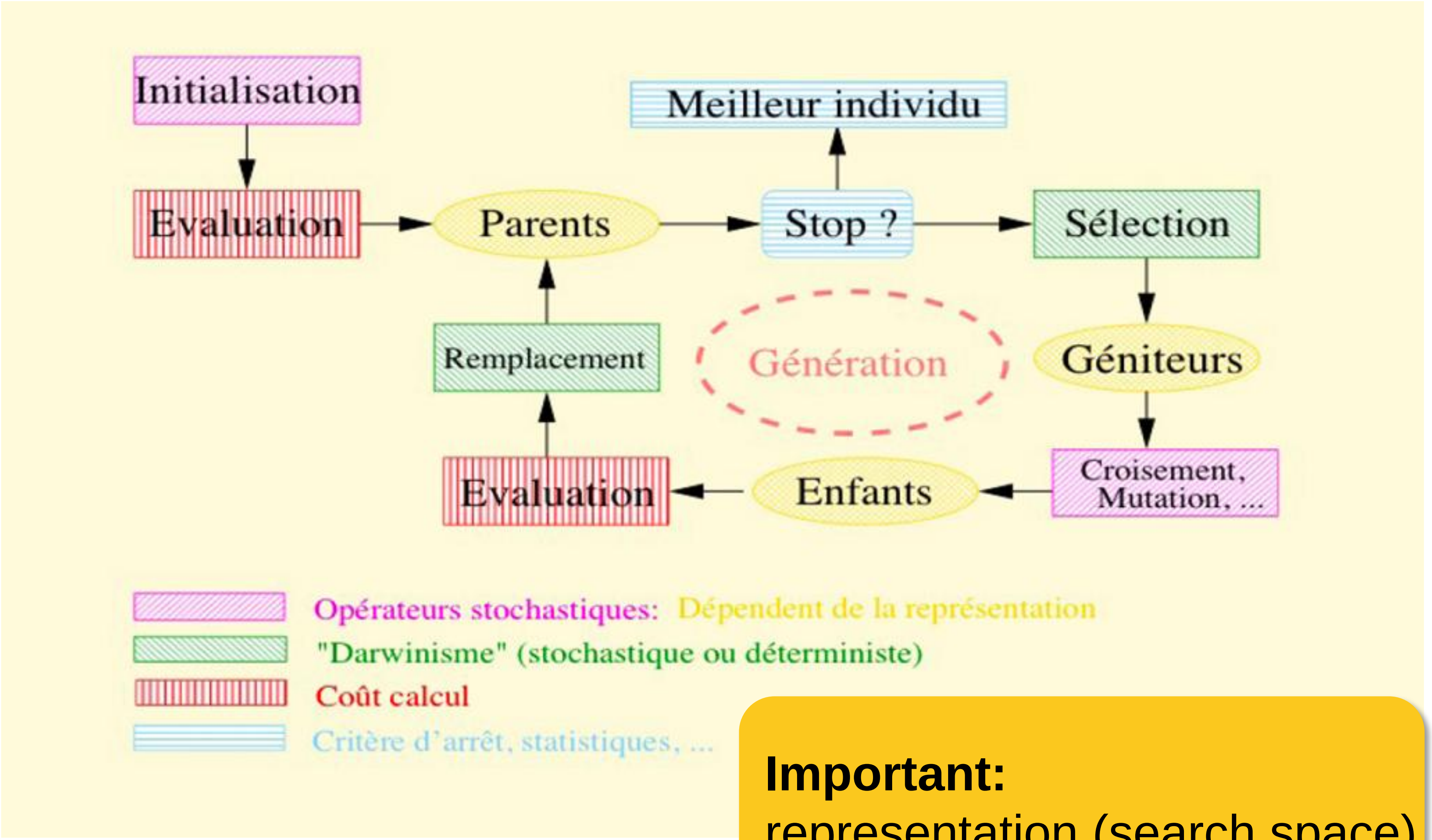


- Class of optimization algorithms inspired by the idea of biological evolution
- selection, mutation, recombination

Metaphors

Classical Optimization	Evolutionary Computation
candidate solution vector of decision variables / design variables / object variables	individual, offspring, parent
set of candidate solutions	population
objective function loss function cost function error function	fitness function
iteration	generation

Generic Framework of an EA



Important:
representation (search space)

The Historic Roots of EAs

Genetic Algorithms (GA)

J. Holland 1975 and D. Goldberg (USA)

$$\Omega = \{0,1\}^n$$

Evolution Strategies (ES)

I. Rechenberg and H.P. Schwefel, 1965 (Berlin)

$$\Omega = \mathbb{R}^n$$

Evolutionary Programming (EP)

L.J. Fogel 1966 (USA)

Genetic Programming (GP)

J. Koza 1990 (USA)

$$\Omega = \text{space of all programs}$$

nowadays one umbrella term: evolutionary algorithms

Examples for some EA parts

Selection

Selection is the major determinant for specifying the trade-off between exploitation and exploration

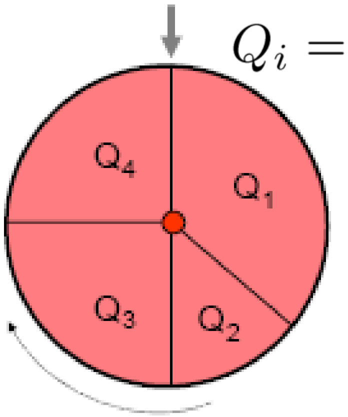
Selection is either

stochastic or deterministic

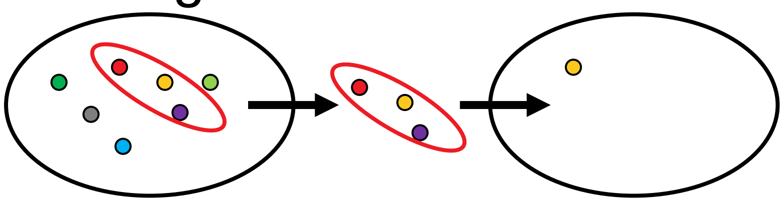
e.g. fitness proportional

$$Q_i = \frac{f(x_i)}{\sum_{j=1}^{\mu} f(x_j)}$$

Disadvantage: depends on scaling of f



e.g. via a tournament



e.g. $(\mu+\lambda)$, (μ,λ)

Mating selection (selection for variation): usually stochastic

Environmental selection (selection for survival): often deterministic

Variation Operators

Variation aims at generating new individuals on the basis of those individuals selected for mating

Variation = Mutation and Recombination/Crossover

mutation: *mut:* $\Omega \to \Omega$

recombination: *recomb:* $\Omega^r \to \Omega^s$ where $r \geq 2$ and $s \geq 1$

- choice always depends on the problem and the chosen representation
- however, there are some operators that are applicable to a wide range of problems and tailored to standard representations such as vectors, permutations, trees, etc.

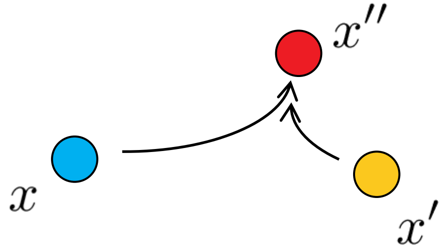
Variation Operators: Guidelines

Two desirable properties for mutation operators:

- every solution can be generated from every other with a probability greater than 0 (“exhaustiveness”)
- $d(x, x') < d(x, x'') => Prob(\mathrm{mut}(x) = x') > Prob(\mathrm{mut}(x) = x'')$ (“locality”)

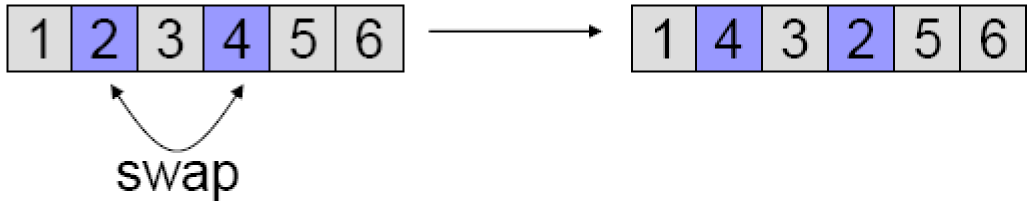
Desirable property of recombination operators (“in-between-ness”):

$$x'' = \ \mathrm{recomb}(x, x') \Rightarrow d(x'', x) \leq d(x, x') \wedge d(x'', x') \leq d(x, x')$$

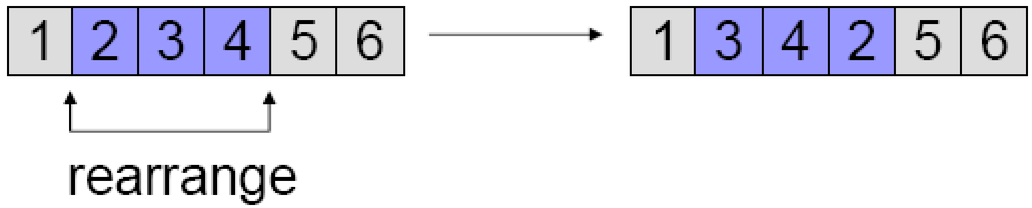


Examples of Mutation Operators on Permutations

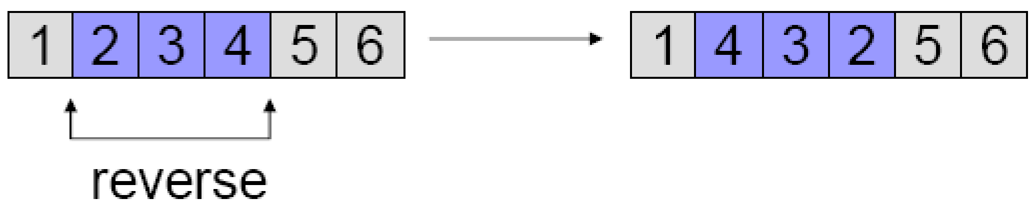
Swap:



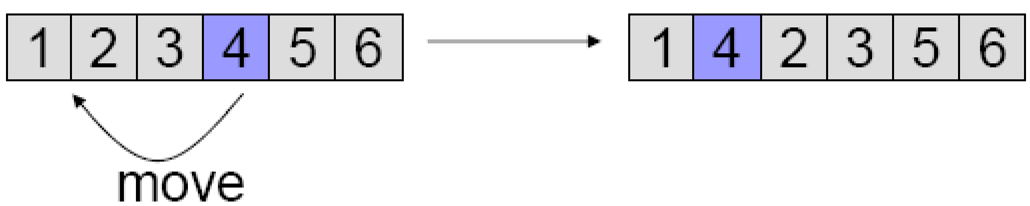
Scramble:



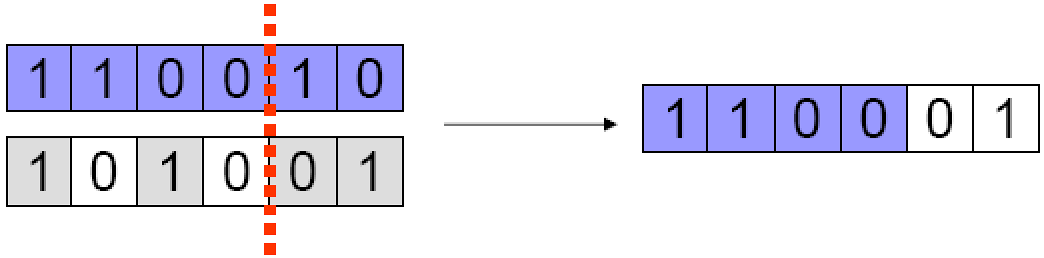
Invert:



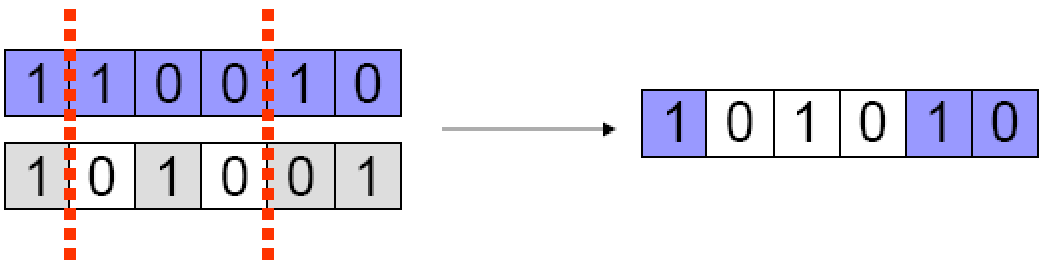
Insert:



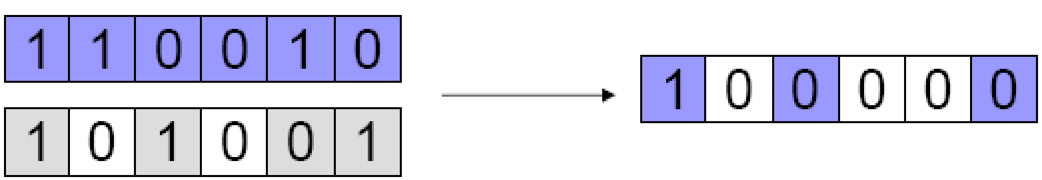
1-point crossover



n-point crossover



uniform crossover



choose each bit independently from one parent or another

A Canonical Genetic Algorithm

- binary search space, maximization
- uniform initialization
- generational cycle: of the population
  - evaluation of solutions
  - mating selection (e.g. roulette wheel)
  - crossover (e.g. 1-point)
  - environmental selection (e.g. plus-selection)

First Conclusions of Introductory Part

- EAs are generic algorithms (randomized search heuristics, meta-heuristics, ...) for black box optimization

  no or almost no assumptions on the objective function

- They are typically less efficient than problem-specific (exact) algorithms (in terms of #funevals)

  not the case in the continuous case (we will see later)

- Allow for an easy and rapid implementation and therefore to find good solutions fast

  easy to incorporate (and recommended!) to incorporate problem-specific knowledge to improve the algorithm

Exercise:
Pure Random Search
and the (1+1)EA

http://researchers.lille.inria.fr/~brockhof/advancedcontrol/

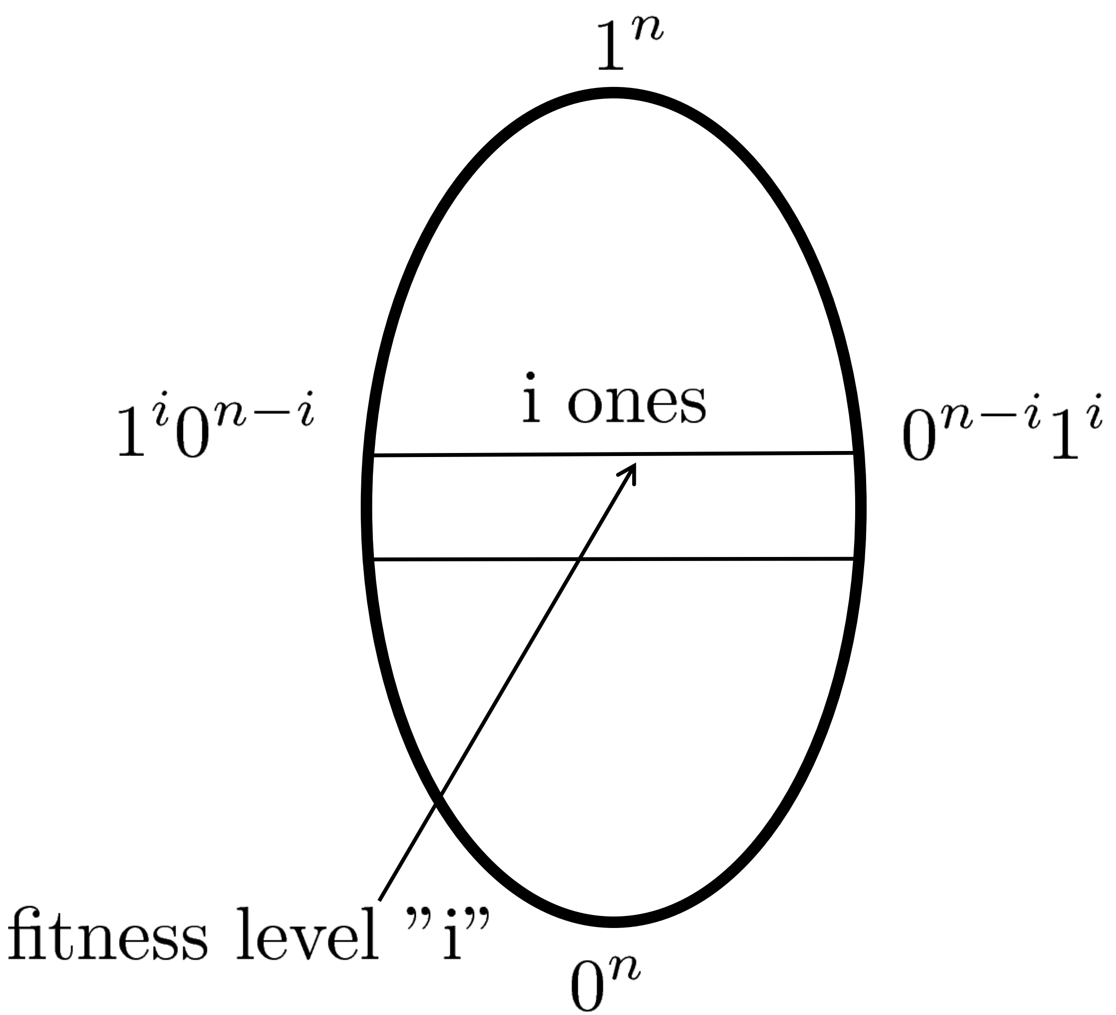
Proof Technique Fitness-based Partitions

$$\Omega = \{0,1\}^n$$

$$f(x) = \sum_{i=1}^{n} x_i$$

$$P_i = \{x | f(x) = i\}$$

$$\Omega = \bigcup_{i=0}^{n} P_i$$



Upper Runtime Bound for (1+1)EA on ONEMAX

$$T = \inf\{t \in \mathbb{N}, X_t = (1, \ldots, 1)\}$$

X_t estimate of solution at iteration t
T_i time to leave fitness level "i"

$$E(T) \leq \sum_{i=0}^{n-1} E(T_i)$$

$\mathrm{Prob}(\mathrm{leave}P_i) \geq \frac{1}{n}(1-\frac{1}{n})^{n-1} \times (n-i)$ (proba to flip one and only one of the $(n-i)$ remaining 0)

$$(1-\frac{1}{n})^{n-1} \geq \frac{1}{e}$$

$\mathrm{Prob}(\mathrm{leave}P_i) \geq \frac{n-i}{en}$

Upper Runtime Bound for (1+1)EA on ONEMAX

$$E(T_i) \leq \frac{en}{n-i}$$

$$E(T) \leq \sum_{i=0}^{n-1} \frac{en}{n-i} \leq e\, n(\log n + 1)$$