

# Algorithmes évolutionnistes

## Optimisation stochastique à base de population

UE méthodes et outils pour l'IA et la RO

L3 informatique

Nicolas Bredeche

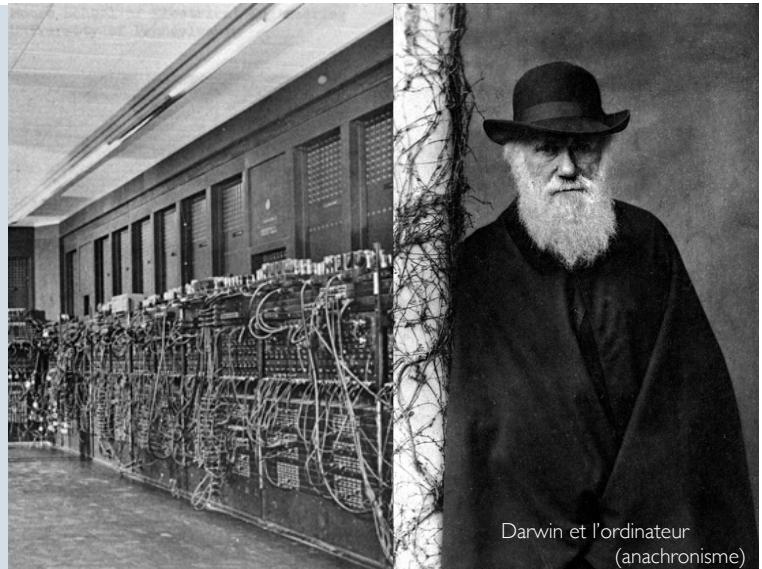
Sorbonne Université

ISIR, UMR 7222

Paris, France

[nicolas.bredeche@upmc.fr](mailto:nicolas.bredeche@upmc.fr)

...



Darwin et l'ordinateur  
(anachronisme)



rev. 2018-03-29

## Préambule

2

Documents autorisés pour l'examen:  
une seule et unique feuille A4 recto-verso manuscrite

mise à jour de multirobot\_muliwars.py

nicolas.bredeche@upmc.fr

## Optimisation: éléments

Problème:  $y^* = \arg \min_{y \in Y} f(y)$

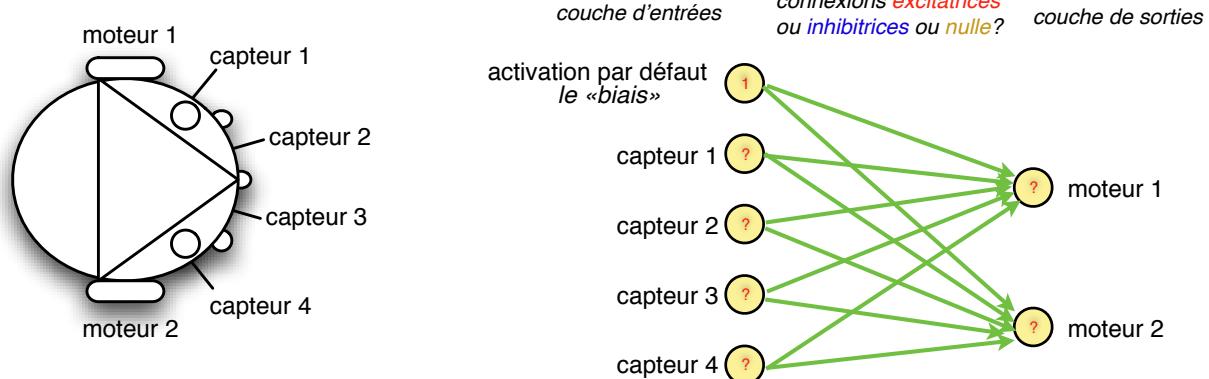
Solution candidate:  $a := (y, f(y))$

Formulation simplifiée

- Des méthodes pour des classes de problèmes
  - ▶ Algorithme de gradient (recherche locale, suit le gradient)
  - ▶ Hill-climbing (recherche locale, change un élément à la fois)
  - ▶ Méthodes énumératives (recherche globale, espace de recherche discret)
  - ▶ Méthodes heuristiques (espace structuré)
  - ▶ Méta-heuristique et méthodes stochastiques
    - recherche aléatoire (recherche globale, sans a priori) [Monte carlo, Tabu]
    - recuit simulé (recherche globale) ["simulated annealing"]
    - méthodes bio-inspirées (recherche globale) [DE, PSO, Algo. évolutif, ...]

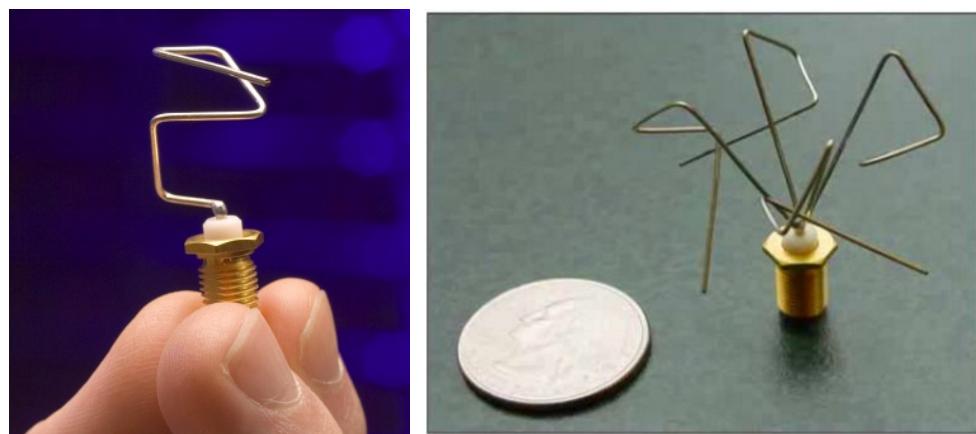
## Recherche aléatoire

6



- Fonction objectif:
  - ▶  $f(y) = \text{distance parcourue pendant } n \text{ itération}$
- Description d'une solution:
  - ▶  $\{-1, 0, 1\}^{10}$ , soit:  $[x_1, x_2, \dots, x_{10}]$ , avec  $x_i \in \{-1, 0, +1\}$
- Algorithme:
  - ▶ recherche aléatoire

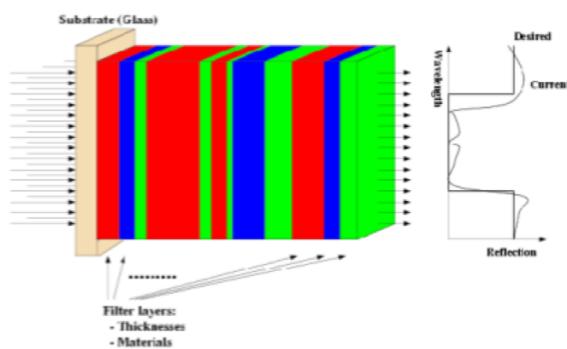
**exemple pratique**  
robot\_randomsearch.py



- Antenne de petite taille pour satellite
  - Objectif: maximisez l'efficacité, minimiser la taille
  - Espace de recherche: structure de l'antenne (graphe)
  - Difficulté: absence de modèles d'interférences

Source: Lohn et al., Genetic Programming Theory and Practice II, 2004

## Filtre optique



- Définir les couches d'un filtre optique
  - Objectif: contraintes fixées par un cahier des charges
  - Espace de recherche: (matériau, épaisseur)<sup>n</sup>
  - Difficulté: espace de recherche discret et continu



crédit photo: Chung-Leng Tran

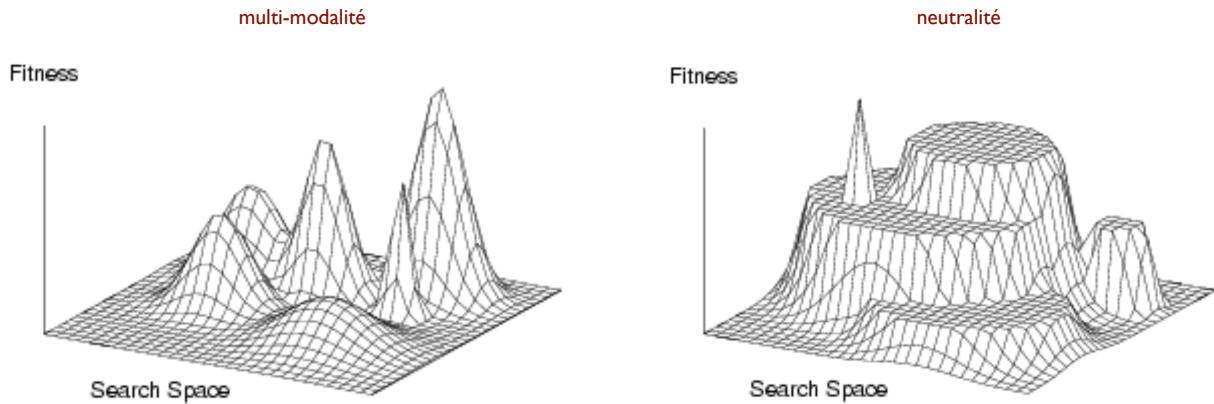
- Retrouver la composition d'un café à l'arôme
  - Objectif: maximiser la satisfaction de l'expert-évaluateur
  - Espace de recherche: mélanges de café
  - Difficulté: évaluation empirique par l'expert

[Herdy, 1997]

## Définition de la classe de problèmes

10

- Propriétés
  - Espace de recherche
    - ▶ binaire, symbolique, continu
    - ▶ structuré ou non
  - Fonction de performance
    - ▶ lien ténu entre représentation et performance
    - ▶ évaluation potentiellement bruitée
  - Relation faible entre espace de recherche et objectif



- espace de recherche “complexe”
  - ▶ multi-modalité, plateaux de neutralité
  - ▶ irrégulière, non différentiable, discontinue
- espace de recherche inconnu
  - ▶ problème type “boîte noire”
  - ▶ MAIS: notion de voisinage (sinon, il ne reste plus que Monte Carlo)

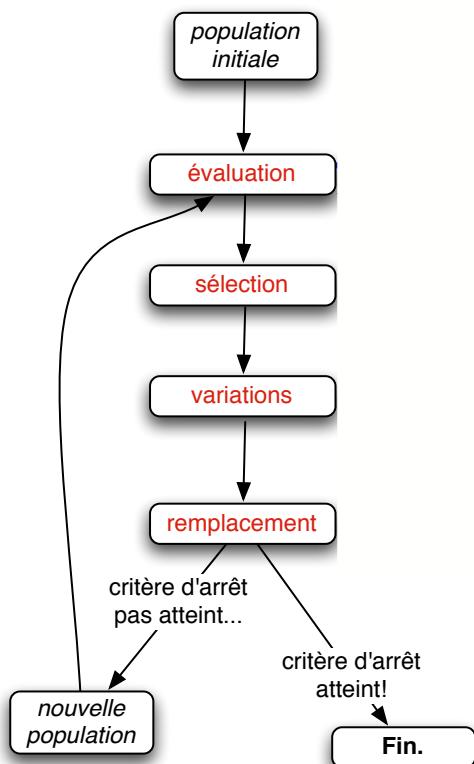
Images “fitness landscape”: Sébastien Verel

## Algorithmes évolutionnistes\*

### Principes généraux

\*Ou: Algorithmes évolutionnaires

\*Ou: Algorithmes pour l'optimisation stochastique à base de population



- Caractérisation
  - Algorithme d'optimisation stochastique à base de population
  - Famille des méta-heuristiques
- Deux mécanismes principaux
  - Pression à la sélection
  - Variations +/- aveugles
- Propriété souhaitée
  - Recherche globale et locale

## Opérateurs de sélection

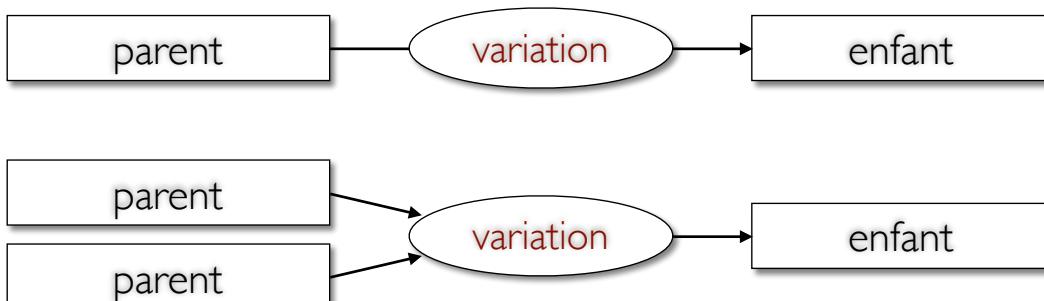


- Définition
  - ▶ Sélectionne une sous-partie des solutions candidates
- Exemple
  - ▶ Renvoie les  $N$  meilleurs individus parmi  $M$
- Propriétés
  - ▶ Déterministe vs. stochastique
  - ▶ Compromis exploration/exploitation
  - ▶ Elitiste ou non



- Méthodes
  - ▶ fitness proportionate
  - ▶ *k*-tournament
  - ▶ plus-selection ( $\mu+\lambda$ )
  - ▶ comma-selection ( $\mu,\lambda$ )
  - ▶ NSGA-2 (multi-objectif)

## Opérateurs de variation



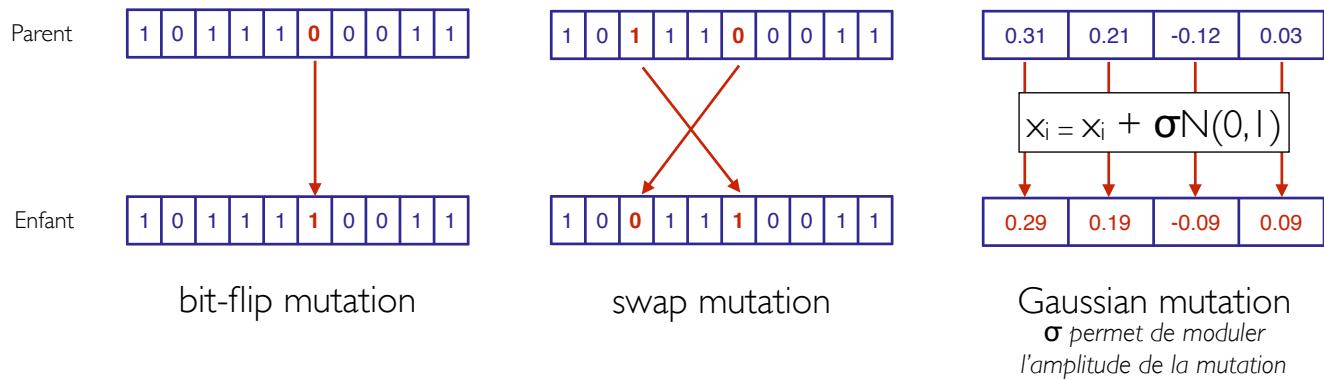
- Définition
  - ▶ Construit un nouvel individu à partir d'un (ou plusieurs) individus
- Exemple
  - ▶ Modifie aléatoirement un élément du génome
- Propriétés
  - ▶ Conservatif vs. disruptif



- Définition

- ▶ Construit un nouvel individu à partir d'un seul individu parent

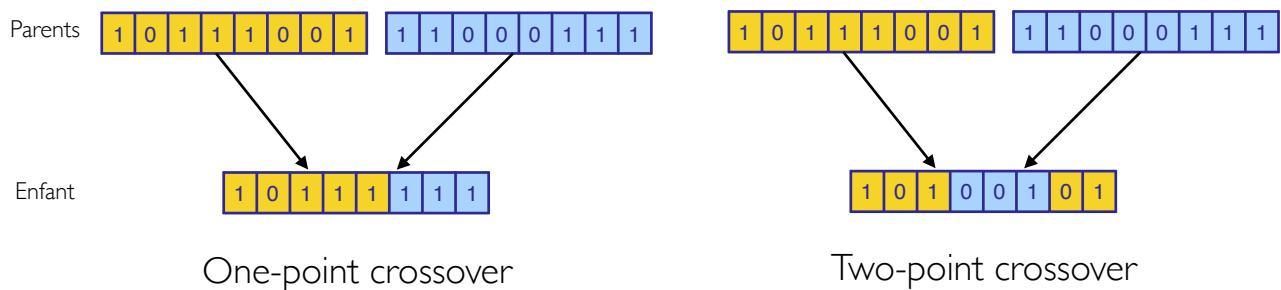
- Exemples



- Définition

- ▶ Construit un nouvel individu à partir de 2 (ou +) individus parents

- Exemples:

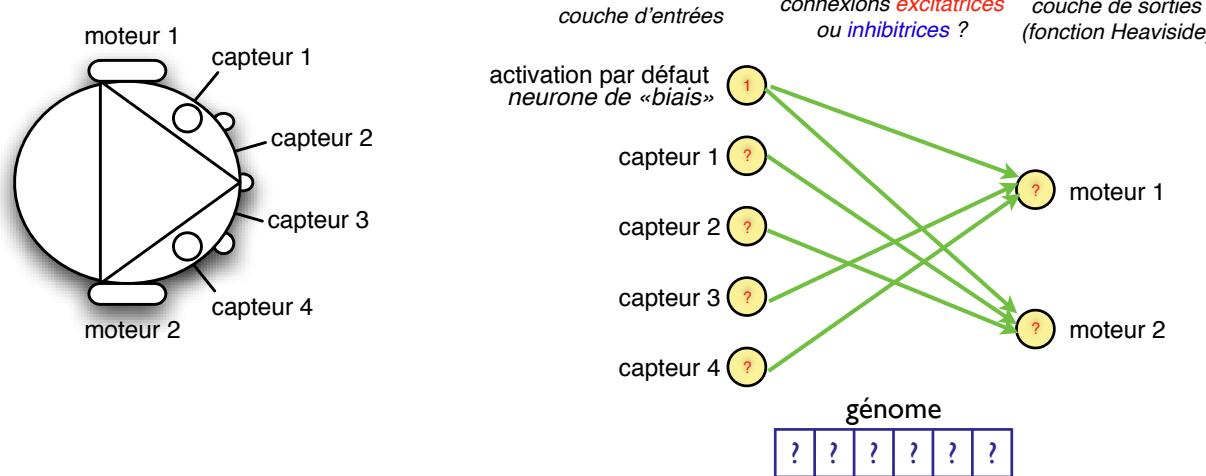


# Cas d'étude

Robotique évolutionniste

Optimiser un robot éviteur d'obstacle

20



$$fitness(x) = \int_T V * (1 - \sqrt{\delta_v}) * (1 - sensor_{max})$$

- ➡ maximiser la vitesse de chaque moteur
- ➡ maximiser la vitesse de translation
- ➡ maximiser la distance au mur

## Problème : trouver le meilleur éviteur d'obstacles

il s'agit d'un problème jouet pour lequel on suppose ici que [1,1,1,1,-1,-1] est la solution optimale

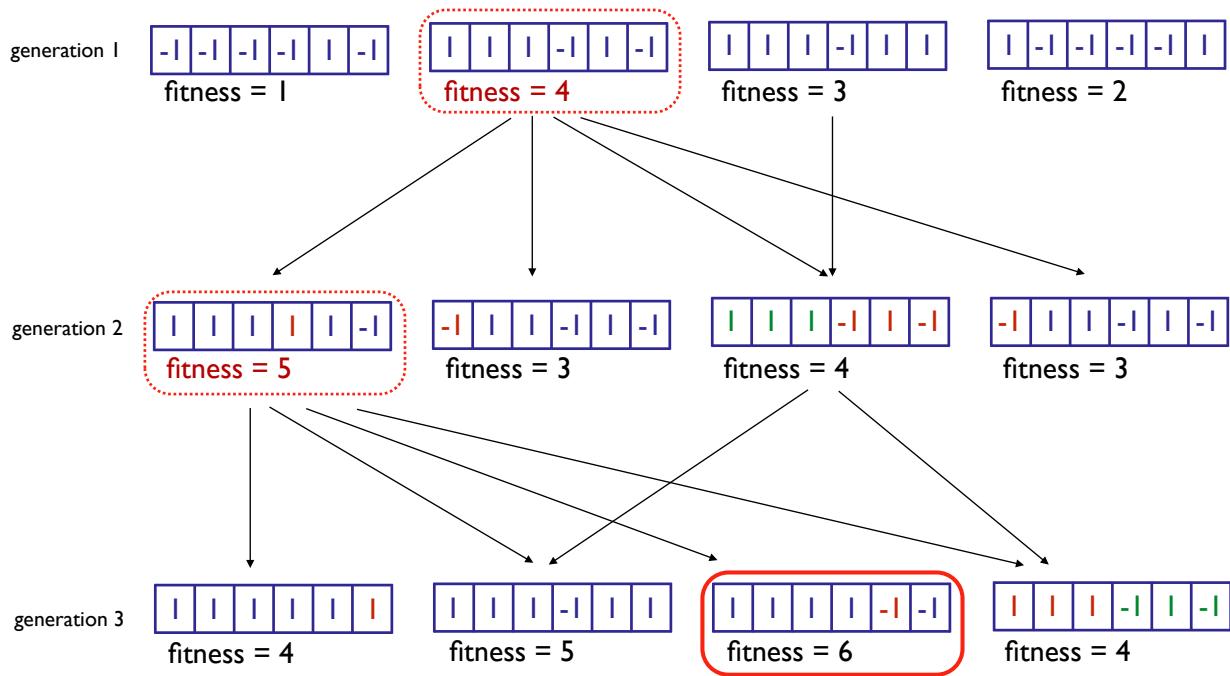
- **Objectif** : maximiser la fonction fitness
- **Population initiale** : 4 individus tirés au hasard
- **Opérateur de Sélection** : prend le meilleur
- **Opérateurs de Variation** : croisement ou mutation
  - Probabilité de croisement: p ; probabilité de mutation: 1-p ; avec p=0.5
  - Croisement: on mélange le début d'un génome et la fin d'un second
  - Mutation: on change une valeur au hasard

Remarques:

1. les opérateurs sont ici choisis arbitrairement. D'autres choix sont possibles.
2. tout aussi arbitrairement, on décide ici que la performance est donnée par le nombre de valeur du génome correctes (ie. [1,1,1,1,-1,-1] vaut 6). En réalité, la performance résulte du comportement du robot.

nicolas.bredeche@upmc.fr random generator :<0.1 0.3 0.6 0.1 0.7 0.3 0.9 0.9 0.7 0.1 0.9 0.2 0.4 0.1 0.9 0.5 0.6 0.9>

## déroulement de l'algorithme



Solution!

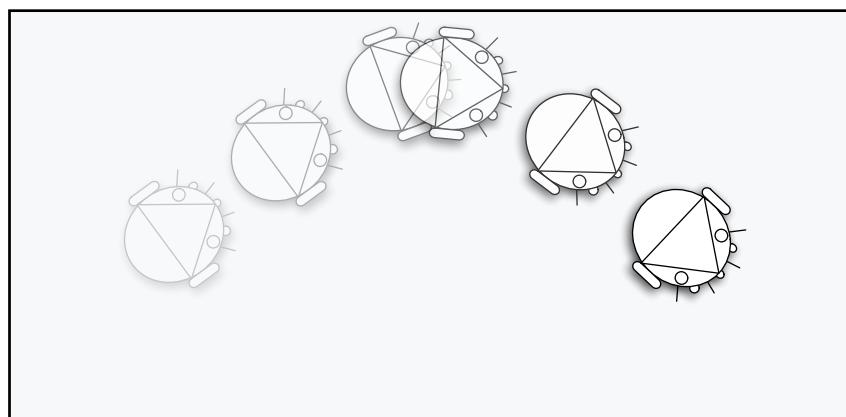
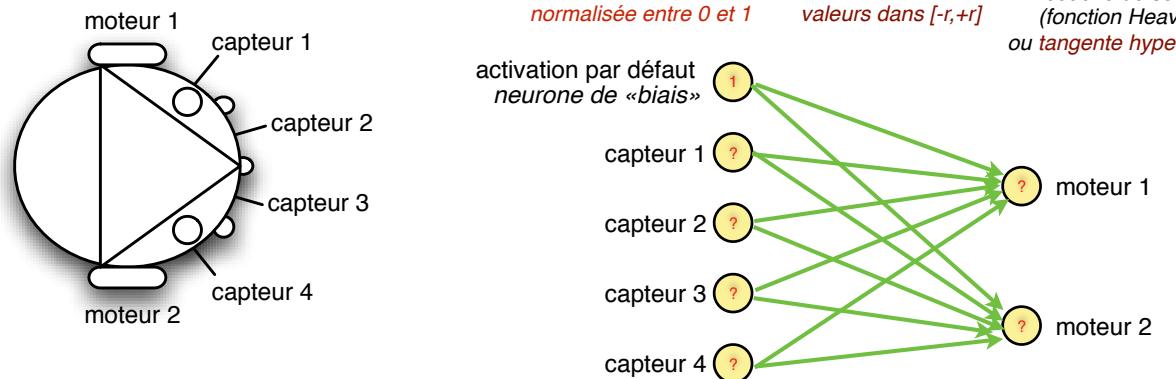
**exemple pratique**  
geneticalgorithm\_maxone.ipynb

- **Objectif** : maximiser la correspondance entre une chaîne de bits et une autre (ex.: [1,1,1,1,1,1,...,1])
- **Représentation d'un individu** :  $\{0,1\}^m$
- **Population initiale** : N individus
- **Opérateur de sélection** : par tournoi de taille k
- **Opérateurs de variation** : mutation bit-flip (probabilité/bits)

en rouge, les paramètres à régler

nicolas.bredeche@upmc.fr

## robot-éviteur



cf.TME

nicolas.bredeche@upmc.fr

## Algorithme (1+1)-ES naïf

```

init:  $x \in \mathbb{R}^n$ 
init:  $\sigma > 0$ 

for  $i$  in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ )  $\geq$  fitness( $x$ ):
         $x = x'$ 

```

nicolas.bredeche@upmc.fr

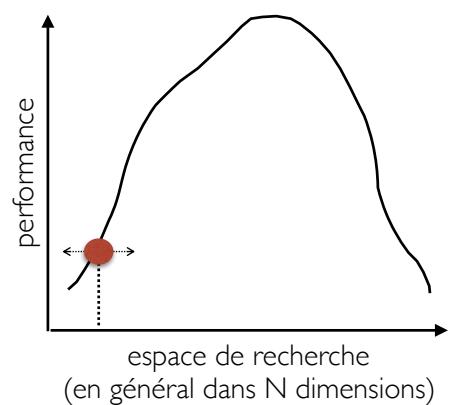
## Algorithme (1+1)-ES naïf

```

init:  $x \in \mathbb{R}^n$ 
init:  $\sigma > 0$ 

for  $i$  in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ )  $\geq$  fitness( $x$ ):
         $x = x'$ 

```



nicolas.bredeche@upmc.fr

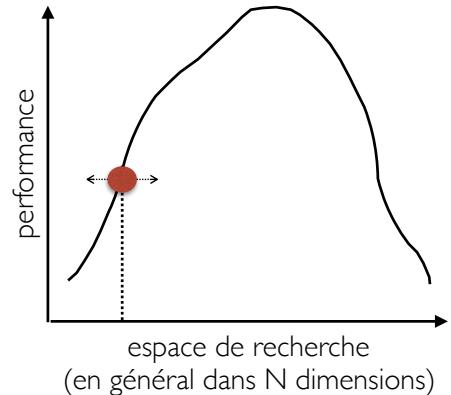
## Algorithme (1+1)-ES naïf

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ ) ≥ fitness( $x$ ):
         $x = x'$ 
```

convergence lente  
sigma trop petit



nicolas.bredeche@upmc.fr

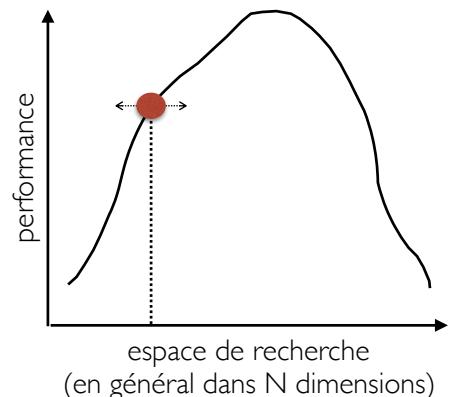
## Algorithme (1+1)-ES naïf

convergence lente  
sigma trop petit

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ ) ≥ fitness( $x$ ):
         $x = x'$ 
```



nicolas.bredeche@upmc.fr

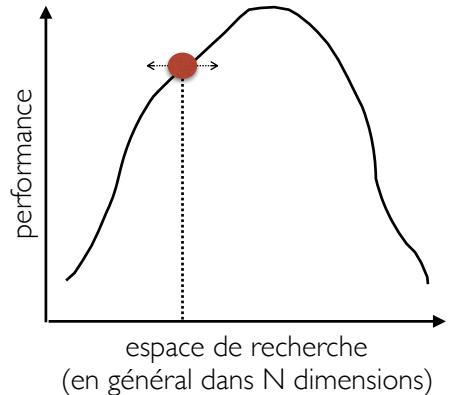
## Algorithme (1+1)-ES naïf

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ ) ≥ fitness( $x$ ):
         $x = x'$ 
```

convergence lente  
sigma trop petit



nicolas.bredeche@upmc.fr

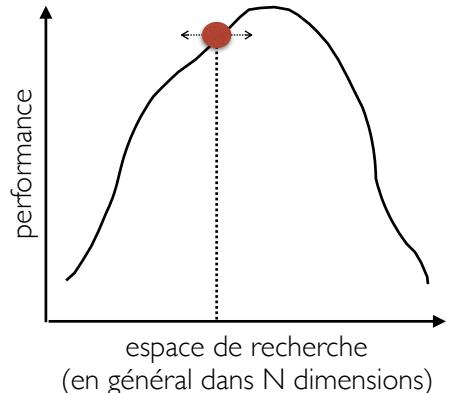
## Algorithme (1+1)-ES naïf

convergence lente  
sigma trop petit

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ ) ≥ fitness( $x$ ):
         $x = x'$ 
```



nicolas.bredeche@upmc.fr

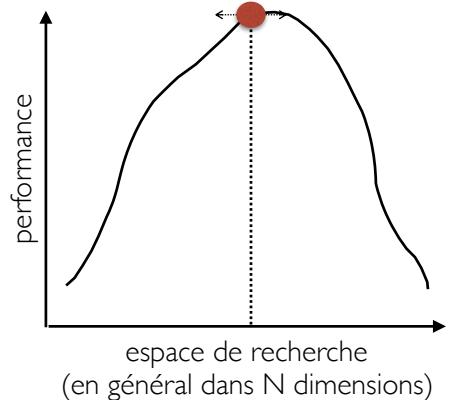
## Algorithme (1+1)-ES naïf

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ ) ≥ fitness( $x$ ):
         $x = x'$ 
```

convergence lente  
sigma trop petit



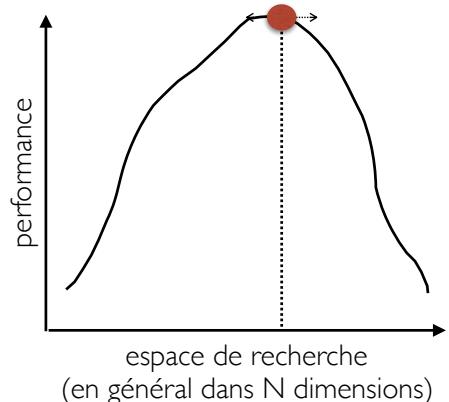
nicolas.bredeche@upmc.fr

## Algorithme (1+1)-ES naïf

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ ) ≥ fitness( $x$ ):
         $x = x'$ 
```



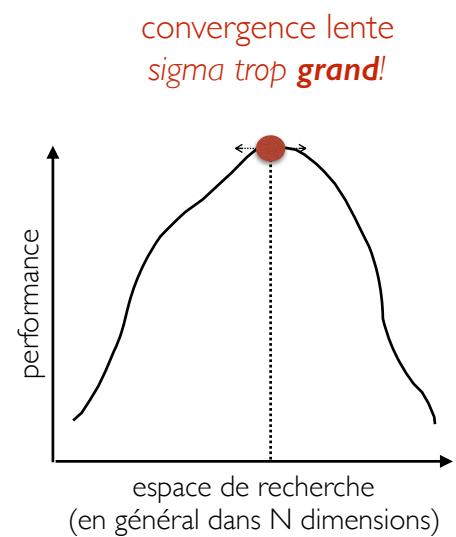
nicolas.bredeche@upmc.fr

## Algorithme (1+1)-ES naïf

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ ) ≥ fitness( $x$ ):
         $x = x'$ 
```



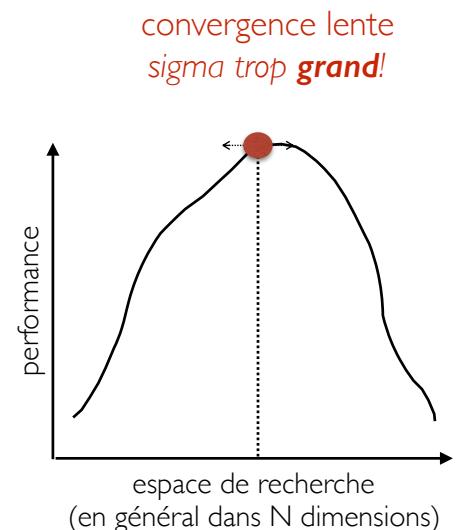
nicolas.bredeche@upmc.fr

## Algorithme (1+1)-ES naïf

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ ) ≥ fitness( $x$ ):
         $x = x'$ 
```



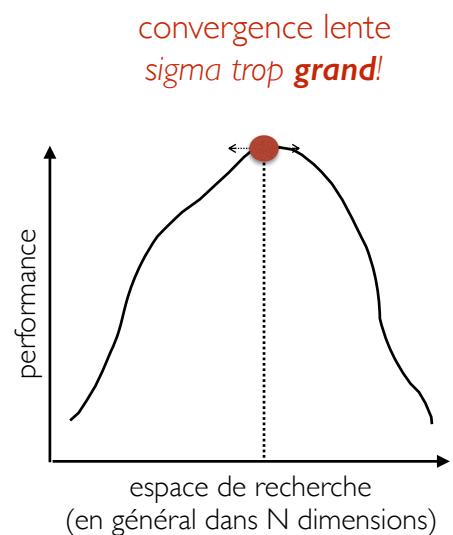
nicolas.bredeche@upmc.fr

## Algorithme (1+1)-ES naïf

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ ) ≥ fitness( $x$ ):
         $x = x'$ 
```



**oscille autour du maximum avant de l'atteindre**

nicolas.bredeche@upmc.fr

## Algorithme (1+1)-ES, règle des 1/5e

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ ) ≥ fitness( $x$ ):
         $x = x'$ 
         $\sigma = 2 * \sigma$ 
    else:
         $\sigma = 2^{-1/4} * \sigma$ 
```

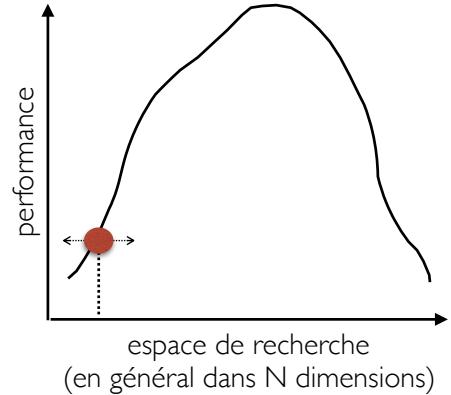
nicolas.bredeche@upmc.fr

## Algorithme (1+1)-ES, règle des 1/5e

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ )  $\geq$  fitness( $x$ ):
         $x = x'$ 
         $\sigma = 2 * \sigma$ 
    else:
         $\sigma = 2^{-1/4} * \sigma$ 
```



nicolas.bredeche@upmc.fr

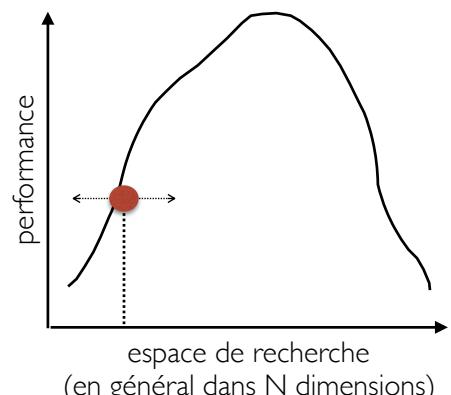
## Algorithme (1+1)-ES, règle des 1/5e

```
init:  $x \in \mathbb{R}^n$ 
```

sigma augmente!

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ )  $\geq$  fitness( $x$ ):
         $x = x'$ 
         $\sigma = 2 * \sigma$ 
    else:
         $\sigma = 2^{-1/4} * \sigma$ 
```



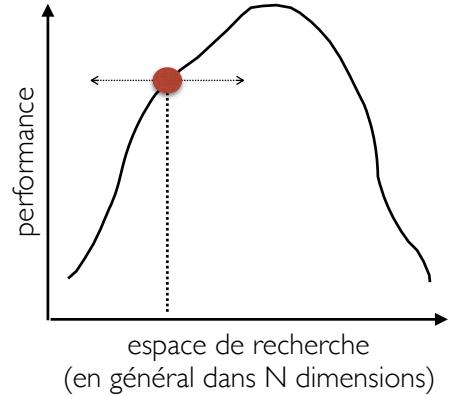
nicolas.bredeche@upmc.fr

## Algorithme (1+1)-ES, règle des 1/5e

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ )  $\geq$  fitness( $x$ ):
         $x = x'$ 
         $\sigma = 2 * \sigma$ 
    else:
         $\sigma = 2^{-1/4} * \sigma$ 
```



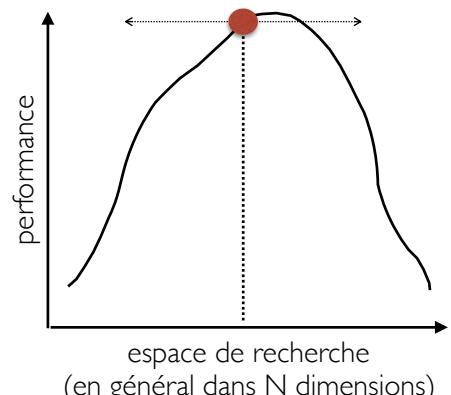
nicolas.bredeche@upmc.fr

## Algorithme (1+1)-ES, règle des 1/5e

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ )  $\geq$  fitness( $x$ ):
         $x = x'$ 
         $\sigma = 2 * \sigma$ 
    else:
         $\sigma = 2^{-1/4} * \sigma$ 
```



nicolas.bredeche@upmc.fr

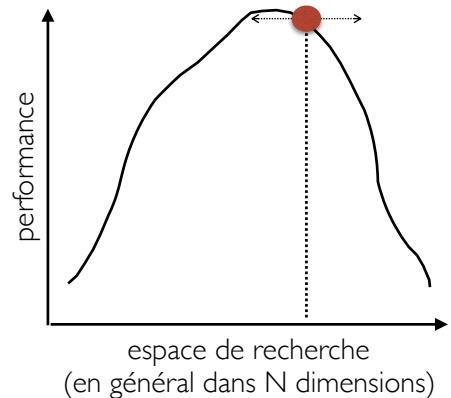
## Algorithme (1+1)-ES, règle des 1/5e

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ )  $\geq$  fitness( $x$ ):
         $x = x'$ 
         $\sigma = 2 * \sigma$ 
    else:
         $\sigma = 2^{-1/4} * \sigma$ 
```

sigma diminue!



nicolas.bredeche@upmc.fr

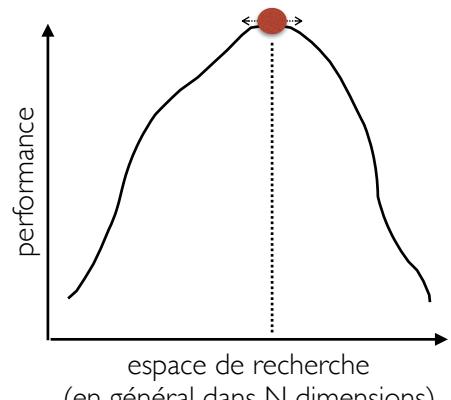
## Algorithme (1+1)-ES, règle des 1/5e

```
init:  $x \in \mathbb{R}^n$ 
```

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ )  $\geq$  fitness( $x$ ):
         $x = x'$ 
         $\sigma = 2 * \sigma$ 
    else:
         $\sigma = 2^{-1/4} * \sigma$ 
```

sigma diminue!



nicolas.bredeche@upmc.fr

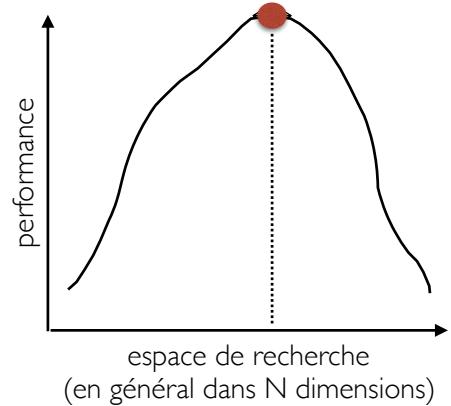
## Algorithme (1+1)-ES, règle des 1/5e

```
init:  $x \in \mathbb{R}^n$ 
```

sigma diminue!

```
init:  $\sigma > 0$ 
```

```
for i in range(evaluations):
     $x' = x + \sigma N(0, I)$ 
    if fitness( $x'$ )  $\geq$  fitness( $x$ ):
         $x = x'$ 
         $\sigma = 2 * \sigma$ 
    else:
         $\sigma = 2^{-1/4} * \sigma$ 
```



**convergence adaptative**

nicolas.bredeche@upmc.fr

# Famille d'algorithmes

Transparents facultatifs

- Points communs

- ▶ Opérateurs de sélection et remplacement
- ▶ Questionnement sur le compromis exploration-exploitation
  - trop d'exploitation: convergence prématuée
  - trop d'exploration: pas de convergence du tout

- Différences

- ▶ Représentations
  - bits, symboles, entiers, réels, arbres, graphes
- ▶ Opérateurs manipulant ces représentations
  - opérateurs unaires (ex.: *mutation*), n-aire (ex.: *recombinaison*)

## Evolution Artificielle (“EC”)

46

algorithmes évolutionniste/évolutionnaire -- “evolutionary computation”

- **Algorithmes génétiques**

- ▶ Holland, 1975 (IA et Biologie)
- ▶ représentation: chaînes de bits

- **Stratégies d'évolution**

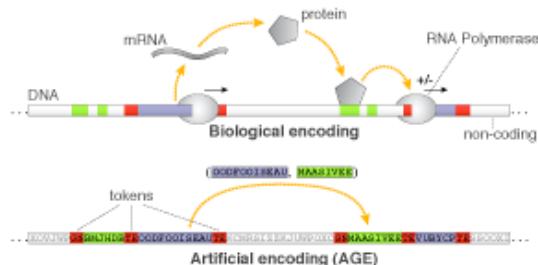
- ▶ Rechenberg, Schwefel, 1965 (Math Appl.)
- ▶ représentation: vecteur de réels

- **Programmation évolutionnaire**

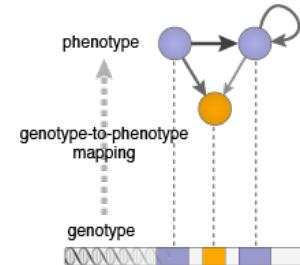
- ▶ Fogel, 1966
- ▶ représentation: Automates

- **Programmation génétique**

- ▶ Koza, 1992
- ▶ représentation: arbre, DAG, graphe, AdF



Génome: une séquence  
de lettres (A-Z)



Réécriture sous la forme d'un  
graphe (p.ex. réseau de neurones)

nicolas.bredeche@upmc.fr

[Mattiussi et al. 2007]

## Stratégies d'évolution ("ES")

- Représentation
  - ▶ Vecteur de réels
- Opérateurs
  - ▶ Mutation gaussienne
  - ▶ Taux de mutation
    - Heuristique (ex.: (1+1) avec la règle des 1/5<sup>ème</sup>)
    - Auto-adaptatif
- Applications:
  - ▶ Optimisation paramétrique (fonctions, modèles)
  - ▶ Poids d'un réseau de neurones

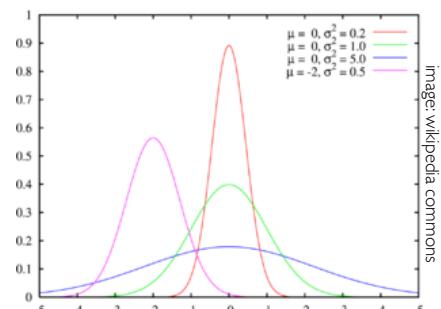
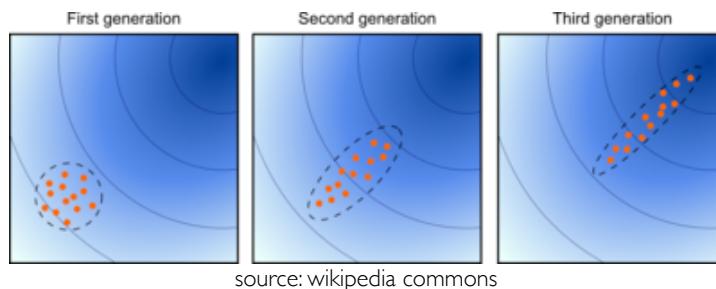


image: wikipedia commons

[Rechenberg, Schwefel, 1965]



- “Co-variance Matrix Adaptation ES”
  - ▶ path length control (cumulative step-size adaptation, CSA)
  - ▶ Matrice de covariance (estimation de la hessienne)
- Caractéristiques
  - ▶ recherche locale et globale
  - ▶ invariance p/r aux rotations/translations
  - ▶ pas/peu de paramètres à régler
  - ▶ nombre de dimensions: de 3 à 100+

cf. <http://www.bionik.tu-berlin.de/user/niko/cmaesintro.html>

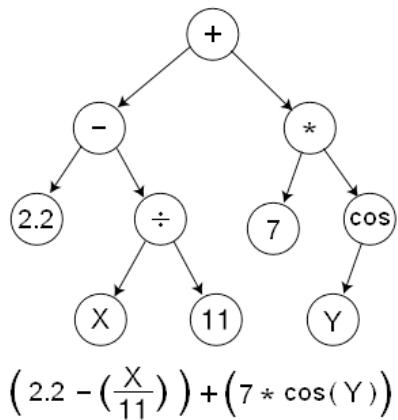
[Hansen, 1995]

## Programmation Génétique (“GP”)

50

[Koza, 92]

- Représentation
  - ▶ Arbre, graphe, DAG
  - ▶ Fonctions et Terminaux
  - ▶ ADF
- Opérateurs
  - ▶ Croisement
  - ▶ (Mutation)
- Applications
  - ▶ Régression symbolique
  - ▶ Circuit électronique (multiplexer)
  - ▶ Construction de structures - cf. transparents suivants



# Conclusions

## Bonnes pratiques

52

- Evaluation

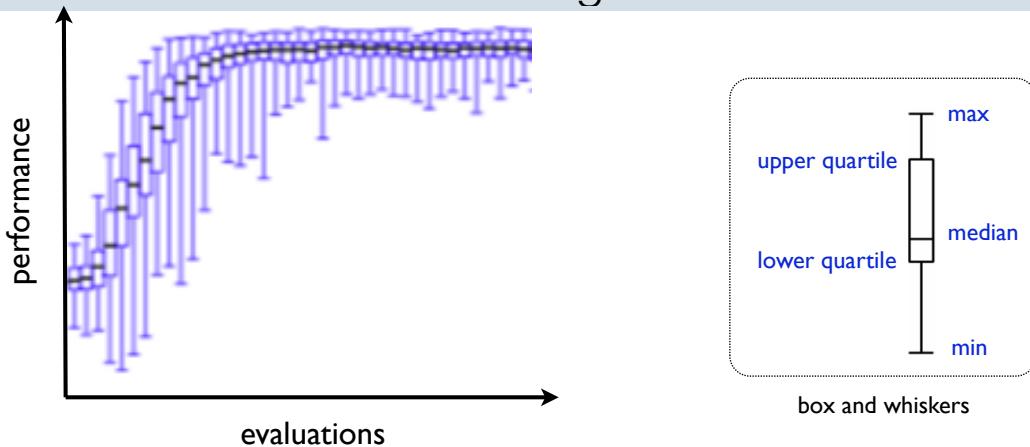
- ▶ Une “bonne” fonction fitness vise un objectif et facilite le chemin
- ▶ Il faut parfois réévaluer plusieurs fois une solution candidate pour avoir une bonne estimation de sa qualité

- Critère d'arrêt

- ▶ budget (en nombre d'évaluations)
- ▶ heuristique/empirique (convergence/stagnation observée)
- ▶ perte de diversité, atteinte de l'optimum (si connu)

- Validité des résultats

- ▶ Il s'agit d'un algorithme stochastique
- ▶ Il faut faire plusieurs runs
- ▶ Tracer: médiane, min, max, quartile... c'est mieux que moyenne et écart type



- A retenir:

- ▶ Médianes plutôt que moyennes, évaluations plutôt que générations
- ▶ Répéter les expériences, poursuivre jusqu'à convergence
- ▶ Donner les résultats pour les meilleures perf's et pour les moyennes
  - meilleures performances au mieux: le choix de la solution
  - meilleures performances en moyenne: le choix de l'algorithme

*Il s'agit d'un exemple à fin d'illustration*

# Fin du cours