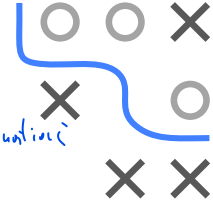
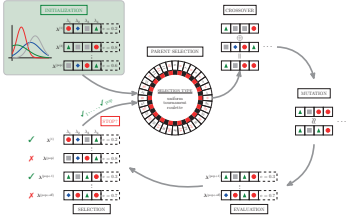
# Optimization in Machine Learning

## Evolutionary Algorithms Introduction



08.01.25

Examples of deriv. free situations
1) Discrete optimization
2) Simulations

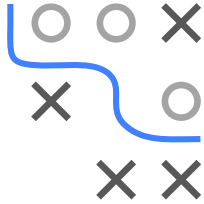



**Learning goals**

- Evolutionary algorithms
- Encoding
- Parent selection, variation, survival selection

# EVOLUTIONARY ALGORITHMS

**Evolutionary algorithms** (EA) are a class of stochastic, metaheuristic optimization techniques whose mode of operation is inspired by the evolution of natural organisms.

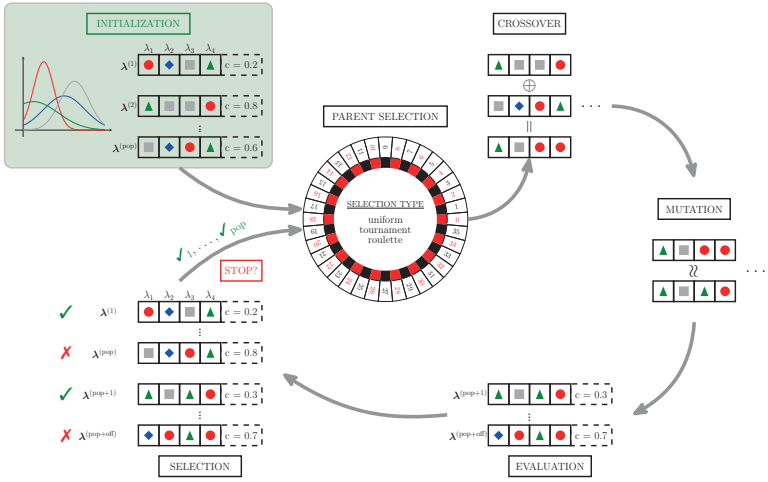


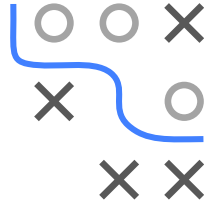
History of evolutionary algorithms:

- **Genetic algorithms**: Use binary problem representation, therefore closest to the biological model of evolution.
- **Evolution strategies**: Use direct problem representation, e.g., vector of real numbers.
- **Genetic programming**: Create structures that convert an input into a fixed output (e.g. computer programs); solution candidates are represented as trees.
- **Evolutionary programming**: Similar to genetic programming, but solution candidates are not represented by trees, but by finite state machines.

The boundaries between the terms become increasingly blurred and are often used synonymously.

# STRUCTURE OF AN EVOLUTIONARY ALGORITHM



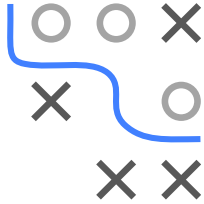
We have $\nu$ elements generate $\lambda$ offsprings.
then merge $\nu, \lambda$ and select fittest $\nu$

$\nu = 1$ | stochastic
$\lambda = 1$ | local search

# NOTATION AND TERMINOLOGY

(encoding)

- A chromosome is a set of parameters which encodes a proposed solution to the problem that the genetic algorithm is trying to solve. The chromosome is often represented as a binary string, although a wide variety of other data structures are also used.
- The set of all solutions is known as the population.

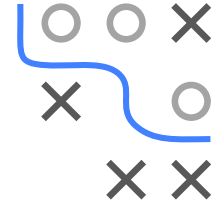


| Symbols | EA Terminology |
|---|---|
| solution candidate $\mathbf{x} \in \mathcal{S}$ | chromosome of an individual |
| $x_j$ | $j$-th gene of chromosome |
| set of candidates $P$ with $\mu = \lvert P \rvert$ | population and size |
| $\lambda$ | number of generated offsprings |
| $f : \mathcal{S} \to \mathbb{R}$ | fitness function |

**Note**: Unintuitively, we are minimizing fitness because we always minimize $f$ by convention.

# ENCODING

Encoding of chromosomes is the first step of solving a problem with EAs. Technically: Mapping from **genotype** to **phenotype**. Encoding depends on the problem, and eventually decides performance of problem solving.



**Encoding methods**:

- Binary encoding: Strings of 0s and 1s
- Real value encoding: Real values

When we choose encoding approach we should think how mutation and crossover will look like.

**Genotype:**

| $x_1$ | $x_2$ |
|---|---|

**Phenotype:**

| 01101 | 11001 |
|---|---|

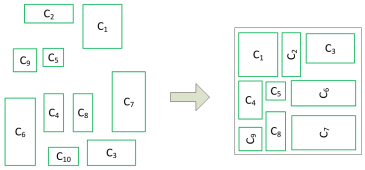
**Binary encoding**

| 13 | 25 |
|---|---|

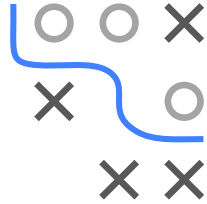
**Real value encoding**

# ENCODING

- Tree encoding: Tree objects



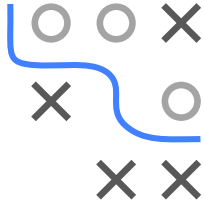

what are V/H?



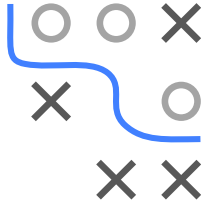
Floor planning problem. Given are *n* circuits of different area requirements. Goal: arrange them into a floor layout so that all circuits are placed in a minimum layout. Each solution candidate can be represented by a tree.
Source: Encoding Techniques in Genetic Algorithms, Debasis Samanta, 2018.

# STEP 1: INITIALIZE POPULATION

- Evolutionary algorithms start with generating initial population $P = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(\mu)}\}$.
- Usually: Initialize uniformly at random. (or use heuristics if available (domain knowledge))
- Introducing prior knowledge possible.
- Population is evaluated: objective function is computed for each initial individual. (the most expensive step, luckily can be parallelized)
- Initialization influences quality of solution, so many EAs employ *restarts* with new randomly generated initial populations.

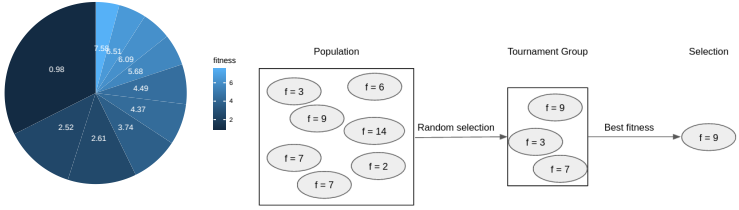


# STEP 2: PARENT SELECTION



Choose a number of $\lambda$ parents pairs creating $\lambda$ offsprings.

- **Neutral selection:** Draw parents uniformly at random.
- **Fitness-proportional / Roulette wheel selection:** Draw individuals with probability proportional to their fitness.
- **Tournament selection:** Randomly select $k$ individuals for a "tournament group" and pick the best one (according to fitness value).



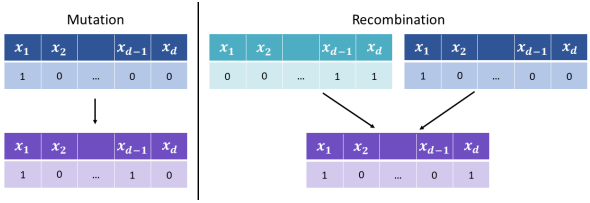

**Left:** Fitness-proportional selection. Fitness values of $\mu = 10$ individuals are converted into probabilities. **Right:** Tournament selection.

# STEP 3: VARIATION

New individuals (offsprings) are generated from parents.

- Recombination/Crossover: Combine two parents into offspring.
- Mutation: Modify the offspring locally.

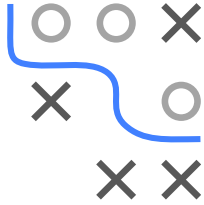
Sometimes only one of both operations is performed.




**Note:** Particular operation depends on encoding. Examples for binary and numeric encodings follow later.

# STEP 4: SURVIVAL SELECTION

Choosing surviving individuals. Two common strategies are:



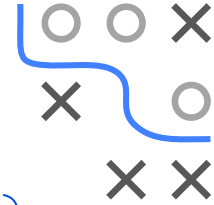
- $(\mu, \lambda)$**-selection:** Select $\mu$ best individuals *only from set of offsprings* ($\lambda \geq \mu$ necessary).
  **But:** Best individual can get lost! (Because can be from population, not offspring)
- $(\mu + \lambda)$**-selection:** Select $\mu$ best individuals from set of $\mu$ parents and $\lambda$ offsprings
  **Now:** Best individual certainly survives.

Mostly (maybe always) this is used

# EVOLUTIONARY ALGORITHMS

## Advantages

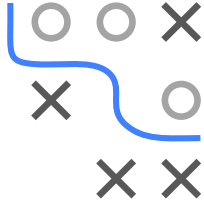
- Simple but enough to solve complex problems
- All parameter types possible in general
- Highly parallelizable
- Flexible through different variation operations



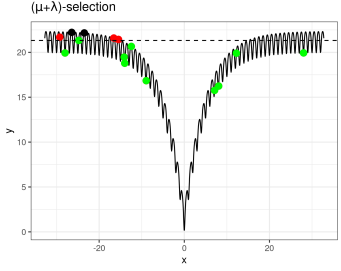
## Disadvantages

- Little mathematical ~~rigor~~ (for realistic, complex EAs)
- Hard to find balance between exploration and exploitation
- Quite some parameters, hard to determine them
- Customization necessary for complex problems
- Not suitable for expensive problems like HPO as large number of function evaluations necessary

# Optimization in Machine Learning



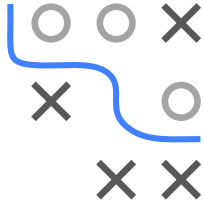
## Evolutionary Algorithms
## ES / Numerical Encodings




**Learning goals**

- Recombination
- Mutation
- A few simple examples

# RECOMBINATION FOR NUMERIC

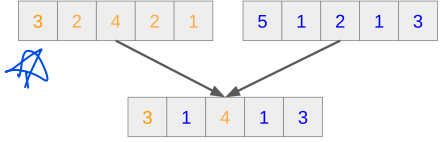


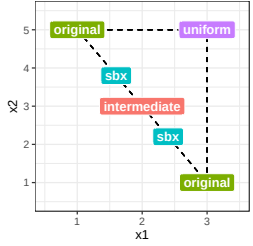
Options for recombination of two individuals $\mathbf{x}, \tilde{\mathbf{x}} \in \mathbb{R}^d$:

- **Uniform crossover**: Choose gene $j$ of parent 1 with probability $p$ and of parent 2 with probability $1 - p$
- **Intermediate recombination**: Offspring is created from mean of two parents: $\frac{1}{2}(\mathbf{x} + \tilde{\mathbf{x}})$
- **Simulated Binary Crossover (SBX)**: generate **two offspring**

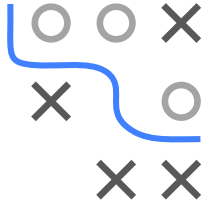
convex combination of 2 parents

$$\bar{\mathbf{x}} \pm \frac{1}{2}\beta(\tilde{\mathbf{x}} - \mathbf{x}), \ \bar{\mathbf{x}} = \frac{1}{2}(\mathbf{x} + \tilde{\mathbf{x}}), \ \beta \in [0, 1] \text{ uniformly at random}$$



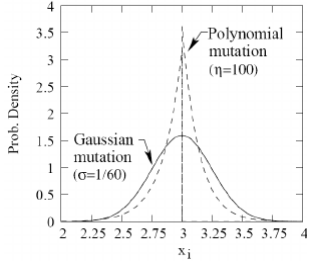
# MUTATION FOR NUMERIC



**Mutation:** Individuals get modified

**Example** for $\mathbf{x} \in \mathbb{R}^d$:

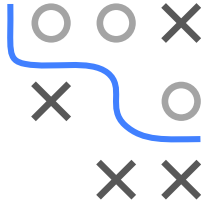
- **Uniform mutation:** Select random gene $x_j$ and replace it by uniformly distributed value (within feasible range). → may cause a large step
- **Gauss mutation:** $\mathbf{x} \pm \mathcal{N}(0, \sigma \mathbf{I})$
- **Polynomial mutation:** Use a different distribution instead of normal distribution

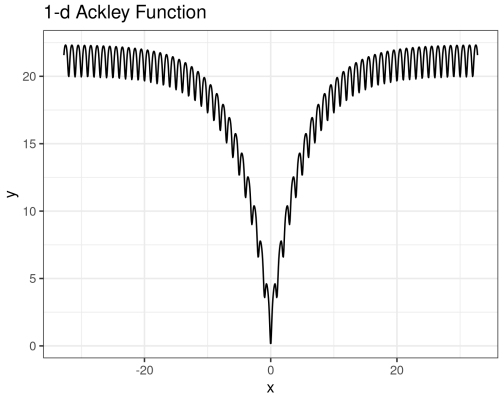



**Source:** K. Deb, D. Deb. Analysing mutation schemes for real-parameter genetic algorithms, 2014

# EXAMPLE OF AN EVOLUTIONARY ALGORITHM

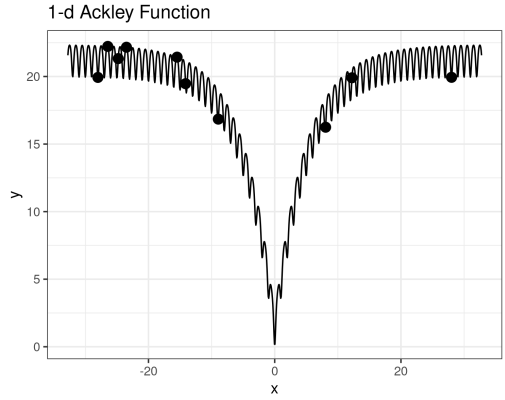
(Simple) EA on 1-dim Ackley function on $[-30, 30]$. Usually, for optimizing a function $f : \mathbb{R}^d \to \mathbb{R}$, individuals are encoded as real vectors $\mathbf{x} \in \mathbb{R}^d$.

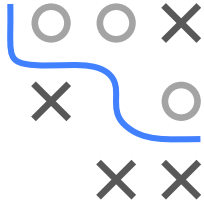




# EXAMPLE OF AN EVOLUTIONARY ALGORITHM

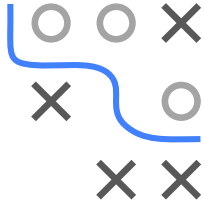
Random initial population with size $\mu = 10$

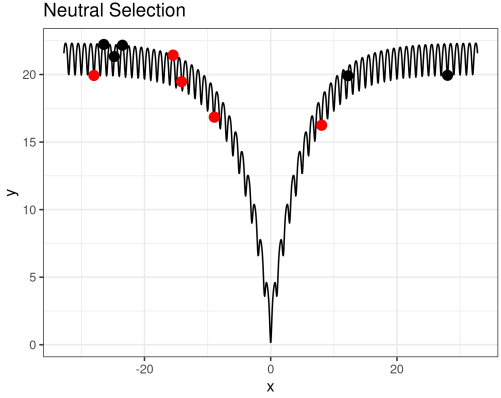


# EXAMPLE 1: ACKLEY FUNCTION

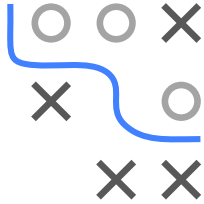
We choose $\lambda = 5$ offsprings by neutral selection (red individuals).

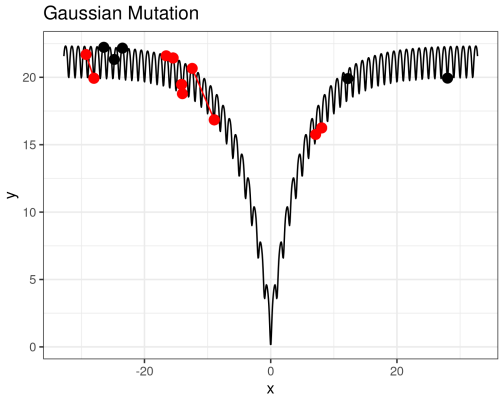




# EXAMPLE 1: ACKLEY FUNCTION

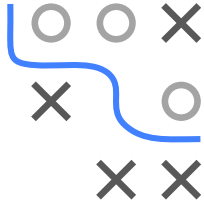
Use Gaussian mutation with $\sigma = 2$, but without recombination.

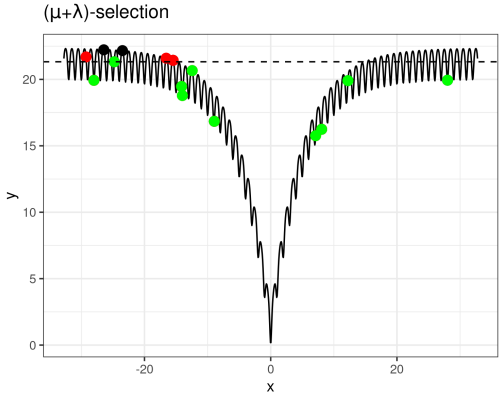




# EXAMPLE 1: ACKLEY FUNCTION

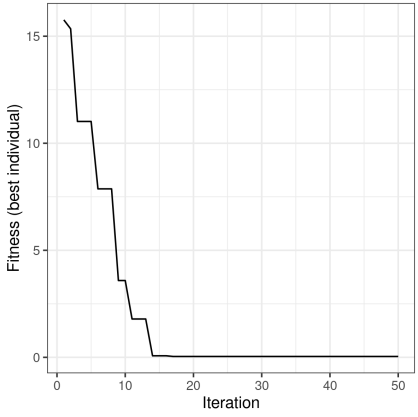
Use $(\mu + \lambda)$ selection. Selected individuals are marked in green.

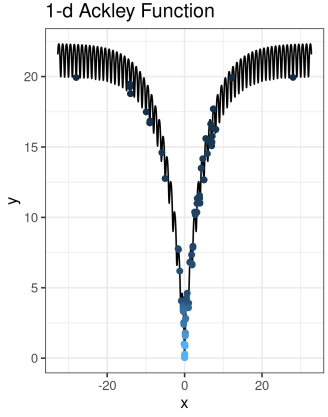


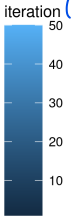


# EXAMPLE 1: ACKLEY FUNCTION

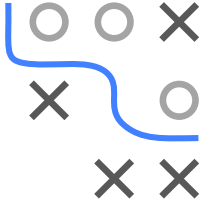
After 50 iterations:



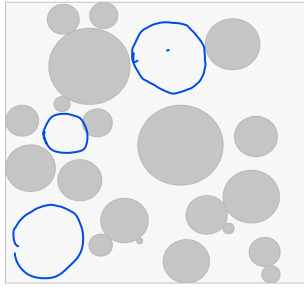



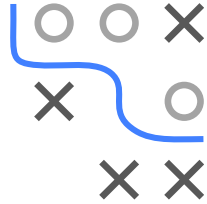

(generation = producing new population)



# EXAMPLE 2: GRID OF BALLS

Consider a grid in which $n$ balls with random radius are placed.





fixed center of radius

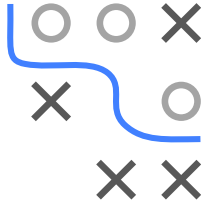
Homework

**Aim:** Find the circle with the largest possible radius in the grid that does **not** intersect with the other existing circles.

- What is the fitness function?
- How is the population defined?

Implementation: `https://juliambr.shinyapps.io/balls/`

# EXAMPLE 2: GRID OF BALLS



In our example, the chromosome of an individual is the center of a circle, so the chromosomes are encoded as 2-dimensional real vectors $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$.

The population $P \subset \mathbb{R}^2$ is given as a set of circle centers.

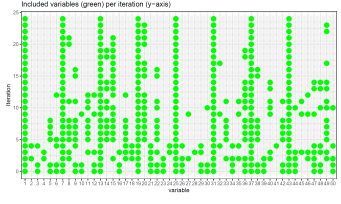
The fitness function evaluates an individual $\mathbf{x} \in P$ based on the distance to the nearest neighboring gray circle $k$.

$$f(\mathbf{x}) = \min_{k \in \text{Grid}} \text{distance}(k, \mathbf{x}),$$

where the distance is defined as 0 if a circle center is within the radius of a circle of the grid.
This function is to be maximized: we are looking for the largest circle that does not touch any of the gray circles.

# Optimization in Machine Learning

## Evolutionary Algorithms
## GA / Bit Strings

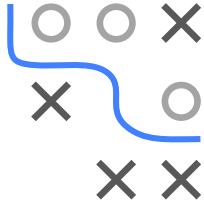



**Learning goals**

- Recombination
- Mutation
- Simple examples

# BINARY ENCODING

- In theory: Each problem can be encoded binary
- In practice: Binary not always best representation (e.g., if values are numeric, trees or programs)



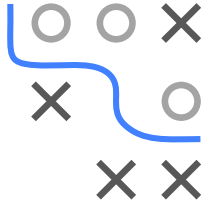
We typically encode problems with **binary decision variables** in binary representation.

**Examples:**

- Scheduling problems
- Integer / binary linear programming
- Feature selection
- ...

# RECOMBINATION FOR BIT STRINGS

Two individuals $\mathbf{x}, \tilde{\mathbf{x}} \in \{0,1\}^d$ encoded as bit strings can be recombined as follows:



- **1-point crossover:** Select crossover $k \in \{1, ..., d-1\}$ randomly. Take first $k$ bits from parent 1 and last $d-k$ bits from parent 2.

$$\begin{matrix} 1 & 1 & & 1 \\ 0 & 0 & & 0 \\ \hline 0 & 1 & \Rightarrow & 1 \\ 1 & 1 & & 1 \\ 1 & 0 & & 0 \end{matrix}$$

for example
For feature selection we don't care about order so 1-point crossover much more sense

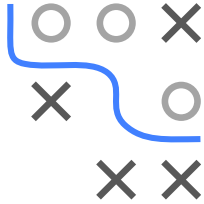
- **Uniform crossover:** Select bit $j$ with probability $p$ from parent 1 and $1-p$ from parent 2.

$$\begin{matrix} 1 & 0 & & 1 \\ 0 & 0 & & 0 \\ 0 & 1 & \Rightarrow & 1 \\ 0 & 1 & & 1 \\ 1 & 0 & & 1 \end{matrix}$$

select randomly

# MUTATION FOR BIT STRINGS

Offspring $\mathbf{x} \in \{0,1\}^d$ encoded as a bit string can be mutated as follows:



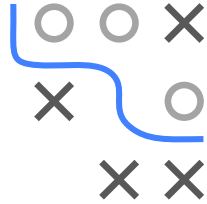
- **Bitflip:** Each bit $j$ is flipped with probability $p \in (0,1)$.

$$\begin{array}{ccc} 1 & & 0 \\ 0 & & 0 \\ 0 & \Rightarrow & 0 \\ 0 & & 1 \\ 1 & & 1 \end{array}$$

# EXAMPLE 1: ONE-MAX EXAMPLE

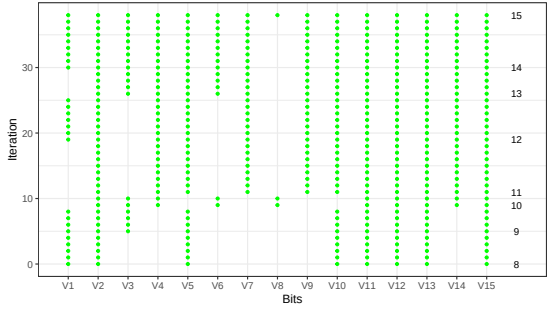
Usual as sanity check

$\mathbf{x} \in \{0,1\}^d$, $d = 15$ bit vector representation.



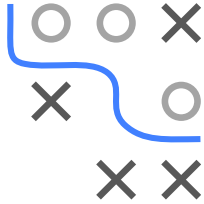
**Goal:** Find the vector with the maximum number of 1's.

- Fitness: $f(\mathbf{x}) = \sum_{i=1}^{d} x_i$
- $\mu = 15$, $\lambda = 5$, $(\mu + \lambda)$-strategy, bitflip mutation, no recombination




**Green:** Representation of best individual per iteration. Right scale shows fitness.

# EXAMPLE 2: FEATURE SELECTION



We consider the following toy setting:

- Generate design matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ by drawing $n = 1000$ samples of $p = 50$ independent normally distributed features with $\mu_j = 0$ and $\sigma_j^2 > 0$ varying between 1 and 5 for $j = 1, \ldots, p$.
- Linear regression problem with dependent variable $\mathbf{y}$:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \epsilon$$

with $\epsilon \sim \mathcal{N}(0, 1)$.

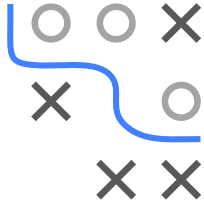
Parameter $\boldsymbol{\theta}$:

$$\theta_0 = -1.2$$

$$\theta_j = \begin{cases} 1 & \text{for } j \in 1, 7, 13, 19, 25, 31, 37, 43 \\ 0 & \text{otherwise} \end{cases}$$

$\Rightarrow$ Only 8 out of 50 equally influential features

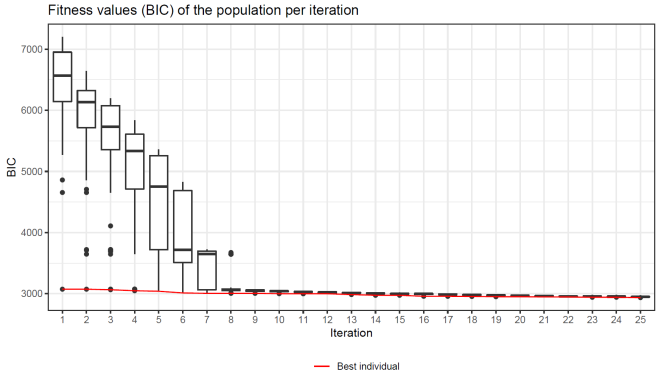
# EXAMPLE 2: FEATURE SELECTION

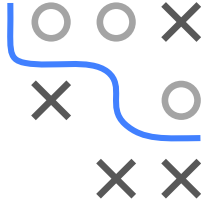


- **Aim:** Find influential features
- **Encoding:** $\mathbf{z} \in \{0,1\}^p$, $z_j = 1$ means $\theta_j$ included in model
- **Fitness function** $f(\mathbf{z})$: BIC of the model belonging to $\mathbf{z}$
- **Mutation:** Bit flip with $p = 0.3$
- **Recombination:** Uniform crossover with $p = 0.5$
- **Survival selection:** $(\mu + \lambda)$ strategy with $\mu = 100$ and $\lambda = 50$

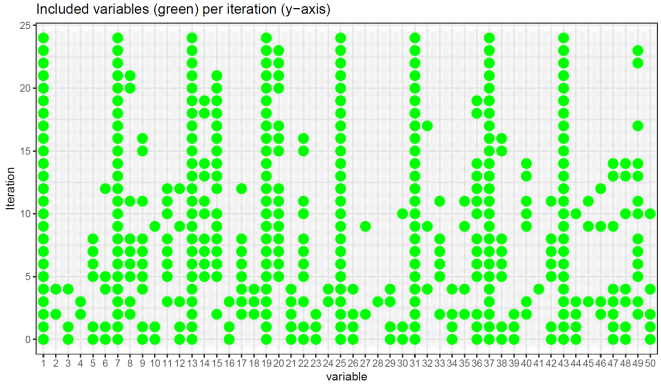
```
## [1] "After 10 iterations:"
##  [1]  1  7 11 13 14 15 19 20 22 25 30 31 36 37 40 43 44 48
## [19] 49 50
## [1] "After 20 iterations:"
##  [1]  1  7  8 13 15 19 20 25 31 37 43
## [1] "Included variables after 24 iterations:"
## [1]  1  7 13 19 25 31 37 43
```

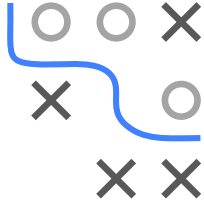
# EXAMPLE 2: FEATURE SELECTION



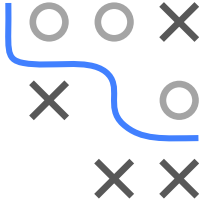



# EXAMPLE 2: FEATURE SELECTION



# Optimization in Machine Learning

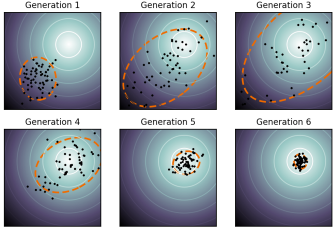


## Evolutionary Algorithms
## CMA-ES Algorithm

14.01.24

from 2001, still SOTA

Author is PhD in Medicine




**Learning goals**

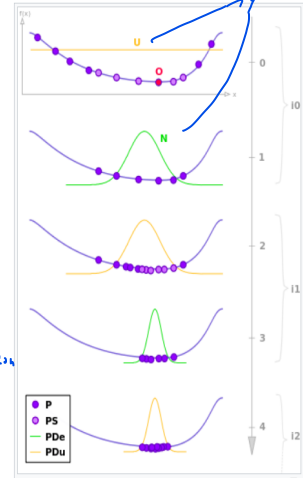
- CMA-ES strategy
- Estimation of distribution
- Step size control

# ESTIMATION OF DISTRIBUTION ALGORITHM

Main difference from EA is

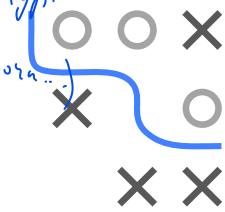
- Instead of population, maintain distribution to sample offspring from

1. Draw $\lambda$ offsprings $\mathbf{x}^{(i)}$ from $p(\cdot|\boldsymbol{\theta}^{[t]})$
2. Evaluate fitness $f(\mathbf{x}^{(i)})$ (we're doing maximization here)
3. Update $\boldsymbol{\theta}^{[t+1]}$ with $\mu$ best offsprings




Estimation of distribution algorithm. For each iteration *i*, a random draw is performed for a population *P* in a distribution *PDu*. The distribution parameters *PDe* are then estimated using the selected points *PS*. The illustrated example optimizes a continuous objective function *f(X)* with a unique optimum *O*. The sampling (following a normal distribution *N*) concentrates around the optimum as one goes along unwinding algorithm.

Sampling type (uniform...)



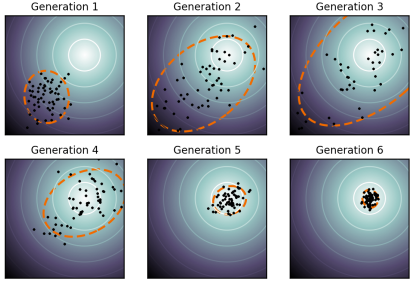
? Do we switch between
• sampling distribution? N → U ...

# COVARIANCE MATRIX ADAPTATION

Sample distribution is multivariate Gaussian

$$\mathbf{x}^{[t+1](i)} \sim \mathbf{m}^{[t]} + \sigma^{[t]}\mathcal{N}(\mathbf{0}, \mathbf{C}^{[t]}) \quad \text{for } i = 1, \ldots, \lambda$$

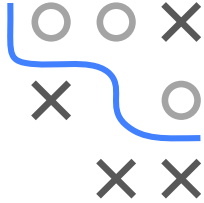
- $\mathbf{x}^{[t+1](i)} \in \mathbb{R}^d$ $i$-th offspring; $\lambda \geq 2$ number of offspring
- $\mathbf{m}^{[t]} \in \mathbb{R}^d$ mean value and $\mathbf{C}^{[t]} \in \mathbb{R}^{d \times d}$ covariance matrix
- $\sigma^{[t]} \in \mathbb{R}_+$ "overall" standard deviation/step size → "mean + stepsize + direction"




**Question:** How to adapt $\mathbf{m}^{[t+1]}$, $\mathbf{C}^{[t+1]}$, $\sigma^{[t+1]}$ for next generation $t + 1$?

# CMA-ES: BASIC METHOD - ITERATION 1

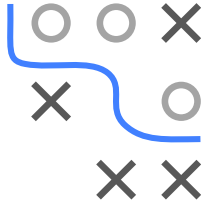
0. Initialize $\mathbf{m}^{[0]}, \sigma^{[0]}$ problem-dependent and $\mathbf{C}^{[0]} = \mathbf{I}_d$

# CMA-ES: BASIC METHOD - ITERATION 1

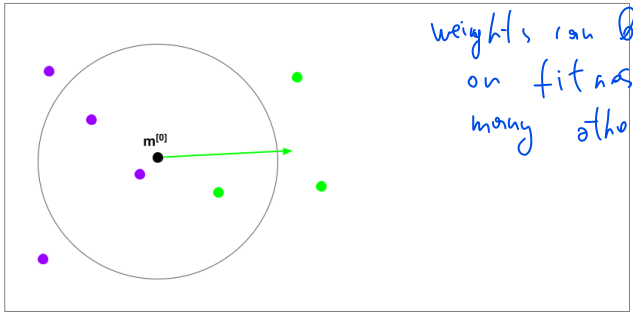
1. **Sample** $\lambda$ offsprings from distribution

$$\mathbf{x}^{[1](i)} = \mathbf{m}^{[0]} + \sigma^{[0]}\mathcal{N}(\mathbf{0}, \mathbf{C}^{[0]})$$

# CMA-ES: BASIC METHOD - ITERATION 1

❷ **Selection and recombination** of $\mu < \lambda$ best-performing offspring using fixed weights $w_1 \geq \ldots \geq w_\mu > 0, \sum_{i=1}^{\mu} w_i = 1$.
$\mathbf{x}_{i:\lambda}$ is $i$-th ranked solution, ranked by $f(\mathbf{x}_{i:\lambda})$.



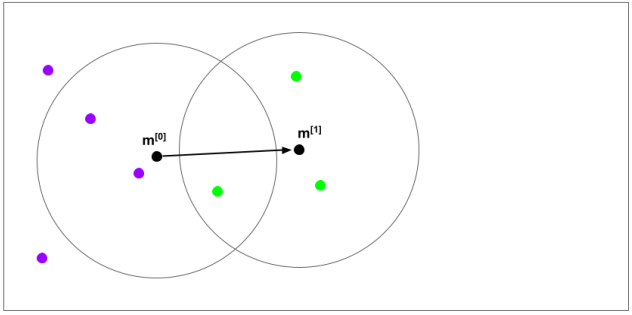

weights can be adjusted based on fitness value, but there are many other approaches

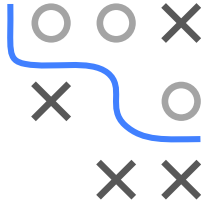
Calculation of auxiliary variables ($\mu = 3$ points)
$\mathbf{y}_w^{[1]} := \sum_{i=1}^{\mu} w_i (\mathbf{x}_{i:\lambda}^{[1]} - \mathbf{m}^{[0]}) / \sigma^{[0]} := \sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda}^{[1]}$

normalizing

# CMA-ES: BASIC METHOD - ITERATION 1

❸ **Update mean**



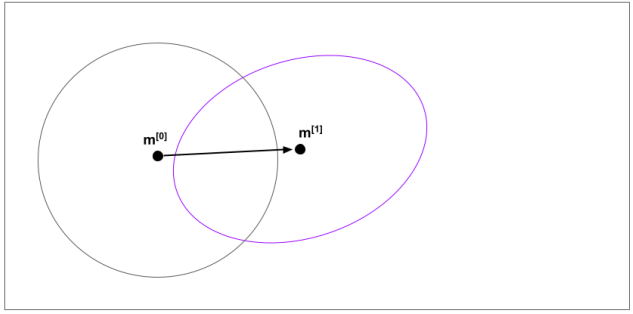



Movement towards the new distribution with mean
$\mathbf{m}^{[1]} = \mathbf{m}^{[0]} + \sigma^{[0]}\mathbf{y}_w^{[1]}$.

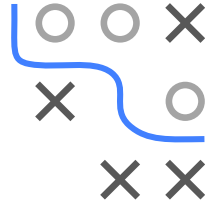
# CMA-ES: BASIC METHOD - ITERATION 1

❹ **Update covariance matrix**

Roughly: elongate density ellipsoid in direction of successful steps.
$\mathbf{C}^{[1]}$ reproduces successful points with higher probability than $\mathbf{C}^{[0]}$.



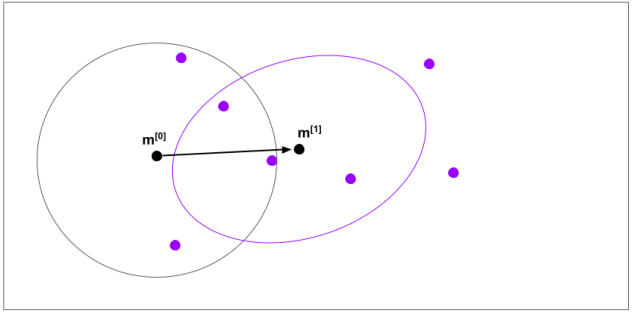

Update $\mathbf{C}^{[0]}$ using sum of outer products and parameter $c_\mu$:
$\mathbf{C}^{[1]} = (1 - c_\mu)\mathbf{C}^{[0]} + c_\mu \sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda}^{[1]} (\mathbf{y}_{i:\lambda}^{[1]})^\top$ (rank-$\mu$ update).

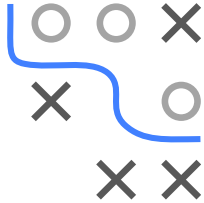


if points are too close we will shrink the cov. matrix, and get stuck. We usually do restarts of the algorithm if we get stuck we restart with bigger pop.

# CMA-ES: BASIC METHOD - ITERATION 2

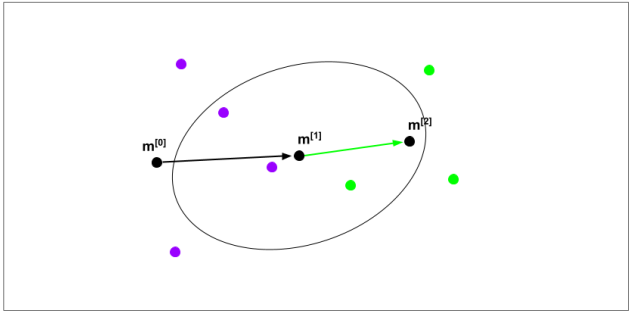
1. **Sample** from distribution for new generation

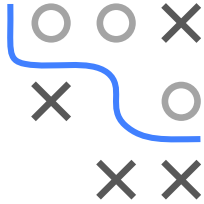


# CMA-ES: BASIC METHOD - ITERATION 2

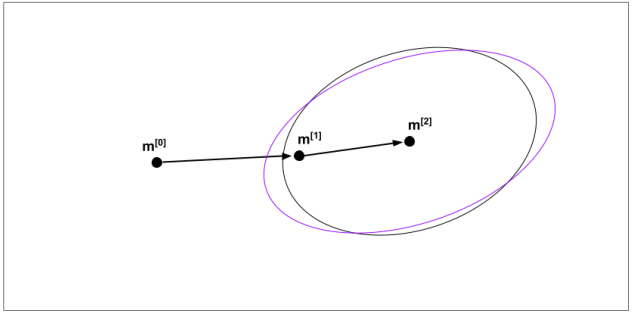
2. **Selection and recombination** of $\mu < \lambda$ best-performing offspring
3. **Update mean**

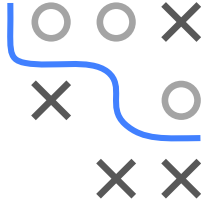


# CMA-ES: BASIC METHOD - ITERATION 2

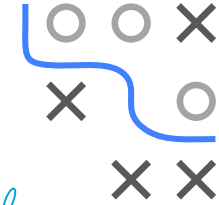
❹ **Update covariance matrix**

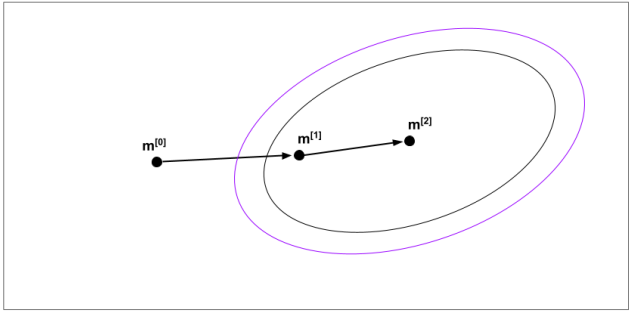


# CMA-ES: BASIC METHOD - ITERATION 2

5. **Update step-size** exploiting correlation in history of steps.
   steps point in similar direction $\Longrightarrow$ increase step-size
   steps cancel out $\Longrightarrow$ decrease step-size

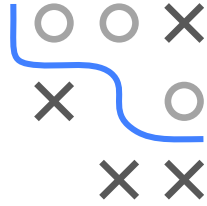





kind of like momentum

# UPDATING C: FULL UPDATE

Full CMA update of $\mathbf{C}$ combines rank-$\mu$ update with a rank-1 update using exponentially smoothed evolution path $\mathbf{p}_c \in \mathbb{R}^d$ of successive steps and learning rate $c_1$:



$$\mathbf{p}_c^{[0]} = \mathbf{0}, \quad \mathbf{p}_c^{[t+1]} = (1 - c_1)\mathbf{p}_c^{[t]} + \sqrt{\frac{c_1(2 - c_1)}{\sum_{i=1}^{\mu} w_i^2}}\,\mathbf{y}_w$$

fitness?

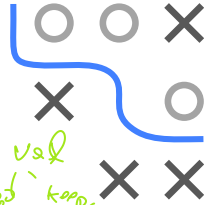
$\sum_{i=1}^{\mu} w_i = 1$ $w_i \in (0,1)$, when $w_s$ are equal

Final update of $\mathbf{C}$ is

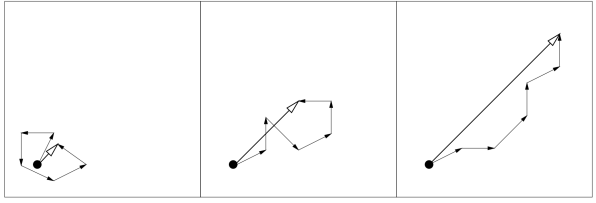
$$\mathbf{C}^{[t+1]} = (1 - c_1 - c_\mu \textstyle\sum_j w_j)\mathbf{C}^{[t]} + c_1 \underbrace{\mathbf{p}_c^{[t+1]}(\mathbf{p}_c^{[t+1]})^\top}_{\text{rank-1}} + c_\mu \underbrace{\sum_{i=1}^{\mu} w_i \mathbf{y}_{i:\lambda}^{[t+1]}(\mathbf{y}_{i:\lambda}^{[t+1]})^\top}_{\text{rank-}\mu}$$

- Correlation between generations used in rank-1 update
- Information from entire population is used in rank-$\mu$ update

# UPDATING $\sigma$: METHODS STEP-SIZE CONTROL



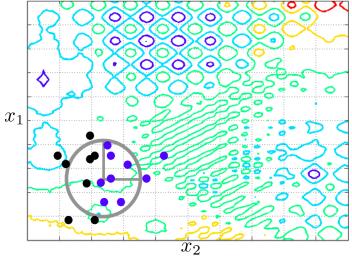
- $1/5$**-th success rule**: increases the step-size if more than 20 % of the new solutions are successful, decrease otherwise
- $\sigma$**-self-adaptation**: mutation is applied to the step-size and the better - according to the objective function value - is selected
- **Path length control via cumulative step-size adaptation (CSA)**
  Intuition:
  - Short cumulative step-size $\triangleq$ steps cancel $\rightarrow$ decrease $\sigma^{[t+1]}$
  - Long cumulative step-size $\triangleq$ corr. steps $\rightarrow$ increase $\sigma^{[t+1]}$



# Optimization in Machine Learning

## Evolutionary Algorithms
## CMA-ES Wrap Up

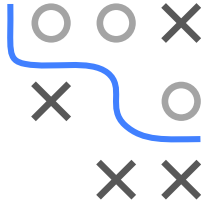
15.01.25




**Learning goals**

- Advantages & Limitations
- IPOP-CMA-ES
- Benchmark

# CMA-ES: WRAP UP

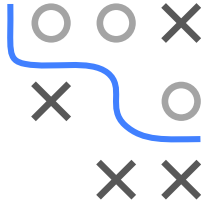


**Algorithm** CMA-ES

1: Input: $m \in \mathbb{R}^d$, $\sigma \in \mathbb{R}_+$, $\lambda$ (problem-dependent)
2: Initialize: $\boldsymbol{C} = \mathbb{I}$, $\boldsymbol{p_c} = \mathbf{0}$, $\boldsymbol{p}_\sigma = \mathbf{0}$
3: Set: $c_{\boldsymbol{C}} \approx 4/d$, $c_\sigma \approx 4/d$, $c_1 \approx 2/d^2$, $c_\mu \approx \mu_w/d^2$, $c_1 + c_\mu \leq 1$, $d_\sigma \approx 1 + \sqrt{\mu_w/d}$ and $w_{i=1,\ldots,\mu}$ such that $\mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2} \approx 0.3\lambda$ → obtained empirically
4: **while** not terminate **do**
5: $\quad \mathbf{x}^{(i)} = \boldsymbol{m} + \sigma\mathcal{N}(\mathbf{0}, \boldsymbol{C})$ for $i = 1, \ldots, \lambda$ — *Sampling*
6: $\quad \boldsymbol{y}_w = \sum_{i=1}^{\mu} w_i \boldsymbol{y}_{i:\lambda}$, where $\boldsymbol{y}_{i:\lambda} = (\mathbf{x}_{i:\lambda} - \boldsymbol{m})/\sigma$ — *Selection/Recombination*
7: $\quad \boldsymbol{m} \leftarrow \boldsymbol{m} + \sigma \boldsymbol{y}_w$ — *Update* $\boldsymbol{m}$
8: $\quad \boldsymbol{p_c} \leftarrow (1 - c_{\boldsymbol{C}})\boldsymbol{p_c} + \sqrt{c_{\boldsymbol{C}}(2 - c_{\boldsymbol{C}})\mu_w}\,\boldsymbol{y}_w$ — *Cumulation of* $\boldsymbol{C}$
9: $\quad \boldsymbol{p}_\sigma \leftarrow (1 - c_\sigma)\boldsymbol{p}_\sigma + \sqrt{c_\sigma(2 - c_\sigma)\mu_w}\,\boldsymbol{C}^{-\frac{1}{2}}\boldsymbol{y}_w$ — *Cumulation of* $\sigma$
10: $\quad \boldsymbol{C} \leftarrow (1 - c_1 - c_\mu \sum w_j)\boldsymbol{C} + c_1 \boldsymbol{p_c}\boldsymbol{p_c}^\top + c_\mu \sum_{i=1}^{\mu} w_i \boldsymbol{y}_{i:\lambda}\boldsymbol{y}_{i:\lambda}^\top$ — *Update* $\boldsymbol{C}$
11: $\quad \sigma \leftarrow \sigma \times \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||\boldsymbol{p}_\sigma||}{\mathbb{E}||\mathcal{N}(\mathbf{0},\mathbb{I})||} - 1\right)\right)$ — *Update* $\sigma$
12: **end while**

(exp length of random vector has closed form)

# CMA-ES: WRAP UP - DEFAULT VALUES



Related to selection and recombination:

- $\boldsymbol{\lambda}$: offspring number, population size $4 + \lfloor 3 \ln d \rfloor$
- $\boldsymbol{\mu}$: parent number, solutions involved in mean update $\lfloor \lambda/2 \rfloor$
- $\boldsymbol{w_i}$: recombination weights (preliminary convex shape) $\ln \frac{\lambda+1}{2} - \ln i, \text{ for } i = 1, \ldots, \lambda$

Related to $\boldsymbol{C}$-update:

- $\mathbf{1} - \boldsymbol{c_C}$: decay rate for evolution path, cumulation factor $1 - \frac{4+\mu_w/d}{d+4+2\mu_w/d}$
- $\boldsymbol{c_1}$: learning rate for rank-one update of $\boldsymbol{C}$ $\frac{2}{(d+1.3)^2+\mu_w}$
- $\boldsymbol{c_\mu}$: learning rate for rank-$\mu$ update of $\boldsymbol{C}$ $\min\left(1 - c_1, 2 \cdot \frac{\mu_w - 2 + 1/\mu_w}{(d+2)^2+\mu_w}\right)$

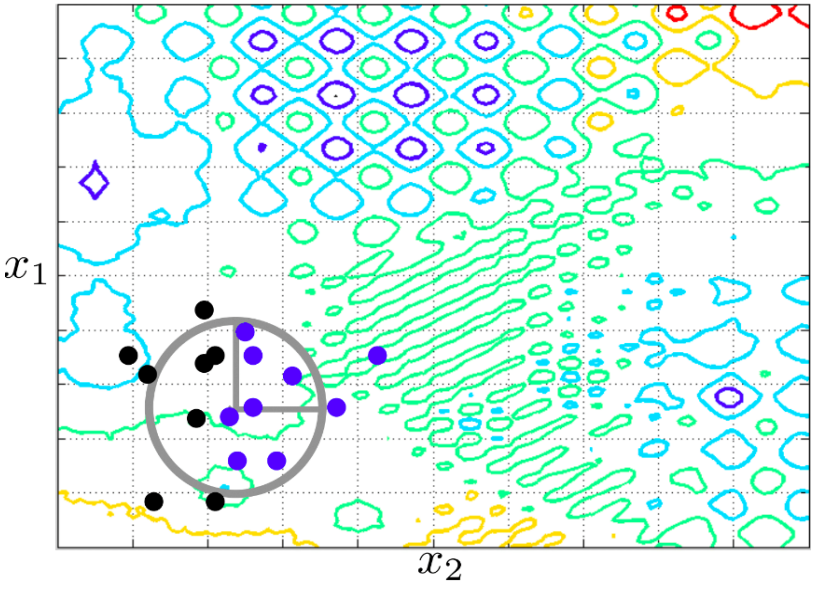
Related to $\sigma$-update:

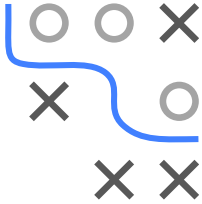
- $\mathbf{1} - \boldsymbol{c_\sigma}$: decay rate for evolution path $1 - \frac{\mu_w+2}{d+\mu_w+5}$
- $\boldsymbol{d_\sigma}$: damping for $\sigma$-change $1 + 2\max\left(0, \sqrt{\frac{\mu_w-1}{d+1}} - 1\right) + c_\sigma$

with $\mu_w = \left(\frac{||\boldsymbol{w}||_1}{||\boldsymbol{w}||_2}\right) = \frac{(\sum_{i=1}^{\mu} |w_i|)^2}{\sum_{i=1}^{\mu} w_i^2} = \frac{1}{\sum_{i=1}^{\mu} w_i^2}$ and typical default parameter values.

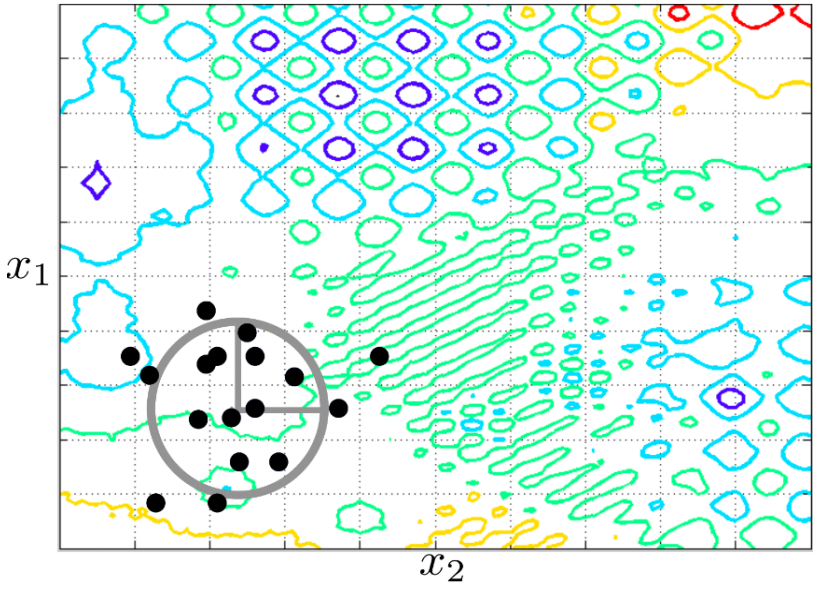
- $\mu_w$ can be extended to $\lambda$ instead of $\mu$ weights, allowing negative weights for the remaining $\lambda - \mu$ points ("active covariance adaption").

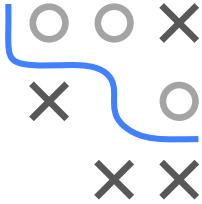
# CMA-ES: WRAP UP



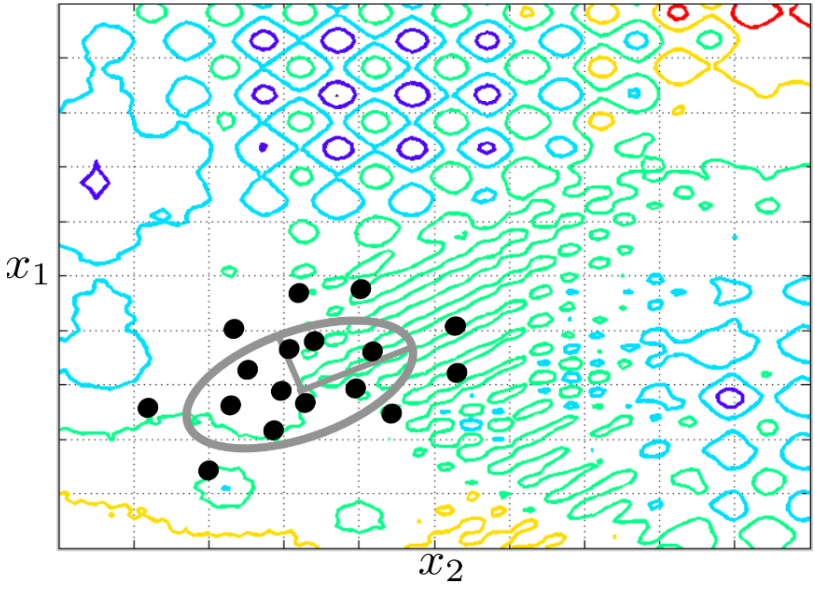



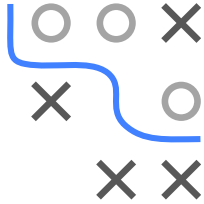
# CMA-ES: WRAP UP



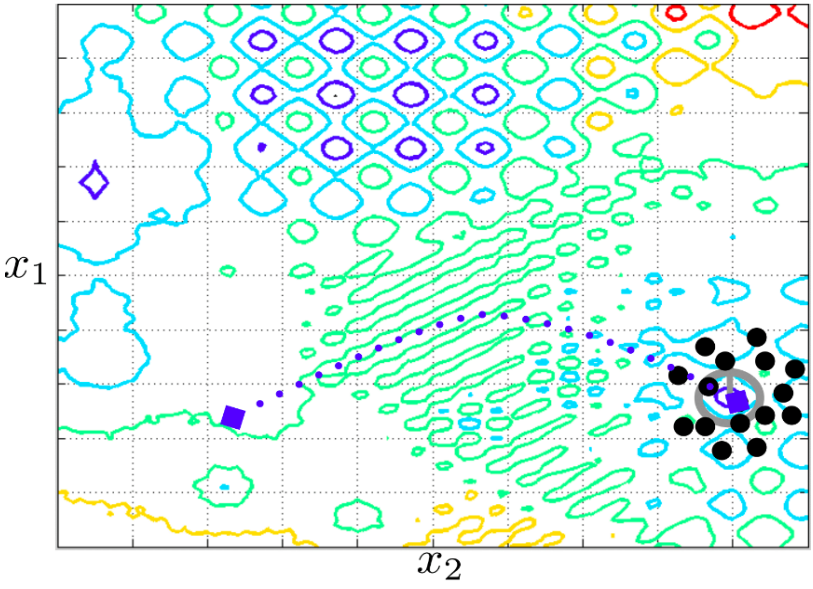



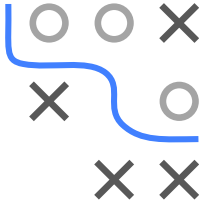
# CMA-ES: WRAP UP



# CMA-ES: WRAP UP

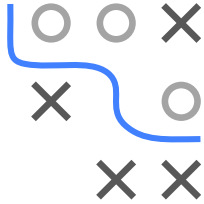


# CMA-ES: WRAP UP - ADVANTAGES

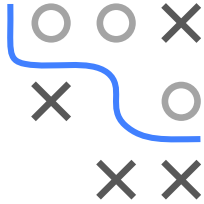
CMA-ES can outperform other strategies in following cases:

- Non-separable problems (parameters of the objective function are dependent)
- Derivative of the objective function is not available
- High-dimensional problems (large $d$)
- Very large search space
- Useful in case “classical” search methods like quasi-Newton methods (BFGS) or conjugate gradient methods fail due to a non-convex or rugged search landscape (e.g. outliers, noise, local optima, sharp bends).



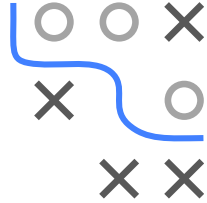
# CMA-ES: WRAP UP - LIMITATIONS

CMA-ES can be outperformed by other strategies in following cases:



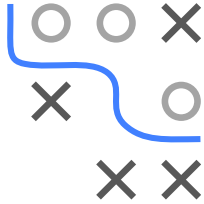
- Partly separable problems (i.e. optimization of $n$-dimensional objective function can be divided into a series of $d$ optimizations of every single parameter)
- Derivative of the objective function is easily available $\rightarrow$ Gradient Descend / Ascend
- Low dimensional problems (small $d$)
- Problems that can be solved by using a relative small number of function evaluations (e.g. $< 10d$ evaluations)

# CMA-ES: IPOP



- Many special forms and extensions of the “basic” CMA-ES exist
- CMA-ES efficiently minimizes unimodal objective functions and is in particular superior on ill-conditioned, non-separable problems
- Default population size $\lambda_{default}$ has been tuned for unimodal functions and however can get stuck in local optima on multi-modal functions, such that convergence to global optima is not guaranteed
- It could be shown that increasing the population size improves the performance of the CMA-ES on multi-modal functions
- **IPOP-CMA-ES** is a special form of restart-CMA-ES, where the *population size is increased for each restart* (IPOP) Double it and give it to the next iteration
- By increasing the population size the search characteristic becomes more global after each restart
- For the restart strategy CMA-ES is stopped whenever some stopping criterion is met, and an independent restart is launched with the population size increased by a factor of 2 (values between 1.5 and 5 are reasonable).

# CMA-ES: WRAP UP - BENCHMARK EAS

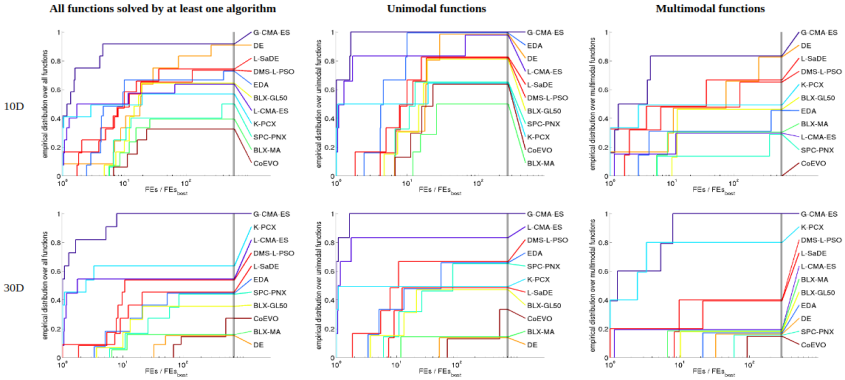


*Example:* Black-box optimization of 25 benchmark functions under thoroughly defined experimental and recording conditions for the 2005 IEEE Congress on Evolutionary Computation: Session on Real-Parameter Optimization.
17 papers were submitted, 11 were accepted, thereunder hybrid methods.

*Two of the Algorithms:*

- L-CMA-ES (Auger and Hansen. 2005a): A CMA evolution strategy with small population size and small initial step-size to emphasize on local search characteristics. Independent restarts are conducted until the target function value is reached or the maximum number of function evaluations is exceeded.
- G-CMA-ES (Auger and Hansen. 2005b): A CMA evolution strategy restarted with increasing population size (IPOP). Independent restarts are conducted with increasing population size until the target function value is reached or the maximum number of function evaluations is exceeded. With the initial small population size the algorithm converges fast, with the succeeding larger population sizes the global search performance is emphasized in subsequent restarts.

# CMA-ES: WRAP UP - BENCHMARK EAS




- Comparison of performance results from 11 algorithms for search space dimension 10 and 30 on different function subsets
- Expected number of function evaluations (FEs) to reach the target function value is normalized by the value of the best algorithm on the respective function $FEs_{best}$
- Calculation of the empirical cumulative distribution function of FEs / $FEs_{best}$ for each algorithm over different sets of functions in 10 and 30D
- Small values for FEs / $FEs_{best}$ and therefore large values of the graphs are preferable.