A0M33EOA:
EAs for Real-Parameter Optimization.
Evolution strategies. CMA-ES.

Petr Pošík

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Many parts adapted or taken from
Kubalík, J. *Real-Parameter Evolutionary Algorithms*.
Lecture slides for A4M33BIA course. 2016

Introduction

EAs for real-parameter optimization

Phenotype:

- Representation that the fitness function understands and is able to evaluate.
- Vector of real numbers.

Genotype?

- Representation to which the "genetic" operators are applied.
- **Binary vector** encoding the real numbers.
  - Discretization. Finite space.
  - Discretized problem is not the same as the original one.
  - Can miss the real function optimum. Results depend on the chosen precision of discretization.
  - Requires encoding and decoding process.
- **Vector of real numbers** (genotype = phenotype).
  - *Infinite domain* (theoretically), even for space with finite bounds.
  - Opportunity to exploit *graduality* or *continuity* of the function (slight changes in variables result in slight changes of the function value).
  - No need for encoding/decoding.

Contents

Contents:

- Standard selecto-recombinative genetic algorithms with binary representation.
- Standard selecto-recombinative genetic algorithms with real representation.
- Evolution strategies.
- ES with Covariance Matrix Adaptation (CMA-ES).

Standard EAs with Binary Encoding

Genotype-Phenotype Mapping

Assume a candidate solution to a function of 2 real numbers encoded as a single binary chromosome. Both numbers are encoded using 10 bits. The range of both numbers is $\langle -5.12, 5.11 \rangle$.

Genotype-Phenotype Mapping

Assume a candidate solution to a function of 2 real numbers encoded as a single binary chromosome. Both numbers are encoded using 10 bits. The range of both numbers is $\langle -5.12, 5.11 \rangle$.

Which pair of real numbers corresponds to chromosome `1000000000 0100000000` ?

A (512, 256)

B (2.56, 0)

C $(10^9, 10^8)$

D (0, -2.56)

Genotype-Phenotype Mapping

Mapping binary to real vector representation (2D example):

- 2D real domain, bound constraints $[x_l, x_r] \times [y_l, y_r]$.
- Using n bits to encode each parameter.

$$x_B = \boxed{x_1 \quad x_2 \quad \ldots \quad x_n \mid y_1 \quad y_2 \quad \ldots \quad y_n}$$

- How to compute phenotype from known genotype?

$$x_R = \boxed{x_l + (x_r - x_l)\frac{\text{bin2int}(x_1, \ldots, x_n)}{2^n - 1} \mid y_l + (y_r - y_l)\frac{\text{bin2int}(y_1, \ldots, y_n)}{2^n - 1}}$$

Genotype-Phenotype Mapping

Mapping binary to real vector representation (2D example):

- 2D real domain, bound constraints $[x_l, x_r] \times [y_l, y_r]$.
- Using n bits to encode each parameter.

$$x_B = \boxed{x_1 \quad x_2 \quad \ldots \quad x_n \;\big|\; y_1 \quad y_2 \quad \ldots \quad y_n}$$

- How to compute phenotype from known genotype?

$$x_R = \boxed{x_l + (x_r - x_l)\frac{\text{bin2int}(x_1,\ldots,x_n)}{2^n - 1} \;\Big|\; y_l + (y_r - y_l)\frac{\text{bin2int}(y_1,\ldots,y_n)}{2^n - 1}}$$

Where in the EA should we place the mapping?

Algorithm 1: Evolutionary Algorithm

begin
  $X \leftarrow$ `InitializePopulation()`
  $f \leftarrow$ `Evaluate(`X`)`
  $x_{BSF}, f_{BSF} \leftarrow$ `UpdateBSF(`X, f`)`
  while not `TerminationCondition()` **do**
    $X_N \leftarrow$ `Breed(`X, f`)` `// using certain breeding pipeline`
    $f_N \leftarrow$ `Evaluate(`X_N`)`
    $x_{BSF}, f_{BSF} \leftarrow$ `UpdateBSF(`X_N, f_N`)`
    $X, f \leftarrow$ `Join(`X, f, X_N, f_N`)` `// aka ‘‘replacement strategy’’`
  return x_{BSF}, f_{BSF}

Genotype-Phenotype Mapping

Mapping binary to real vector representation (2D example):

- 2D real domain, bound constraints $[x_l, x_r] \times [y_l, y_r]$.
- Using n bits to encode each parameter.

$$x_B = \boxed{x_1 \quad x_2 \quad \ldots \quad x_n \;\big|\; y_1 \quad y_2 \quad \ldots \quad y_n}$$

- How to compute phenotype from known genotype?

$$x_R = \boxed{x_l + (x_r - x_l)\frac{\text{bin2int}(x_1, \ldots, x_n)}{2^n - 1} \;\Big|\; y_l + (y_r - y_l)\frac{\text{bin2int}(y_1, \ldots, y_n)}{2^n - 1}}$$

Where in the EA should we place the mapping?

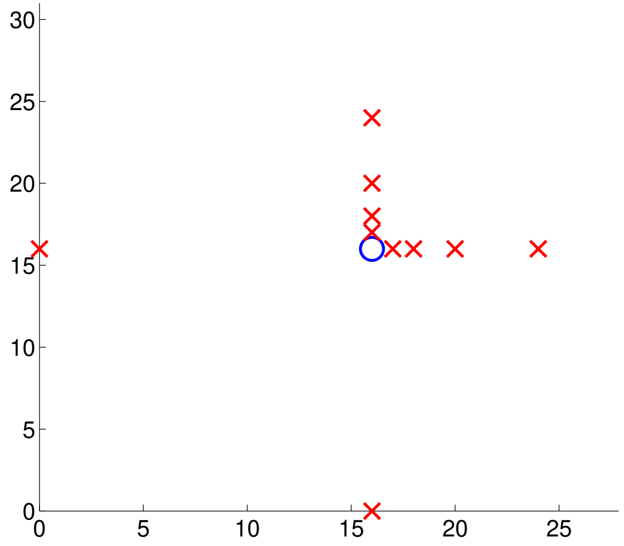
Algorithm 1: Evolutionary Algorithm with Genotype-Phenotype Mapping

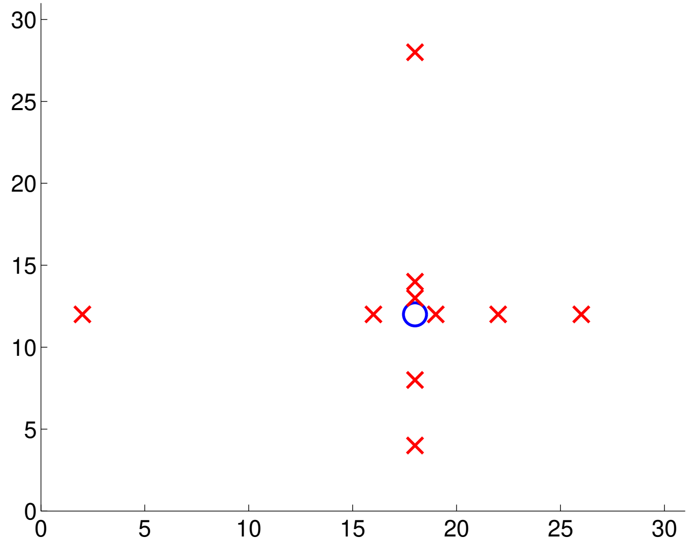
```
begin
    X ← InitializePopulation()
    f ← MapAndEvaluate(X)
    x_BSF, f_BSF ← UpdateBSF(X, f)
    while not TerminationCondition() do
        X_N ← Breed(X, f)                     // using certain breeding pipeline
        f_N ← MapAndEvaluate(X_N)
        x_BSF, f_BSF ← UpdateBSF(X_N, f_N)
        X, f ← Join(X, f, X_N, f_N)           // aka ‘‘replacement strategy’’
    return x_BSF, f_BSF
```

Effect of bit-flip mutation

The neighborhood of a point in the phenotype space generated by an operation applied on the genotype.

- Genotype: 10bit binary string.
- Phenotype: vector of 2 real numbers (in a discretized space).
- Operation: "bit-flip" mutation.





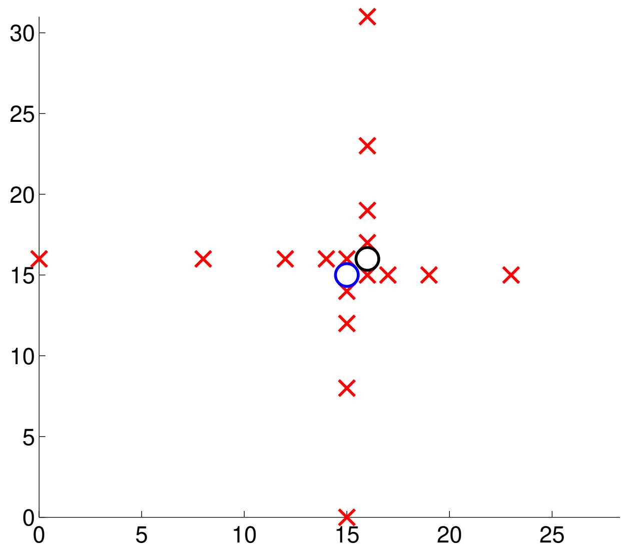
A very common situation:

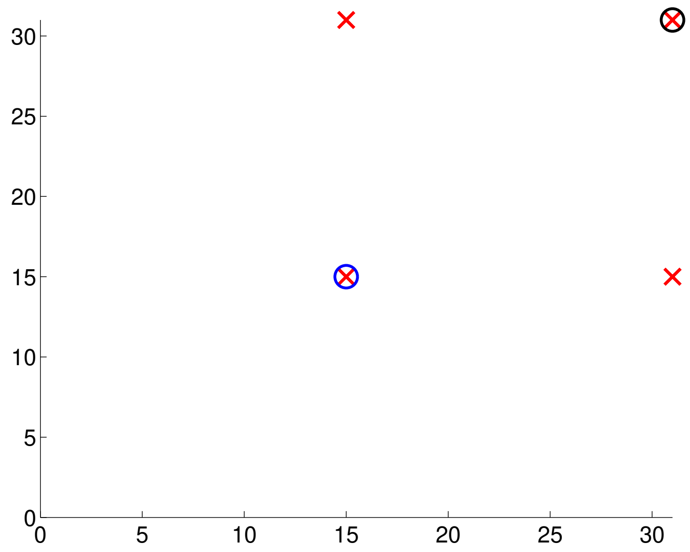
- Point which is locally optimal w.r.t. the phenotype is not locally optimal w.r.t. the genotype recombination operators. (GOOD! An opportunity to escape from LO!)
- Point which is locally optimal w.r.t. the genotype recombination operators is not locally optimal w.r.t. the phenotype. (BAD: Even the best solutions found by EA do not have to correspond to the real optima we look for!)

Effect of 1-point crossover

The neighborhood of a point in the phenotype space generated by an operation applied on the genotype.

- Genotype: 10bit binary string.
- Phenotype: vector of 2 real numbers (in a discretized space).
- Operation: 1-point crossover.

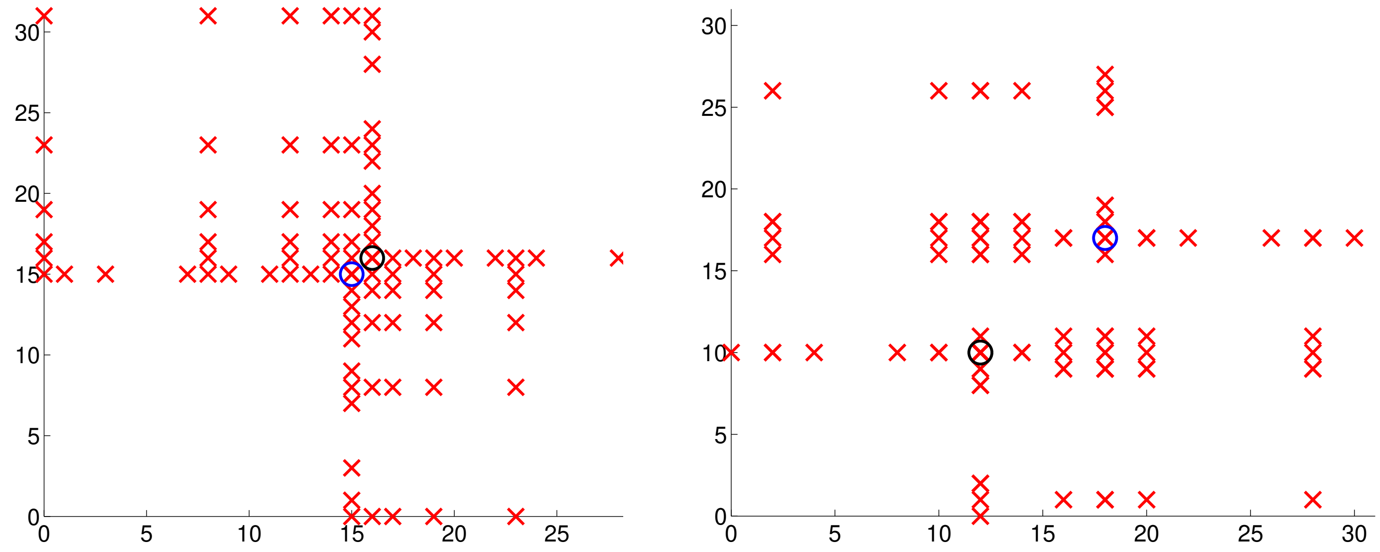




Effect of 2-point crossover

The neighborhood of a point in the phenotype space generated by an operation applied on the genotype.

- Genotype: 10bit binary string.
- Phenotype: vector of 2 real numbers (in a discretized space).
- Operation: 2-point crossover.



Summary

Binary encoding for real-parameter optimization:

- Results depend on the chosen discretization.
- The neighborhoods generated by binary crossover and mutation operators do not fit well to the "usual structures" of real-parameter functions.
- Can be useful for a rough exploration of the search space. (Then we can increase the resolution, or switch to real representation.)
- Using **Gray code** may help in certain situations, but does not solve the fundamental issues.

Standard EAs with Real Encoding

Recombination Operators for ESs with Real Encoding

Genotype = Phenotype = Vector of real numbers!

Standard mutation operators:

- Gaussian mutation
- Cauchy mutation

Standard recombination operators:

- Simple (1-point) Crossover: same as for binary strings
- Uniform Crossover: same as for binary strings
- Average Crossover
- Arithmetic Crossover
- Flat Crossover
- Blend Crossover BLX-(α)

Advanced recombination operators:

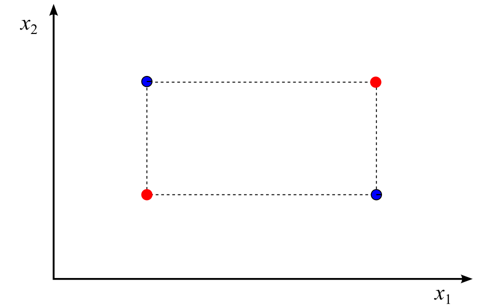
- Simplex Crossover (SPX)
- Unimodal Normal Distribution Crossover (UNDX)
- Parent-Centric Crossover (PCX)

Standard Recombination Operators for Real EAs

Assume that $\boldsymbol{x}^1 = (x^1_1, \ldots, x^1_n)$ and $\boldsymbol{x}^2 = (x^2_1, \ldots, x^2_n)$ are two parents.

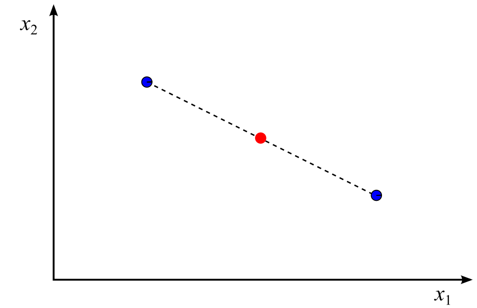
- **Simple (1-point) Crossover**: a position $i \in 1, 2, \ldots, n-1$ is randomly chosen, and two offspring chromosomes $\boldsymbol{y}^1$ and $\boldsymbol{y}^2$ are built as follows:

$$\boldsymbol{y}^1 = (x^1_1, \ldots, x^1_i, x^2_{i+1}, \ldots, x^2_n)$$
$$\boldsymbol{y}^2 = (x^2_1, \ldots, x^2_i, x^1_{i+1}, \ldots, x^1_n)$$



- **Average Crossover**: an offspring $\boldsymbol{y}$ is created as and average of the parents:

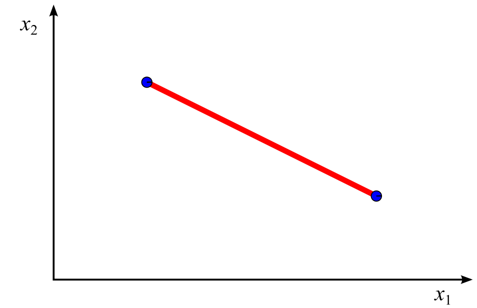
$$\boldsymbol{y} = \frac{1}{2}(\boldsymbol{x}^1 + \boldsymbol{x}^2)$$



- **Arithmetic Crossover**: an offspring is created as a *weighted average* of the parents:

$$\boldsymbol{y} = r \cdot \boldsymbol{x}^1 + (1 - r) \cdot \boldsymbol{x}^2,$$

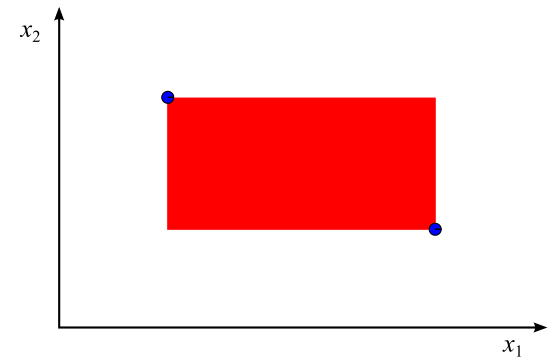
where $r \in (0, 1)$ is a constant, or varies with regard to the number of generations made, or is randomly chosen.



Standard Recombination Operators for Real EAs (cont.)

- **Flat Crossover:** an offspring $\boldsymbol{y} = (y_1, \ldots, y_n)$ is created such that each y_i is sampled with uniform distribution from interval

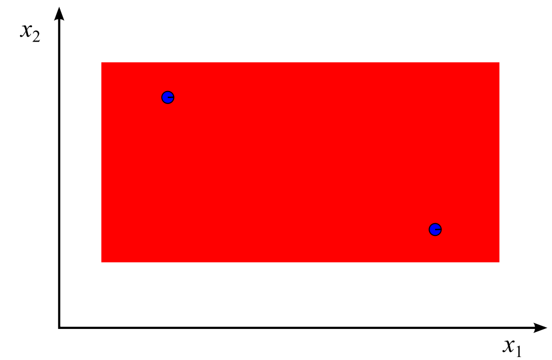
$$y_i \in [\min(x_i^1, x_i^2), \max(x_i^1, x_i^2)].$$



- **Blend Crossover:** an offspring $\boldsymbol{y} = (y_1, \ldots, y_n)$ is created such that each y_i is sampled with uniform distribution from interval

$$y_i \in [c_{\min} - \alpha I, c_{\max} + \alpha I],$$

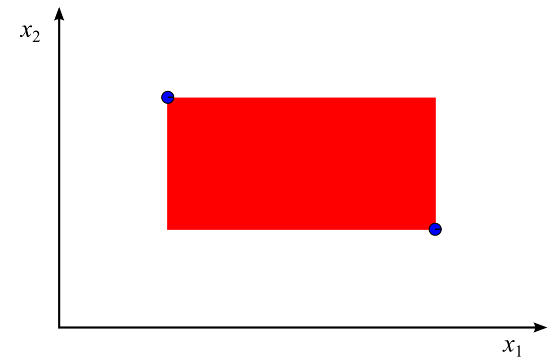
where $c_{\min} = \min(p_i^1, p_i^2)$, $c_{\max} = \max(p_i^1, p_i^2)$, $I = c_{\max} - c_{\min}$, and $\alpha > 0$.



Standard Recombination Operators for Real EAs (cont.)

- **Flat Crossover:** an offspring $\boldsymbol{y} = (y_1, \ldots, y_n)$ is created such that each y_i is sampled with uniform distribution from interval

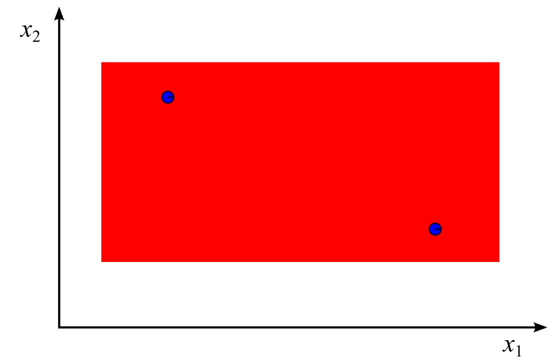
$$y_i \in [\min(x_i^1, x_i^2), \max(x_i^1, x_i^2)].$$



- **Blend Crossover:** an offspring $\boldsymbol{y} = (y_1, \ldots, y_n)$ is created such that each y_i is sampled with uniform distribution from interval

$$y_i \in [c_{\min} - \alpha I, c_{\max} + \alpha I],$$

where $c_{\min} = \min(p_i^1, p_i^2)$,
$c_{\max} = \max(p_i^1, p_i^2)$, $I = c_{\max} - c_{\min}$,
and $\alpha > 0$.

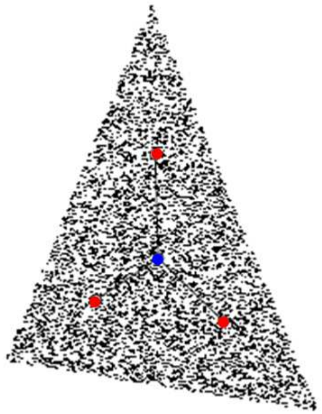


Characteristics:

- Simple, and average crossovers are deterministic; arithmetic crossover does not introduce enough diversity either.
- Simple, flat, and blend crossovers are not rotationally invariant.

Advanced Operators

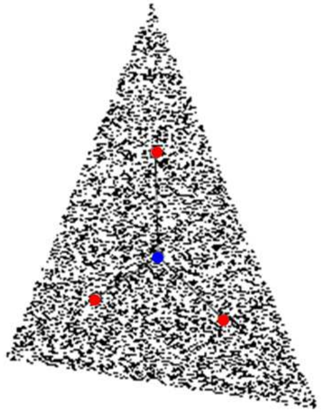
Simplex Crossover (SPX):



- Generates offspring around the mean of the μ parents
- with uniform distribution
- in a simplex which is $\sqrt{\mu+1}$ times bigger than the parent simplex.

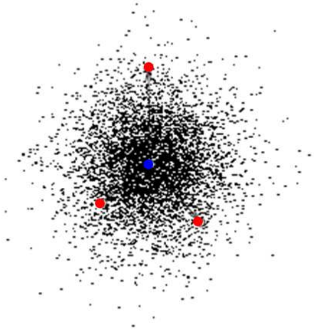
Advanced Operators

Simplex Crossover (SPX):



- Generates offspring around the mean of the μ parents
- with uniform distribution
- in a simplex which is $\sqrt{\mu+1}$ times bigger than the parent simplex.

Unimodal Normal Distribution Crossover (UNDX):

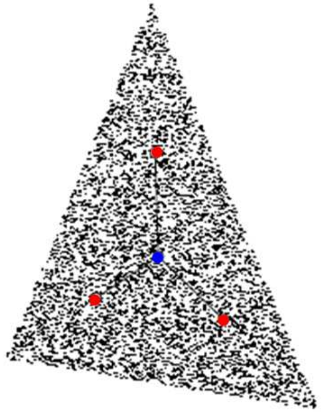


- Generates offspring around the mean of the μ parents
- with multivariate normal distribution.
- Preserves the correlation among parameters well.

Advanced Operators

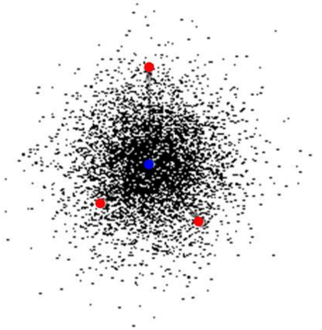
Simplex Crossover (SPX):

- Generates offspring around the mean of the μ parents
- with uniform distribution
- in a simplex which is $\sqrt{\mu+1}$ times bigger than the parent simplex.



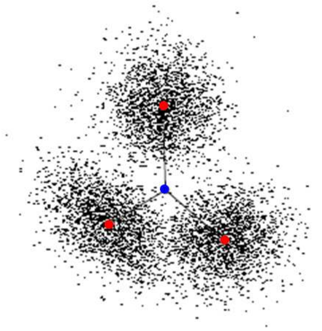
Unimodal Normal Distribution Crossover (UNDX):

- Generates offspring around the mean of the μ parents
- with multivariate normal distribution.
- Preserves the correlation among parameters well.



Parent-Centric Crossover (PCX):

- Generates offspring around one of the parents
- with multivariate normal distribution.
- The distribution shape is determined by the relative positions of the parents.
- Similar to adaptive mutation.



Generalized Generation Gap (G3) Algorithm

G3 [Deb05]: Elite preserving, steady-state, computationally fast. Special breeding pipeline and replacement operator.

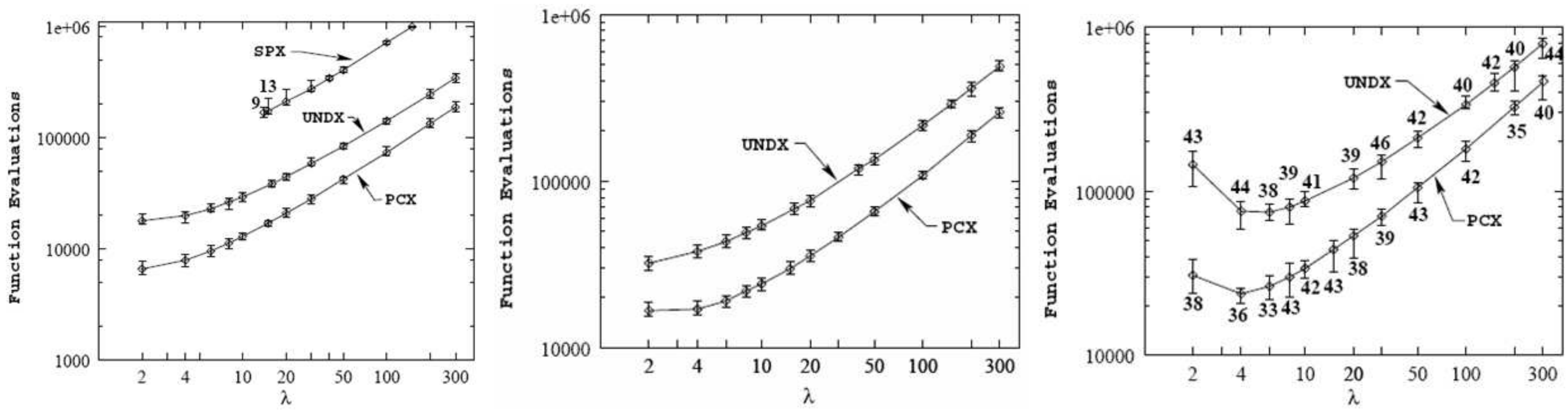
1. From the population $P(t)$, select the best parent and $(\mu - 1)$ other parents randomly.
2. Generate λ offspring from μ parents using a recombination scheme.
3. Choose two parents at random from μ parents.
4. Form a combined subpopulation of chosen two parents and λ offspring, choose the best two solutions and replace the chosen two parents with these solutions.

Generalized Generation Gap (G3) Algorithm

G3 [Deb05]: Elite preserving, steady-state, computationally fast. Special breeding pipeline and replacement operator.

1. From the population $P(t)$, select the best parent and $(\mu - 1)$ other parents randomly.
2. Generate λ offspring from μ parents using a recombination scheme.
3. Choose two parents at random from μ parents.
4. Form a combined subpopulation of chosen two parents and λ offspring, choose the best two solutions and replace the chosen two parents with these solutions.

Comparisons of UNDX, SPX and PCX with the G3 model on Ellipsoidal, Schwefel's, and Generalized Rosenbrock's functions for $D = 20$.



[Deb05] K. Deb. A population-based algorithm-generator for real-parameter optimization. *Soft Computing*, 9(4):236–253, April 2005.

Summary

Selecto-recombinative standard EAs with real encoding

- often use the same algorithm and breeding pipeline as binary EAs,
- although a specialized pipeline can be designed (e.g., G3).
- They use different mutation and crossover operators.

Operators for real encoding:

- Much wider range of possibilities than in binary space.
- Generally, there is no single best operator for all problems.
- Operators resulting in normal distribution of offspring usually work better for practical problems.

Evolution Strategies (ES)

Evolution Strategies: Introduction

"The European branch of Evolutionary Computation."

- Originated in Germany in 1960's (Ingo Rechenberg and Hans-Paul Schwefel).
- ES use the natural representation of vectors in R^D as "chromosomes".
- ES originally relied on *mutation and selection* only; recombination was added later.
- Mutation is performed by adding a random vector distributed according to multivariate Gaussian with covariance matrix $\sigma\mathbf{I}$, $\text{diag}(\sigma_1, \ldots, \sigma_D)$, or general C.
- Special feature: *built-in* adaptation of mutation parameters!

Notation: $(\mu \overset{+}{,} \lambda)$-ES

- μ is the *population size* (and number of parents),
- λ is the *number of offspring* created each generation,
- $+$ or $,$ denote the *replacement strategy*:
  - $,$ is *generational* strategy: old population is discarded, new population of μ parents is chosen from the λ generated offspring.
  - $+$ is *steady-state* strategy: old population is joined with the new offspring, new population of μ parents is chosen from the joined $\mu + \lambda$ individuals.

Notation: $(\mu/\rho \overset{+}{,} \lambda)$-ES

- *Recombination* (usually deterministic), choose ρ individuals out of μ parents, $\mu \geq \rho$.
- Sometimes, subscript to ρ is used to denote the type of recombination, e.g., ρ_I for intermediate recombination (average), or ρ_W for weighted recombination (weighted average). Other recomb. ops from Real EAs can be used in principle.

Evolution Strategy Algorithm

ES use ordinary EA template (see lecture 1), with only slightly changed pipeline:

Algorithm 2: ES Breeding Pipeline

Input: Population X of μ individuals, with their fitness in f.
Number of parents ρ. Number of offspring λ.
Output: Population X_N of λ offspring.

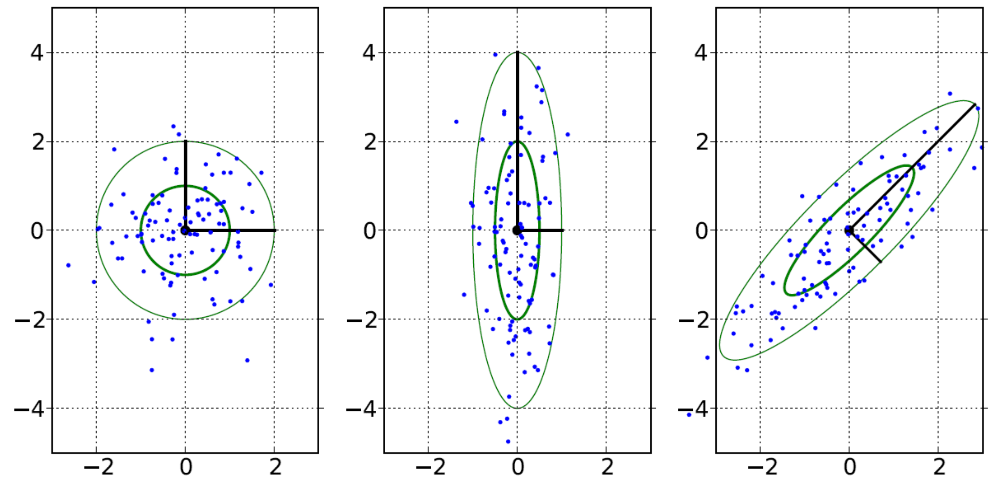
```
1 begin
2     X_N ← ∅
3     for i ← 1, ..., λ do
4         X_S ← SelectParents(X, f)                 // ρ parents
5         x_R ← Recombine(X_S)                      // usually only single offspring
6         x_N ← Mutate(x_R)
7         X_N ← X_N ∪ {x_N}
8     return X_N
```

- The `join()` operation then forms new population for the next generation by choosing the best μ individuals either from X_N (comma strategy) or from $X \cup X_N$ (plus strategy).
- Very often $\rho = \mu$, resulting in $(\mu/\mu \overset{+}{,} \lambda) - ES$. All offspring are then centered around a single vector $\mathbf{x}_R$. Lines 4 and 5 can thus be removed from the `for`-loop and placed before it.

Question: Gaussian mutation

Which of the following figures depict Gaussian distributions with covariance matrices given by

- $\boldsymbol{C} = \sigma^2 \boldsymbol{I}$ and
- $\boldsymbol{C} = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_D^2)$?



A left and middle

B left and right

C middle and right

D right and left

Gaussian Mutation

Gaussian mutation: the mutated offspring $\boldsymbol{y}$ are distributed around the original individual $\boldsymbol{x}$ as

$$\boldsymbol{y} \sim N(\boldsymbol{x}, \boldsymbol{C}) \sim \boldsymbol{x} + N(0, \boldsymbol{C}) \sim \boldsymbol{x} + \boldsymbol{C}^{\frac{1}{2}} N(0, \boldsymbol{I}),$$

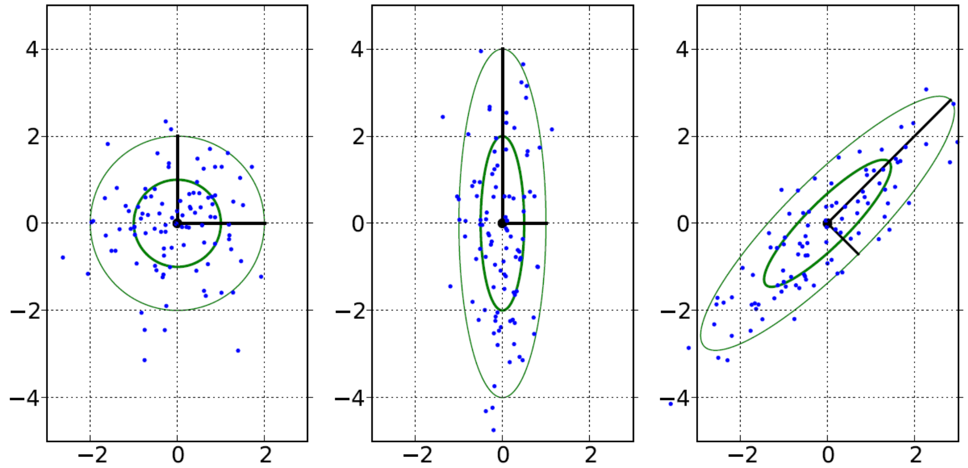
where $N(\boldsymbol{\mu}, \boldsymbol{C})$ is a **multivariate Gaussian distribution** with probability density function in R^D

$$f_D(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{C}) = \frac{1}{\sqrt{(2\pi)^D \det(\boldsymbol{C})}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{C}^{-1} (\boldsymbol{x}-\boldsymbol{\mu})\right)$$

Parameters:

- $\boldsymbol{\mu}$: location of the distribution. When used for mutation, $\boldsymbol{\mu} = 0$ to prevent bias.
- $\boldsymbol{C}$: Covariance matrix; determines the shape of the distribution:
  - **Isotropic:** $\boldsymbol{C} = \sigma^2 \boldsymbol{I}$
  - **Axis-parallel:** $\boldsymbol{C} = \operatorname{diag}(\sigma_1^2, \ldots, \sigma_D^2)$
  - **General:** $\boldsymbol{C}$ positive definite

  How many degrees of freedom (free parameters) do these have?



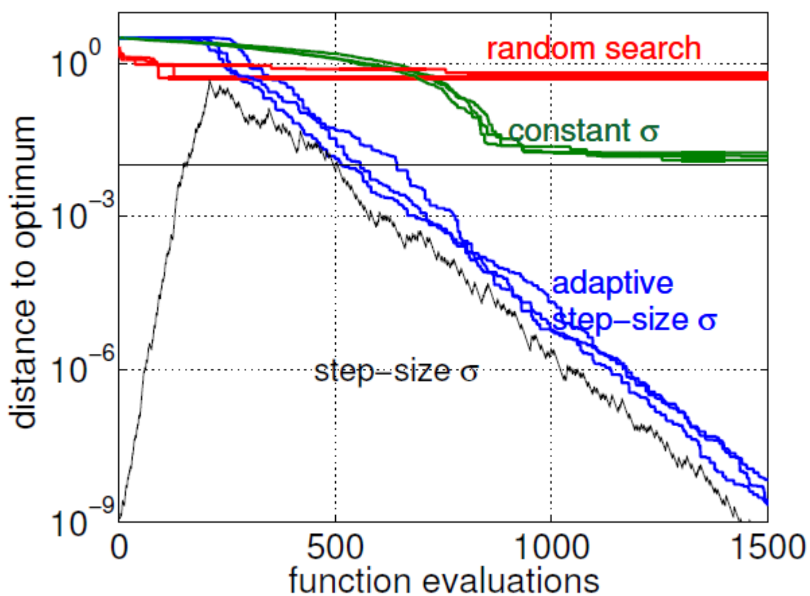
How to set up the parameters of covariance matrix?

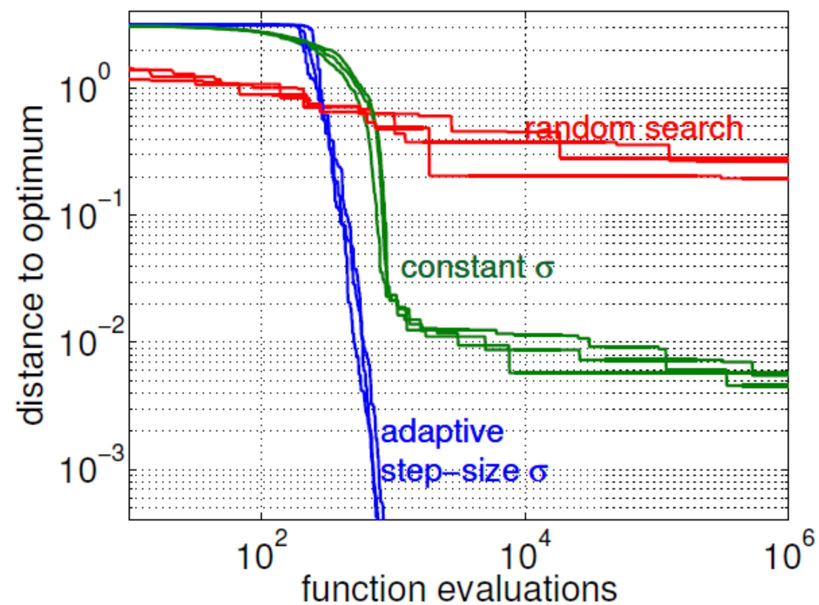
Adaptation of Mutation Parameters

Adaptation of mutation parameters is **key to ES design!**

Example: $(1+1)$-ES (hill-climber) with isotropic mutation on Sphere function: $f = \sum_i x_i^2$

- Random search vs
- $(1+1)$-ES with constant $\sigma = 10^{-2}$ vs
- $(1+1)$-ES with σ adapted using $\frac{1}{5}$-rule with $\sigma_0 = 10^{-9}$





- Random search: inefficient.
- Constant σ: initially too small value, appropriate value between 600 and 800 evals, too large value at the end.
- Adaptive σ: near-optimal value during (almost) the whole run!

1/5 Success Rule

Progress rate φ: a ratio of the distance covered towards the optimum and the number of evaluations required to reach this distance.

Rechenberg analyzed the behavior of (1+1)-ES on 2 simple functions:

- Corridor function: $f_1(\boldsymbol{x}) = x_1$ if $|x_i| < 1$ for $i \in (2, \ldots, D)$, otherwise $f_1(\boldsymbol{x}) = \infty$
- Sphere function: $f_2(\boldsymbol{x}) = \sum_i x_i^2$

Findings:

- In both cases, the optimal step size σ^{opt} is inversely proportional to the dimension of the space D (number of variables).
- The maximum progress rate φ^{max} is also inversely proportional to D.
- For the optimal step sizes, the following probabilities of a successful mutation were obtained:
  - $p_{S,1}^{opt} = 1/(2e) \approx 0.184$
  - $p_{S,2}^{opt} \approx 0.270$

1/5 Success Rule

Progress rate φ: a ratio of the distance covered towards the optimum and the number of evaluations required to reach this distance.

Rechenberg analyzed the behavior of (1+1)-ES on 2 simple functions:

- Corridor function: $f_1(\boldsymbol{x}) = x_1$ if $|x_i| < 1$ for $i \in (2, \ldots, D)$, otherwise $f_1(\boldsymbol{x}) = \infty$
- Sphere function: $f_2(\boldsymbol{x}) = \sum_i x_i^2$

Findings:

- In both cases, the optimal step size σ^{opt} is inversely proportional to the dimension of the space D (number of variables).
- The maximum progress rate φ^{max} is also inversely proportional to D.
- For the optimal step sizes, the following probabilities of a successful mutation were obtained:
  - $p_{S,1}^{opt} = 1/(2e) \approx 0.184$
  - $p_{S,2}^{opt} \approx 0.270$

1/5 success rule: *To obtain nearly optimal (local) performance of the (1+1)-ES in real-valued search spaces, tune the mutation step in such a way that the (measured) success rate is about 1/5.*

- If it is greater than $1/5$, increase the mutation step σ; if it is less, decrease σ.

In practice, the $1/5$ success rule has been mostly superseded by more sophisticated methods. However, its conceptual insight remain remarkably valuable.

(1+1)-ES with 1/5 rule

Algorithm 3: (1+1)-ES with 1/5 rule

```
Input: D ∈ N⁺, d ≈ √(D+1)
begin
    x ← Initialize()
    while not TerminationCondition() do
        x_N ← x + σ𝒩(0, I)                          // mutation/perturbation
        b ← BetterThan(x_N, x)                       // Mutation successful?
        σ ← σ (exp (𝟙(b) − 1/5))^(1/d)              // 1/5 rule
        if b then
            x ← x_N
```

- $\mathbb{1}(b)$ is an indicator function:

$$\mathbb{1}(b) = \begin{cases} 1 & \text{iff } b \text{ is true,} \\ 0 & \text{iff } b \text{ is false.} \end{cases}$$

- In $(1+\lambda)$-ES, replace b with the proportion r of improving mutations in generation.
- Other implementations are possible.

Self-adaptation

Self-adaptation:

- **Strategy parameters are part of the chromosome!** $\mathbf{x} = (x_1, \dots, x_D, \sigma_1, \dots, \sigma_D)$
- Parameters undergo evolution together with the decision variables.
- Each individual holds information how it shall be mutated.

Example: assuming axis-parallel normal distribution is used,

- mutation of $\mathbf{x} = (x_1, \dots, x_D, \sigma_1, \dots, \sigma_D)$ creates an offspring individual

$$\mathbf{x}' = (x'_1, \dots, x'_D, \sigma'_1, \dots, \sigma'_D)$$

by mutating each part in a different way:

$$\sigma'_i \leftarrow \sigma_i \cdot \exp(\tau \cdot \mathcal{N}(0,1)) \qquad x'_i \leftarrow x_i + \sigma'_i \cdot \mathcal{N}(0,1)$$

- Intuition: a "bad" σ' probably generates bad $\mathbf{x}'$ and is eliminated by selection.

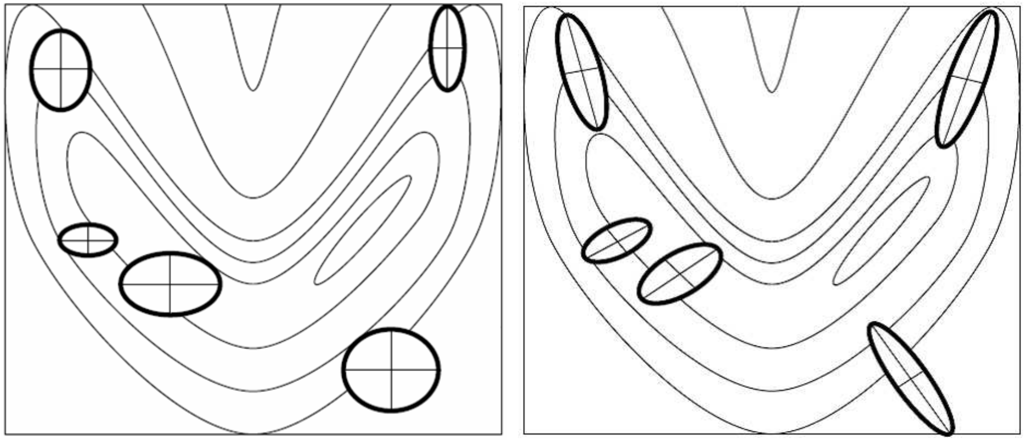
Remarks:

- An algorithm can adapt a global step size σ and coordinate-wise step sizes separately, such that the resulting coordinate-wise st. dev. is given as $\sigma \cdot \sigma_i$.
- The global step size may be adapted e.g. by the 1/5-rule.

Generalizations and issues

Generalizing from

- axis-parallel mutation distributions with D strategy parameters to
- general normal mutation distributions with full cov. matrix requires adaptation of $\frac{1}{2}D(D+1)$ strategy parameters!



Issues with self-adaptation: **selection noise** (the more parameters, the worse)!

- The intuition from the previous slide does not work much!
- A good offspring may be generated with poor strategy parameter settings (poor setting survives), or a bad offspring may be generated with good parameter settings (good setting is eliminated).

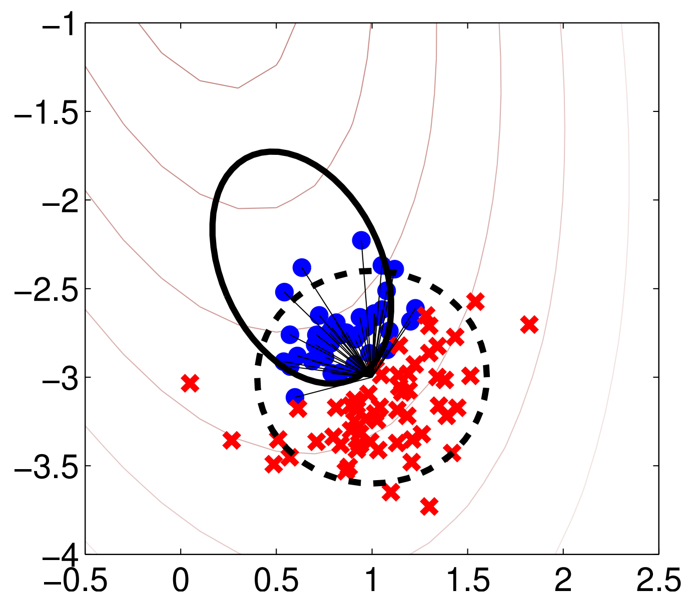
Solutions: derandomization via

- reducing the number of mutation distribution: (1,λ)-ES, ($\mu/\mu,\lambda$)-ES, and
- accumulating info in time (evolution paths).

CMA-ES

Evolutionary strategy with covariance matrix adaptation [HO01]:

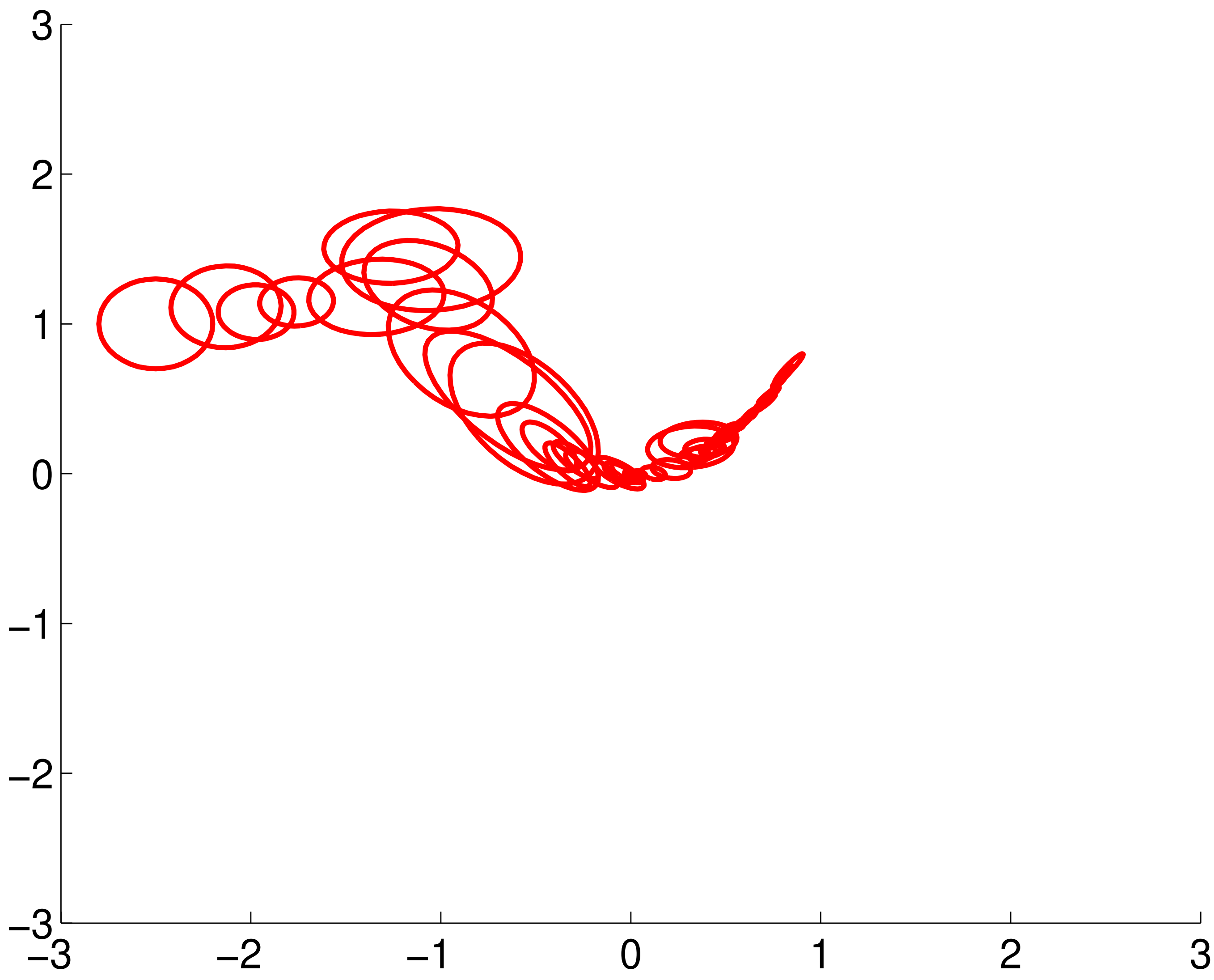
- Currently, *de facto* standard in real-parameter optimization.
- $(\mu/\mu_W, \lambda)$-ES: recombinative, mean-centric
- Offspring is created by sampling from a single normal distribution.
- Successful mutation steps are used to adapt the mean $\boldsymbol{x}$ and the covariance matrix $\boldsymbol{C}$ of the distribution.
- Accumulates the successful steps over many generations.



[HO01] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

CMA-ES Demo

CMA-ES on the Rosenbrock function:



CMA-ES Code (1)

CMA-ES is a **complex, but carefully designed** and tuned algorithm!

Really? It does not seem so from the pseudocode below...

Algorithm 4: CMA-ES

begin
  Initialize: $\boldsymbol{x} \in \mathbb{R}^D, \sigma \in \mathbb{R}^D_+, \boldsymbol{C} = \boldsymbol{I}$.
  while not `TerminationCondition()` **do**
    $\mathcal{M} \leftarrow$ `SampleDistribution(`$\lambda, \mathcal{N}(\boldsymbol{x}, \sigma^2\boldsymbol{C})$`)`
    $\mathcal{P} \leftarrow$ `SelectBest(`$\mu, \mathcal{M}$`)`
    $(\boldsymbol{x}, \sigma, \boldsymbol{C}) \leftarrow$ `UpdateModel(`$\boldsymbol{x}, \sigma, \boldsymbol{C}, \mathcal{P}$`)`
  return $\boldsymbol{x}$

Hm, ok, how is the Normal distribution actually sampled?

CMA-ES Code (2)

CMA-ES with the distribution sampling step expanded:

Algorithm 5: CMA-ES

begin
  Initialize: $\boldsymbol{x} \in \mathbb{R}^D, \sigma \in \mathbb{R}^D_+, \boldsymbol{C} = \boldsymbol{I}$.
  while not `TerminationCondition()` **do**
    for $k \in 1, \ldots \lambda$ **do**
      $\boldsymbol{z}_k \leftarrow \mathcal{N}(0, \boldsymbol{I})$
      $\boldsymbol{x}_k \leftarrow \boldsymbol{x} + \sigma \boldsymbol{C}^{\frac{1}{2}} \times \boldsymbol{z}_k$
    $\mathcal{P} \leftarrow$ `SelectBest(`$\mu, \{\boldsymbol{z}_k, f(\boldsymbol{x}_k)) | 1 \le k \le \lambda\}$`)`
    $(\boldsymbol{x}, \sigma, \boldsymbol{C}) \leftarrow$ `UpdateModel(`$\boldsymbol{x}, \sigma, \boldsymbol{C}, \mathcal{P}$`)`
  return $\boldsymbol{x}$

Remarks:

- All individuals exist in 2 “versions”: $\boldsymbol{z}_k$ distributed as $\mathcal{N}(0, \boldsymbol{I})$, and $\boldsymbol{x}_k$ distributed as $\mathcal{N}(\boldsymbol{x}, \sigma^2 \boldsymbol{C})$.
- $\boldsymbol{x}_k$ are used just as an intermediate step for evaluation!
- $\boldsymbol{z}_k$ are used for model update via the population of selected parents $\mathcal{P}$.

OK, that’s not that complex. What about the model update?

CMA-ES Code (3)

CMA-ES with the model update step expanded:

Algorithm 6: CMA-ES

begin
  Initialize: $\boldsymbol{x} \in \mathbb{R}^D, \sigma \in \mathbb{R}^D_+, \boldsymbol{C} = \boldsymbol{I}, \boldsymbol{s}_\sigma = 0, \boldsymbol{s}_c = 0.$
  while not `TerminationCondition()` **do**
    for $k \in 1, \ldots \lambda$ **do**
      $\boldsymbol{z}_k \leftarrow \mathcal{N}(0, \boldsymbol{I})$
      $\boldsymbol{x}_k \leftarrow \boldsymbol{x} + \sigma \boldsymbol{C}^{\frac{1}{2}} \times \boldsymbol{z}_k$
    $\mathcal{P} \leftarrow$ `SelectBest`$(\mu, \{\boldsymbol{z}_k, f(\boldsymbol{x}_k)) | 1 \le k \le \lambda\})$
    $\boldsymbol{s}_\sigma \leftarrow (1 - c_\sigma)\boldsymbol{s}_\sigma + \sqrt{c_\sigma(2 - c_\sigma)}\sqrt{\mu_w} \sum_{\boldsymbol{z}_k \in \mathcal{P}} w_k \boldsymbol{z}_k$ `// search path for` σ
    $\boldsymbol{s}_c \leftarrow (1 - c_c)\boldsymbol{s}_c + h_\sigma \sqrt{c_c(2 - c_c)}\sqrt{\mu_w} \sum_{\boldsymbol{z}_k \in \mathcal{P}} w_k \boldsymbol{C}^{\frac{1}{2}} \boldsymbol{z}_k$ `// search path for` $\boldsymbol{C}$
    $\sigma \leftarrow \sigma \cdot \exp^{c_\sigma / d} \left(\frac{\|\boldsymbol{s}_\sigma\|}{\mathbb{E}\|\mathcal{N}(0, \boldsymbol{I})\|} - 1 \right)$ `// update` σ
    $\boldsymbol{C} \leftarrow (1 - c_1 + c_h - c_\mu)\boldsymbol{C} + c_1 \boldsymbol{s}_c \boldsymbol{s}_c^T + c_\mu \sum_{\boldsymbol{z}_k \in \mathcal{P}} w_k \boldsymbol{C}^{\frac{1}{2}} \boldsymbol{z}_k (\boldsymbol{C}^{\frac{1}{2}} \boldsymbol{z}_k)^T$ `// update` $\boldsymbol{C}$
    $\boldsymbol{x} \leftarrow \boldsymbol{x} + c_m \sigma \boldsymbol{C}^{\frac{1}{2}} \sum_{\boldsymbol{z}_k \in \mathcal{P}} w_k \boldsymbol{z}_k$ `// update` $\boldsymbol{x}$
  return $\boldsymbol{x}$

Remark: Two search paths, $\boldsymbol{s}_\sigma$ and $\boldsymbol{s}_c$, are part of the algorithm state, together with $\boldsymbol{x}, \sigma$, and $\boldsymbol{C}$. They accumulate the algorithm moves accross iterations.

And what are all those $c_1, c_h, c_\mu, \ldots$?

CMA-ES Code

The full CMA-ES pseudocode:

Algorithm 7: CMA-ES

Given: $D \in \mathbb{N}_+, \lambda \geq 5, \mu \approx \lambda/2, w_k = w'(k)/\sum_{k=1}^{\mu} w'(k), w'(k) = \log(\lambda/2 + 1/2) - \log \operatorname{rank}(f(\boldsymbol{x}_k)),$
$\mu_w = 1/\sum_{k=1}^{\mu} w_k^2, c_\sigma \approx \mu_w/(D+\mu_w), d \approx 1 + \sqrt{\mu_w/D}, c_c \approx (4+\mu_w/D)/(D+4+2\mu_w/D),$
$c_1 \approx 2/(D^2+\mu_w), c_\mu \approx \mu_w/(D^2+\mu_w), c_m = 1.$

begin
- Initialize: $\boldsymbol{x} \in \mathbb{R}^D, \sigma \in \mathbb{R}^D_+, \boldsymbol{C} = \boldsymbol{I}, \boldsymbol{s}_\sigma = 0, \boldsymbol{s}_c = 0.$
- **while not** `TerminationCondition()` **do**
  - **for** $k \in 1, \ldots \lambda$ **do**
    - $\boldsymbol{z}_k \leftarrow \mathcal{N}(0, \boldsymbol{I})$
    - $\boldsymbol{x}_k \leftarrow \boldsymbol{x} + \sigma \boldsymbol{C}^{\frac{1}{2}} \times \boldsymbol{z}_k$
  - $\mathcal{P} \leftarrow$ `SelectBest`$(\mu, \{\boldsymbol{z}_k, f(\boldsymbol{x}_k)) | 1 \leq k \leq \lambda\})$
  - $\boldsymbol{s}_\sigma \leftarrow (1-c_\sigma)\boldsymbol{s}_\sigma + \sqrt{c_\sigma(2-c_\sigma)}\sqrt{\mu_w} \sum_{\boldsymbol{z}_k \in \mathcal{P}} w_k \boldsymbol{z}_k$ `// search path for` σ
  - $\boldsymbol{s}_c \leftarrow (1-c_c)\boldsymbol{s}_c + h_\sigma \sqrt{c_c(2-c_c)}\sqrt{\mu_w} \sum_{\boldsymbol{z}_k \in \mathcal{P}} w_k \boldsymbol{C}^{\frac{1}{2}} \boldsymbol{z}_k$ `// search path for` $\boldsymbol{C}$
  - $\sigma \leftarrow \sigma \cdot \exp^{c_\sigma/d}\left(\frac{\|\boldsymbol{s}_\sigma\|}{\mathbb{E}\|\mathcal{N}(0,\boldsymbol{I})\|} - 1\right)$ `// update` σ
  - $\boldsymbol{C} \leftarrow (1 - c_1 + c_h - c_\mu)\boldsymbol{C} + c_1 \boldsymbol{s}_c \boldsymbol{s}_c^T + c_\mu \sum_{\boldsymbol{z}_k \in \mathcal{P}} w_k \boldsymbol{C}^{\frac{1}{2}} \boldsymbol{z}_k (\boldsymbol{C}^{\frac{1}{2}} \boldsymbol{z}_k)^T$ `// update` $\boldsymbol{C}$
  - $\boldsymbol{x} \leftarrow \boldsymbol{x} + c_m \sigma \boldsymbol{C}^{\frac{1}{2}} \sum_{\boldsymbol{z}_k \in \mathcal{P}} w_k \boldsymbol{z}_k$ `// update` $\boldsymbol{x}$
- return $\boldsymbol{x}$

where $h_\sigma = \mathbb{1}(\|\boldsymbol{s}_\sigma\|^2/D < 2 + 4/(D+1)), c_h = c_1(1-h_\sigma^2)c_c(2-c_c)$, and $\boldsymbol{C}^{\frac{1}{2}}$ is the unique symmetric positive definite matrix obeying $\boldsymbol{C}^{\frac{1}{2}} \times \boldsymbol{C}^{\frac{1}{2}} = \boldsymbol{C}$. All c-coeficients are ≤ 1.

CMA-ES Summary

CMA-ES is **quasi parameter-free:**

- It has a lot of internal parameters, but almost all of them are carefully set by the algorithm itself.
- The user has to specify only
  - initial solution $\boldsymbol{x}$,
  - initial step size σ, and
  - the number of offspring λ (but even that can be set based on the search space dimension).

CMA-ES Variants:

- Reducing the local search character of CMA-ES:
  - **IPOP-CMA-ES:** Restart CMA-ES several times, making the population twice as large each time.
  - **BIPOP-CMA-ES:** Restart CMA-ES many times in 2 regimes: IPOP, and small-pop (spend similar number of evaluations in IPOP and small-pop modes.
- Reducing the number of parameters to be adapted:
  - **L-CMA-ES:** Smaller memory requirements, suitable for high-dimensional spaces, limited adaptation.
- Learning from unsuccessful mutations:
  - **Active CMA-ES:** negative weights allowed during covariance update. Gotcha: $\boldsymbol{C}$ may lose positive definiteness!

Relations to other algorithms

Estimation of Distribution Algorithms (**EDA**):

- CMA-ES can be considered an instance of EDA.
- EDAs template: sample from probabilistic model, and update model based on good individuals (i.e., the same as CMA-ES uses).

Natural Evolution Strategies (**NES**):

- Idea: the update of all distribution parameters should be based on the same fundamental principle.
- NES proposed as more principled alternative to CMA-ES.
- Later it was found that CMA-ES actually implements the underlying NES principle.

Information Geometric Optimization (**IGO**):

- Framework unifying many successful algorithms from discrete and continuous domains.
- CMA-ES and NES (and many EDA variants, see the next lecture) can be derived as special instances of IGO.

Summary

Learning outcomes

After this lecture, a student shall be able to

- perform the mapping of chromosomes from binary to real space when using binary encoding for real-parameter optimization;
- describe and exemplify the effects of such a genotype-phenotype mapping on the neighborhood structures induced by mutation and crossover;
- give examples and describe some mutation and crossover operators designed for spaces of real number vectors;
- explain the main features of ES and differences to GAs;
- explain the notation $(\mu/\rho \overset{+}{,} \lambda)$-ES;
- describe the differences between mutation with isotropic, axis-parallel, and general Gaussian distribution, including the relation to the form of the covariance matrix, and the number of parameters that must be set/adapted for each of them;
- explain and use two simple methods of mutation step size adaptation (1/5 rule and self-adaptation);
- write a high-level pseudocode of CMA-ES and describe CMA-ES in the $(\mu/\rho \overset{+}{,} \lambda)$ notation;
- implement DE algorithm;
- explain the basic forms of DE mutation and crossover.