Metaheuristics
Evolution Strategies

Fernando Lobo

Universidade do Algarve

Overview

- Like GAs, ESs are a major class of Evolutionary Algorithms
- GAs initially developed in the late 60s, early 70s in the USA (by John Holland and his students)
- ESs initially developed around the same time in Germany (Ingo Rechenberg and Hans-Paul Schwefel)
- Mostly applied to continuous representations (i.e., vectors of real-valued variables)
- But its ideas can be adapted to other representations

Major difference from GAs

- More emphasis on mutation, less on crossover
  - Mutation strength is adaptive
  - Introduced the notion of self-adaptation
- Uses two types of selection
  - Parent selection: solutions are selected uniformly at random (i.e. ignoring fitness values)
  - Survival selection: (μ, λ) or $(\mu + \lambda)$

Simplest case: $(1 + 1)$-ES

- It is an iterative improvement algorithm, similar to a stochastic hillclimber
- Algorithm maintains a single solution (the parent)
- It produces a child by doing mutation
- The better of two becomes the parent for the next iteration

A note on the notation

- The $(1 + 1)$ notation looks a bit weird
- In general we can have a $(\mu + \lambda)$-ES
  - It means we have μ parents and will be producing λ children
  - The $+$ sign means we will keep the best μ out of the $\mu + \lambda$ parents and children
  - It's an elitist strategy

$(1+1)$-ES

$x =$ solution generated uniformly at random (u.a.r.)
$v_x = f(x)$
while *not terminate()* **do**
    $y =$ mutation(x)
    $v_y = f(y)$
    if $v_y \geq v_x$ **then**
        $x = y$
    end
end
return x

Mutation

- x and y are vectors of real numbers
- $x = [x_1, \ldots, x_n]$, $y = [y_1, \ldots, y_n]$
- We mutate x by mutating each of its components x_i
- We do so by drawing n samples $[z_1, \ldots, z_n]$ from the Normal distribution with mean 0 and standard deviation σ
  - And then we let $y_i = x_i + z_i$, for all $i \in \{1..n\}$
- σ is often referred to as the mutation step size

Gaussian mutation

- Small changes are more frequent than large changes
- Step size σ controls if the mutation strength
- A larger σ value allows larger changes (larger steps) in a solution

Normal Distribution

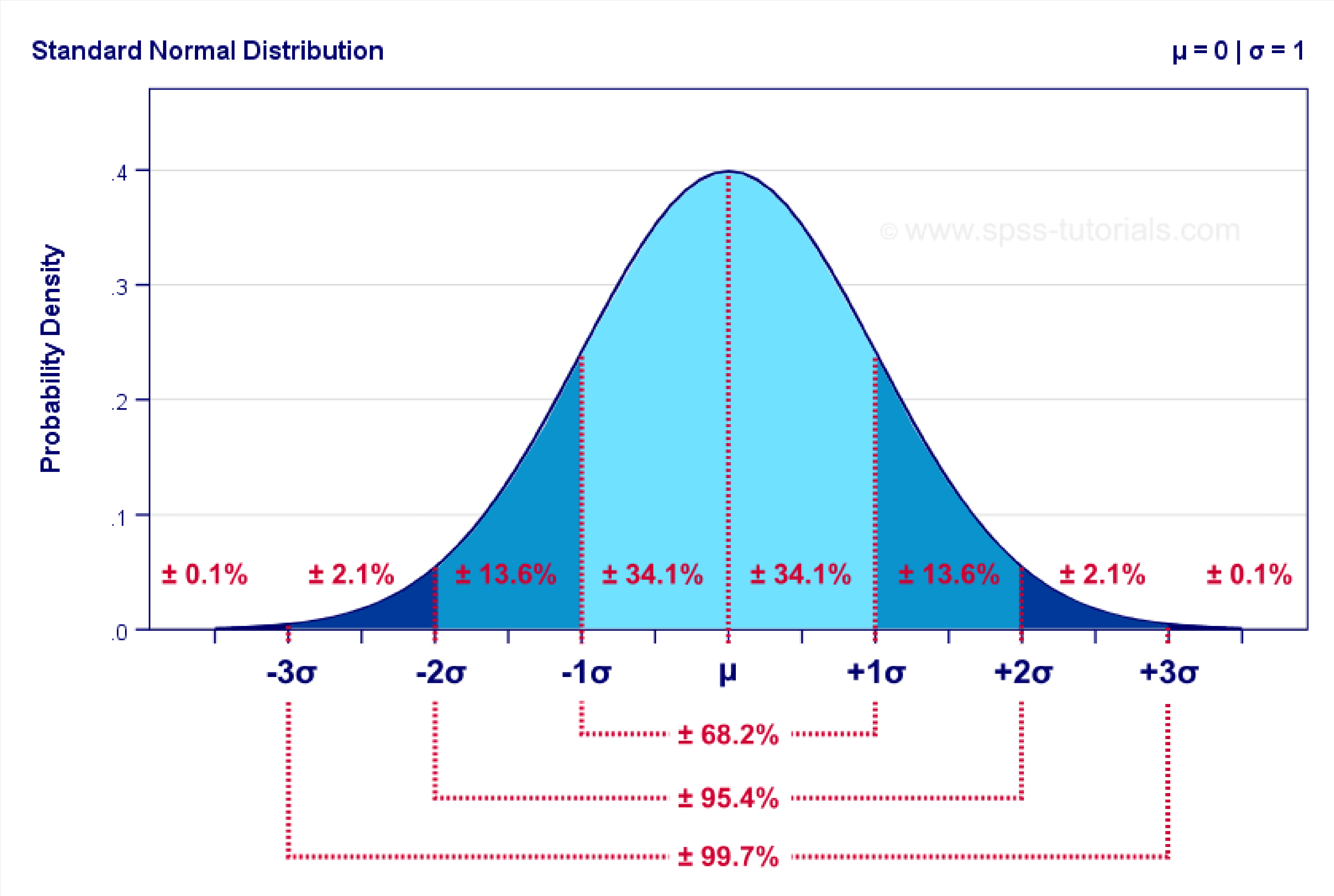

Image taken from `https://www.spss-tutorials.com/normal-distribution/`

Normal Distribution

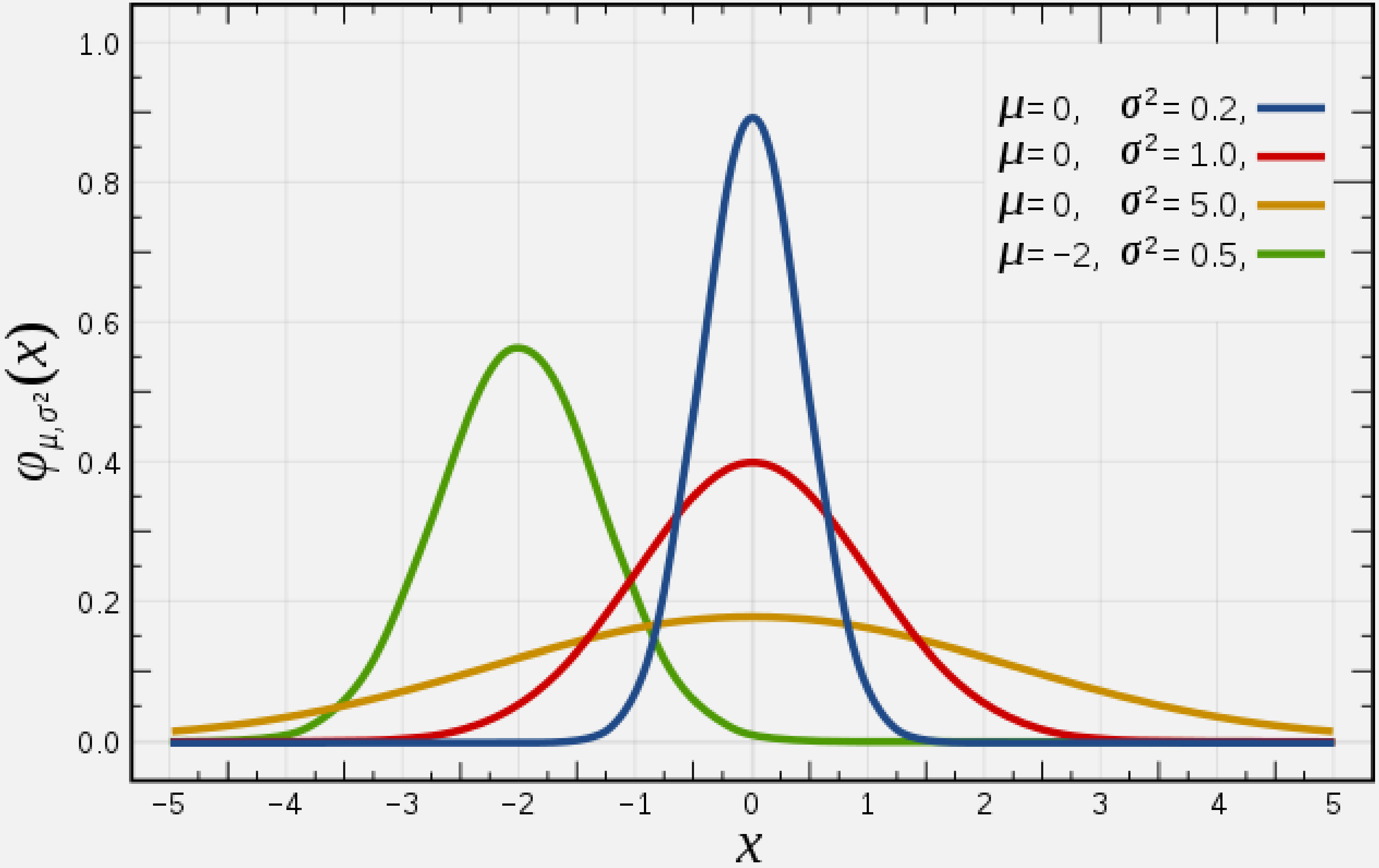

Image taken from `https://en.wikipedia.org/wiki/Normal_distribution`

1/5 success rule

- Adapt the step size σ during the execution of a run
- Idea: Allow large changes at the beginning, but prefer small changes at the end of a run (to fine tune the solution)
- Keep track of the number of "successful mutations" during the execution of the run
  - A successful mutation occurs when the child becomes parent for the next iteration
- Then update σ after every k iterations, depending on the amount of successful mutations.

1/5 success rule

- The update is done according to the following idea:
  - if we had a lot of successful mutations, enlarge σ
    - by doing so, we might speed up the time to approach a good solution
  - if we had just few successful mutations, diminish σ
    - we are likely to be close to a (local optimal) solution, and we might as well make small changes to it to fine tune it

1/5 success rule

- The 1/5 success rule implements that idea as follows:

$$\sigma = \begin{cases} \sigma \;/\; c & \text{if } p_s > 1/5 \\ \sigma \;\cdot\; c & \text{if } p_s < 1/5 \\ \sigma & \text{if } p_s = 1/5 \end{cases}$$

- where c is a constant less than 1, typically $0.8 \leq c < 1.0$
- and p_s is the percentage of successful mutations during the last k iterations

Population-based ES

- The $(1+1)$ strategy is the simplest case
- In general we have a $(\mu+\lambda)$ or (μ,λ) strategy

Plus strategy: $(\mu + \lambda)$-ES

$P =$ generated μ solutions u.a.r.
evaluate(P)
while *not terminate()* **do**
  $C =$ generate λ children from P
  evaluate(C)
  $P =$ select best μ of $(P \cup C)$
end
return P

Comma strategy: (μ, λ)-ES

```
P = generated μ solutions u.a.r.
evaluate(P)
while not terminate() do
    C = generate λ children from P
    evaluate(C)
    P = select best μ of C
end
return P
```

Population-based ES

- Plus strategy is elitist, comma strategy is not
- The ration λ/μ determines the selection pressure
- A commonly used rule-of-thumb is to set $\lambda \approx 7 \cdot \mu$

Generating children (in more detail)

- C = generate λ children from P
- . . . is implemented as:

---

for $i = 1$ **to** λ **do**
  x = select string from P uniformly at random
  $C[i]$ = mutation(x)
end

- Mutation is done as explained previously

Population-based ES

- Plus strategy is elitist, comma strategy is not
- Ratio λ/μ determines the selection pressure
- A commonly used rule-of-thumb is to set $\lambda \approx 7 \cdot \mu$
- Can also use recombination to create children (more on this later)

Self-adaptation

- We have seen the basic ES
  - The step size σ acts as a global parameter
  - Can be adaptive (by using the $1/5$ success rule)
- Self-adaptation is a technique that encodes the step size(s) in the solution themselves
  - The step size(s) will then “evolve” from generation to generation, along with the regular decision variables

Self-adaptation: simplest case (one σ per individual)

- Solution is represented by n decision variables (ES people call them object variables), and one σ

- Each solution if the population will have its own σ (i.e., its own mutation rate)

- The value of σ will change from generation to generation

$$x = [\underbrace{x_1, x_2, \ldots, x_n}_{\text{object variables}}, \sigma]$$

Self-adaptation: simplest case (one σ per individual)

- How to do mutation on $x = [x_1, x_2, \ldots, x_n, \sigma]$?
  - First mutate σ
    - $\sigma = \sigma \cdot \exp(\tau \cdot \mathrm{randN}(0, 1))$
  - Then mutate each x_i using the mutated σ
    - $x_i = x_i + \sigma \cdot \mathrm{randN}(0, 1)$
- Notes:
  - τ is a control parameter, usually set proportional to $1/\sqrt{n}$
  - $N(0, 1)$ gets a sample from the Standard Normal Distribution
  - Use a boundary rule so that σ never goes below a small positive ϵ
    - if $\sigma < \epsilon$ then $\sigma = \epsilon$

Self-adaptation: n σ's per individual

- Solution is represented by n decision variables, and n σ's, one for each variable
- Each x_i will be mutated according to its own σ_i

$$x = [\underbrace{x_1, x_2, \ldots, x_n}_{\text{object variables}}, \underbrace{\sigma_1, \sigma_2, \ldots, \sigma_n}_{\text{strategy parameters}}]$$

Self-adaptation: n σ's per individual

- How to do mutation on $x = [x_1, x_2, \ldots, x_n, \sigma_1, \sigma_2, \ldots, \sigma_n]$?
  - First mutate the σ's
    - ★ $z = \text{randN}(0, 1)$
    - ★ $\sigma_i = \sigma_i \cdot \exp(\tau' \cdot z + \tau \cdot \text{randN}(0, 1))$
  - Then mutate each x_i using the mutated σ_i
    - ★ $x_i = x_i + \sigma_i \cdot \text{randN}(0, 1)$
- Notes:
  - Uses two learning rates: a global one (τ') and a local variable-wise (τ).
  - τ' is usually set proportional to $1/\sqrt{2n}$
  - τ is usually set proportional to $1/\sqrt{2\sqrt{n}}$
  - Use boundary rule for σ, as before

Recombination in ES

- Creates one child z, from two parents x and y
- It is done independently for each variable. Two options:

  Intermediary: z_i is the average of the parents' values: $(x_i + y_i)/2$

  Discrete: $z_i = x_i$ or $z_i = y_i$. The choice is made at random.

- Parents can be fixed, or chosen randomly for each position:

  Local: Select two parents at random, and use them for all positions

  Global: Select two parents at random for each variable x_i

More advanced ES

- Covariance-Matrix Adaptation Evolution Strategy (CMA-ES)
- Developed by Nikolaus Hansen
- It's an alternative to self-adaptation, and tends to work better than it
- Adapts a covariance matrix that is able to model pairwise correlations between variables

More advanced ES

- We are really trying to model a Multivariate Normal Distribution.
- Children can be seen as variations of the mean vector of the population
- They are obtained by making perturbations controlled by the various σ
- Adaptation of the covariance matrix plays an important role in making the search more effective