UiO : Department of Informatics
University of Oslo

Biologically inspired computing - Lecture 2

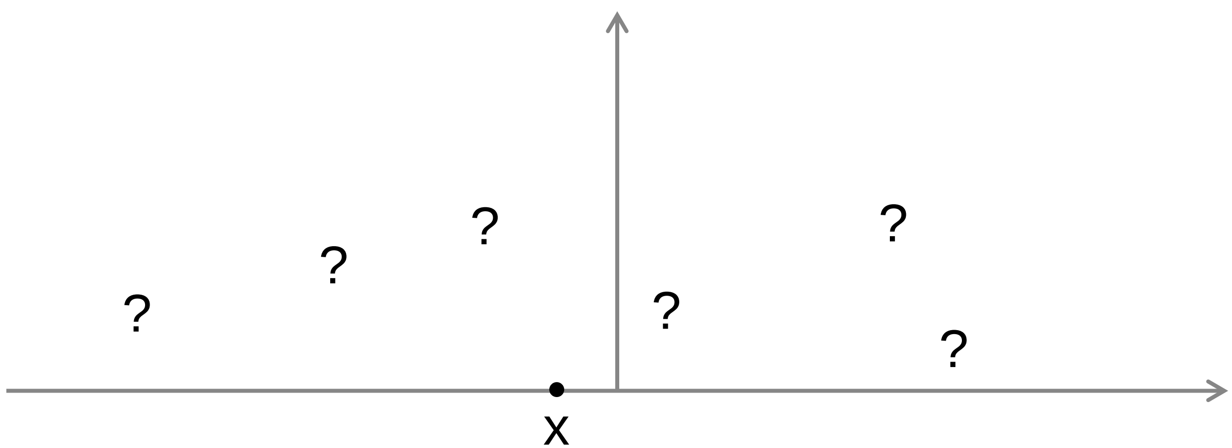
Evolution strategies & Evolutionary programming

This lecture

- Random optimization
- Evolution strategies (+ EAs in general)
  - The strategy parameter
  - Random displacements as mutation
  - Selection and recombination
- Evolutionary programming

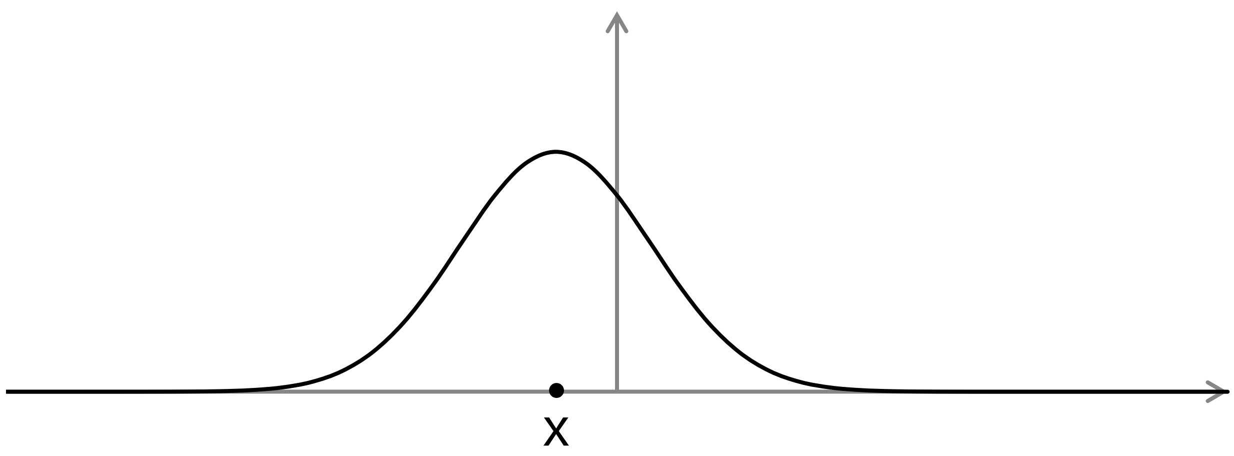
Hill climbing in $\mathbb{R}^n$

- Hill climbing:
  - Randomly select one neighboring solution
  - Continuous space: need to define neighborhood



Random optimization

- The entire space is the neighborhood
  - Selection probabilities are normally distributed: Closer solutions are more likely to be selected

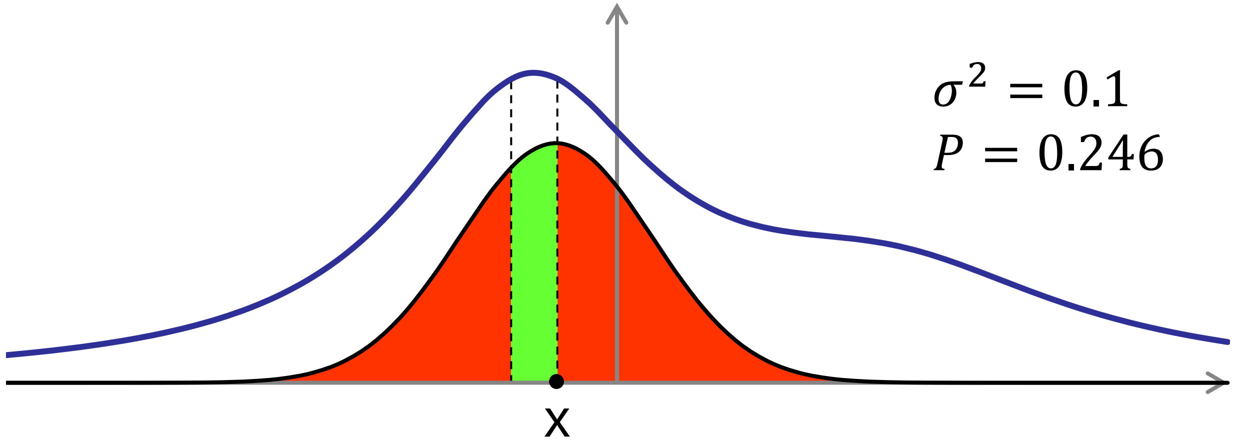


Random optimization

```
def random_opt():
   X = random_vector()
   while not_done():
      Y = X + normal(0,sigma)
      if (f(X) < f(Y)):
           X = Y
   return X
```

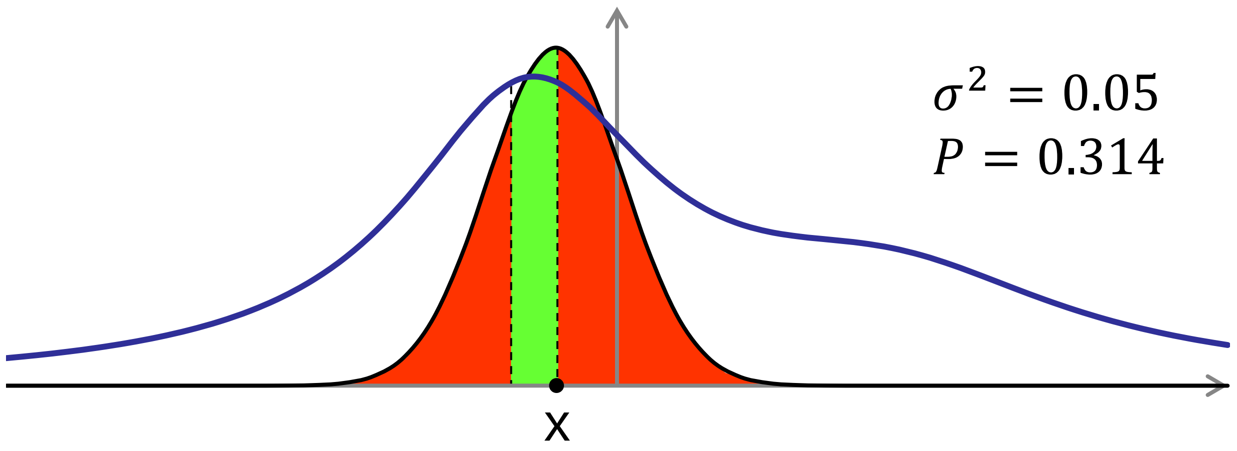
The $(1+1)$ evolution strategy

When the current solution gets close to an optima the probability of a better solution being selected decreases



The $(1+1)$ evolution strategy

- To compensate, we can reduce the spread of the distribution
- σ afftects our *search strategy*



The $(1+1)$ evolution strategy

- Add a strategy parameter σ to random optimization and you get the (1+1) ES:

```
def random_opt():
   X.x = random_vector()
   X.s = initial_sigma()
   while not_done():
      Y.s = X.s*exp(normal(0,tau))
      Y.x = X.x + normal(0,Y.s)
      if (f(X) < f(Y)):
           X = Y
   return X.x
```

The evolution analogy

Optimization	Biology
Candidate solution	Individual
Old solution	Parent
New solution	Offspring
Solution quality	Fitness
Random displacements added to offspring	Mutation
Search strategy	Mutation rate, gene robustness

Robustness

- The (1+1) ES is more efficient at finding accurate solutions, but it remains vulnerable to local optima

- Solution: Run multiple times?
  - Sometimes referred to as *(1+1) reset*

- Even better: Do multiple runs in parallel and make use of the extra information from having multiple solutions available at once

The evolution analogy

Optimization	Biology
Candidate solution	Individual
Old solution	Parent
New solution	Offspring
Solution quality	Fitness
Random displacements added to offspring	Mutation
Search strategy	Mutation rate, gene robustness
A set of solutions	Population

Evolutionary algorithm outline

```
def evolve():
   P.x = initialize_population()
   P.fitness = evaluate(P.x)
   while not_done():
      Q.x = reproduce(P)
      Q.x = mutate(Q.x)
      Q.fitness = evaluate(Q.x)
      P = survival(P,Q)
   return best(P).x
```

Evolutionary algorithm outline

- `initialize_population()`
  - Generates a set of starting points
  - May be completely random solutions, or some hand-crafted selection

- `evaluate(P)`
  - Applies the objective function to all elements in `P`
  - Problem-dependent

Evolutionary algorithm outline

- `reproduce(P)`
  - Creates a new population from `P`
- `mutate(X)`
  - Applies random changes to the individuals in `X`
- `survival(P,Q)`
  - Creates a new population from `P` and `Q`

Evolution strategies

- Each individual is composed of n solution parameters and n_σ strategy parameters:

$$\langle x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_{n_\sigma} \rangle$$

  - Usually n_σ is either 1 (all x_i share one strategy) or n (each x_i have a separate search strategy)
  - Sometimes an additional set of parameters α_i is used to model correlations between strategies

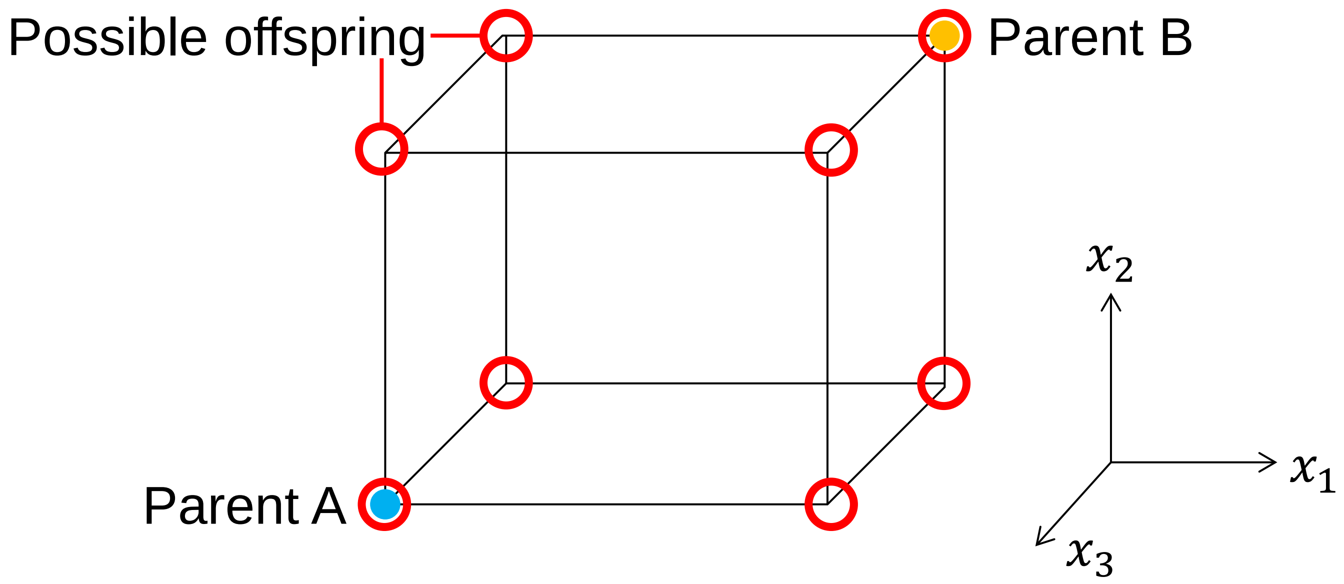
Evolution strategies

- Recombination creates λ offspring
- Each one draws two parents at random and recombines them using intermediary or discrete recombination
- It is common to mix, i.e. use discrete for x_i and intermediate for σ_i

```
def reproduce(P):
   Q = []
   for i in range(1,lambda):
      parents = draw(2,P)
      offspring = recombine(parents[0], parents[1])
      Q.append(offspring)
   return Q
```

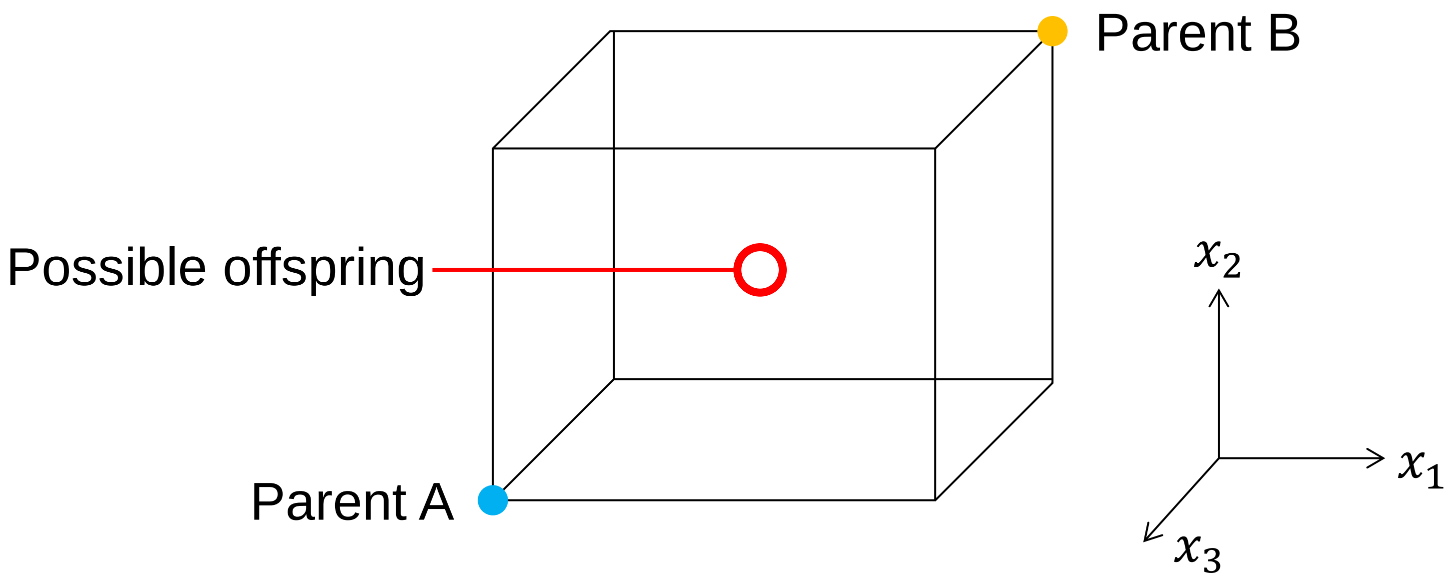
Discrete recombination

- Each parameter is chosen from one of the parents at random



Intermediary recombination

- Each parameter is chosen as the average of from the parents



Evolution strategies

- Two survivor selection methods:
  - (μ, λ): from the offspring only
  - $(\mu + \lambda)$: from both parents and offspring

- In both cases the survivor candidates are sorted by fitness, and the μ best are kept

Evolution strategies

- (μ, λ) is often preferred, for several reasons:
  - Better able to escape local optima
  - Able to adapt to changing fitness functions
  - Since solutions aren't evaluated for how good the strategy parameters are, bad strategies can linger in the population if parents can survive indefinitely

Evolutionary programming

- Historically, evolutionary programming was mainly concerned with prediction problems
- More recently the field has diversified a lot, and is used for all kinds of different problems and with many different representations and mutation schemes
- Here we will focus on a variant for continuous optimization

Evolution strategies vs. Evolutionary programming

	Evolution strategies	Evolutionary programming
Representation	Vector of solution and strategy parameters	
Parent selection	Probabilistic	Deterministic
Recombination	Probabilistic	None
Mutation	$\sigma_i' = \sigma_i \cdot e^{N(0,\tau)}$ $x_i' = x_i + N(0,\sigma_i')$	$\sigma_i' = \sigma_i \cdot \left(1 + N(0,\alpha)\right)$ $x_i' = x_i + N(0,\sigma_i')$
Survivor selection	Deterministic	Probabilistic

Evolutionary programming

- In EP each solution is seen as a species instead of an individual
  - Recombination does not make sense!
  - Each solution gives rise to exactly one new solution each generation

Evolutionary programming

Survivor selection is done by tournaments

- Each solution is compared to q other randomly selected solutions (q is typically about 10)
- The best half, ranked by the number of “wins” survives

```
def survival(P,Q):
   PQ = [P,Q]
   for i in range(1, 2*mu):
      vs = draw(q, PQ)
      score = sum( PQ[i].fitness > vs.fitness )
   return best(mu, PQ, score)
```