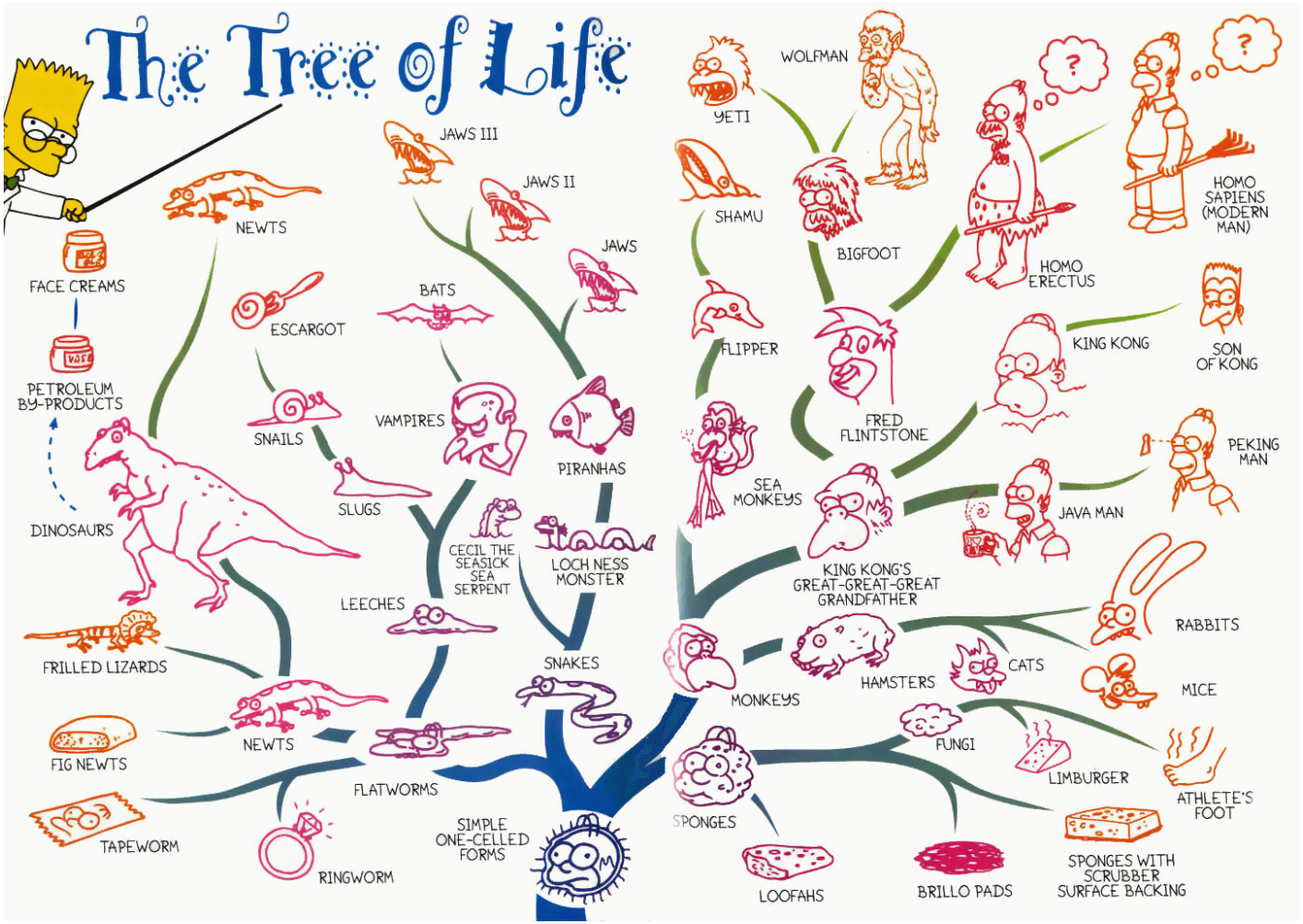
Evolutionary Algorithms (Vienna 2022)

Christian Blum

Artificial Intelligence Research Institute (IIIA-CSIC)

Evolution

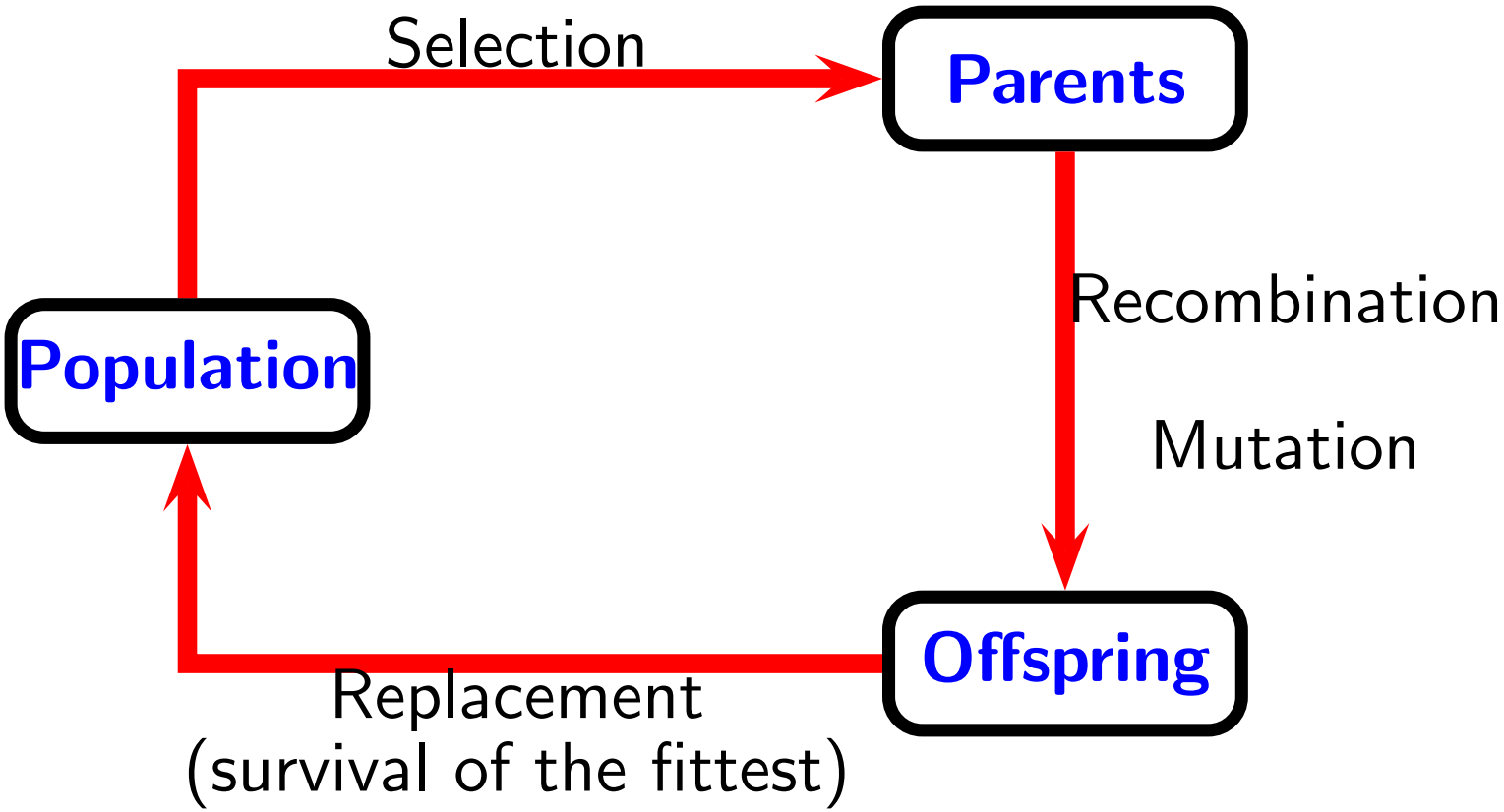

The Tree of Life
JAWS III
JAWS II
JAWS
NEWTS
FACE CREAMS
ESCARGOT
BATS
PETROLEUM BY-PRODUCTS
SNAILS
VAMPIRES
PIRANHAS
DINOSAURS
SLUGS
CECIL THE SEASICK SEA SERPENT
LOCH NESS MONSTER
LEECHES
FRILLED LIZARDS
SNAKES
NEWTS
FIG NEWTS
FLATWORMS
TAPEWORM
SIMPLE ONE-CELLED FORMS
RINGWORM
YETI
WOLFMAN
SHAMU
BIGFOOT
HOMO ERECTUS
HOMO SAPIENS (MODERN MAN)
FLIPPER
FRED FLINTSTONE
KING KONG
SON OF KONG
SEA MONKEYS
PEKING MAN
JAVA MAN
KING KONG'S GREAT-GREAT-GREAT GRANDFATHER
RABBITS
MONKEYS
HAMSTERS
CATS
MICE
FUNGI
LIMBURGER
ATHLETE'S FOOT
SPONGES
LOOFAHS
BRILLO PADS
SPONGES WITH SCRUBBER SURFACE BACKING

Evolution: essential concepts

Evolution: an optimization process

- Selection. Examples:
  1. Mammals prefer mates with a very different immune system
  2. Plants with a nicer flower (or a better smell) attract more flies
- Survival of the fittest. Examples:
  1. Slow rabbits are more easily caught by foxes
  2. In a cold winter the one with more fat reserves survive
- Recombination and mutation. Introduce innovation and diversity into a population

Evolutionary cycle



Technical algorithm

Pseudo código

$P \leftarrow$ GenerateInitialPopulation()
Evaluate(P)
while termination conditions not met **do**
  $P' \leftarrow$ Recombination(P)
  $P'' \leftarrow$ Mutation(P')
  Evaluate(P'')
  $P \leftarrow$ Selection(P'', P) (Replacement)
end while

Evolutionary algorithms: notation

Notation

- **Selection:**
  1. The choice of the individuals (i.e., solutions) that act as parents to produce offspring for the next generation
  2. The choice of the individuals that replace old individuals. (Also called **replacement**).
- **Recombination:** The recombination (or crossover) of two or more individuals for producing offspring.
- **Mutation:** The change that an individual undergoes (either randomly or in a heuristically guided way)
- **Evaluation:** The evaluation of the fitness of an individual (usually inversly proportional to the objective function)

Historical note

Evolutionary algorithms are among the oldest metaheuristic methods.

Families of evolutionary algorithms

- **Evolutionary Programming (EPs)** introduced by [Fogel et al., 1966]
- **Evolutionary Strategies (ESs)** introduced by [Rechenberg, 1973]
- **Genetic Algorithms (GAs)** introduced by [Holland, 1975], [Goldberg, 1989]
- **Genetic programming (GP)** introduced by [Koza, 1992]
- **Scatter search (SS)** introduced by [Glover, 1977]
- **Estimation of distribution algorithms (EDAs)** introduced by [Mühlenbein, 1996]
- **Differential evolution (DE)** introduced by [Storn, 1995]

Families of evolutionary algorithms

Differences between these families/types of EAs

- **Purpose:**
  1. **Continuous optimization:** ES, SS, EDA, DE
  2. **Combinatorial optimization:** GA, GP, SS
  3. **Programs/trees:** GP

- **Solution representation**. Examples:
  1. **Binary strings:** GA
  2. **Tree structures:** GP

- **Evolutionary operators:**
  1. **Exclusively mutation:** ES
  2. **2-parent crossover:** GA

Literature

Seminal books

- **[Fogel et al., 1966]** L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966
- **[Rechenberg, 1973]** I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973
- **[Holland, 1975]** J. H. Holland. *Adaption in natural and artificial systems*. The University of Michigan Press, Ann Harbor, MI, 1975
- **[Goldberg, 1989]** D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, Reading, MA, 1989

Useful introduction to EAs for combinatorial optimization

[Hertz and Kobler, 2000] A. Hertz and D. Kobler. A framework for the description of evolutionary algorithms. *European Journal of Operational Research*, 126:1–12, 2000

Evolutionary algorithms: 7 features

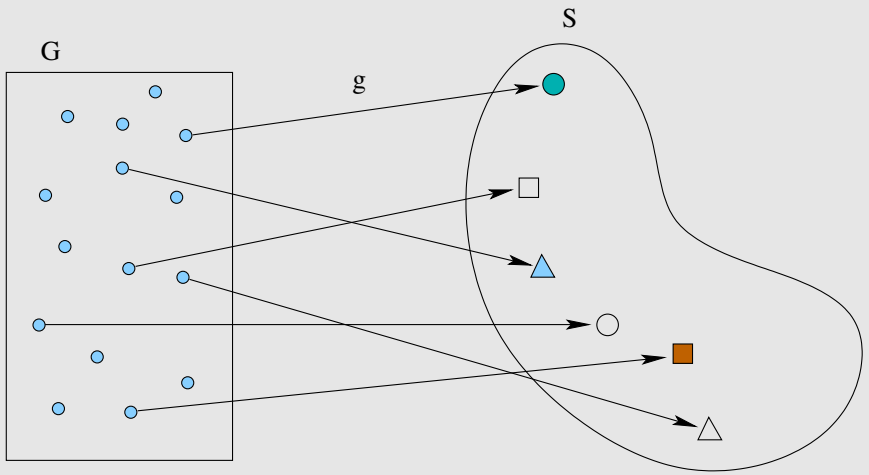
1. Representation of the individuals (solution representation)
2. Evolution process (replacement strategy)
3. Neighborhood structure (which individuals can be recombined)
4. Information sources (how to recombine?)
5. How to deal with infeasible individuals?
6. Intensification strategy
7. Diversification strategy

Representation: genotype vs. phenotype

Dos espacios distintos

1. Genotype space $\mathcal{G}$: the space of all individuals
2. Phenotype space $\mathcal{S}$: the space of all solutions to the problem

Mapping



Genotype/Phenotype space

Requirements on the mapping

- **Surjectivity:** For each solution in the phenotype space there should be an individual from the genotype space that maps to it
- **Non-disruptiveness:** Genotypes (individuals) that are similar should map to similar solutions

Note

The representation of individuals is crucial for the success of an evolutionary algorithm.

Example: direct representations

Direct representations: on the whiteboard!

- **TSP:** binary strings of length $m = |E|$
  Position i contains a binary value for $e_i \in E$
  Disadvantages:
  1. Strings are of length $\frac{n(n-1)}{2}$
  2. Many possible binary strings do not correspond to feasible solutions
- **GSS:** binary strings that have a position for every edge that needs to be given a direction
- **KCT:** edge list that contains all edges of the corresponding tree
- **Continuous optimization:** real-valued vectors (standard)

Example: indirect representations

Indirect representations: on the whiteboard!

- **TSP:** permutations of the n cities/nodes
  The sequence of nodes shows the order in which they are visited
  Advantages (when compared to binary strings):
  1. Permutations are of length n (much shorter)
  2. Every possible permutation represents a feasible solution
- **GSS:** permutations of all operations
  But: not every permutation represents a feasible solution
- **TSP, GSS:** real-valued vectors (*gray encoding*)
  Note: every vector can be converted into a permutation
  Advantage: enables the use of standard evolutionary operators
- **GSS:** vectors that indicate for each construction step the greedy function to be used in this specific step.

Ex.: representations with low locality

Prüfer codes

- **Given:** a complete graph $G = (V, E)$ where $|V| = n$
- Each sequence $a_1, \ldots, a_{n-2}$ with $a_i \in \{1, \ldots, n\}$ $(i = 1, \ldots, n-2)$ is called a **Prüfer code**
- Every Prüfer code encodes a spanning tree of G
- The degree of each node is equal to the number of occurrences of its index in the Prüfer code + 1
- For each spanning tree of G there exists exactly one Prüfer code encoding this tree.

Ex.: representations with low locality

Decodification of a Prüfer code

1. Scan the Prüfer code for the initialization of the degrees (d_i) of the nodes.
2. Initialize a counter: $i := 1$.
3. Identify the node v with minimal current degree greater than zero. In case of ties, choose the one with minimal index.
4. Add the edge (v, v_{a_i}) the tree under construction
5. Decrement the degrees of v and v_{a_i}.
6. Increment i
7. Repeat points 3–6 until there are only two nodes with degree one left. Introduce an edge between these two nodes.

The evolution process

What is the evolution process?

The type of the evolution process determines the way of assembling the new generation.

Well-known types of evolution processes

- **Generational replacement:** The offspring entirely replaces the old generation.
- **Steady state:** The best offspring individuals replace a percentage (not all) of the worst individuals of the old generation.

Note

The population size can be **constant** or **varying**

Recombination: neighborhood functions

Neghborhood functions in EAs

A **neighborhood function** $\mathcal{N}_{\mathcal{EC}} : \mathcal{I} \rightarrow 2^{\mathcal{I}}$ defines, for every individual $i \in \mathcal{I}$, the set of individuals $\mathcal{N}_{\mathcal{EC}}(i) \subseteq \mathcal{I}$ which can be recombined with it.

Different types of neighborhood functions

- **Unstructured population:** every individual can be recombined with any other one (e.g., simple GA)
- **Structured population:**
  - **Natural:** for example, in the *k-cardinality tree (KCT)* problem
  - **Artificial:** parallel evolutionary algorithms (PEAs)

Parallel EAs (PEAs)

Motivation

The intrinsic parallelism of EC algorithms has given rise to PEAs, whose main feature is the geographical decentralization of the population. One of the aims is to save computation time.

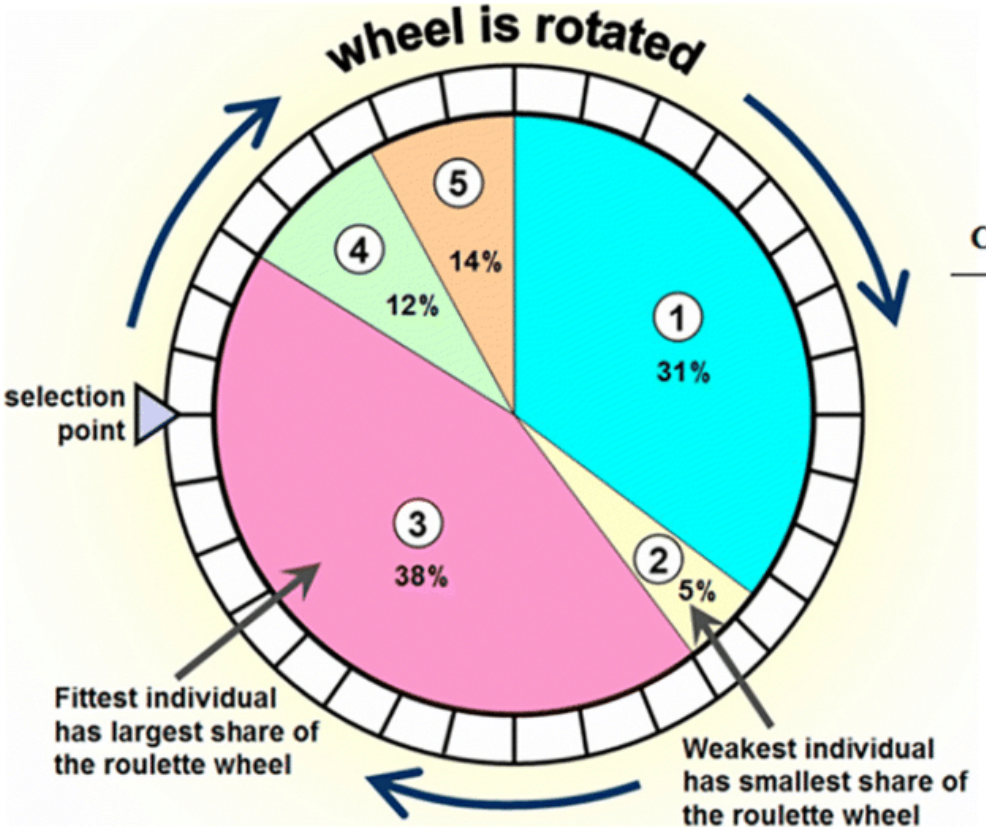
Different PEAs

- **Coarse-grained or distributed EAs (dEAs):** Population is partitioned into different sub-populations (islands). They operate independently, but exchange information.
- **Fine-grained or cellular EAs (cEAs):** Individuals are placed on a toroidal grid; each individual on a different grid position. Individuals can only recombine with neighboring individuals.

Different selection schemes (1)

Rhoulette-wheel-selection

Choose the parent individuals randomly, biased by their quality



Chromosome	Fitness value
C1	31%
C2	5%
C3	38%
C4	12%
C5	14%

Different selection schemes (2)

Tournament selection

Choose m individuals uniformly at random. Choose the best one among them. Do the same for choosing all the parent individuals.

Chromosome #	C_1	C_2	C_3	C_4	C_5	C_6	C_7
Fitness value	10	1	8	6	9	4	7

Tournament size= 3

Randomly 3 chromosomes are selected

Chromosome #	C_2	C_6	C_7
Fitness value	1	4	7

Chromosome with best Fitness is selected

Winner Chromosome #	C_7
Fitness value	7

Recombination/Crossover

Crossover for bit strings: whiteboard!

- 1-*point crossover*
- *n-point crossover*

KCT problem: whiteboard!

- Only trees T and T' with edges in common can be recombined
- Generate a child tree T'' with a Greedy heuristic, with a preference for edges that both parents have in common.

TSP problem: whiteboard!

Just one of many examples: PMX (partially mapped crossover)

How to deal with infeasible solutions?

Note

The recombination of individuals might result in unfeasible individuals.

Three possible ways of dealing with infeasibility

- **Reject**: discard infeasible solutions
- **Punish**: decrease the **fitness** of infeasible solutions
- **Repair**: apply some operators to change the solution trying to obtain a feasible one

Example for repairing (GSS problem): whiteboard!

- Unfeasible permutations can be repaired using the *list scheduler* algorithm

Intensification of the search process

Problem

Evolutionary algorithms might sometimes be slow in comparison to metaheuristics based on local search.

Possible solution to this problem

Intensification strategies that apply operators or algorithms to improve the fitness of single individuals

Examples

- Before replacement, apply local search to all (or some) individuals of the current population (**Memetic Algorithms**)
- Instead of a random mutation, apply mutation operators based on local search (some steps).

Diversification of the search process

Problem

Early convergence, which means that all individuals are becoming quickly very similar

A possible solution to this problem

Apply mechanisms to diversify the search. For example:

- Random mutation
- Introduce into the population new individuals from not yet explored areas of the search space
- **Niching**, **Crowding**

How to generate the initial population?

- Population of **random solutions**
- Construct a population with a randomized greedy heuristic (e.g., GRASP)

TSP example: a simple EA

- **Solution representation:** permutations of the n cities
- **Evolution process:** generational replacement
- **Neighborhood function:** each individual can reproduce with any other individual
- **Selection and information sources:** *Tournament selection*, PMX crossover (two-parent crossover)
- **Infeasibility:** does not apply as all offspring are feasible
- **Intensification strategy:** 2-opt local search
- **Diversification strategy:** with a probability of 0.1 choose a city randomly and insert it at a random point in the permutation

Advantages/disadvantages of EAs

General advantage

EAs are good in discovering high-quality areas of the search space

General disadvantage

EAs have deficiencies in exploiting good solutions (danger of a disrupting crossover)

Motivation of EDAs

Overcome the drawbacks of usual recombination operators likely to break good building blocks

Principle

First, produce a **probability distribution** over the search space. Then, iterate over the following two steps:

1. **Sample the current probability distribution**, generating in this way a set of new solutions
2. **Adapt/change the probability distribution** using the newly sampled solutions

EDAs directly working on populations

Pseudo code

$P \leftarrow$ GenerateInitialPopulation()
while termination conditions not met **do**
  $P_{sel} \leftarrow$ Selection(P) $\{P_{sel} \subseteq P\}$
  $\mathbf{p}(\cdot \mid P_{sel}) \leftarrow$ EstimateProbabilityDistribution(P_{sel})
  $P \leftarrow$ SampleProbabilityDistribution($\mathbf{p}(\cdot \mid P_{sel})$)
end while

EDAs working on distributions

Pseudo code

$\mathbf{p}(\cdot) \leftarrow$ GenerateProbabilityDistribution()
while termination conditions not met **do**
  $P \leftarrow$ SampleProbabilityDistribution($\mathbf{p}(\cdot)$)
  UpdateProbabilityDistribution(P)
end while

Example: PBIL

Population-based incremental learning (PBIL)

- PBIL is a simple EDA for problems in which solutions can be represented by binary vectors of length n
- PBIL works with a simple distribution $\vec{p} = (p_1, \ldots, p_n)$ where $p_i \in [0, 1]$ for all $i = 1, \ldots, n$

PBIL: pseudo code

$p_i := 0.5$ for all $i = 1, \ldots, n$
while termination conditions not met **do**
  $P :=$ generate m individuals by sampling $\vec{p}$
  Let $\vec{s}^*$ be the best solution in P
  $p_i := p_i + \delta_{\text{lern}}(s_i^* - p_i)$ for all $i = 1, \ldots, n$
  With probability p_{mut}: $p_i := p_i + \delta_{\text{mut}}(r\{0,1\} - p_i)$ for all $i = 1, \ldots, n$
end while

Differential evolution (DE)

DE: algorithm type

DE is an evolutionary algorithm for continuous optimization

Algorithm components

- **Population of solutions:** $\vec{x}_i$, $i = 1, \ldots, m$.
  Each position $x_{i,j}$ is a real number
- **Mutation**
- **Crossover**
- **Selection**

Mutation

For each solution $\vec{x}_i$, $i = 1, \ldots, m$, the following is done:

1. Randomly choose 3 different solutions from the current population: $\vec{x}_r$, $\vec{x}_s$, $\vec{x}_l$ $(i \neq r, s, l)$
2. Generate a mutated solution:

$$\vec{x}_i^{mut} := \vec{x}_r + F \cdot (\vec{x}_s - \vec{x}_l)$$

with $F \in [0, 2]$

Crossover in DE

Crossover

For each solution $\vec{x}_i$, $i = 1, \ldots, m$, the following is done:

- Generate a trial solution $\vec{x}_i^{pr}$.
- For this purpose determine $x_{i,j}^{pr}, j = 1, \ldots, n$ in the following way:

$$x_{i,j}^{pr} := \begin{cases} x_{i,j}^{mut} & : \text{ if } (r(j) \leq q) \text{ or } j = k \\ x_{i,j} & : \text{ otherwise } . \end{cases}$$

Where: $q \in [0, 1]$, $r(j)$ are random numbers (for all j), and k is a randomly chosen position

Selection in DE

Possibilities for implementing selection

- **Most often used:** the trial solution $\vec{x}_i^{pr}$ replaces $\vec{x}_i$ in the population only if better
- **Also possible:** even if the trial solution $\vec{x}_i^{pr}$ is worse than $\vec{x}_i$, it still has some probability of replacing it in the current population

Questions?