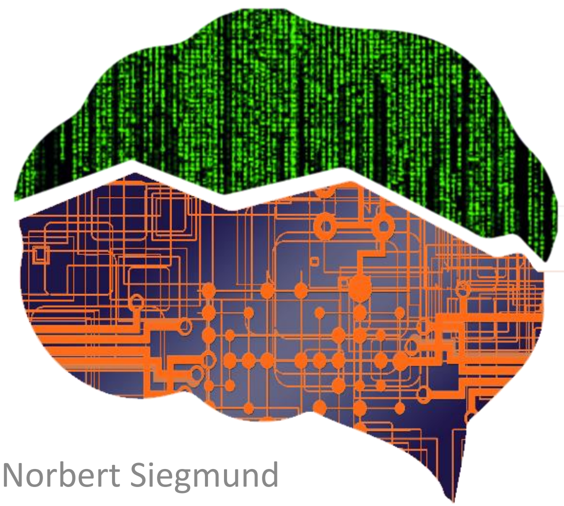
Machine Learning for Software Engineering

Multi-State Meta-Heuristics Continued



Prof. Dr.-Ing. Norbert Siegmund
Intelligent Software Systems

Bauhaus-Universität Weimar

Recap I

- What is Gaussian Convolution?
  - With a certain probability change a gene of an individual
  - Use the Gaussian distribution for the actual change
  - Adjust σ^2 to control exploration vs. exploitation

- What is the approach of Tabu Search and Iterative Local Search to leave a local optimum?

- What is the difference between single- and multi-state meta-heuristics?

Recap II

- What does (μ,λ) stand for?
- Difference between (μ,λ) and $(\mu+\lambda)$?

  $P \leftarrow \{\}$ $\quad$ $P \leftarrow \{Q\}$

- Relation to Steepest Ascent Hill Climbing (with Replacement)?

- Basic operations of evolutionary algorithms?
  - Breed (how to select parents and how to tweak them to make children)
  - Join (replacing parents with children? How?)
  - Initialization (random? With bias?)

Genetic Algorithms (GA)

Introduction to GA

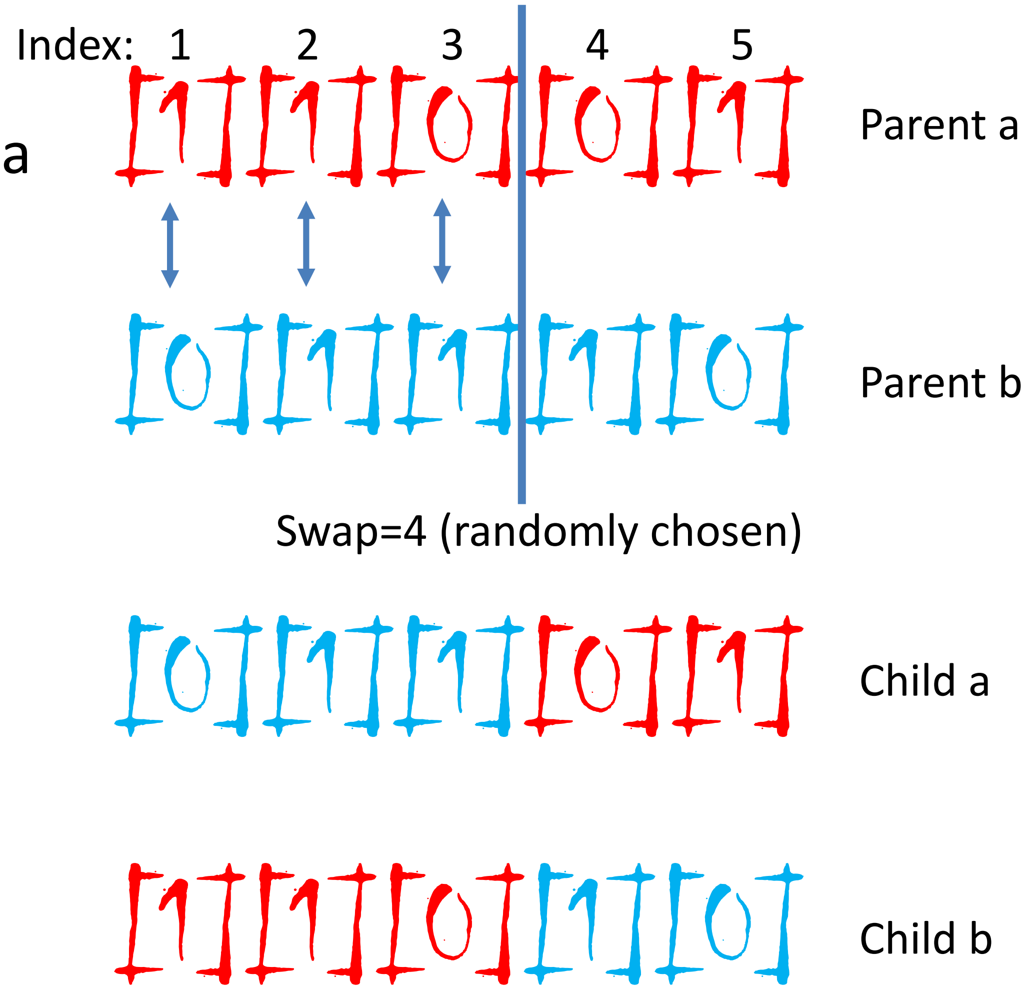
- Invented by John Holland in 1970s
- Approach is similar to the (μ,λ) algorithm
- Difference in selection and breeding operation
  - ES selects parents before breeding children
  - GA selects little-by-little parents to breed new children
- Breeding:
  - Select two parents, copy them, crossover them, mutate results, and add the two children to the new population
  - Repeat until population is full

GA Algorithm

$size \leftarrow$ population size
$P \leftarrow \{\}$
for $size$ times **do**
  $P \leftarrow P \cup \{$random individual$\}$
$Best \leftarrow empty$
repeat
  for each individual $P_i \in P$ **do**
    $AssessFitness(P_i)$
    if $Best == empty$ or $Fitness(P_i) > Fitness(Best)$ **then**
      $Best \leftarrow P_i$
  $Q \leftarrow \{\}$ ← From here it deviates from (μ,λ)
  for $size/2$ times **do**
    $Parent\ P_a \leftarrow SelectWithReplacement(P)$
    $Parent\ P_b \leftarrow SelectWithReplacement(P)$
    $Children\ C_a, C_b \leftarrow Crossover\big(Copy(P_a), Copy(P_b)\big)$
    $Q \leftarrow \cup \{Mutate(C_a), Mutate(C_b)\}$
  $P \leftarrow Q$
until $Best$ is optimum or out of time
return $Best$

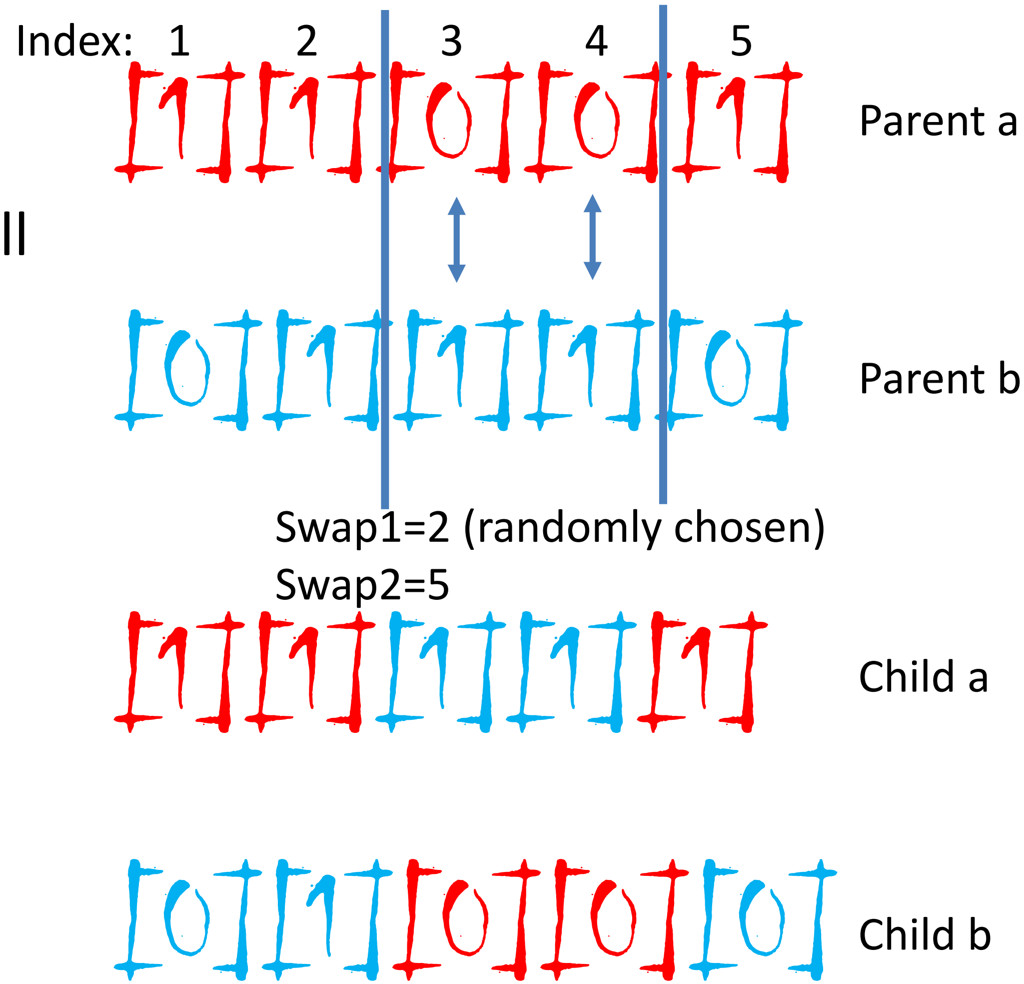
How to Do the Crossover?

- One-Point Crossover
  - Swap everything below a randomly chosen index



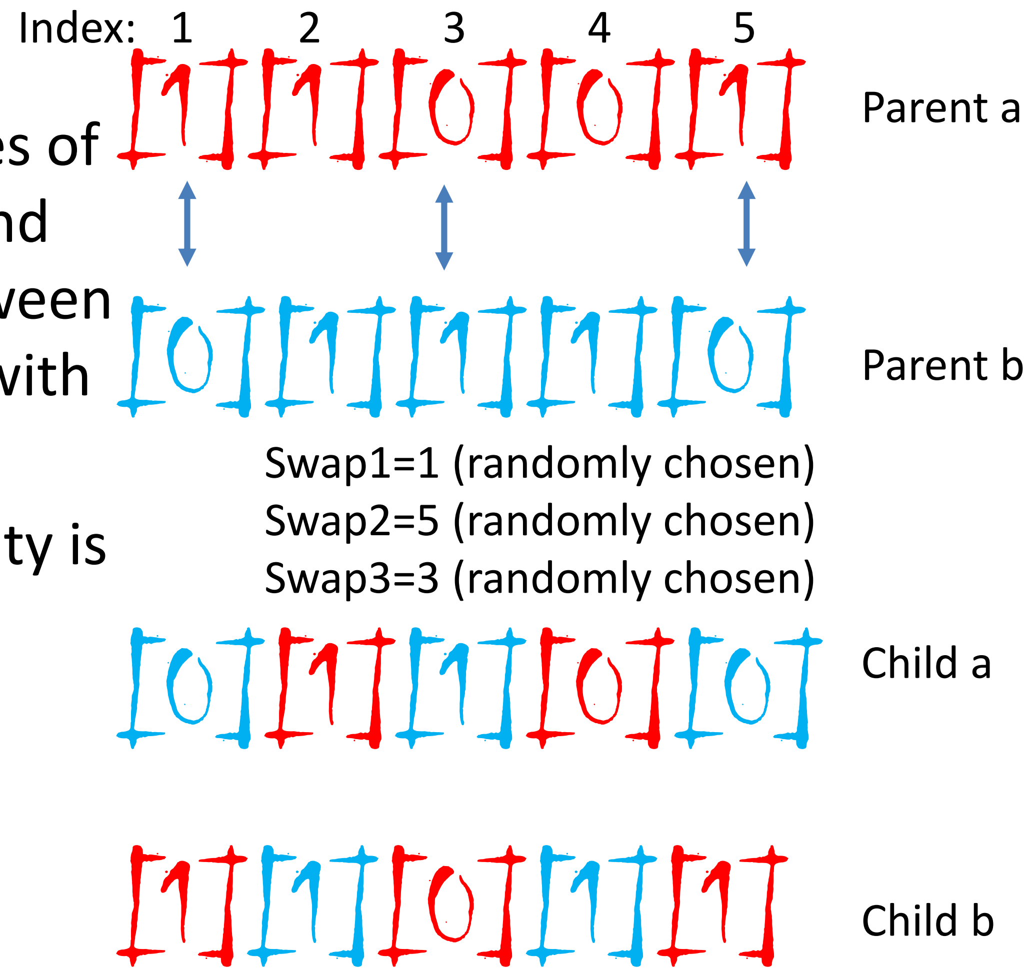
A More Flexible Crossover

- Two-Point Crossover
  - Select two random indexes and switch all genes in between



Crossover Arbitrary Genes

- Uniform Crossover
  - Go through the genes of the first individual and swap the genes between the two individuals with a certain probability
  - Usually the probability is just 0.5



Algorithms Overview

One-Point Crossover

$\vec{x} \leftarrow$ first parent: $\langle x_1, \dots, x_l \rangle$
$\vec{v} \leftarrow$ second parent: $\langle v_1, \dots, v_l \rangle$
$c \leftarrow$ random int chosen uniformly from 1 to l
if $c \neq 1$ **then**
  for i from 1 to $c - 1$ **do**
    swap the values of x_i and v_i
return $\vec{x}$ and $\vec{v}$

Two-Point Crossover

$\vec{x} \leftarrow$ first parent: $\langle x_1, \dots, x_l \rangle$
$\vec{v} \leftarrow$ second parent: $\langle v_1, \dots, v_l \rangle$
$c \leftarrow$ random int chosen uniformly from 1 to l
$d \leftarrow$ random int chosen uniformly from 1 to l
if $c > d$ **then**
  swap c with d
if $c \neq d$ **then**
  for i from c to $d - 1$ **do**
    swap the values of x_i and v_i
return $\vec{x}$ and $\vec{v}$

Uniform Crossover

$p \leftarrow$ probability of swapping a gene
$\vec{x} \leftarrow$ first parent: $\langle x_1, \dots, x_l \rangle$
$\vec{v} \leftarrow$ second parent: $\langle v_1, \dots, v_l \rangle$
for i from 1 to l **do**
  if $p \geq$ uniform random nb (0 to 1) **then**
    swap the values of x_i and v_i
return $\vec{x}$ and $\vec{v}$

Why is Crossover Alone not Sufficient?

- Children will be constrained to the hyper space that the parents span
- Hyper space might be significantly smaller than the overall search space
- Best solutions might lie outside the hyper space
  - We won't find the global optimum

- So, we need an operation to break out of the hyper space
- Still, crossover has its benefits to share high-performing building blocks of individuals
  - Building blocks are combinations of genes that are linked (i.e., interact positively wrt. the objective function)
  - One- and two-point crossover assumes that the linked genes are encoded as neighbors in the vector representing the individual (often unlikely, though)

Going Beyond Binary for Crossover

- Swapping the exact floating-point number makes not so much sense
- What can we do?
  - Use the average between two floating-point values
  - Use a random number between two floating-point values

- Can we generate also new values to break out of the hyper cube?
  - Idea: **Line Recombination**

Line Recombination Algorithm

$\vec{x} \leftarrow$ first parent: $\langle x_1, \dots, x_l \rangle$
$\vec{v} \leftarrow$ second parent: $\langle v_1, \dots, v_l \rangle$
$p \leftarrow$ positive value defining how far we outrach the hyper cube (e.g., 0.25)

Input

$\alpha \leftarrow$ random value from $-p$ to $1 + p$ inclusive
$\beta \leftarrow$ random value from $-p$ to $1 + p$ inclusive
for i from 1 *to* l **do**
  $t \leftarrow \alpha x_i + (1 - \alpha) v_i$
  $s \leftarrow \beta v_i + (1 - \beta) x_i$
    if t and s are within bounds **then**
      $x_i \leftarrow t$
      $v_i \leftarrow s$
return $\vec{x}$ and $\vec{v}$

Example for $p = 0.25$: range: [-0.25;1.25]
E.g. with random: $\alpha = 0.37$ and $\beta = 0.11$

$x_i = 3.5; v_i = 1.0$
$t = 0.37 * 3.5 + (1 - 0.37) * 1.0 = 1.925$
$s = 0.11 * 1.0 + (1 - 0.11) * 3.5 = 3.21$

Extension: Intermediate Recombination

- Just shifting two lines allows us to generate children not only on the line vector between two parents, but in the whole hyper cube

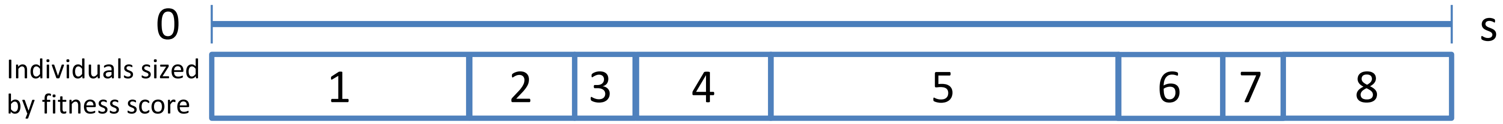
$\vec{x} \leftarrow$ first parent: $\langle x_1, \dots, x_l \rangle$
$\vec{v} \leftarrow$ second parent: $\langle v_1, \dots, v_l \rangle$
$p \leftarrow$ positive value defining how far we outrach the hyper cube (e.g., 0.25)

for i from 1 *to* l **do**
  repeat
    $\alpha \leftarrow$ random value from $-p$ to $1+p$ inclusive
    $\beta \leftarrow$ random value from $-p$ to $1+p$ inclusive
    $t \leftarrow \alpha x_i + (1-\alpha) v_i$
    $s \leftarrow \beta v_i + (1-\beta) x_i$
  until t and s are within bounds
  $x_i \leftarrow t$
  $v_i \leftarrow s$
return $\vec{x}$ and $\vec{v}$

Moved lines mean that we use different α and β values for each element

A Better Selection Operation

- So far: SelectWithReplacement
  – Can lead to selecting the same individual multiple times
  – Can select some low-fitness individuals
- Better: Select with a higher probability an individual with a high fitness score: Fitness Proportionate Selection (or Roulette Selection)
  – Idea:
    - Span a value range that is proportional to an individual's score
    - Concatenate all value ranges
    - Compute a random number in the all-value range and look up the corresponding individual



Fitness-Proportionate Selection (FPS)

$\vec{p} \leftarrow$ population consisting of a vector of individuals: $\langle \overrightarrow{p_1}, \ldots, \overrightarrow{p_l} \rangle$
$\vec{f} \leftarrow$ fitness score of each individual (same order as $in\ \vec{p}$): $\langle f_1, \ldots, f_l \rangle$

forall $f\ in\ \vec{f}$ **do**
  if $f == 0$ **then**
    $f \leftarrow 1.0$
for i from 2 to l ***do***
  $f_i \leftarrow f_i + f_{i-1}$

Deal with 0 fitness score to have at least a tiny chance to be accepted
Build the value range of all fitness scores as a cumulative density function (CDF)

$n \leftarrow$ random number from 0 to f_l inclusive
for i from 2 to l **do**
  if $f_{i-1} < n \leq f_i$ **then**
    return p_i
return p_1

Repeat this for each parent to be selected for crossover
Select the parent individual based on a random number falling into its corresponding interval

Note: That this is always an 1-based index value (not zero-based)

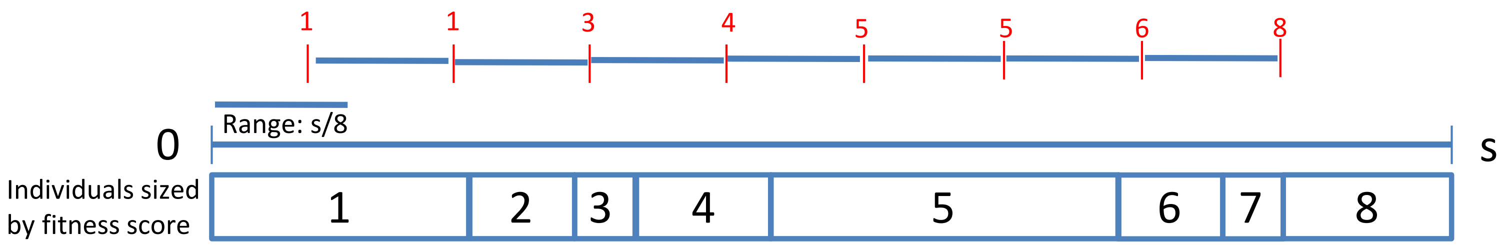
Problems of FPS

- Weak solutions can still be selected very often
- We might never the select the best solutions

- => Stochastic Universal Sampling (SUS)
  - Fit individuals get selected at least once
  - Also used in other areas (Particle Filters) under the term low variance resampling

Stochastic Universal Sampling (SUS) Algorithm

- Build fitness array as in FPS
- Draw a random number between 0 and s/n (here, s/8)
- Select individual at this position (here, 1)
- Increment current position by, s/n and repeat till n individuals have been selected
- Benefit: O(n) effort vs. O(n log n) for FPS
- Benefit: SUS guarantees that if an individual has a high score (>s/n), it will get chosen by the algorithm



In Code (for you to do at home)

$\vec{p} \leftarrow$ population consisting of a vector of individuals: $\langle \overrightarrow{p_1}, \ldots, \overrightarrow{p_l} \rangle$
$\vec{f} \leftarrow$ fitness score of each individual (same order as $in\ \vec{p}$): $\langle f_1, \ldots, f_l \rangle$
$index \ \leftarrow 0$
forall $f\ in\ \vec{f}$ **do**
  if $f\ == 0$ **then**
    $f \leftarrow 1.0$
for i from 2 to l ***do***
  $f_i \leftarrow f_i + f_{i-1}$

$offset \leftarrow$ random number from 0 to $\frac{f_l}{n}$ inclusive (where usually $n = l$)
for $f_{index} < offset$ do
  $index \leftarrow index + 1$
$offset \ \leftarrow offset + \frac{f_l}{n}$
return p_{index}

Repeat this for each parent to be selected for crossover

Nature of Fitness Value

- Assumption so far: Fitness value is on a metric scale
  - Distances between two fitness value has a meaning
  - Also called parametric function
- Often not the case: Consider the property reliability in software engineering
  - Systems that run reliably are up to 98.99, 99.97, 99.98, or 99.99 percent of a year (the peak is 99.99)
  - But using SUS all individuals have nearly the same probability to be selected
- What can we do?

Non-Parametric Selection Algorithm

- Non-parametric tests in statistics are based only on ranking
- There is no notion of distances
- Tournament Selection: Bigger is better

$P \leftarrow$ population of any representation
$t \leftarrow$ tournament size with $t \geq 1$
$Best \leftarrow$ individual picked at random from P with replacement
for i from 1 to t ***do***
  $Next \leftarrow$ individual picked at random from P with replacement
  if $Fitness(Next) > Fitness(Best)$ **then**
    $Best \leftarrow Next$
return $Best$

- Primary selection technique for a genetic algorithm!
  - Great tuning capability with tournament size (usually t=2)

Take Home Message:

- Evolutionary strategies use only mutation as tweak and select individuals using a truncate operation
- Genetic algorithms go a step further be recombining parents using a crossover operations
- Many variants to implement crossover, selection of individuals for the next generation, and mutation
  - Depends on the encoding of a solution (e.g., if nearby genes are correlated)
  - On the fitness function (e.g., if metric scale or ranking scale)
  - On exploration vs. exploitation

Next Lecture

- Exploitative algorithms of population based optimization techniques
  - Elitism
  - The Steady-State Genetic Algorithm
  - Tree-Style Genetic Programming Pipeline
  - Hybrid Optimization
  - Scatter Search
- Differential Evolution
- Particle Swarm Optimization