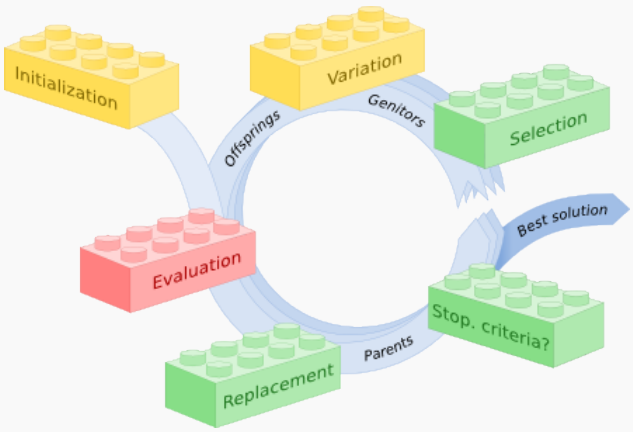
Evolutionary Programming

Evolutionary Programming (EP)

Generalities:

- Oldest idea of EA [Fogel et al., 1964];
- population based thought as parallel search (not be confused with parallel implementation);
- oriented towards continuous/real-valued optimisation problems!



EP: representation

- Each individual is represented as a real-valued vector:

$$\langle \mathbf{x}, \boldsymbol{\sigma} \rangle = \langle x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n \rangle$$

- where the first part represents a solution $\mathbf{x}$ and the second is a set of self-adapting parameters $\boldsymbol{\sigma}$, also referred to as *meta-EP*.
- This representation can be seen as n "evolving" normal distributions (one for each i^{th} design variable)

$$\mathcal{N}(\mathbf{x}[i], \boldsymbol{\sigma}[i])$$

EP: initialisation

- A population of μ individuals is generated.
- The design variables x_i are uniformly sampled within the decision space.
- The moving strategies σ_i are uniformly sampled between 0 and 1: $\mathcal{U}(0, 1)$

EP: parent selection

- Each solution generates a point, namely offspring!
  - i.e. No actual selection needed, all individuals are singularly considered!
  - μ individuals $\Rightarrow$ μ offspring!

EP: recombination

- There is only one variation operator, i.e. mutation (no crossover!), which is more appropriately referred to as recombination.
- Each individual $\langle \mathbf{x}, \boldsymbol{\sigma} \rangle$ generates an offspring $\left\langle \mathbf{x}', \boldsymbol{\sigma}' \right\rangle$ according to the formulas:

$$\begin{cases} \sigma_i' = \sigma_i \left(1 + \alpha \cdot \mathcal{N}(0,1)\right) \\ x_i' = x_i + \sigma_i' \cdot \mathcal{N}(0,1) \end{cases} \quad i = 1, 2, \ldots, n \quad \forall \langle \mathbf{x}, \boldsymbol{\sigma} \rangle \in Pop$$

- Each individual contains the information about the following move (early self-adaptation) with usually $\alpha = 0.2$.

EP: survivor selection
(deterministic variant)

- The so-called $(\mu + \mu)$ selection strategy is usually applied:
  - the μ parents and the μ offspring are taken into account and sorted on the basis of their fitness value;
  - those μ solutions that have the best fitness/objective function value are selected for the subsequent generation.

This approach is elitist!

The point displaying the best fitness value always find a place in the next generation!

EP: survivor selection
(stochastic variant)

- Each solution is evaluated against q other randomly chosen solutions (round-robin);
- for each comparison, a "win" is assigned to the solution under examination when it outperforms one of the randomly selected solution;
- the solutions who scored most wins are selected for the following generation.

Still elitist

but some (good/bad) solutions are selected depending on the random choice of the competitors.

Genetic Algorithm

Genetic Algorithm (GA)

Generalities:

- Developed: USA in the 1970's.
- Early names: J. Holland, K. DeJong, D. Goldberg [Goldberg, 1989][1].
- Typically applied to:
  - discrete optimization;
  - but also to real-valued problems.
- Attributed features:
  - not too fast;
  - versatile, good metaheuristic for combinatorial problems.

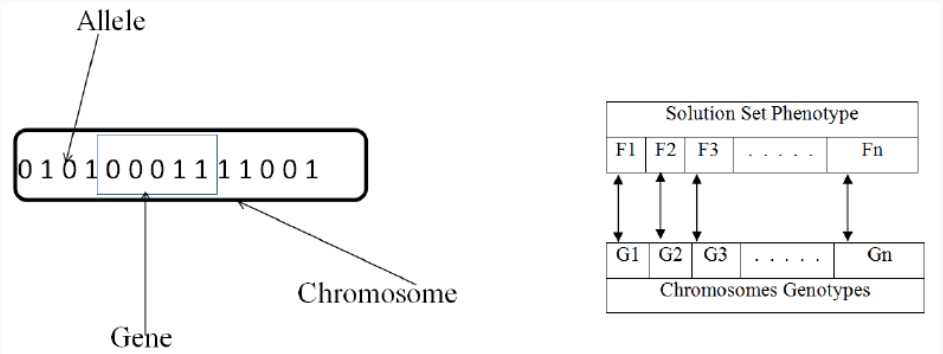
[1] Interesting book about GAs. Consider also [Eiben and Smith, 2003].

Simple Genetic algorithm
[Holland, 1975]

- Holland's original GA is now known as the simple genetic algorithm (SGA).
- The main features of this algorithm is the use of binary encoding.
- This choice was done in the 1970's to remark the analogy with biology (it is indeed inspired from biology!).
- This choice is generally VERY INEFFICIENT and should never applied unless the problem is naturally binary (here reported for historical reasons).

Representation

- A chromosome is encoded as a binary number: "chromosome", composed of "genes" representing a given parameter to be optimised, and encoded with a given number of "alleles"[2].
- Due to the inner discretisation of the binary encoding GA can turn out more efficient in discrete optimization.



[2]N.B. If a gene represents a real number, we need to fix a precision (i.e. number of binary places): higher the precision higher the dimensionality of the discrete problem $\Rightarrow$ difficult problem, slow evolution!

Parent selection mechanism

- The individuals that are undergoing recombination are selected by means of a Selection Mechanism.
- Classical selection mechanisms (there is many more but we will focus on these three):
  - Tournament;
  - Fitness Proportionate;
  - Ranking.

Selection pressure:

the property of the selection component in following the promising search directions (in other words, a parent selection mechanism which selects the best individuals many times has a high selection pressure).

Tournament selection

- Pick up a k solutions (at random) and compare their fitness, the better individual is in the mating pool.

Widely used! (usually the tournament size is $k = 2$)

- It does not require a sorting or a global knowledge (e.g. in a parallel implementation only a sub-portion of the population could be available) of the fitness distribution over the individuals of the population.
- The higher k, the higher the probability of selecting o good solution.
- It can be performed with or without replacement (see [Eiben and Smith, 2003] for details).

Fitness Proportionate Selection (FPS)

i.e. better individuals have higher probabilities to be selected! ✓

- It is given a probability P_i to be chosen to each i^{th} solution according to $P_i = \frac{f_i}{\sum_{i=1}^{\mu} f_i}$;
- the individual/individuals are selected through a mechanisms considering such probability (usually "roulette wheel").

In some scenarios:

- outstanding individuals take over the entire population quickly: premature convergence ✗
- when fitness values are all similar (or equal → plateau) there is no selection pressure ✗
- If this happens we adjust the previous formula with Goldberg's **sigma scaling**, (see [Eiben and Smith, 2003] and [Goldberg, 1989] for details).

Ranking selection
(always preserves some selection pressure ✓)

- Individuals are sorted according to their fitness value and a probability is assigned according to their position in the list (rank), and a probability is assigned to each i^{th} solution by means of:
  - *Linear Ranking*: $p_{lin-rank}(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$, $1.0 < s \leq 2.0$.
  - *Exponential Ranking*: $p_{exp-rank}(i) = \frac{1-e^{-i}}{c}$, c is a normalisation factor.

We still need a mechanism using this probability.

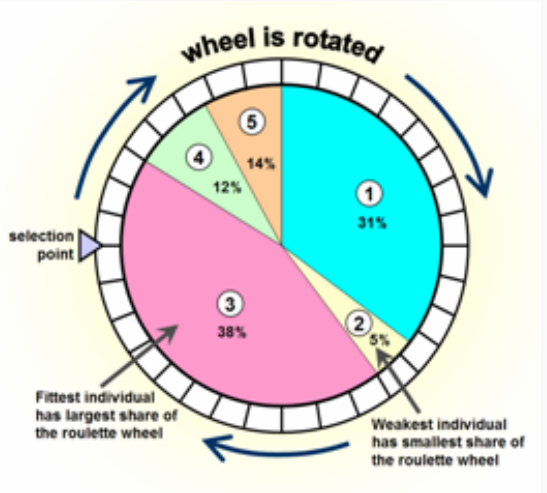
- s can be tuned to adjust the probability.
- Exponential ranking is preferred if an higher emphasis on selecting above-average individuals is required.

Implementing probability selection:
roulette wheel algorithm.

- We work out probabilities P_i according to FPS or Ranking, then we select one or more individuals by spinning a one-armed roulette wheel with μ sectors, whose size is equal to P_i.

Roulette-Wheel algorithm (single spin)

$i = 1$
while $\sum_{\mathbf{1}}^{i} P_i < \mathcal{U}(0, 1)$ **do**
  $i = 1 + 1$
end while
Output i^{th} **individual in the population**



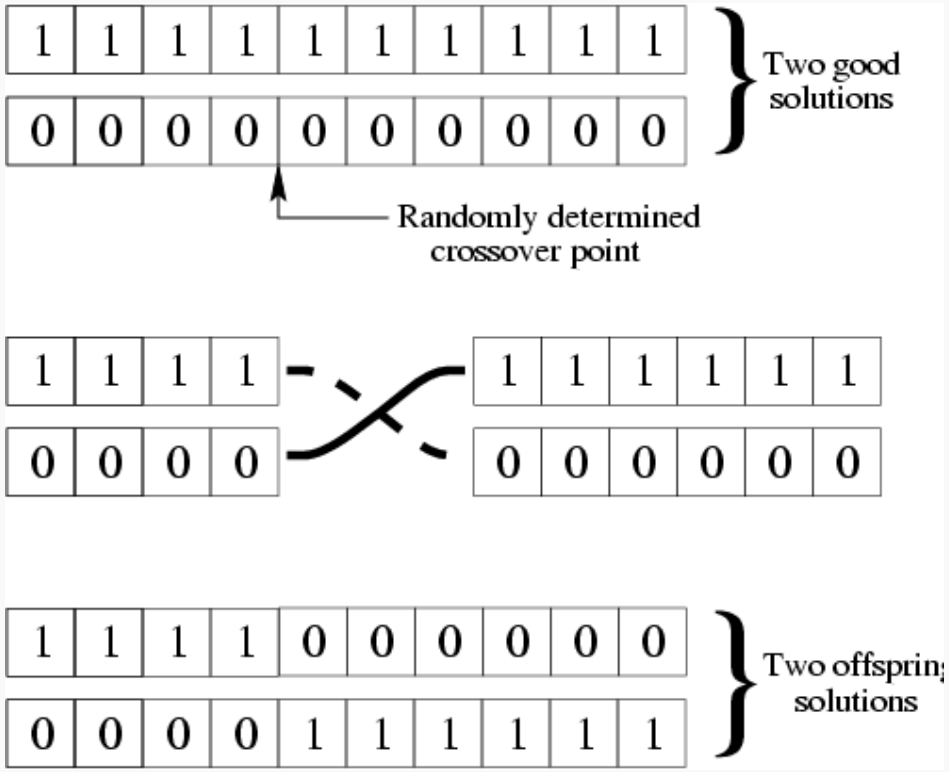
- First spin $\Rightarrow$ first element, second spin $\Rightarrow$ second element,...
- Spins are independent! (not suitable when multiple solution have to be selected ✗)
  - in this case Stochastic universal Sampling can be adopted (see [Eiben and Smith, 2003])

Crossover

- The selected parents undergo recombination.
- In a SGA, the recombination is the crossover.
- Crossover is an operator which combines two parents in order to produce one, two or more offspring.
- The analogy of biological crossover in binary encoding is very straightforward.

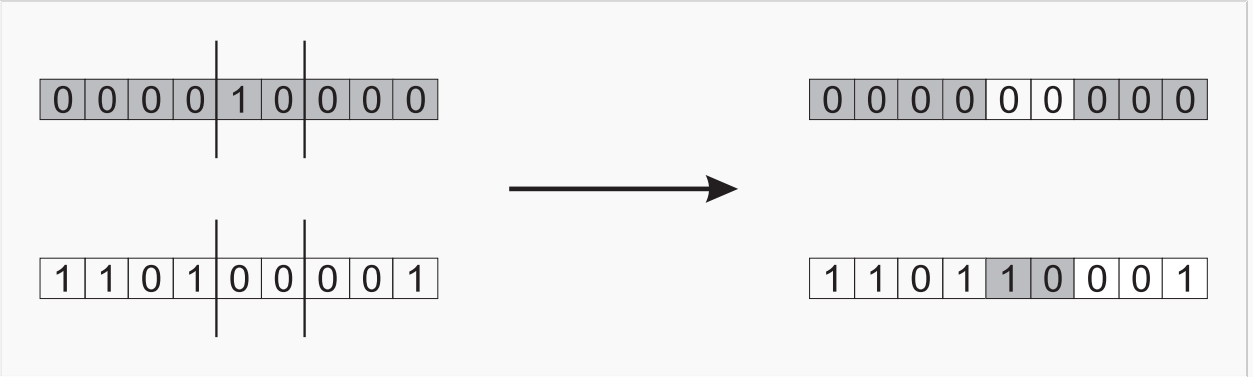
Crossover
1-*point crossover*

- It selects a random "cut-point" and switch head and tail of two chromosomes:



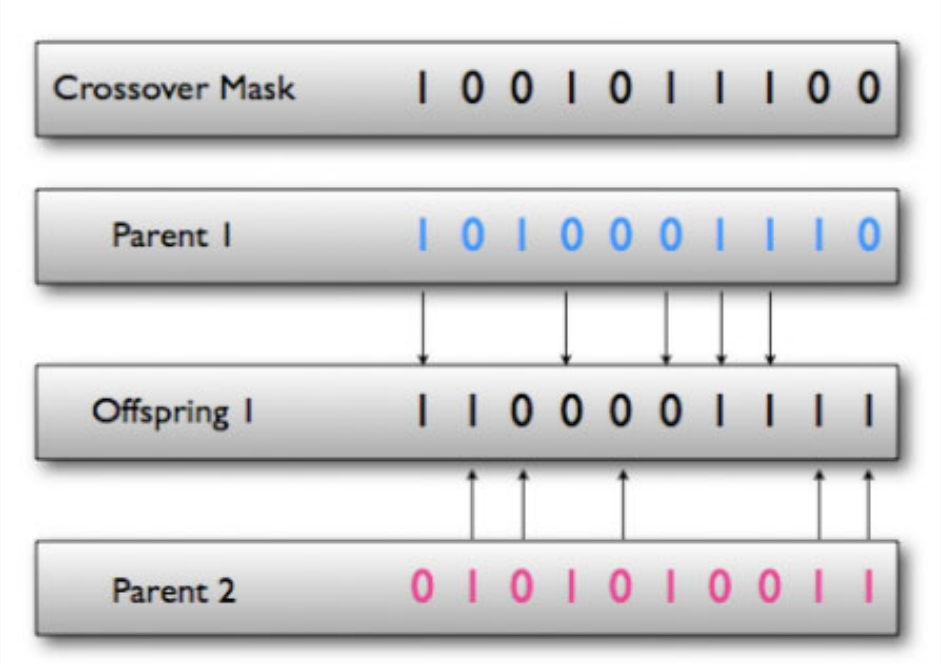
Crossover
N-point crossover

- Choose n random crossover points;
- split along those points;
- glue parts, alternating between parents:



Crossover
Uniform crossover

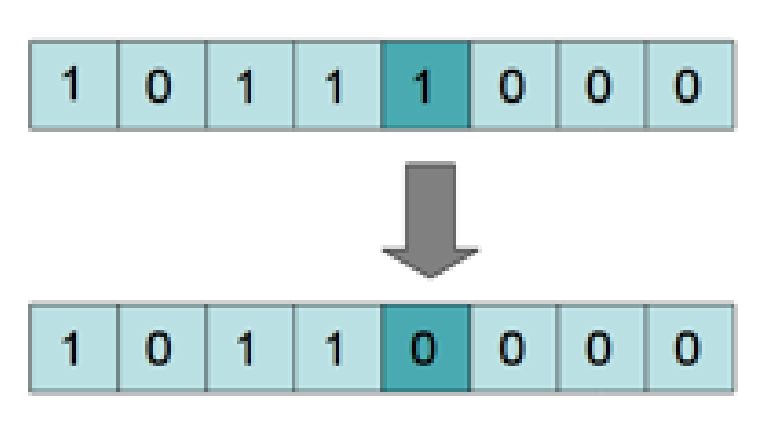
- Randomly (uniform distribution) initialise a mask for generating the first child:
  - we do not know how many cut-points we are going to have (they are uniformly distribute!)



- Make an inverse copy of the output if a second child is needed!

Mutation

- It is usually applied to the newly generated offspring before calculating its fitness value.
- Alter each gene independently with a probability p_m (referred to as "mutation rate").



Crossover or mutation?

Which one is better? Which one is the most important?

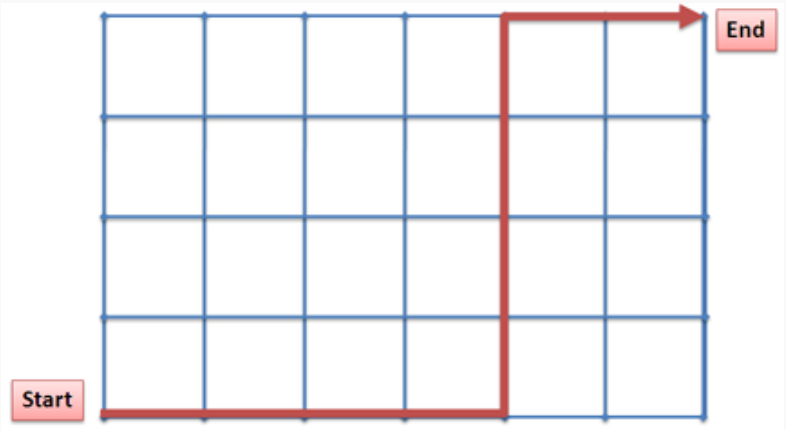
- It depends on the problem, but, in general, it is good to have both!
- Both have a role but mutation-only-EA is possible, crossover-only-EA seems not to work.

Survivor selection

- The main feature of a GA is that the algorithm must be generational (also called age-based):
  - in other words, the parents must be replaced by the newly generated offspring.
- Some implementations employ elitism: a restricted number of parents, the best, are copied for the subsequent generation.

Integer representation

- Some problems (e.g. image processing paramters, evolve a path in a grid...) naturally have integer variables:



in the example every gene is a step taken from
$\{North, South, West, East\}$
$\Rightarrow \{1, 2, 3, 4\}$

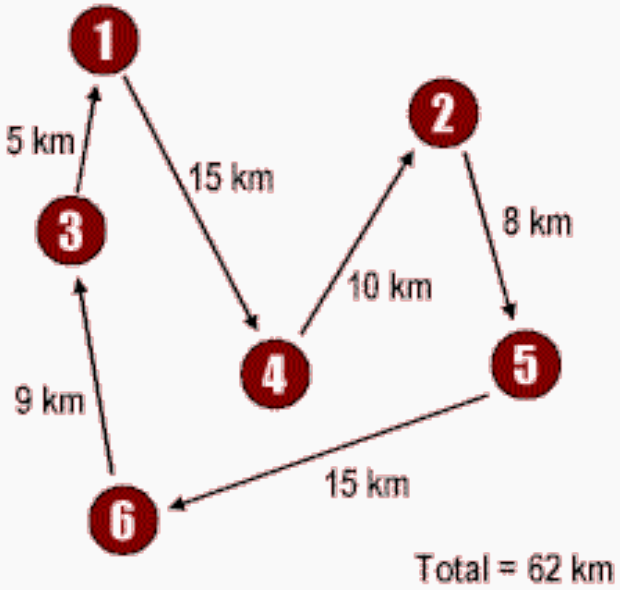
- We use the same selection and crossover strategies of binary GAs!
  - But different mutations...

Mutation
(integer representation)

- *Creep mutation*: $\forall$ genes in the chromosome, there is probability p of adding or subtracting a small random integer.
- *Random reset*: $\forall$ genes in the chromosome, there is probability p to be replaced with a random integer chosen from the set of permissible values (no need to suture solution!).

Permutation representation

- Some problems (e.g. e.g. TSP, scheduling...) naturally take the form of deciding on the order in which a sequence of events has to occurs:



$$\Rightarrow [1, 4, 2, 5, 6, 3]$$

- Still "integer" coded, but the same integer value never appears twice!

Crossover
(permutation representation)

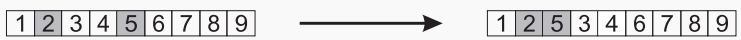
- It is difficult to be applied in a permutation based GA:
  - if we exchange variables we can have the same value multiple twice!
- The most popular were designed for specific applications:
  - Order 1 crossover;
  - Partially mapped crossover;
  - Cycle crossover;
  - Edge crossover.

A description is given in [Eiben and Smith, 2003].

Mutation
(permutation representation)

- *Swap*: Pick two design variables at random and swap their positions:

- *Insert*: Pick two design variables at random and move the second to follow the first, shifting the rest along to accommodate:



- *Inversion*: Pick two design variables at random and then invert the substring between them:

- *Scramble*: Pick two design variables at random and randomly rearrange the design variables in the substring:

Real-valued representation
(or of floating-point representation)

- Many problems occur as real valued problems, e.g. continuous parameter optimisation, design of an aircraft etc.
- In these cases, the most reasonable representation is a real valued as it would guarantee to work directly on the problem and not in a "deformed space".
- Parent and survivor selections are the same of SGA!

Crossover
(real-valued representation)

For a real-valued GA we distinguish between

- *Deterministic* crossover: generally inefficient as it does not use the real-valued nature of the problem.
- *Arithmetic* crossover: the value of the design variables is varied due to some arithmetic operations.

Given a child $\mathbf{z}$ obtained from parents $\mathbf{x}$ and $\mathbf{y}$:

deterministic $\Rightarrow \mathbf{z}[i] = \mathbf{x}[i]$ **or** $\mathbf{y}[i]$

All those seen for SGA!

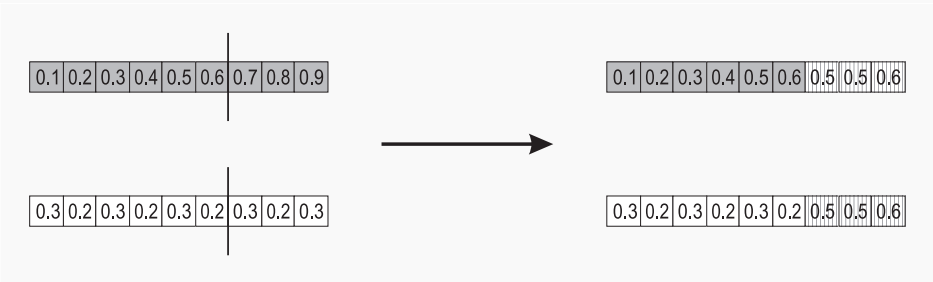
arithmetic $\Rightarrow \mathbf{z}[i] = \alpha\mathbf{x}[i] + (1-\alpha)\mathbf{y}[i]$, $\alpha \in [0,1]$

It can be:

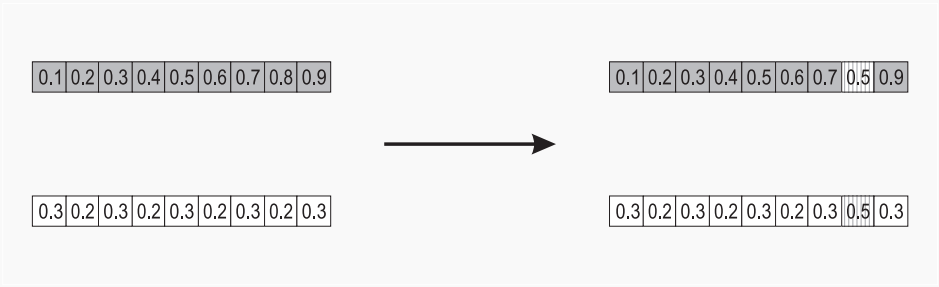
- randomised arithmetic recombination ($\alpha = \mathcal{U}(0,1)$);
- simple, single or whole arithmetic recombination.

Crossover

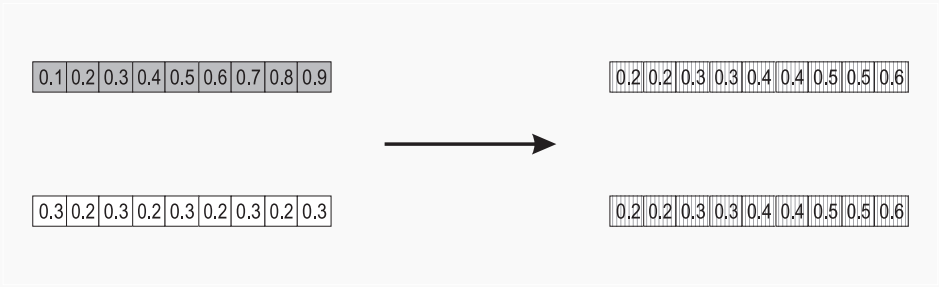
Simple, single and whole arithmetic, ($\alpha = \frac{1}{2}$)



Simple arithmetic recombination



Single arithmetic recombination



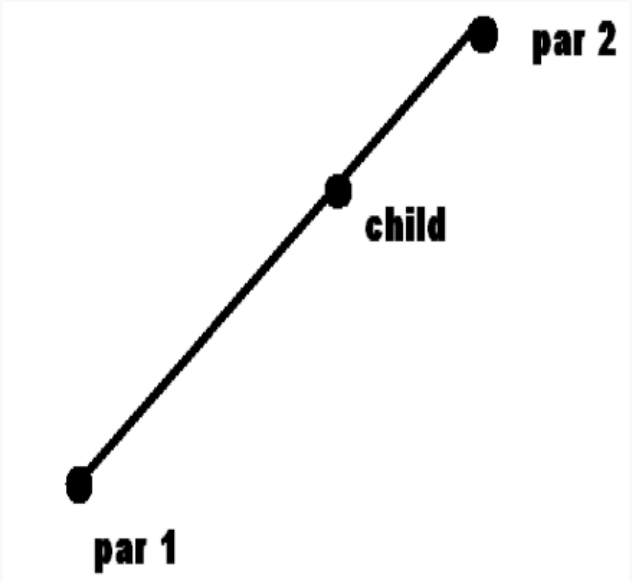
Whole arithmetic recombination

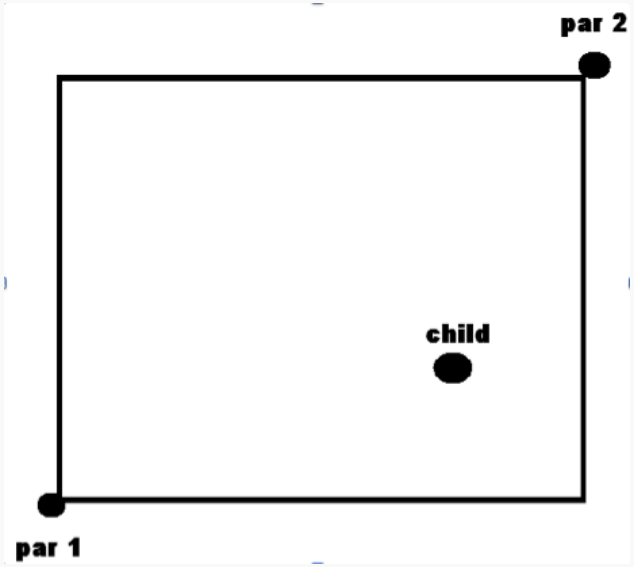
Box crossover
(real-valued representation)

- The box crossover is another important method where:

$$\mathbf{z}[i] = min\left(\mathbf{x}[i], \mathbf{y}[i]\right) + \alpha|\mathbf{x}[i] - \mathbf{y}[i]|,\ \alpha = \mathcal{U}\left(0, 1\right)$$

- It samples one (or more) points within the hyper-rectangle identified by the two parents, while the algorithm crossover works on a straight line:





Mutation
(real-valued representation)

As for the binary case, there is a probability p f applying a mutation scheme that:

- replace the design variable with a uniformly distributed random number within the bounds (no need to saturate!):
  $\mathbf{x_j}[i] = \mathcal{U}\left(\mathbf{x^L}[i], \mathbf{x^U}[i]\right), \forall i = 0, 1, 2, \ldots, n;$

or

- add a small (rarely a big) random quantity the variable from Gaussian (or Cauchy) distribution: e.g.
  $\mathbf{x_j}[i] = \mathbf{x_j}[i] + \mathcal{N}\left(0, \sigma_{small}\right), \forall i = 0, 1, 2, \ldots, n.$

Laboratory and participation work:

- For the four problems under consideration in this module in 10D and 50D:
  - implement a real-valued GA with 1)tournament selection; 2) box crossover; 3) Gaussian mutation; 4) Replacement of the population;
  - for the two dimensionality levels, tune the parameters (e.g. population size,. . .) that appear the most convenient, and draw your own conclusion.

References I

Eiben, A. E. and Smith, J. E. (2003).
Introduction to Evolutionary Computation.
Springer-verlag, Berlin.

Fogel, L. J., Owens, A., and Walsh, M. (1964).
On the evolution of artificial intelligence(artificial intelligence generated by natural evolution process).
In *National Symposium on Human Factors in Electronics, 5th, San Diego, California*, pages 63–76.

Goldberg, D. E. (1989).
Genetic Algorithms in Search, Optimization and Machine Learning.
Addison-Wesley Publishing Co., Reading, MA, USA.

Holland, J. (1975).
Adaptation in Natural and Artificial Systems.
University of Michigan Press.