Lecture 5:
GOSET

What is GOSET?

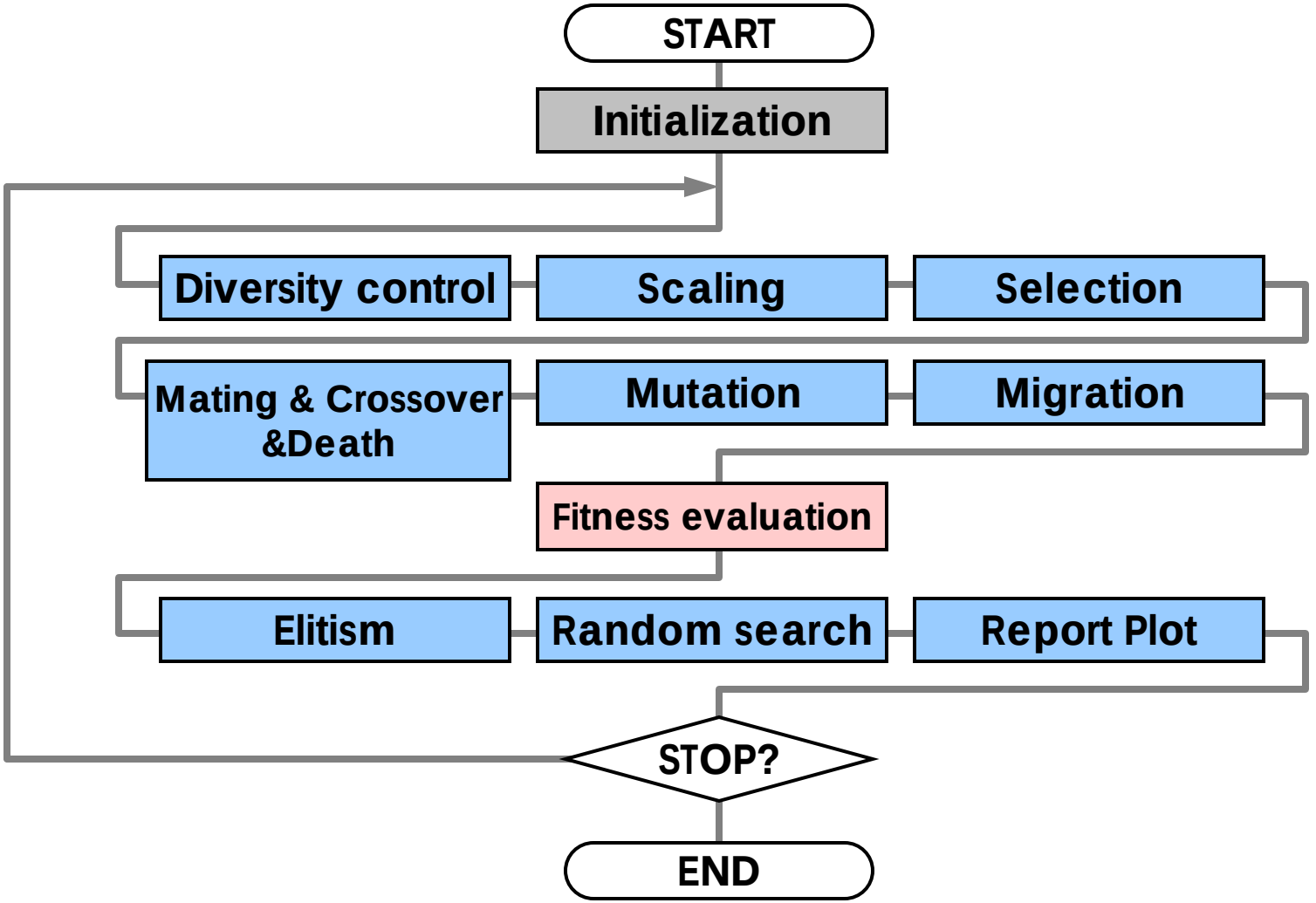
- **GOSET** stands for **G**enetic **O**ptimization **S**ystem **E**ngineering **T**ool

- GOSET is a MATLAB based genetic algorithm toolbox for solving optimization problems

GOSET Features

- Wide range of choices for genetic operators
- Single-objective optimization
- Multi-objective optimization
- Modular Structure
- GUI Interface
- GOSET DLL

Algorithm Execution



Data Structure

Data Structures

To conveniently process the information used in GOSET, the following data structures are employed

Data structure	Contents	No. of fields
P	Population	15
GAP	GA Parameters	76
GAS	GA Statistics	6

Population Data Structure (P)

P.blckeval	Block evaluation flag
P.fithandle	Handle to the fitness function
P.size	The number of individuals in the population
P.nobj	Number of objectives
P.mfit	Fitness function values
P.fit	Aggregated fitness function values
P.eval	Fitness evaluation flag
P.age	Age of individuals

Population Data Structure (P)

P.ngenes	Number of genes in an individual
P.min	Minimum value of genes
P.max	Maximum value of genes
P.type	Types of genes
P.chrom_id	Chromosome ID of genes (for multiple chromosome)
P.normgene	Normalized gene values
P.gene	Gene values

Population Data Structure (P)

P.region	Geographic region of individuals
P.pen	Fitness weight values for penalizing in the diversity control

Basic Information

- P.size
- P.ngenes
- P.nobj

Population Fitness

- P.mfit (# obj. by pop. size)

- P.fit (1 by pop. size)

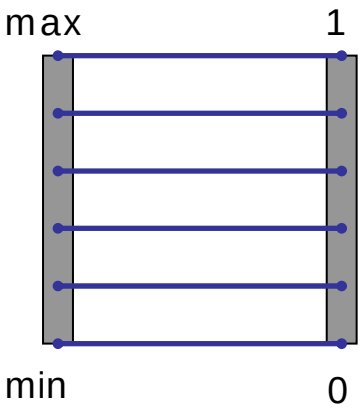
Gene Representation

- P.gene (# genes by pop. size)
- P.min (# genes by 1)
- P.max (# genes by 1)
- P.normgene (# genes by pop. size)

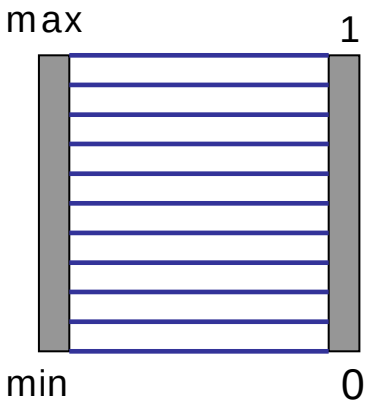
Encoding

- P.type (# genes by 1)
  - Determine the mapping method of the normalized gene value to its actual value
  - There are three different types of mapping

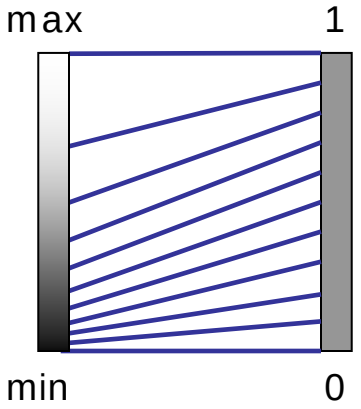
Integer



Real

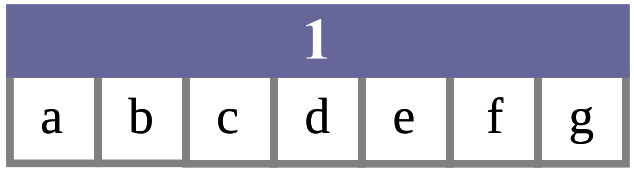
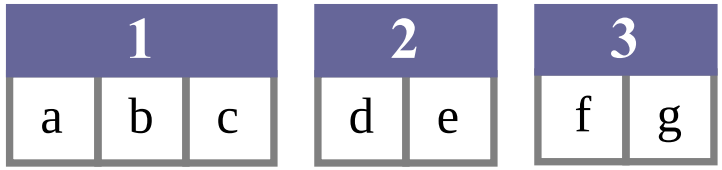
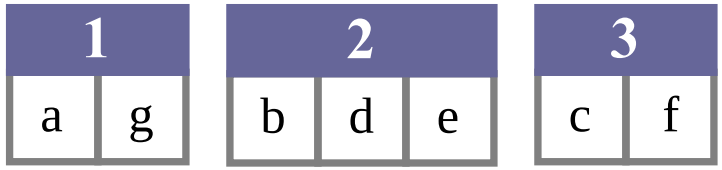


Logarithmic



Chromosome ID

Given the genes of an individual

P.chrom_id	Chromosome structure
P.chrom_id = [1 1 1 1 1 1 1]^T	
P.chrom_id = [1 1 1 2 2 3 3]^T	
P.chrom_id = [1 2 3 2 2 3 1]^T	

Genetic Algorithm Parameters (GAP)

Category	Field names
Fundamental parameters	GAP.fp_ngen GAP.fp_ipop GAP.fp_npop GAP.fp_nobj GAP.fp_obj
Diversity control parameters	GAP.dc_act GAP.dc_alg GAP.dc_spc GAP.dc_mnt GAP.dc_mxt GAP.dc_ntr GAP.dc_mnb GAP.dc_mxb GAP.dc_dc GAP.dc_nt
Selection algorithm parameters	GAP.sl_alg GAP.sl_nts GAP.sl_cah
Death algorithm parameters	GAP.dt_alg GAP.dt_nts GAP.dt_cah
Mating and crossover parameters	GAP.mc_pp GAP.mc_fc GAP.mc_alg GAP.mc_gac GAP.mc_ec
Mutation parameters	GAP.mt_ptgm GAP.mt_prgm GAP.mt_srgm GAP.mt_pagm GAP.mt_sagm GAP.mt_prvm GAP.mt_srvm GAP.mt_pavm GAP.mt_savm GAP.mt_pigm
Migration parameters	GAP.mg_nreg GAP.mg_tmig GAP.mg_pmig
Evaluation Parameters	GAP.ev_bev GAP.ev_are GAP.ev_ssd
Scaling parameters	GAP.sc_alg GAP.sc_kln GAP.sc_cst GAP.sc_kmxq GAP.sc_kmnq
Elitism parameters	GAP.el_act GAP.el_fgs GAP.el_fpe
Random search parameters	GAP.rs_fgs GAP.rs_fps GAP.rs_srp GAP.rs_sap GAP.rs_frp GAP.rs_fea
Reporting parameters	GAP.rp_lvl GAP.rp_gbr GAP.rp_crh
Objective plot parameters	GAP.op_list GAP.op_style GAP.op_sign
Pareto plot parameters	GAP.pp_list GAP.pp_xl GAP.pp_yl GAP.pp_title GAP.pp_style GAP.pp_sign GAP.pp_axis
Distribution plot parameters	GAP.dp_type GAP.dp_np GAP.dp_res
Gene data parameters	GAP.gd_min GAP.gd_max GAP.gd_type GAP.gd_cid

Genetic Algorithm Parameters (GAP)

- GAP.fp_ngen
- GAP.fp_ipop
- GAP.fp_npop
- GAP.fp_nobj
- GAP.fp_obj

Genetic Algorithm Statistics (GAS)

GAS.[Field name]	Description
GAS.cg	Current generation number
GAS.medianfit	The median fitness values of each objective (No. of objectives × No. of generations)
GAS.meanfit	The average fitness values of each objective (No. of objectives × No. of generations)
GAS.bestfit	The best fitness values of each objective (No. of objectives × No. of generations)
GAS.bestgenes	The best gene values for each objective over the generations (No. of genes × No. of generations × No. of objectives)
GAS.ne	The number of the total objective function evaluations

Genetic Algorithm Statistics (GAS)

GAS.cg	Current generation number
GAS.medianfit	Median fitness values (obj × gen)
GAS.meanfit	Average fitness values (obj ×gen)
GAS.bestfit	Best fitness values (obj × gen)
GAS.bestgenes	Best gene values (genes ×gen× obj)
GAS.ne	Number of objective function evaluations

GOSET
Genetic Operators

GOSET Genetic Operators

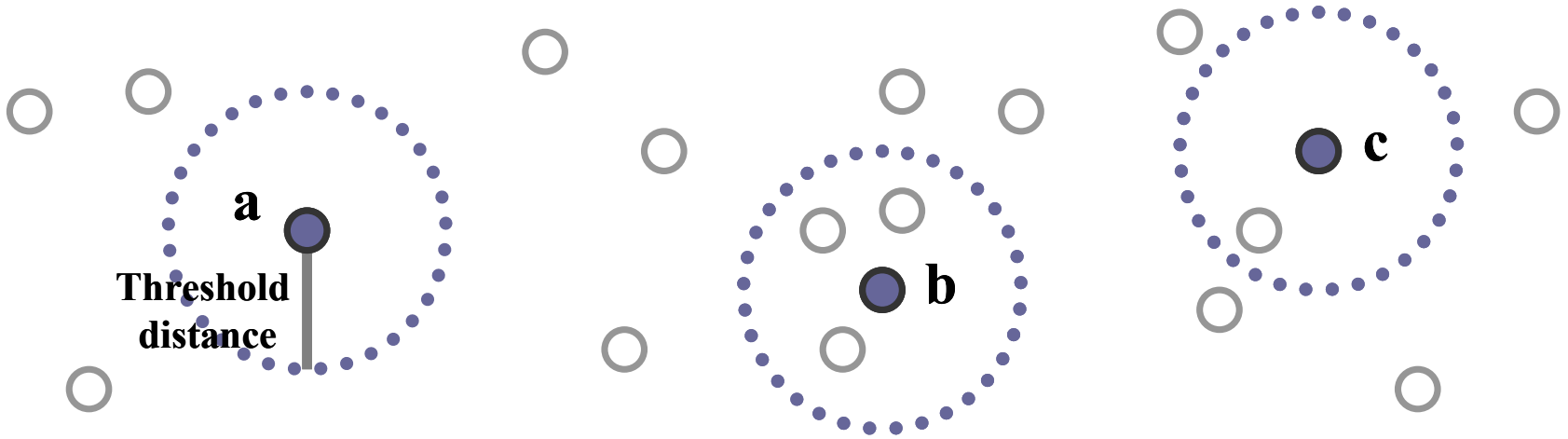
- Diversity control
- Scaling
- Selection
- Death
- Mating & crossover
- Mutation
- Migration
- Elitism
- Random search

Diversity Control

- Maintain population diversity
- Penalize individuals with many neighbors
- Four different diversity controls are available in GOSET

Diversity Control Method 1

- Fitness weight is inversely proportional to the number of neighboring individuals within the threshold distance

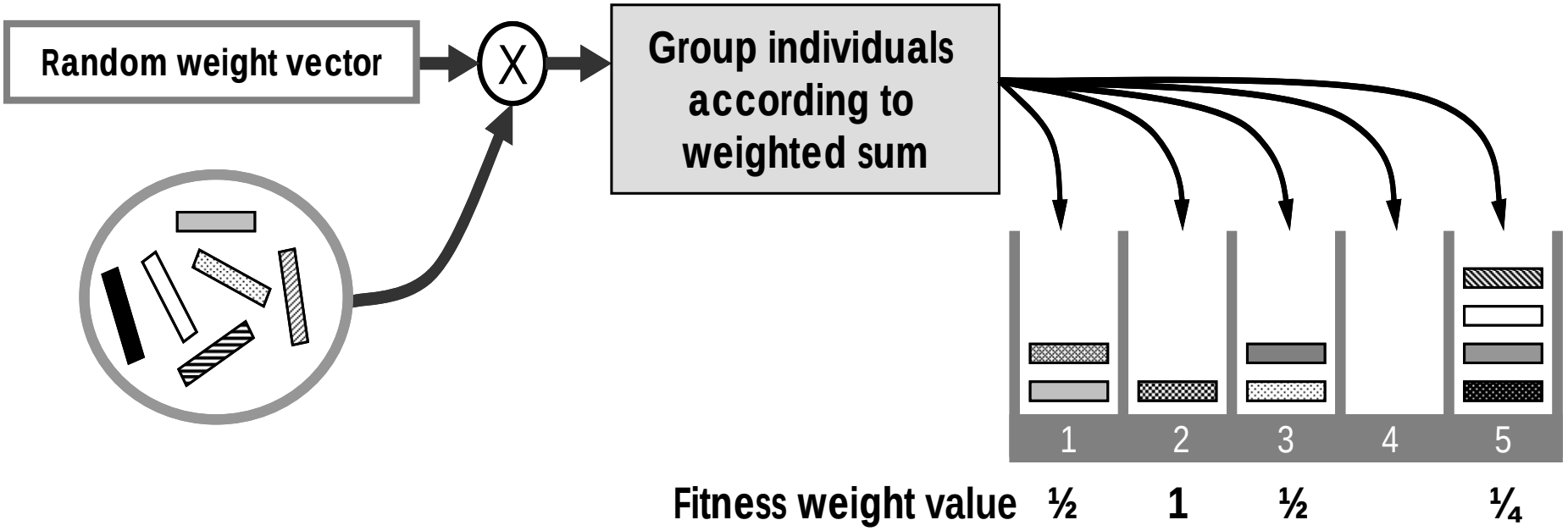


Fitness weight for a = 1 **Fitness weight for b = 1/4** **Fitness weight for c = 1/2**

Diversity Control Method 2

- Given an arbitrary weight vector, evaluate the weighted sum of genes for each individuals
- Group individuals with similar weighted sum
- Repeat multiple times with different weight vectors and the largest penalty function value is used for final fitness weight value

Diversity Control Method 2



$$\text{Fitness weight value} = \frac{1}{\text{Number of individuals in the bin}}$$

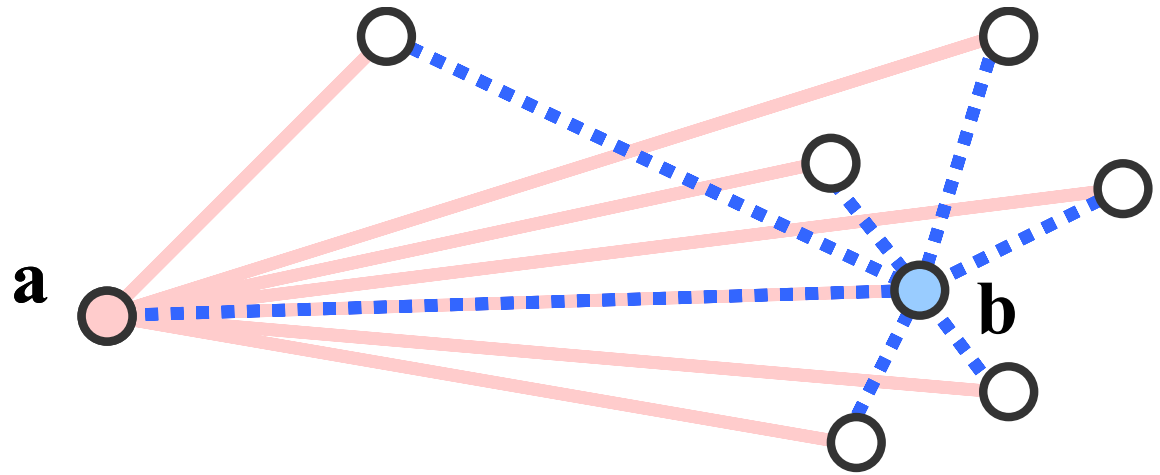
Diversity Control Method 3

- Evaluate the sum of infinity norm between the individual of interest and all other individuals
- Fitness weight is increasing as the distance sum increases

$$\text{Fitness weight for } k\text{'th individual} = \frac{1}{\sum_{i \in I} \exp\left(- \frac{\left\| \boldsymbol{\theta}_k - \boldsymbol{\theta}_i \right\|_\infty}{d_c} \right)}$$

where d_c is the distance constant (`GAP.dc_dc`)

Diversity Control Method 3



Fitness weight for a = 0.8

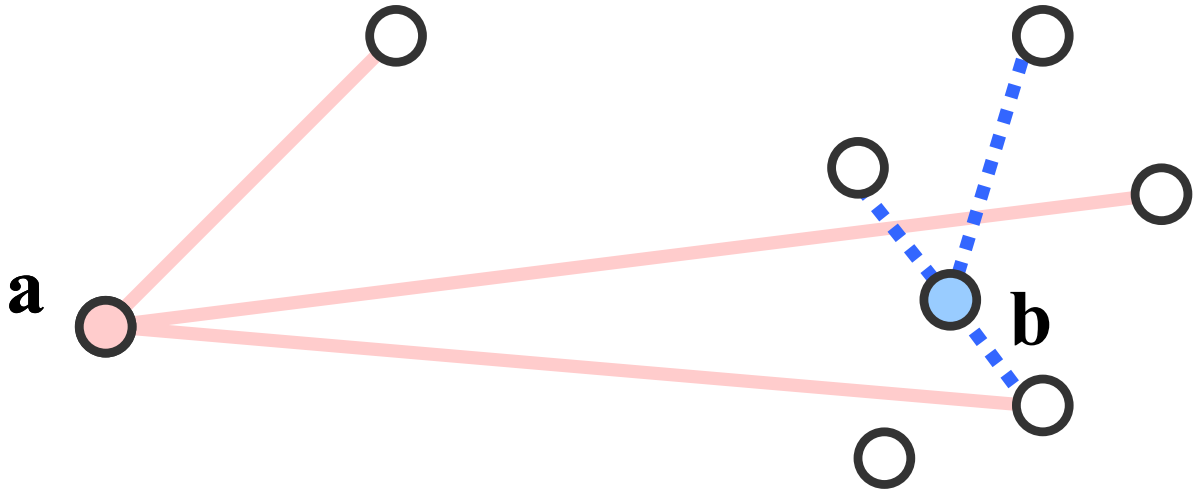
Fitness weight for b = 0.2

Diversity Control Method 4

- Similar to Diversity Control Method 3
- Only evaluate the sum of distances between the individual of interest and a certain number (`GAP.dc_nt`) of randomly chosen individuals

Diversity Control Method 4

GAP.dc_nt =3



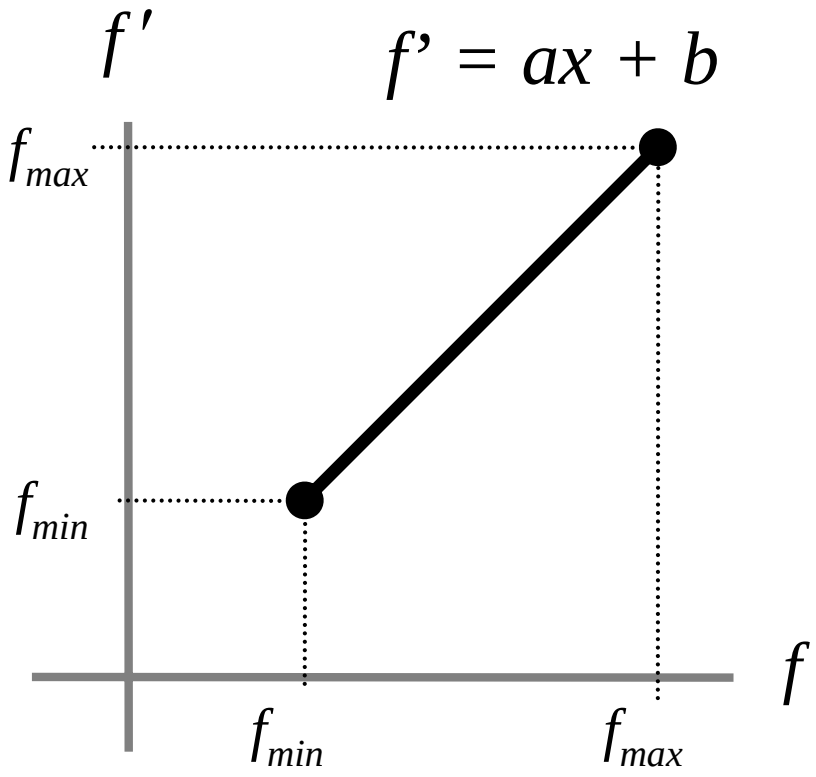
Fitness weight for a = 0.7

Fitness weight for b = 0.1

Scaling

- Purpose: maintain appropriate evolution pressure throughout evolution process
- Without scaling
  - Early Evolution: a few strong individuals usually dominate population quickly
  - Late Evolution: most individuals have similar fitness values and the evolution slows
- Seven scaling methods available

No Scaling

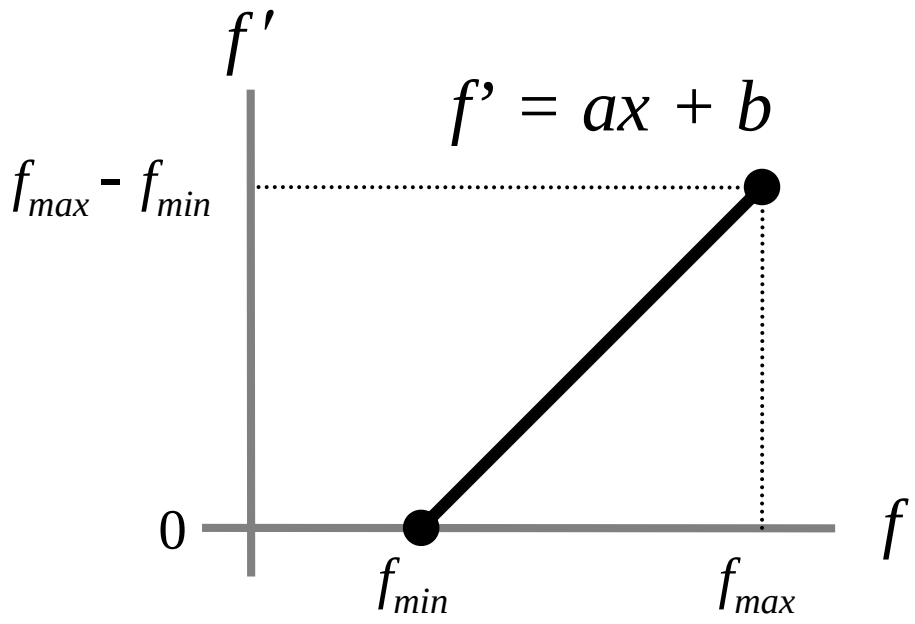


$a = 1$

$b = 0$

- Scaling is not applied and the actual fitness value is used
- Fitness functions must be constructed carefully
- Good for tournament selection

Offset Scaling

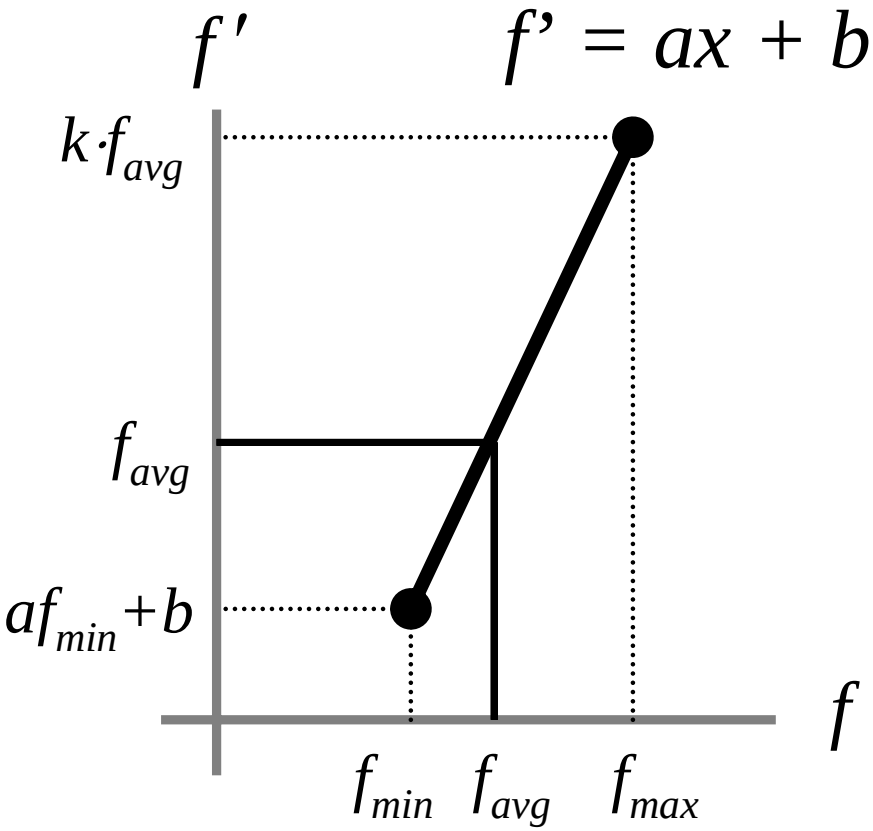


$a = 1$

$b = -f_{min}$

- Linear scaling
- Minimum fitness value is mapped to zero
- Default scaling algorithm

Standard Linear Scaling

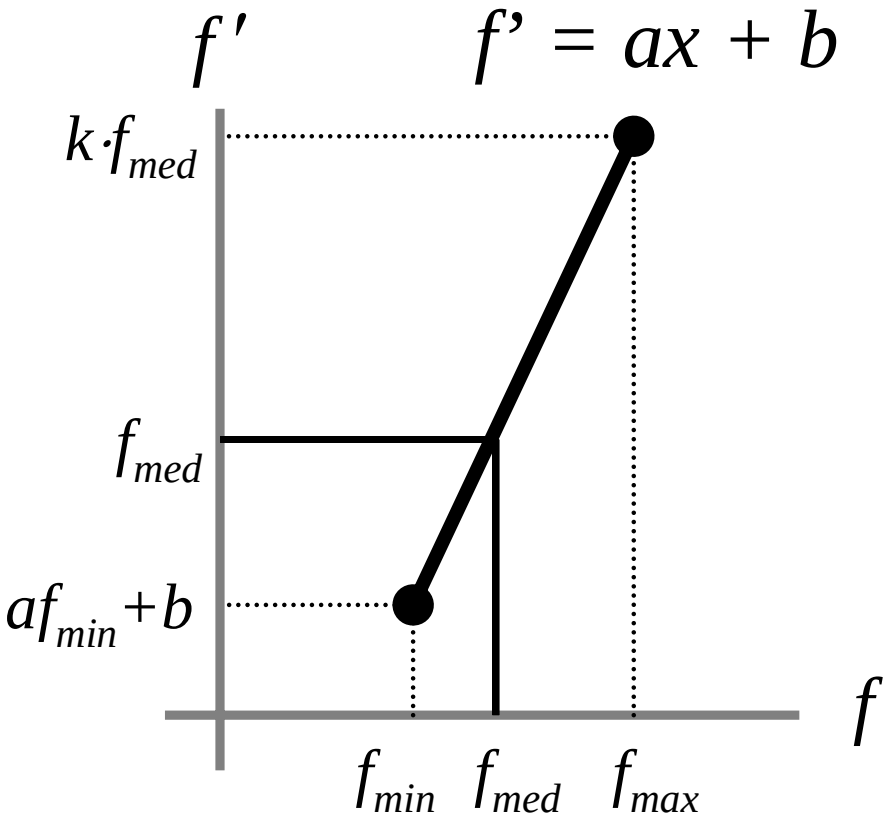


$$a = \frac{(k-1)\,f_{avg}}{f_{\max} - f_{avg}} \qquad b = f_{avg}\,(1-a)$$

$k =$ `GAP.sc_kln`

- Linear scaling
- Average fitness value does not change after scaling
- Most fit individual has mapped fitness k times bigger than that of average fit individual

Modified Linear Scaling

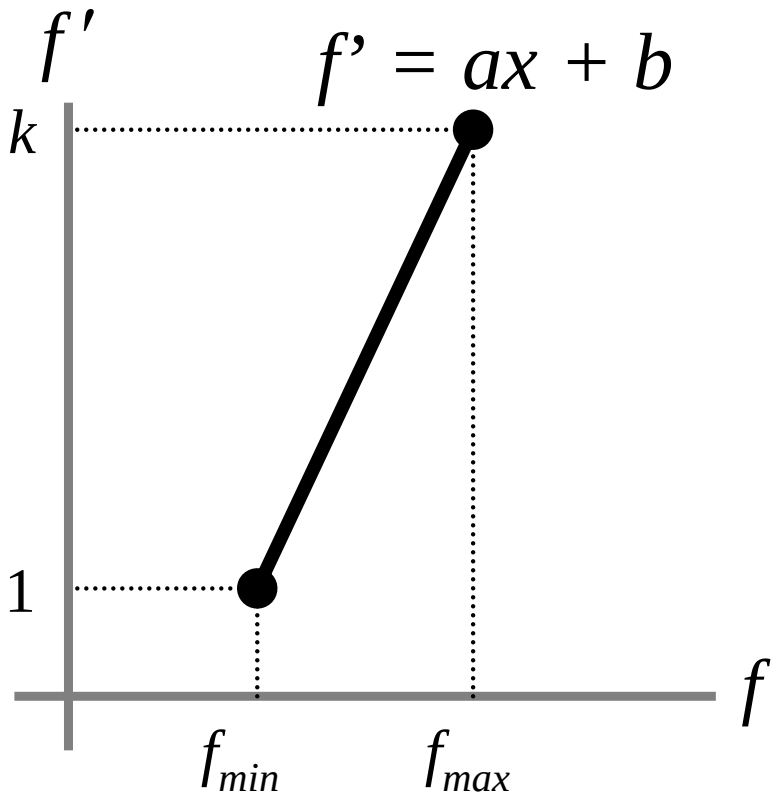


$$a = \frac{(k-1) f_{med}}{f_{\max} - f_{med}} \qquad b = f_{med}(1-a)$$

$k = \texttt{GAP.sc\_kln}$

- Linear scaling
- Median fitness value is preserved after scaling
- Most fit individual is mapped so that its fitness value is *k* times bigger than the median fit individual

Mapped Linear Scaling

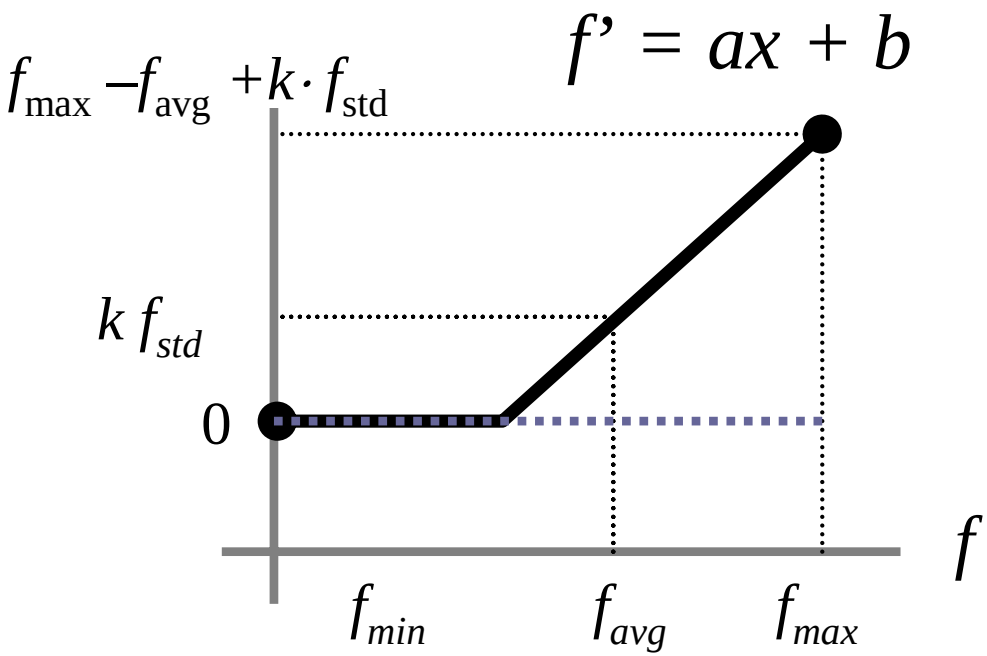


$$a = \frac{k-1}{f_{\max} - f_{\min}} \qquad b = -f_{\min} \cdot a + 1$$

k = `GAP.sc_kln`

- Linear scaling
- The minimum fitness value is mapped to 1, and the maximum fitness value is mapped to k

Sigma Truncation

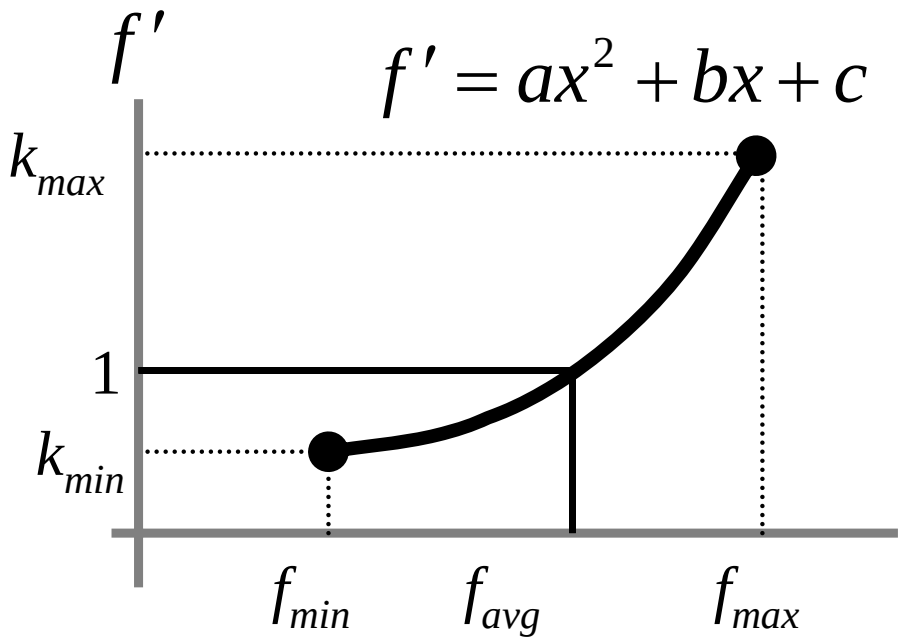


$a = 1 \quad b = -(f_{avg} - k_c \cdot f_{std})$

$k_c = \texttt{GAP.sc\_cst}$

- Linear scaling
- Fitness values offset so that f_{avg} is mapped to $k \cdot f_{std}$
- Resulting negative fitness values are clipped to zero
- Useful when most individuals have large fitness value but there are few individuals with small fitness values

Quadratic Scaling



- Non-linear scaling

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} f_{\max}^2 & f_{\max} & 1 \\ f_{\text{avg}}^2 & f_{\text{avg}} & 1 \\ f_{\min}^2 & f_{\min} & 1 \end{bmatrix}^{-1} \begin{bmatrix} k_{\max} \\ 1 \\ k_{\min} \end{bmatrix}$$

$k_{\max} = \texttt{GAP.sc\_kmxq}$

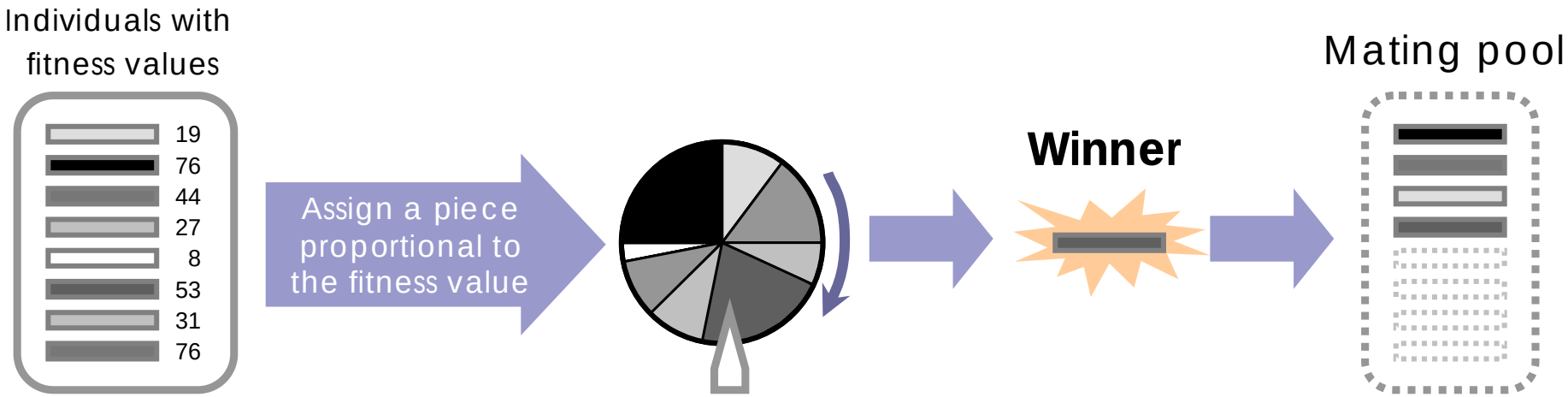
$k_{\min} = \texttt{GAP.sc\_kmnq}$

Selection

- Purpose: select individuals from the population to constitute a mating pool
- When the multiple regions are used, selection operation is restricted to each region and picks the same number of individuals as those in the current region
- Roulette wheel selection
- Tournament selection
- Custom selection

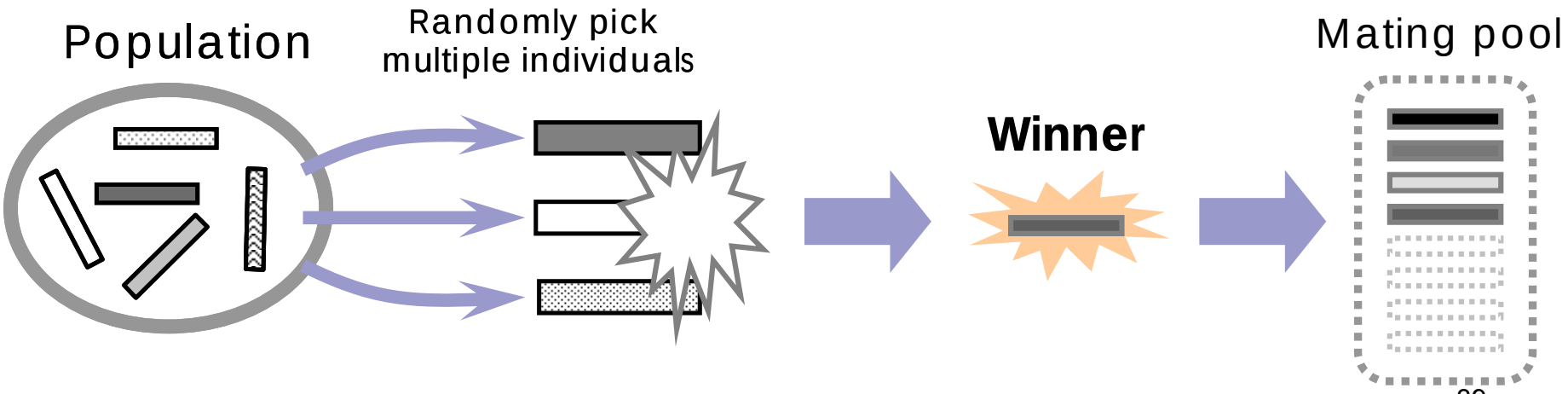
Roulette Wheel Selection

- An individual is selected with the probability proportional to its fitness value
- It is more likely that the better individual is selected - the principle of the survival of the fittest



Tournament Selection

- Individuals are randomly chosen from the population and the one with best fitness value is selected
- The number of individuals for the tournament is a design parameter (GAP.sl_nts)



Death

- Purpose: Create a list of individuals who die and are replaced by the children
- There are six different death algorithms
- Custom death algorithm can be used

Death Algorithms

- Replacing parents
  - Parents are replaced by their own children
- Random selection
  - The parents to be replaced are randomly chosen
- Tournament of fitness
  - Individuals to be replaced based on aggregate fitness
  - GAP.dt_nts parents are randomly chosen for tournament
  - Individual with worst aggregate fitness value marked for death

Death Algorithms

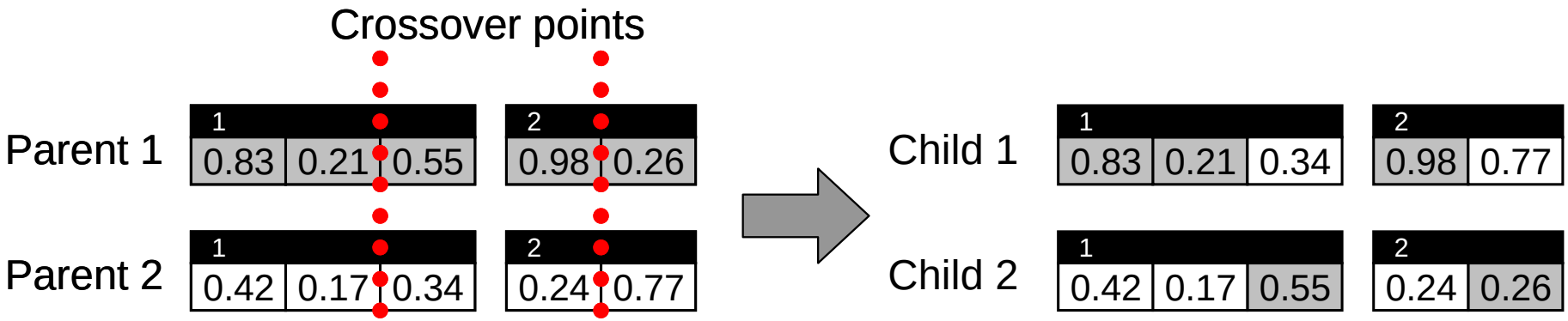
- Tournament on age
  - Tournament based on the age. Among randomly chosen `GAP.dt_nts` parents, the oldest dies.
- Custom algorithm
  - The custom function handle is assigned to GAP.dt_cah
- Random algorithm
  - The death algorithm is randomly chosen among the first four death algorithms at each generation.

Mating-Crossover

- Crossover operation is performed on the normalized gene values and the actual gene values are refreshed based on the normalized gene values
- When the resulting gene value is illegal, it is automatically adjusted using the **ring-mapping**
  - Ring-mappng maps a value to the modulus after division by 1
  - Example 1.2 $\rightarrow$ 0.2 and -2.1 $\rightarrow$ 0.9
- There are five different mating-crossover algorithms

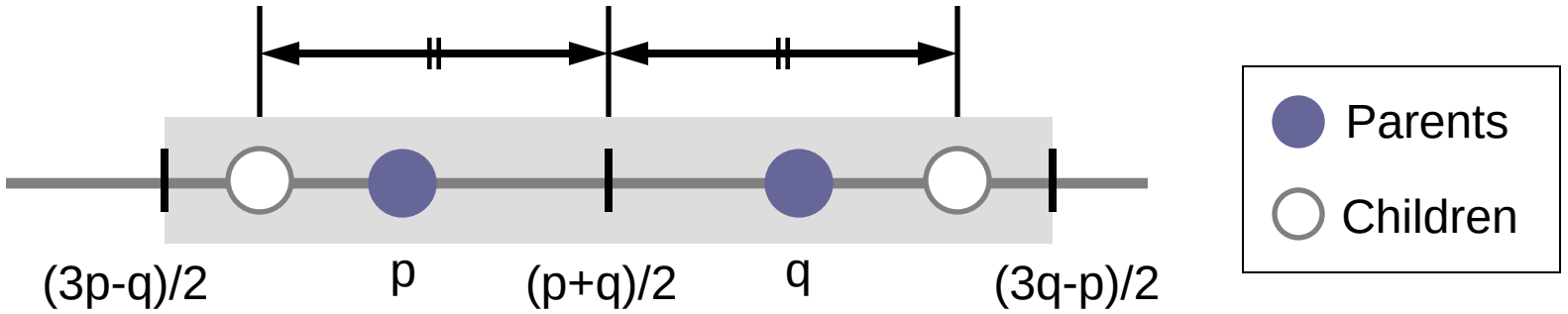
Single-Point Crossover

- Similar to the single point crossover operator in binary-coded GAs
- In multiple-chromosome setting, single point crossover occurs in each chromosome



Simple Blend Crossover

- Children are generated from the weighted sum of their parents
- Gene values of children have same distance from the average gene value of parents

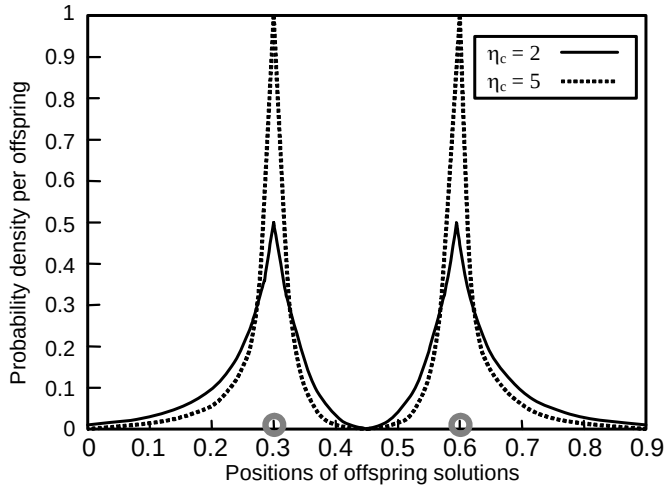


Types of Simple Blend Crossover

- **Scalar simple blend crossover**
  - Each gene has different ratio of blending
  - Example
    - P1=[0 0.8 0.3] P2=[1 0.2 0.5]
    - C1=[0.25 0.95 0.38] C2=[0.75 0.05 0.42]
- **Vector simple blend crossover**
  - All genes are blended using same ratio
  - Example
    - P1=[0 0.8 0.3] P2=[1 0.2 0.5]
    - C1=[0.25 0.65 0.35] C2=[0.75 0.55 0.45]

Simulated Binary Crossover

- Mimics the effect of single-point crossover operator in binary-coded GA
- Simulated binary crossover uses probability density function that simulates the single-point crossover in binary-coded GA



Simulated Binary Crossover

- **Scalar simulated binary crossover**
  - Each gene has different ratio

- **Vector simulated binary crossover**
  - All genes use same ratio

Random Algorithm

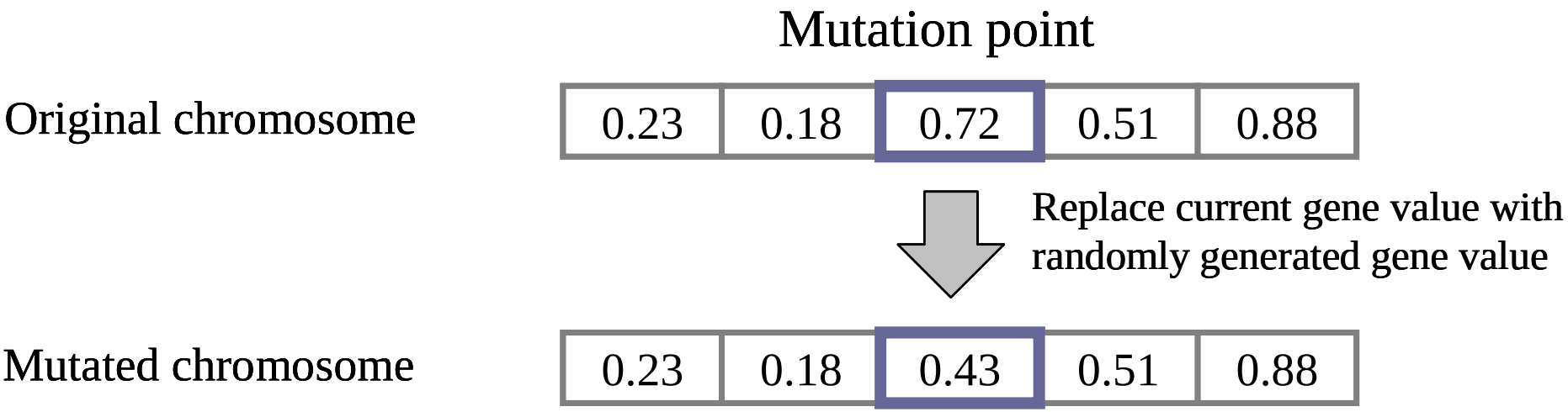
- Mating crossover algorithm changes randomly among the five methods
  - Single point crossover
  - Scalar simple blend crossover
  - Vector simple blend crossover
  - Scalar simulated binary crossover
  - Vector simulated binary crossover
- The interval of changing algorithm is determined by GAP.mc_gac

Mutation

- Applied to the normalized gene values
- Actual gene values are updated based on mutated normalized gene values
- When resulting gene value is illegal, it is adjusted automatically using the ring-mapping
  - Ring-mapping maps a value to the modulus after division by 1
  - Examples: 1.2 $\rightarrow$ 0.2 and -2.1 $\rightarrow$ 0.9
- There are six different mutation algorithms

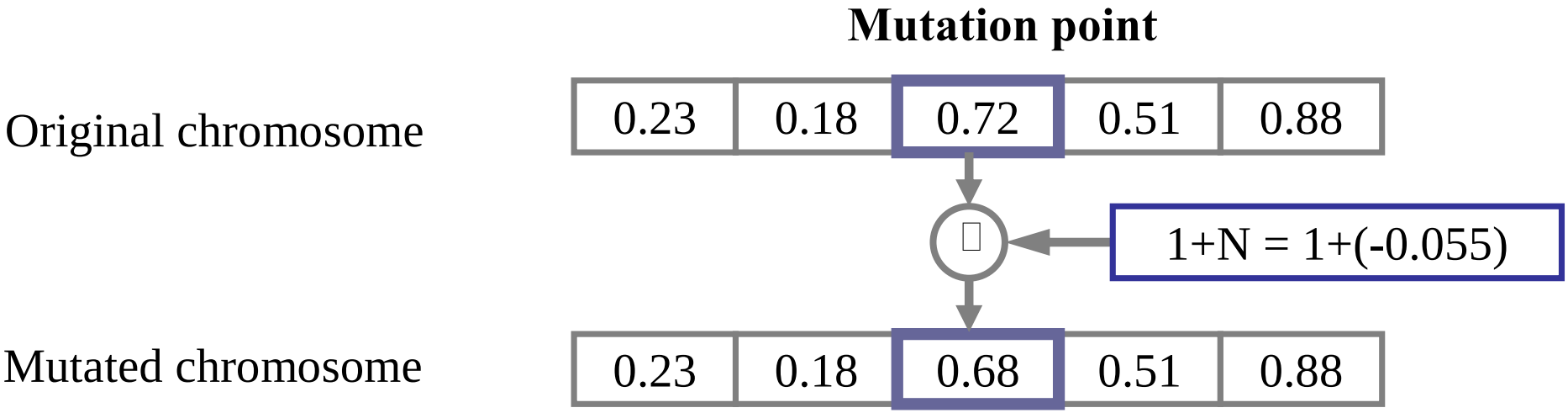
Total Mutation

- Each gene value is replaced by a new randomly generated gene value
- New value has no relationship to old value



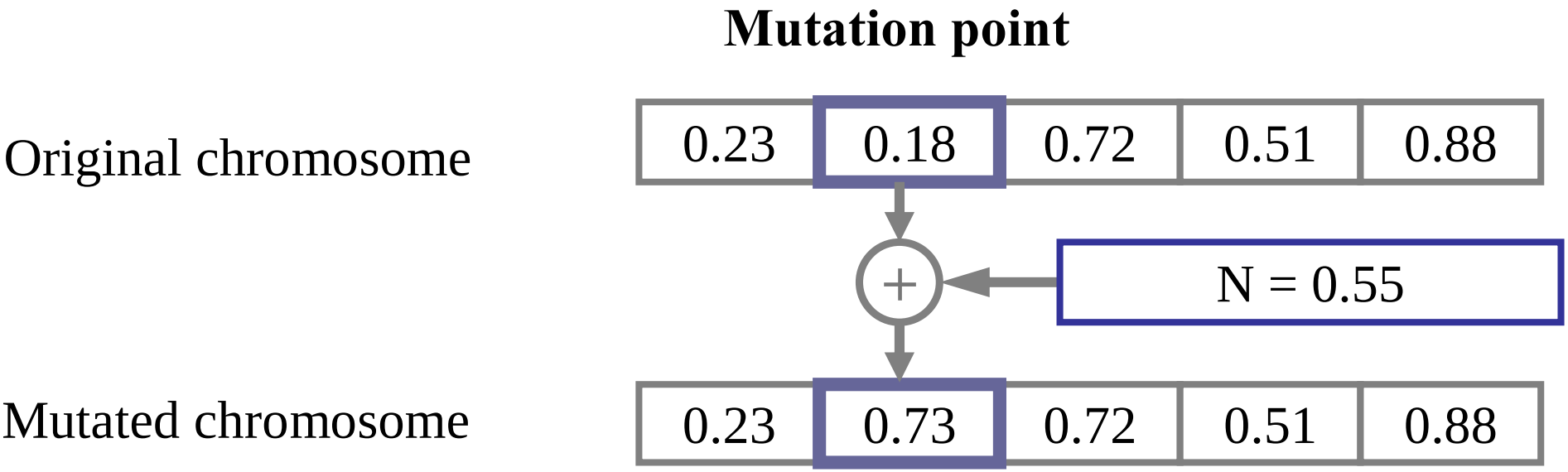
Relative Partial Mutation

- Under mutation is multiplied by (1+N) where N is a Gaussian random variable with the standard deviation of GAP.mt_prgm



Absolute Partial Mutation

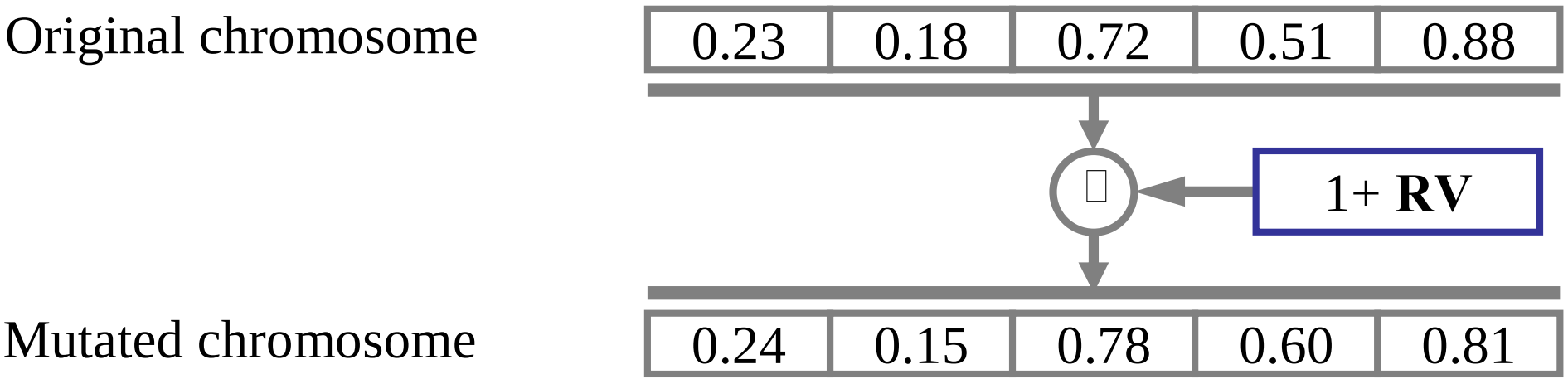
- Gene under mutation is added with a Gaussian random variable N whose standard deviation is `GAP.mt_pagm`



Relative Vector Mutation

- Chromosome multiplied by RV= v_{dir} □ N(0,GAP.mt_srvm) where v_{dir} is a normalized random vector and N is a Gaussian random variable with stand. dev. of `GAP.mt_srvm`

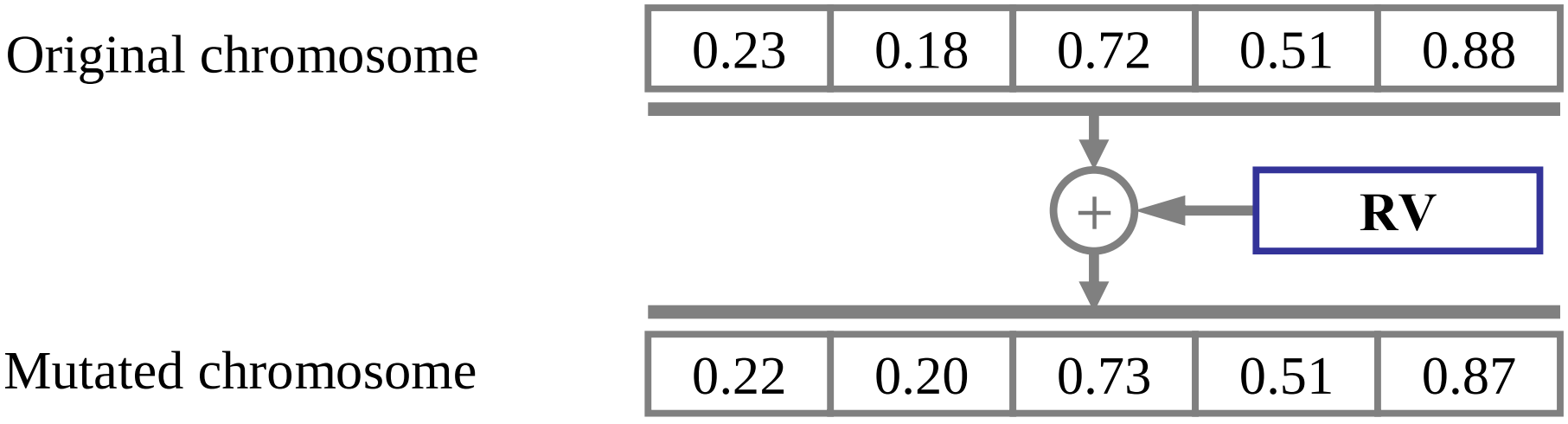
IF RV =[0.03 -0.15 -0.08 0.18 -0.08]



Absolute Vector Mutation

- Chromosome under mutation is added **RV** = v_{dir} N(0,GAP.mt_savm) where v_{dir} is a normalized random vector and N is a Gaussian random variable with the stand. dev. of `GAP.mt_savm`

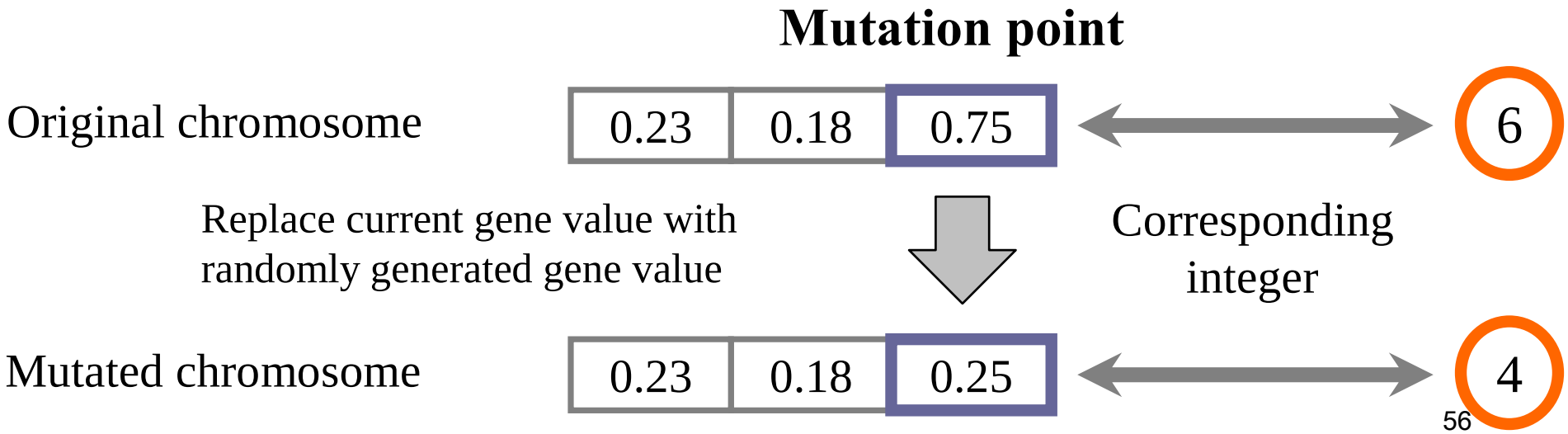
IF RV =[-0.01 0.02 0.01 0 -0.01]



Integer Mutation

- Other mutation operators do not act on integers
- Integer gene value mutated with probability of `GAP.mt_pigm`
- Gene value replaced by a randomly generated value

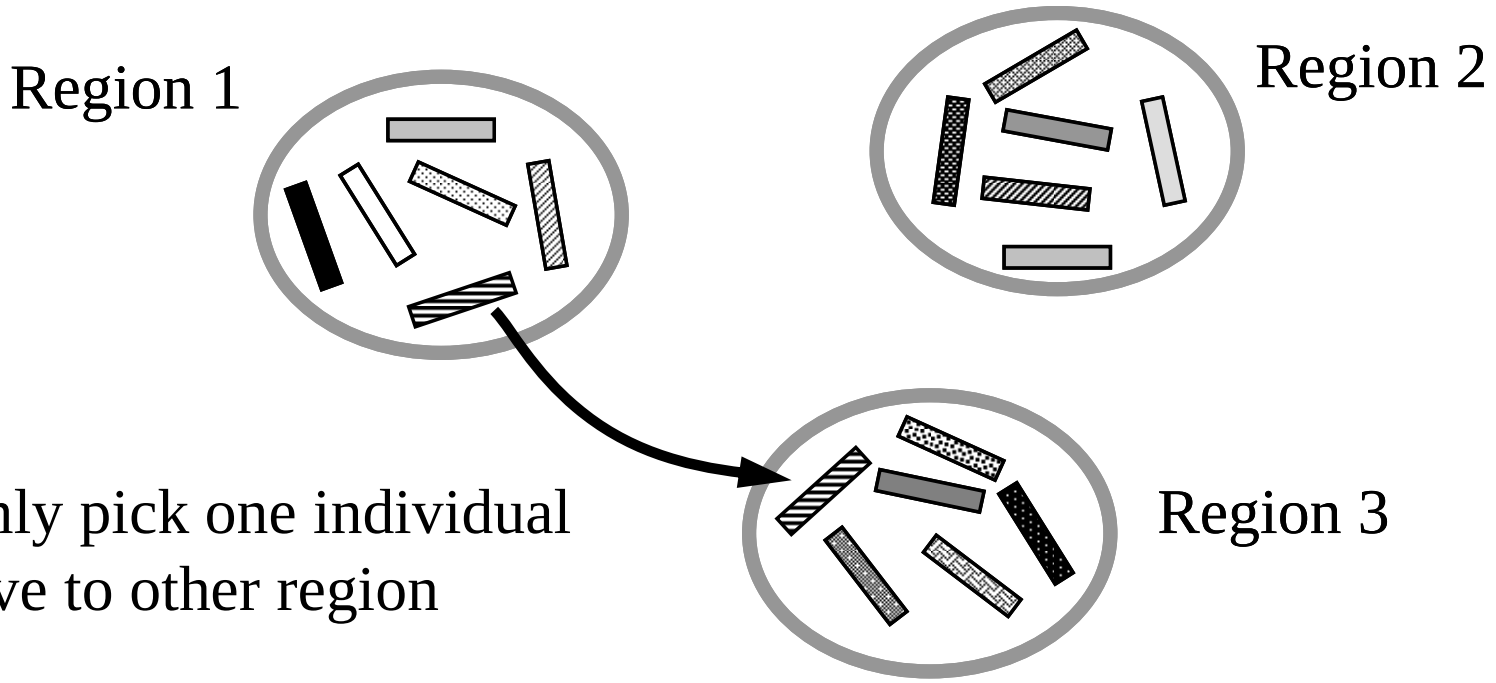
Range for third gene = { 3, 4, 5, 6, 7 }



Migration

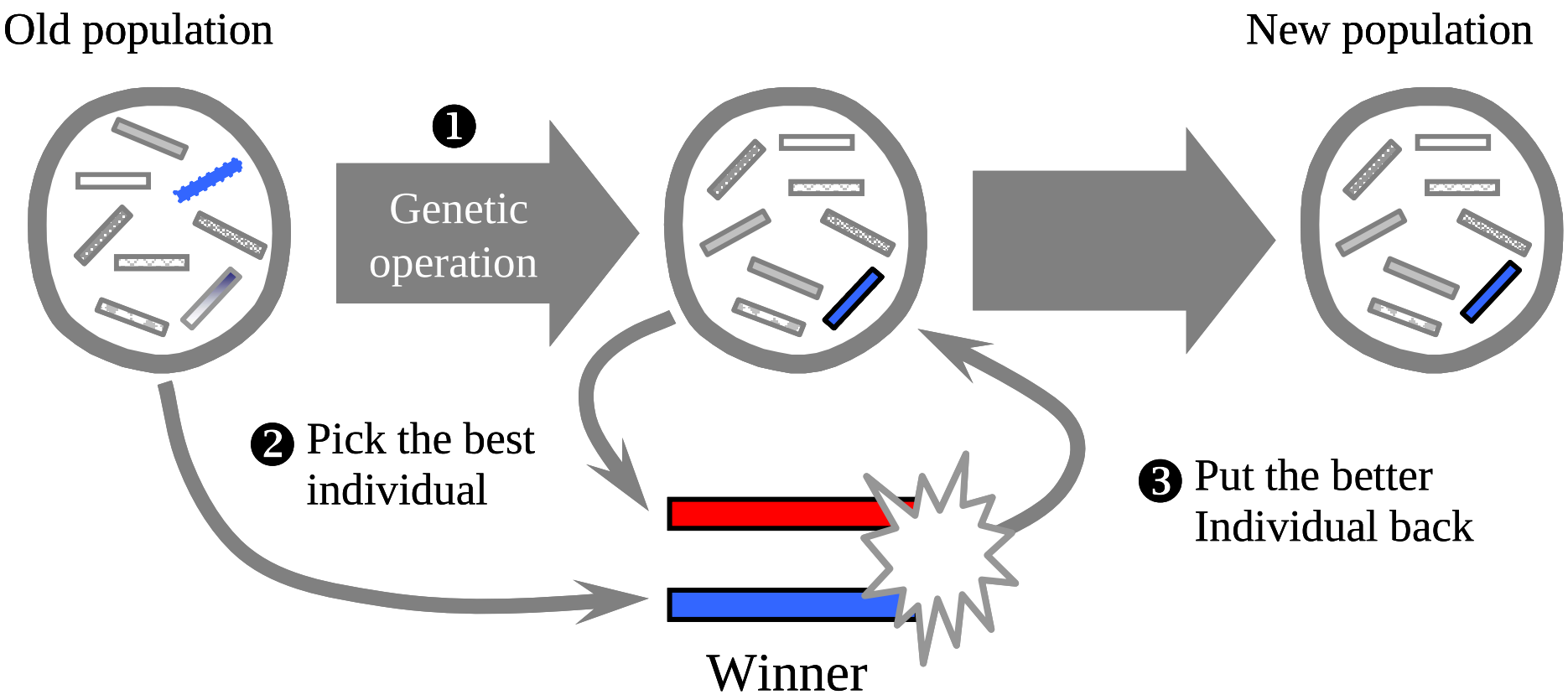
- Multiple regions / migration captures effects of geographical separation in biological systems
- An individual is migrated with the probability of `GAP.mg_pmig`
- Migration occurs every `GAP.mg_tmig` generations

Migration


Region 1
Region 2
Region 3
Randomly pick one individual
and move to other region

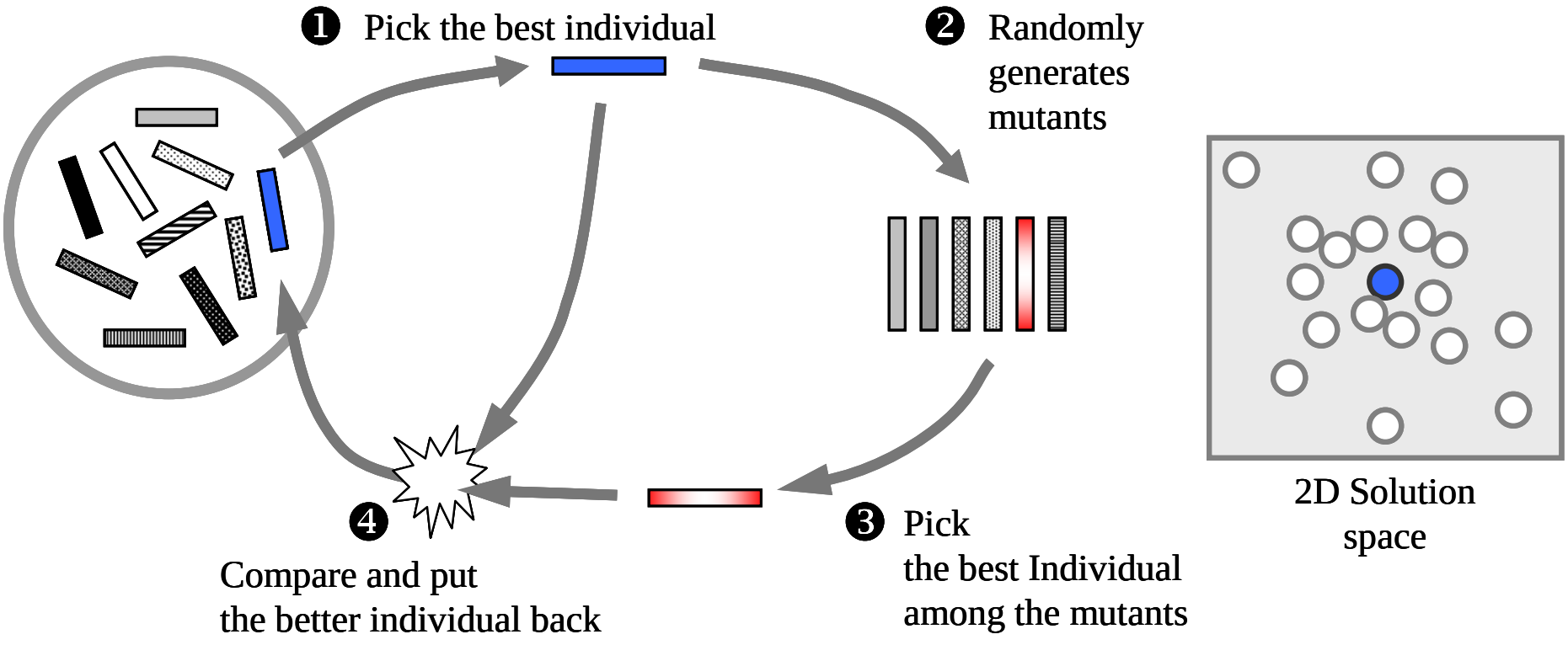
Elitism (Single Objective)

- Purpose: protect the best individual



Random Search

- Purpose: search the vicinity of the best individual for better solution



Random Search

- Relative random search – mutants are generated using relative vector mutation with the standard deviation of `GAP.rs_srp`
- Absolute random search - mutants are generated using absolute vector mutation with the standard deviation of `GAP.rs_sap`
- The relative random search is selected with the probability of GAP.rs_frp and the absolute random search is chosen with the probability of `(1-GAP.rs_frp)`

Random Search

- Random search starts from (`GAP.rs_fgs` × `GAP.fp_ngen`)'th generation
- (`GAP.rs_fps` × `GAP.fp_npop`) mutants are randomly generated