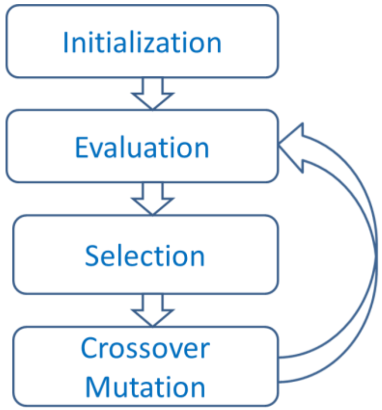
# Genetic Algorithms (2)

Winter 2024

# Review: GA structure




**Program begins**

  $t \leftarrow 0$

  initialize P($t$)

  evaluate P($t$)

  **while** (**not** termination-condition) **do**

    $t \leftarrow t+1$

    select P($t$) from P($t$-1)

    reproduce P($t$) — Crossover Mutation

    evaluate P($t$)

  **end**

**end**

# Sophisticated Control

- Selection
- Coding
- Crossover
- Mutation
- Termination
- Initial population

- Tuning parameters
  - Population size
  - Crossover rate
  - Mutation rate
- Constraint handling
  - Penalty

There are many variations. You are encouraged to check out the related references for general or specific problems

# Selection Method

- Tournament selection:
  - Runs a "tournament" among a few individuals chosen at random from the population and selects the winner, may according to their fitness

# Tournament Selection

- Choose *k* individuals from the population at random
- Pick an individual from tournament
  1. Always pick the best
  2. Give secondary solutions an opportunity to be picked: selection probability may be fixed or based on its fitness (mini Rolette-Wheel)
- Use the picked individual as one of the parents

# Problem Category and Coding

- Binary
- Permutation
- Real-valued (Floating point representation)

# Binary Representation

- You already knew how to code a 0-1binary problem
- Knapsack problem
  - (0 1 0 1 1) $\rightarrow$ pick the $2^{nd}$, $4^{th}$, and $5^{th}$ elements
- Assignment problem
  - $\left[0\,1\,0;1\,0\,0;0\,0\,1\right]$ $\rightarrow$ assign $1^{st}$ project to team 2; $2^{nd}$ project to team 1; $3^{rd}$ project to team 3
  - Note: constraints are required to ensure feasibility

# Binary Problem: Crossover

- One-point
- Two-point
- Multiple point
- Uniform

# Binary Problem: One-point Crossover

Parent 1: [0 0 1 0 1 0 1 1 0 0 0 1 1 0] ~ $2758_{10}$

Parent 2: [0 1 1 1 1 1 0 0 0 0 1 1 0 0] ~ $7948_{10}$

Offspring 1 [0 1 1 1 1 1 1 1 0 0 0 1 1 0] ~ $8134_{10}$

Offspring 2 [0 0 1 0 1 0 0 0 0 0 1 1 0 0] ~ $2572_{10}$

# Binary Problem: Two-point Crossover

Parent 1: [0 0 1 0 | 1 0 1 1 0 0 | 0 1 1 0] ~ $2758_{10}$

Parent 2: [0 1 1 1 | 1 1 0 0 0 0 | 1 1 0 0] ~ $7948_{10}$

Offspring 1 [0 0 1 0 | 1 1 0 0 0 0 | 0 1 1 0] ~ $2822_{10}$

Offspring 2 [0 1 1 1 | 1 0 1 1 0 0 | 1 1 0 0] ~ $7884_{10}$

# Binary Problem: Uniform Crossover

- Generate a random binary mask, with the same length as the parents
  - When the bit in the mask is 0, corresponding bit in *Parent* 1 is passed to *Offspring* 1 and corresponding bit in *Parent* 2 is passed to *Offspring* 2
  - When the bit in the mask is 1, corresponding bit in *Parent* 1 is passed to *Offspring* 2 and corresponding bit in *Parent* 2 is passed to *Offspring* 1

# Uniform Crossover Example

Parent 1: [0 0 1 0 1 0 1 1 0 0 0 1 1 0] ~ $2758_{10}$

Parent 2: [0 1 1 1 1 1 0 0 0 0 1 1 0 0] ~ $7948_{10}$

Mask [0 0 1 1 0 1 1 0 0 0 1 1 1 0]

Offspring 1 [0 0 1 1 1 1 0 1 0 0 1 1 0 0] ~ $3916_{10}$

Offspring 2 [0 1 1 0 1 0 1 0 0 0 0 1 1 0] ~ $6790_{10}$

# Caution for Uniform Crossover

- Uniform crossover has been shown to outperform one or two-point crossover, in terms of genetic variability
- However, uniform crossover may destroy "building blocks" of genes
- Building blocks: short, low-order, high fitness schemata

Parent 1: [1 1 1 0 1 0 1 1 0 0 0 1 1 0]
Parent 2: [1 1 1 1 1 1 0 0 0 0 1 1 0 0]

Objective Maximize f(X)=2X

# Illustration of Crossover

Population: 4 chromosomes; Pc=0.9 (crossover rate=0.9)

#1 [1 0 1 1 0]

#2 [0 1 1 0 0]

#3 [1 1 0 1 1]

#4 [0 1 0 1 1]

1st attempt: pick #1 and #3; generate r=0.23, Do crossover

2nd attempt: pick #1 and #4; generate r=0.96, **Skip crossover**

3rd attempt: pick #2 and #3; generate r=0.62, Do crossover

Stop because the number of offspring reaches 4

Remember, chromosomes may be picked for multiple times

# Binary Problem: Mutation

- Randomly (according to the mutation rate) choose a bit
- Change 0 to 1 or 1 to 0

[0 0 1 **0** 0 0 1 1 0 0 0 1 1 0] ~ $2758_{10}$

[0 0 1 **1** 0 0 1 1 0 0 0 1 1 0] ~ $3270_{10}$

# Mutation (1): bit-wise

Population: 4 chromosomes; Pm=0.2 (mutation rate=0.2)
#1 [1 0 1 1 0]
#2 [0 1 1 0 0]
#3 [1 1 0 1 1]
#4 [0 1 0 1 1]

pick the 1st bit; generate r=0.96, No mutation
pick the 2nd bit; generate r=0.78, No mutation
pick the 3rd bit; generate r=0.29, No mutation
…
pick the 8th bit; generate r=0.05, **Do mutation**
#2 chromosome becomes [0 1 **0** 0 0]
…
for all the bits

# Mutation (2): chromosome-wise

Population: 4 chromosomes; Pm=0.2
#1 [1 0 1 1 0]
#2 [0 1 1 0 0]
#3 [1 1 0 1 1]
#4 [0 1 0 1 1]

for the #1 chromosome; generate r=0.36, No mutation
for the #2 chromosome; generate r=0.18, Do mutation
    randomly pick the $4^{th}$ gene to mutate
    #2 becomes [0 1 1 **1** 0]
for the #3 chromosome; generate r=0.85, No mutation
…
for all the chromosomes

# Permutation Problem

- List of integer values (not repeatable)
  - Typical path representation 5-1-7-8-9-4-6-2-3
  - Ordinal representation ~~1~~ ~~2~~ 3 4 5 6 7 8 9

    1 2 3 4 5 6 7 8 9

    (1 1 2 1 4 1 3 1 1) $\rightarrow$ 1-2-4-3-8-5-9-6-7

    take the 3rd element as an example:

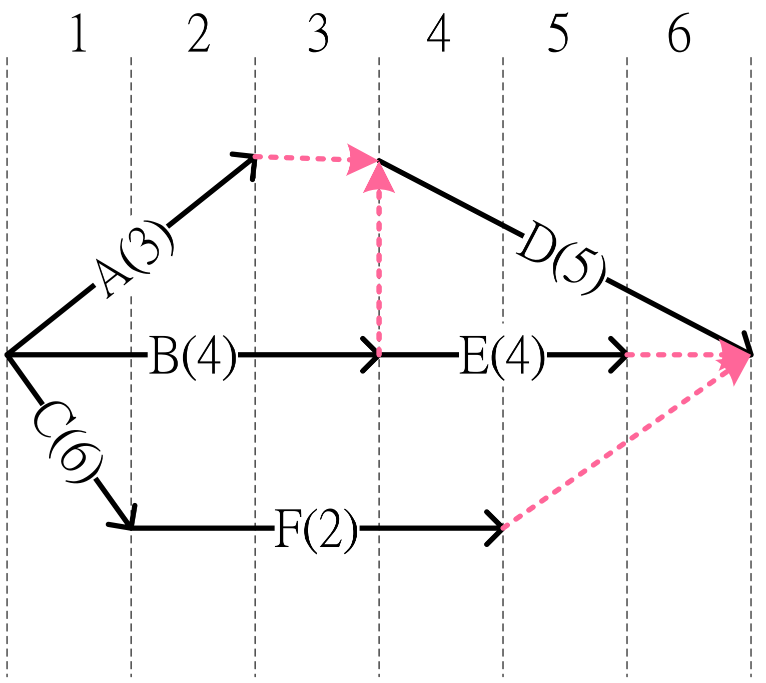
    "2" denotes the second in the current list, city 4

# Permutation Problem: limited resource scheduling

- Priority list:
  - Activities are ordered based on their priority

    (6 2 4 5 3 1)

    → (F B D E C A)
  - When activities A, B, and C compete, B will get resources first, then C, then A
  - How to resolve conflict between D, E, and F?



# Permutation Problem

- Ordinary crossover operations would fail here

Parent 1: [1 2 5 | 4 3]

Parent 2: [4 3 1 | 5 2]

Offspring 1 [1 2 5 5 2]

Offspring 2 [4 3 1 4 3]

?

# Permutation Problem: Crossover

- Partially-mapped (PMX)
- Order (OX)
- Cycle (CX)

# Partially-mapped Crossover

1. Pick 2 crossover points
2. Exchange the values; establish a series of mappings
3. Fill further elements, from the original parent, with no conflict
4. Change those in conflict based on the mappings

KEY: Exploit the mappings of elements

1. P1: [1 2 3 4 5 6 7 8 9]
   P2: [4 5 2 1 8 7 6 9 3]
2. O1:[x x x 1 8 7 6 x x]
   O2:[x x x 4 5 6 7 x x]
3. O1:[x 2 3 1 8 7 6 x 9]
   O2:[x x 2 4 5 6 7 9 3]
4. O1:[4 2 3 1 8 7 6 5 9]
   O2:[1 8 2 4 5 6 7 9 3]

# Order Crossover

1. Offspring maintain the part in-between 2 cross points
2. Choose a subsequence from the 2nd parent, preserving the relative order
3. Remove those already in O1
4. Insert the elements by the sub-sequence
5. Repeat 2~4 for O2

KEY: Maintain the relative order

1. P1: [1 2 3 | 4 5 6 7 | 8 9]
   P2: [4 5 2 | 1 8 7 6 | 9 3]
2. P2: 9-3-4-5-2-1-8-7-6
3. P2: 9-3-~~4~~-~~5~~-2-1-8-~~7~~-~~6~~
4. O1:[2 1 8 | 4 5 6 7 | 9 3]
5. P1: 8-9-1-2-3-4-5-6-7
   P1: ~~8~~-9-~~1~~-2-3-4-5-~~6~~-~~7~~
6. O2:[3 4 5 1 8 7 6 9 2]

# Cycle Crossover

1. Start from the beginning of P1
2. Check the corresponding position in P2; mark it in O1; continue until the element has already been listed
3. Fill in the remaining elements from P2
4. Repeat for P1

KEY: Maintain the position of element

1. P1: [1 2 3 4 5 6 7 8 9]
   P2: [4 1 2 8 7 6 9 3 5]

2. O1: [1 x x x x x x x x]
3. O1: [1 x x 4 x x x x x]
4. O1: [1 x x 4 x x x 8 x]
5. O1: [1 x 3 4 x x x 8 x]
6. O1: [1 2 3 4 x x x 8 x]
7. O1: [1 2 3 4 7 6 9 8 5]

8. O2: [4 1 2 8 x x x 3 x]
9. O2: [4 1 2 8 5 6 7 3 9]

# Permutation Problem: Mutation

- Randomly pick two elements
- Swap the two elements

O1: [1 2 3 4 5 6 7 8 9]

is mutated to

O1: [1 7 3 4 5 6 2 8 9]

# Real-valued Problem

- Save efforts in coding and decoding
- Mapping to a simpler function; treat different domains in one united framework (same length)
  - Linear
    - $23 \le x \le 75 \rightarrow x = x'(75-23)+23 \quad 0 \le x' \le 1$
  - Nonlinear
    - $23 \le x \le 75 \rightarrow x = 23\exp(1.182x') \quad 0 \le x' \le 1$

    Note: $1.182 \approx \ln(75)-\ln(23)$
  - Discrete
    - $x = [12, 36, 99] \rightarrow$

      $x = 12$ if $0.00 < x' < 0.33$

      $x = 36$ if $0.33 \le x' < 0.67$

      $x = 99$ if $0.67 \le x' < 1$

# Real-valued Problem: Crossover

$O_1 = \alpha \times P_1 + (1-\alpha) \times P_2$

$\alpha \sim U[0, 1]$

Parent 1: [3.5, 5.7, 11.8]

Parent 2: [2.6, 2.9, 12.4]

$\alpha$=0.28

Offspring 1: [2.852, 3.684, 12.232]

3.5*0.28+2.6*(1-0.28)

# Real-valued Problem: Mutation

$$Var_i = Var_i + s_i \cdot r_i \cdot a_i$$

$$s_i \in \{-1, +1\}$$

$$r_i = r \cdot range_i \text{ ; } r = \text{specified propotion} \in \boxed{\{10^{-6}, 10^{-5}, \ldots 10^{-1}\}}$$

$$a_i = 2^{-u \cdot k}, u \sim \mathrm{U}[0,1], k = \text{mutation precision} \in \boxed{\{4, 5, \ldots 20\}}$$

$s_i$ and $u$ are determined randomly

$r$ and $k$ are specified by users

$range_i$ depends on the problem

# Real-valued Problem: Mutation Example

$$Var_i = Var_i + s_i \cdot r_i \cdot a_i$$

Offspring: [**3.365**, 5.280, 11.890]

$s_1$=-1; $r_1$=0.1; given range of variable 1: 2.5~4.0;

$u$=-0.236; $k$=4; $a_1 = 2^{-0.236 \cdot 4} = 0.5197$

Variable 1 is mutated to

3.365+(-1)*(0.1*(4.0-2.5))*0.5197

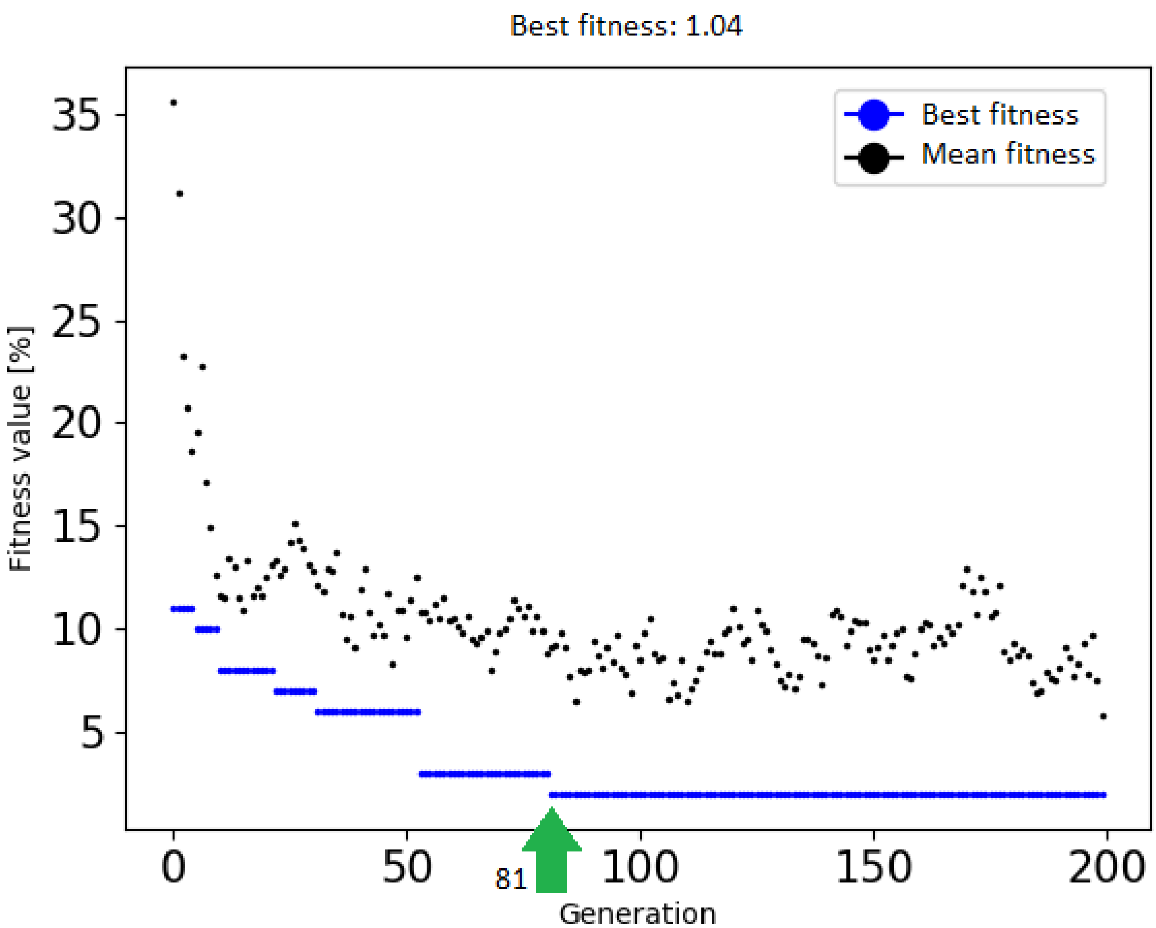
≈ 3.365-0.078

= **3.287**

# Termination

- Fixed number of generations reached (guarantee to stop)
- Allocated computation time reached (guarantee to stop)
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive generations no longer produce better results (convergence)
  - e.g., less than 0.01% improvement in last 10 generations
- Manual inspection
- Combinations of the above

# Convergence



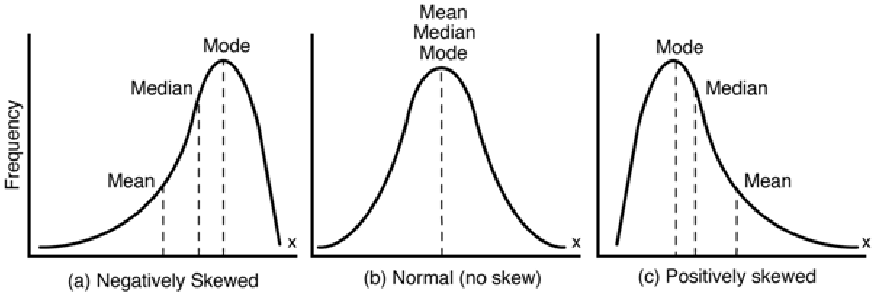

For each generation, a group of solutions would have different fitness values

Best fitness in generations is a **monotonically increasing/decreasing** function

**"Identifying an Emotional State from Body Movements Using Genetic-Based Algorithms"**

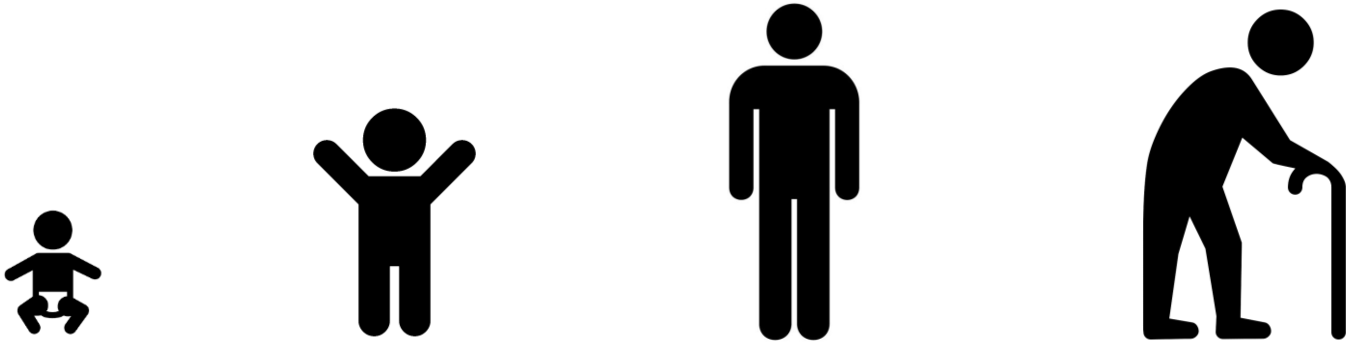
# Initial Population

- Population is initialized randomly
  - e.g., generate a population within [1.2~3.8, 0~10]
  - $[v_1, v_2]$ = **U(0,1)** $\times$ [2.6, 10] + [1.2, 0]
  - How to control the "**density**" of the population?
- Sample and add complement; promote diversity
  - e.g., sample the first half of a 10-bit chromosome as [1 0 0 1 1]; The other half will be [0 1 1 0 0]



# Population Size

- Population size is usually fixed
- Population size may vary from generation to generation
  - Chromosomes age. Their life depends on the fitness
  - More fit chromosomes stay alive longer.

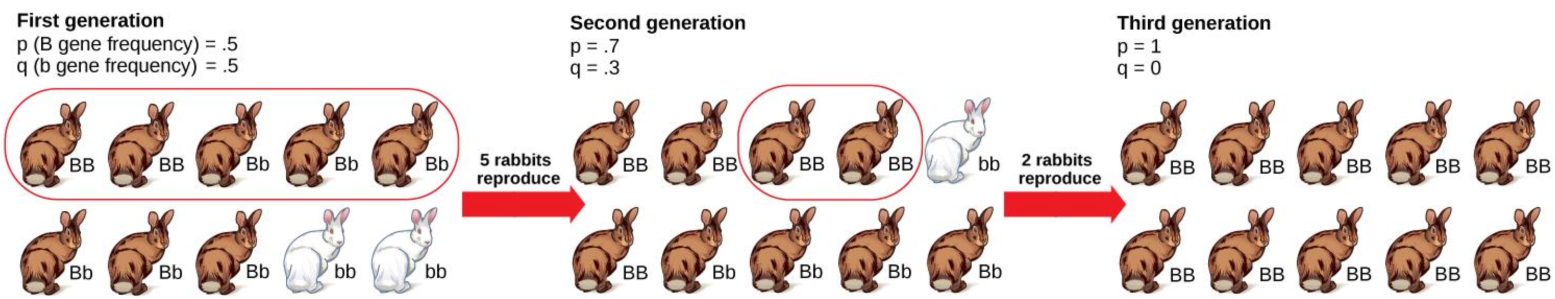


# Tuning Parameters: Crossover

- Crossover rate $p_c$ (usually >0.8):
  - The higher the value of $p_c$, the quicker are the new solutions introduced into the population.
  - If too high, may lead to premature convergence (trapped at local optima)
  - If too low, not much progress is made

# Tuning Parameters: Mutation

- Mutation rate $p_m$ (usually <0.2):
  - High values of $p_m$ transform the GA into a purely random search algorithm causing loss of good solutions.
  - Low values lead to "genetic drift" (few variations are available to respond to sudden environmental change)
  - Suggested: per mutation only one variable per individual is changed/mutated




https://courses.lumenlearning.com/suny-wmopen-biology2

# Tuning Parameters: Mutation

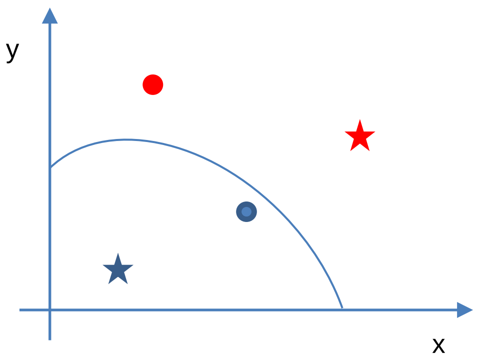
- Mutation rate may be associated with
  - Bit-wise mutation
    - Check bit by bit
  - Chromosome-wise mutation
    - Pick a chromosome and then a bit to mutate
- Suppose we want to mutate 5 chromosomes out of 10; each chromosome has 10 bits
  - Mutation rate for bit-wise mutation is 5/10*1/10 = 0.05
  - Mutation rate for chromosome mutation is 5/10 = 0.5

# Fine-tune GA

- Tuning parameters: crossover rate, mutation rate
- Sensitivity and scenario analysis is in need to choose the best set of tuning parameters
- Conclusion is usually empirical, based on experiments
- How to tune the parameters is of certain interests, and will be covered later

# Constraint Types

- Constraints
  - Hard constraint
    - $g_j(x) \leqq 0$, e.g., $x-y-6 \leqq 0$
    - $h_j(x) = 0$, e.g., $x+y-2=0$
  - Soft constraint
    - e.g., project is expected to complete within 120 days; liquidated damage is $1,000/day
  - Finite set of feasible options
    - e.g., # of trucks: {2, 3, 4}
    - e.g., dimension of H beams, business standard



# Constraint Handling

- Pre-censoring (death penalty)
  - Don't allow any infeasible chromosomes.
  - Easiest but may be a waste of computation time
- Transform the constraint to make the problem unconstrained
  - $0 \leqq x \leqq 10 \leftarrow x=5\sin(y)+5$
- Repair infeasible chromosomes
  - Change the chromosome to ensure feasibility
  - Constraint: $x \leqq 10$. If $x=11$; force it to be 10.
- Add penalty to infeasible chromosomes

# Penalty

- Penalty is computed in relation with violation $\varphi_j(x)$
  - Violation of equality
    - $\varphi_j(x) = |h_j(x)|$ e.g., x+y-2=0; if x+y=0; $\varphi_j(x)=2$
  - Violation of inequality
    - $\varphi_j(x) = \max\{0, g_j(x)\}$ e.g., x-y-6 ≦ 0; if x-y=8; $\varphi_j(x)=2$
- Penalty is positive for a minimization problem; negative for a maximization problem
- Hereafter, we assume positive penalty

# Penalty Types

- Static penalty
  - Penalty is defined based on user-specified levels of violations
- Dynamic penalty
  - Penalty changes over time; increase the penalty as progress
- Adaptive penalty
  - Penalty takes feedback from the search process

# Static Penalty

$$fitness(x) = f(x) + \sum_{j=1}^{m} R_{ij}\varphi_j^2(x)$$

$R_{ij}$ = penalty coefficient for each level of violation and for each constraint

$\varphi_j(x)$ = violation function of constraint $j$

$m$ = number of consrtaints

Example:

Constraint $x+y \le 12 \rightarrow x+y-12 \le 0$

$R=1$ if $12 < x+y \le 20$; $R=4$ if $20 < x+y \le 30$; $R=16$ if $x+y > 30$

Suppose a chromosome leads to $x+y=22$; the fitness value is

$\text{fitness}(x) = f(x) + 4 \times |22-12|^2 = f(x) + 400$

# Dynamic Penalty

$$fitness(x) = f(x) + (C \times t)^{\alpha} \sum_{j=1}^{m} \varphi_j^{\ \beta}(x)$$

$C = \text{constant (e.g., 0.5)}$

$\boxed{\text{t} = \text{generation}}$

$\alpha, \beta = \text{constant(e.g., 2)}$

$\varphi_j(x) = \text{violation function of constraint } j$

$m = \text{number of consrtaints}$

Example:
Constraint x+y≦12

Suppose a chromosome results in x+y=22; the fitness value at generation 10
$= f(x)+(0.5 \times 10)^2|22-12|^2$
$= f(x)+2500$

# Adaptive Penalty (1)

$$fitness(x) = f(x) + \lambda(t)\sum_{j=1}^{m}\varphi_j^{\ 2}(x)$$

$$\lambda(t+1) = \begin{cases} (1/\beta_1)\cdot\lambda(t) & \boxed{\text{case } 1} \\ \beta_2\cdot\lambda(t) & \boxed{\text{case } 2} \\ \lambda(t) & \text{otherwise} \end{cases}$$

case 1 : best individual in the past k generations was $\boxed{\text{feasible}}$

case 2 : best individual in the past *k* generations was $\boxed{\text{infeasible}}$

$\beta_1, \beta_2 = \text{constant } (\beta_1, \beta_2 > 1; \beta_1 \neq \beta_2)$

# Adaptive Penalty (2)

$$fitness(x) = f(x) + (B_{feasible} - B_{all})\sum_{j=1}^{m}[\frac{\varphi_j(x)}{NFT(t)}]^k$$

$B_{feasible}$ = best (only feasible solutions) objective value at generation $t$

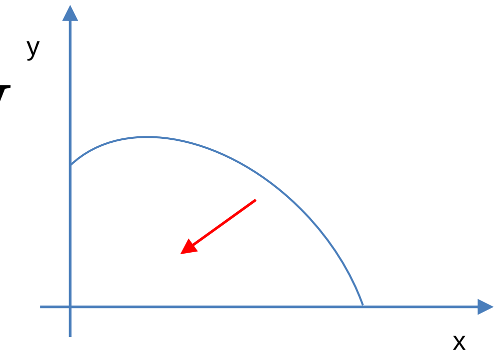
$B_{all}$ = best (including infeasible ones) objective value at generation $t$

$NFT(t)$ = User-specified threshold distance

$NFT$ : how close the infeasible solution to the feasible region is considered reasonable?

$k$ is a constant (e.g., $k = 2$)

# Choice of Penalty




- If the penalty is too high
  - The solutions will be pushed inside the feasible region very quickly and hardly reach the boundary
  - Dangerous when the optimum lies at the boundary of the feasible region
- If the penalty is too low
  - Search time will be mostly spent exploring the infeasible region because the penalty will be negligible with respect to the objective function.
- Again, empirical trials are necessary

# References


- Michalewicz, Z. (1998) *Genetic Algorithms + Data Structures = Evolution Programs*, Springer.
- Haupt, R.L. and Haupt, S.E. (2004) *Practical Genetic Algorithms*. John Wiley & Sons.
- Journals
  - Evolutionary Computation Journal
  - Genetic Programming and Evolvable Machine Journal
  - IEEE Transaction on Evolutionary Computation
  - Other cross-disciplinary journals