

# Optimum Design

Fall 2016

**Myung-II Roh**

Department of Naval Architecture and Ocean Engineering  
Seoul National University

## Contents

---

- ☑ Ch. 1 Introduction to Optimum Design
- ☑ Ch. 2 Unconstrained Optimization Method: Gradient Method
- ☑ Ch. 3 Unconstrained Optimization Method: Enumerative Method
- ☑ Ch. 4 Constrained Optimization Method: Penalty Function Method
- ☑ Ch. 5 Constrained Optimization Method: LP (Linear Programming)
- ☑ Ch. 6 Constrained Optimization Method: SQP (Sequential Quadratic Programming)
- ☑ **Ch. 7 Metaheuristic Optimization Method: Genetic Algorithms**
- ☑ Ch. 8 Case Study of Optimal Dimension Design
- ☑ Ch. 9 Case Study of Optimal Route Design
- ☑ Ch. 10 Case Study of Optimal Layout Design

## Ch. 7 Metaheuristic Optimization Method: Genetic Algorithms

- 7.1 Overview
- 7.2 Generals
- 7.3 Cycles of Genetic Algorithms
- 7.4 Genetic Operators
- 7.5 Example of Traveling Salesman Problem
- 7.6 Multi-Objective Genetic Algorithms

## 7.1 Overview

## Genetic Algorithms (GA) (1/3)

- ☑ Adaptive metaheuristic search algorithm based on the evolutionary ideas of natural selection and genetics
- ☑ One of guided random search algorithms based on the **mechanics of biological evolution**
- ☑ Part of evolutionary computing, a rapidly growing area of artificial intelligence
- ☑ Inspired by Darwin's theory about evolution. That is, In nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones. ("**Survival of the fittest**")
- ☑ Based on an intelligent exploitation of a random search used to solve optimization problems.

\* W. Williams, Genetic Algorithms: A Tutorial, 1995  
Topics in Ship Design Automation, Fall 2016, Myung-Il Roh

**sydlab**  
SEOUL NATION UNIVERSITY 5

## Genetic Algorithms (GA) (2/3)

- ☑ Although randomized, it exploits historical information to direct the search into the region of better performance within search space
- ☑ Find an optimum by **repeating a series of genetic operators**; selection, crossover, mutation, etc.
- ☑ Yield a global optimum for complex optimization problems having a number of local optima ➔ **Global Optimization Method**

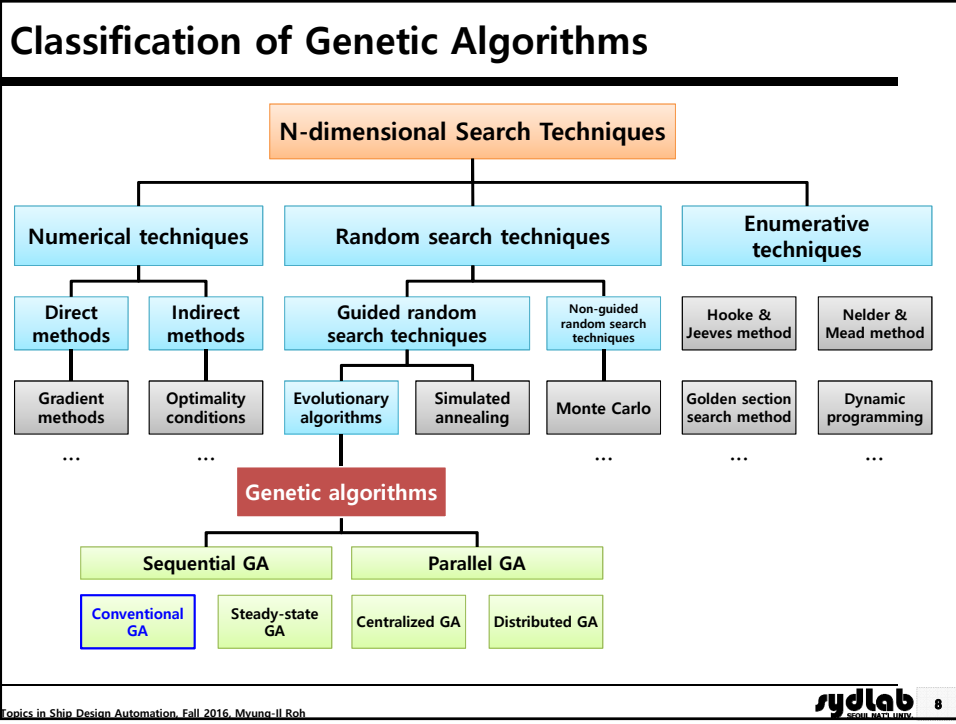
Topics in Ship Design Automation, Fall 2016, Myung-Il Roh

**sydlab**  
SEOUL NATION UNIVERSITY 6

## Genetic Algorithms (GA) (3/3)

- ☑ Suitable for solving complex optimization problems and for applications that require adaptive problem solving strategies
- ☑ Provide efficient, effective techniques for optimization and machine learning applications
- ☑ Widely used today in engineering, business, and scientific fields
- ☑ Developed by John Holland at University of Michigan in 1975
  - To understand the adaptive processes of natural systems
  - To design artificial systems software that retains the robustness of natural systems

Topics in Ship Design Automation, Fall 2016, Myung-Il Roh
**sydlab** 7



## Components of Genetic Algorithms

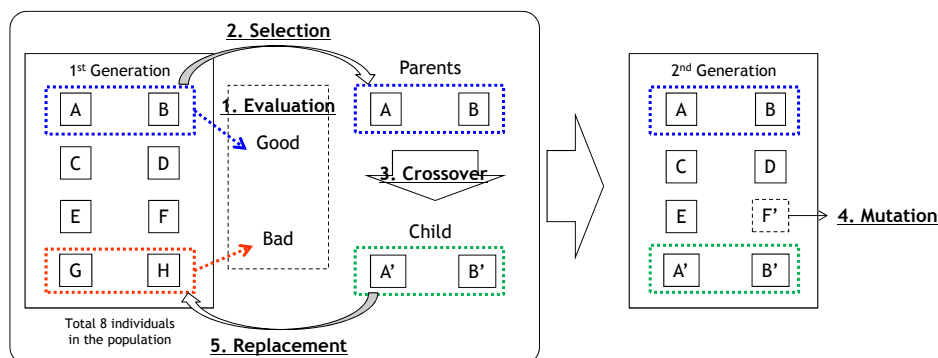
- ☑ To solve a problem with GA, the followings are needed.
- ☑ **Encoding technique:** Gene, chromosome
- ☑ **Initialization procedure:** Initialization (or creation)
- ☑ **Evaluation function:** Fitness calculation
- ☑ **Selection:** Selection of parents
- ☑ **Modification:** Crossover and mutation for the parents
- ☑ **Deletion:** Update of the population
- ☑ **Parameter setting:** Practice and experience

\* W. Williams, Genetic Algorithms: A Tutorial, 1995  
Topics in Ship Design Automation, Fall 2016, Myung-Il Roh

sydlab  
SEOUL NATION UNIVERSITY

9

## Procedure of Finding an Optimum by Using GA



1. **Evaluation:** Evaluate the quality (fitness) of each individual in the population.
2. **Selection:** Select two individuals of good quality as parents.
3. **Crossover:** Generate two child from the parents.
4. **Mutation:** Generate a new child by introducing the property which the parents don't have. (Optional)
5. **Replacement:** Replace individuals of bad quality with the child.

**The optimum is found by repeating the evaluation, selection, crossover, mutation, and replacement.**

Topics in Ship Design Automation, Fall 2016, Myung-Il Roh

sydlab  
SEOUL NATION UNIVERSITY

10

## Example of Finding an Optimum by Using GA (1/5)

**Minimize**  $f(\mathbf{x}) = x_1^2 + x_2^2 - 8x_1 - 8x_2$

1<sup>st</sup> Generation Evaluation

A	( 1 , 1 )	-14
B	( 2 , 5 )	-27
C	( 3 , 4 )	-31
D	( 4 , 2 )	-28
E	( 6 , 2 )	-24
F	( 6 , 7 )	-19

Parents  
( 3 , 4 )  
( 4 , 2 )

**1. Evaluation:** Evaluate the quality (fitness) of each individual in the population

**2. Selection:** Select two individuals of good quality as parents.

- In this example, two individuals having higher fitness were selected as parents.
- **Selection by probability.** The better fitness, the higher selection probability.

**Fitness proportionate selection (Roulette wheel selection)**

- Genetic operator used in genetic algorithms for selecting potentially useful solutions for crossover
- The analogy to a roulette wheel can be envisaged by imagining a roulette wheel in which **each candidate solution represents a pocket on the wheel**; the size of the pockets are proportionate to the probability of selection of the solution.

$$P_{selection}(i) = \frac{Ft(i)_i}{\sum_{j=1}^N Ft(i)_i}$$

$Ft(i)$ : Fitness of individual  $i$  in the population  
 $N$ : Number of individuals in the population  
 $P_{selected}(i)$ : Probability of being selected of individual  $i$

Topics in Ship Design Automation, Fall 2016, Myung-Il Roh

11

## Example of Finding an Optimum by Using GA (2/5)

**Minimize**  $f(\mathbf{x}) = x_1^2 + x_2^2 - 8x_1 - 8x_2$

1<sup>st</sup> Generation Evaluation

A	( 1 , 1 )	-14
B	( 2 , 5 )	-27
C	( 3 , 4 )	-31
D	( 4 , 2 )	-28
E	( 6 , 2 )	-24
F	( 6 , 7 )	-19

Parents  
( 3 , 4 )  
( 4 , 2 )

Child  
C' ( 3 , 2 )  
D' ( 4 , 4 )

Crossover

Replacement

**3. Crossover:** Generate two child from the parents.

- In this example, a method of switching  $x_2$  was used as crossover.

**4. Replacement:** Replace individuals of bad quality with the child.

- In this example, two individuals having bad quality was replaced with the child.

**Replacement methods of individuals**

- + Ex 1) Evaluate the quality of the child. If its quality is better than that of parent, replace it with the parent, **or do not replace (it is dropped).**
- + Ex 2) Evaluate the quality of the child. If its quality is better than that of parent, replace it with the parent, **or replace it with one of the parent having worse quality.**

Topics in Ship Design Automation, Fall 2016, Myung-Il Roh

12

### Example of Finding an Optimum by Using GA (3/5)

*Minimize*  $f(x) = x_1^2 + x_2^2 - 8x_1 - 8x_2$

2<sup>nd</sup> Generation Evaluation

A	( 3 , 2 )	-27
B	( 2 , 5 )	-27
C	( 3 , 4 )	-31
D	( 4 , 2 )	-28
E	( 6 , 2 )	-24
F	( 4 , 4 )	-32

Parents: (3, 4) and (4, 4)

Child: C' (3, 6) and F' (4, 4)

3<sup>rd</sup> Generation Evaluation

A	( 3 , 6 )	
B	( 2 , 5 )	
C	( 3 , 4 )	
D	( 4 , 2 )	
E	( 4 , 4 )	
F	( 4 , 4 )	

By repeating evaluation, selection, crossover, and replacement, the optimum can be found.

**Mutation:** Generate a new child by introducing the property which the parents don't have.

**Stopping Criteria:**

- Stop optimization after repeating the generation by a certain number.
- + Assumption that the solution will converge if the generation is repeated by a certain number.
- Stop optimization by check if most of individuals in the population (e.g., 70%) are almost same or not.

Topics in Ship Design Automation, Fall 2016, Myung-Il Roh **sydlab** 13

### Example of Finding an Optimum by Using GA (4/5)

*Minimize*  $f(x) = x_1^2 + x_2^2 - 8x_1 - 8x_2$

3<sup>rd</sup> Generation Evaluation

A	( 3 , 6 )	-27
B	( 2 , 5 )	-27
C	( 3 , 4 )	-31
D	( 4 , 2 )	-28
E	( 4 , 4 )	-32
F	( 4 , 4 )	-32

Parents: (4, 4) and (4, 4)

Child: E' (4, 4) and F' (4, 4)

4<sup>th</sup> Generation Evaluation

A	( 4 , 4 )	
B	( 4 , 4 )	
C	( 3 , 4 )	
D	( 4 , 2 )	
E	( 4 , 4 )	
F	( 4 , 4 )	

By repeating evaluation, selection, crossover, and replacement, the optimum can be found.

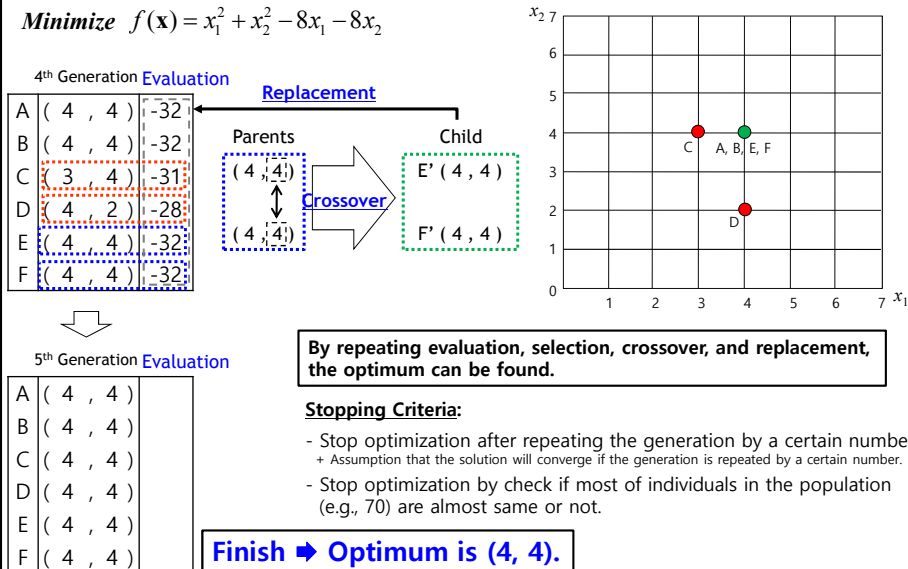
**Stopping Criteria:**

- Stop optimization after repeating the generation by a certain number.
- + Assumption that the solution will converge if the generation is repeated by a certain number.
- Stop optimization by check if most of individuals in the population (e.g., 70) are almost same or not.

Topics in Ship Design Automation, Fall 2016, Myung-Il Roh **sydlab** 14

## Example of Finding an Optimum by Using GA (5/5)

$$\text{Minimize } f(\mathbf{x}) = x_1^2 + x_2^2 - 8x_1 - 8x_2$$



Topics in Ship Design Automation, Fall 2016, Myung-Il Roh

sydlab 15

## Differences from Other Search Techniques

- ☑ Search from a population of points (“individuals”), not a single point.
- ☑ Use probabilistic (and not deterministic) transition rules and thus can be considered to be randomized algorithms.
  - Selection probability, Crossover probability, Mutation probability, etc.
- ☑ Use payoff information (fitness value) from an objective function, not intermediate information (such as derivatives or domain knowledge).
- ☑ Work with a coding of the parameter set (variables), not the parameters themselves.
  - Chromosome

Topics in Ship Design Automation, Fall 2016, Myung-Il Roh

sydlab 16



## 7.2 Generals

### Benefits of Genetic Algorithms

---

- ☑ Concept is easy to understand.
- ☑ Modular, separate from application
- ☑ **Support multi-objective optimization**
- ☑ Good for noisy environments (hard problem to be solved)
- ☑ Always an answer; answer gets better with time
- ☑ Inherently parallel; easily distributed
- ☑ Many ways to speed up and improve a GA-based application as knowledge about problem domain is gained
- ☑ Easy to exploit previous or alternate solutions
- ☑ Flexible building blocks for hybrid applications
- ☑ Substantial history and range of use
- ☑ **No gradient information needed** for objective function and constraint

## When to Use Genetic Algorithms

- ☑ Alternate solutions are too slow or overly complicated.
- ☑ Need an exploratory tool to examine new approaches.
- ☑ A problem is similar to one that has already been successfully solved by using a GA.
- ☑ Want to hybridize with an existing solution.
- ☑ Benefits of the GA technology meet key problem requirements.

## Applications Genetic Algorithms in Various Fields

- ☑ Combinatorial optimization: Set covering, travelling salesman, **routing (e.g., piping)**, bin packing, graph coloring and partitioning
- ☑ Design: **Layout design (e.g., compartment layout, topside layout)**, aircraft design, keyboard configuration, communication networks
- ☑ Control: Gas pipeline, pole balancing, missile evasion, pursuit
- ☑ Scheduling: Manufacturing, **facility scheduling (e.g., gantry crane, transporter)**, resource allocation
- ☑ Machine learning: Designing neural networks, improving classification algorithms, classifier systems
- ☑ Robotics: Trajectory planning
- ☑ Signal processing: Filter design
- ☑ Game playing: Poker, checkers, prisoner's dilemma

## Some Issues for Genetic Algorithms (1/2)

- Selection of Chromosome Representation**
  - Binary-string coding vs. decimal coding
- Selection of Suitable Genetic Operators**
  - Selection: fitness proportionate Selection (roulette wheel selection) vs. tournament selection
  - Crossover, mutation
  - Application of elitism: Conserving superior individuals in the population without the replacement of them
- Selection of Suitable Optimization Parameters**
  - Population no: Number of individuals in the population
  - Maximum generation or iteration no: How many iterations should be made for finding optimum
  - Crossover probability
  - Mutation probability

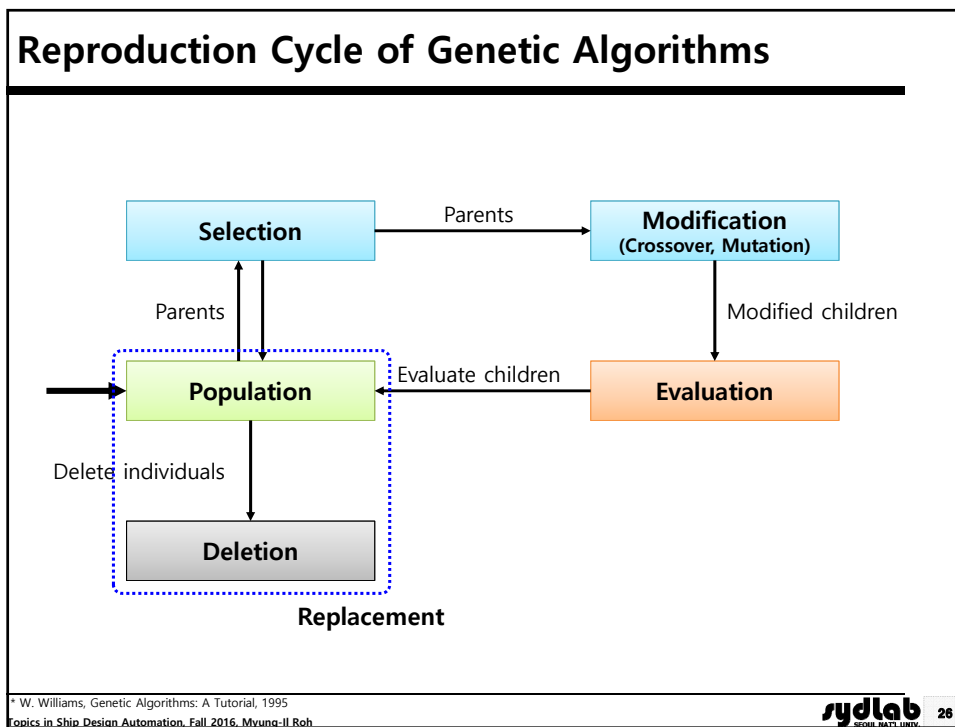
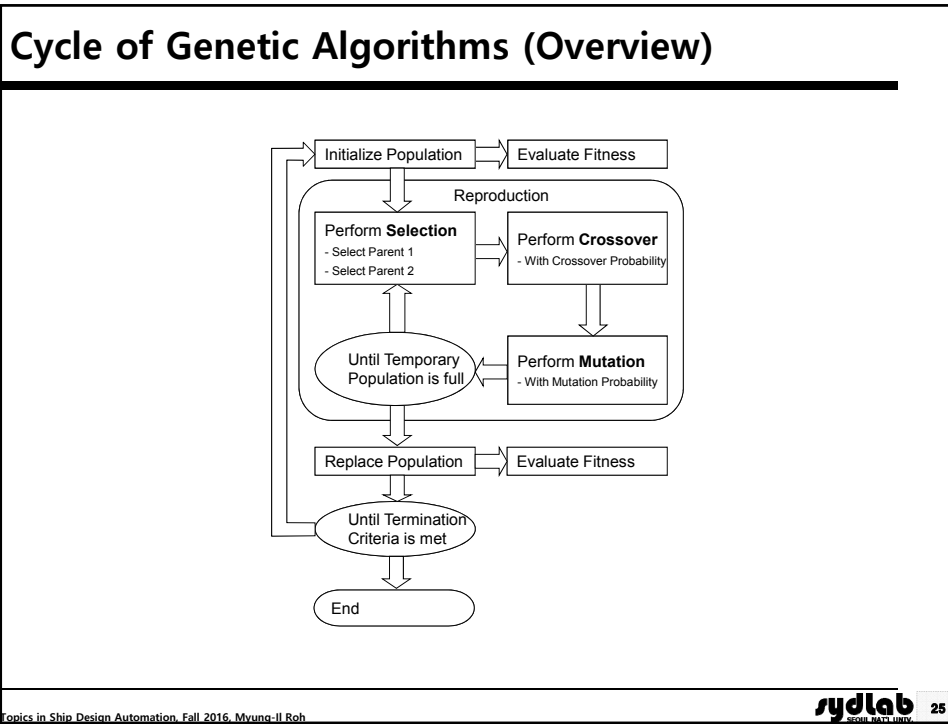
## Some Issues for Genetic Algorithms (2/2)

- Termination Criteria**
  - Number of generations
  - Convergence ratio
- Performance and Scalability**
  - Objective function and Fitness value
- It needs many trials and errors** to select suitable genetic operations and optimization parameters for the corresponding problem.
- Optimum is only as good as the evaluation function** (often hardest part). That is, the better evaluation function, the better optimum.

## 7.3 Cycles of Genetic Algorithms

### Pseudo Code for General Genetic Algorithms

```
Initialize population;  
Evaluate population;  
while StoppingCriteriaNotSatisfied  
{  
    Select parents for reproduction;  
    Perform Crossover; (with small probability)  
    Perform Mutation; (with small probability)  
    Evaluate population;  
    Replace population;  
}
```



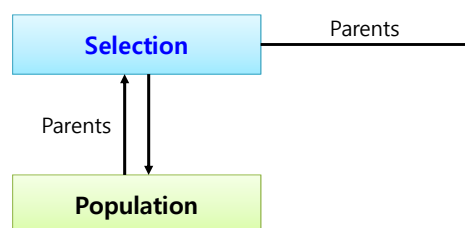
## Population

- ☑ Representation of chromosome
  - Bit strings (Binary-string coding): e.g., 0101 ... 1100
  - Real numbers (Decimal coding): e.g., 37.1 0.0 ... -19.4
  - Permutations of element: e.g., S12 S7 ... S9)
  - Lists of rules: e.g., R1 R2 R3 ... R22 R23
  - Program elements: Tree structure (genetic programming)
  - Any data structure...



## Selection

- ☑ Parents are selected at random with selection chances biased in relation to chromosome evaluations.
- ☑ The selected parents are subjected to be used for generating new child by modification operators such as crossover and mutation.

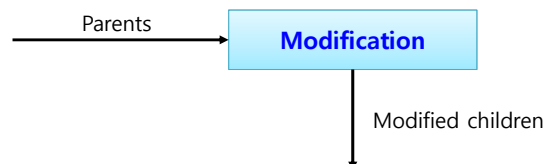


## Selection Operator

- ☑ Genetic operator used in genetic algorithms for **selecting potentially useful solutions for crossover**
  
- ☑ **Some Selection Operators**
  - Fitness proportionate selection (roulette wheel selection)
  - Tournament selection
  - Stochastic universal sampling
  - Reward-based selection

## Modification

- ☑ After parents are selected, modifications for the parents are stochastically triggered.
  
- ☑ **Operators for Modification**
  - Crossover
  - Mutation



## Crossover Operator

- ☑ Genetic operator which is a critical feature of genetic algorithms
- ☑ Greatly **accelerate search early in evolution of a population.**
- ☑ Lead to effective combination of schemata (subsolutions on different chromosomes)

Parents	<b>Before</b>	➔	<b>After</b>	Children
$P_1$	(0 1 <span style="border: 1px dashed red; padding: 2px;">1</span> 0 1 0 0 0)		(0 1 <span style="border: 1px dashed red; padding: 2px;">0</span> 0 1 0 0 0)	$C_1$
$P_2$	(1 1 <span style="border: 1px dashed red; padding: 2px;">0</span> 1 1 0 1 0)		(1 1 <span style="border: 1px dashed red; padding: 2px;">1</span> 1 1 0 1 0)	$C_2$

Example of binary-string coding

## Mutation Operator

- ☑ Cause **local or global movement in the search space.**
- ☑ Restore lost information to the population.

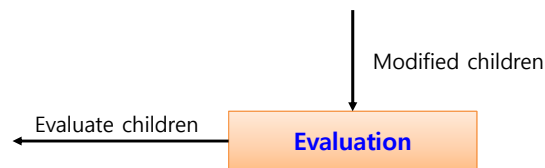
	<b>Before</b>	➔	<b>After</b>	
$C_1$	(0 1 0 <span style="border: 1px dashed red; padding: 2px;">0</span> 1 0 0 0)		(0 1 0 <span style="border: 1px dashed red; padding: 2px;">1</span> 1 0 0 0)	$C_1'$

Example of binary-string coding



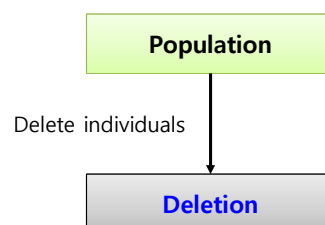
## Evaluation

- ☑ The evaluator decodes a chromosome and assigns it a fitness measure.
- ☑ The evaluator is the only link between genetic algorithms and the problem to be solved.



## Deletion

- ☑ Genetic operator for updating the current population
- ☑ In conventional (generational) GA, entire populations replaced with each iteration.
- ☑ In steady-state GA, a few members (e.g., two individuals) replaced each generation.

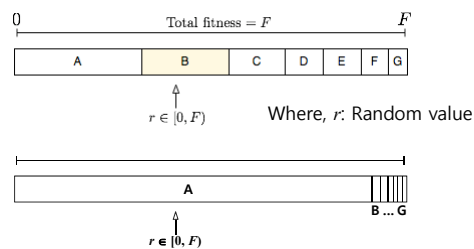


## 7.4 Genetic Operators

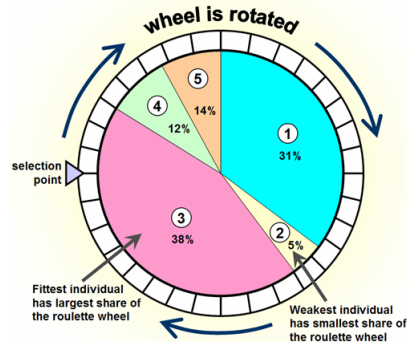
### Selection Operator

#### - Fitness Proportionate Selection (Roulette Wheel Selection) (1/2)

- ☑ The individuals are mapped to contiguous segments of a line, such that **each individual's segment is equal in size to its fitness**.
  - **The better fitness, the longer size.**
- ☑ A random number is generated and the individual whose segment includes the random number is selected.
- ☑ The process is repeated until the desired number of individuals is obtained (called mating population).
- ☑ This technique is similar to a roulette wheel with each slice proportional in size to the fitness.



## Selection Operator - Fitness Proportionate Selection (Roulette Wheel Selection) (2/2)



$$p_{selection}(i) = \frac{Ft(i)_i}{\sum_{j=1}^N Ft(i)_i} \Rightarrow p_{selection}(i) \propto Ft(i)$$

where,

$Ft(i)$ : Fitness of individual  $i$  in the population

$N$ : Number of individuals in the population

$P_{selection}(i)$ : Probability of being selected of individual  $i$

## Fitness Function

- ☑ A function to measure the quality of each individual
- ☑ It can be made from an objective function of the problem to be solved.
- ☑ In the case of a constrained optimization problem, the penalty function method can be used.

$$Ft(i) = M - F(i) \quad \text{or} \quad Ft(i) = \frac{1}{F(i)}, \quad \text{if } F(i) > 0$$

$$F(i) = f(i) + \sum_{j=1}^q R_j \cdot \max\{g_j(i), 0\} \quad \rightarrow \text{Penalty Function Method}$$

where,

$M$ : Large number

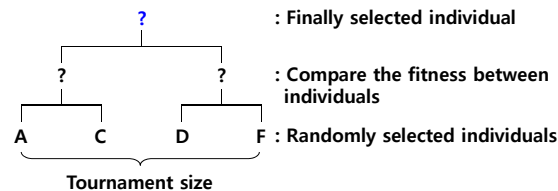
$f(i)$ : Objective function value of individual  $i$  in the population

$g_j(i)$ :  $j^{\text{th}}$  constraint value of individual  $i$  in the population

$R_j$ : Penalty factor or penalty parameter

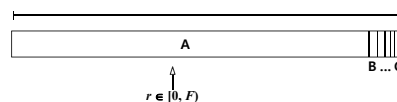
## Selection Operator - Tournament Selection

- ☑ Tournament selection involves **running several tournaments among a few individuals** chosen at random from the population.
- ☑ The winner of each tournament (the one with the best fitness) is selected for crossover.
- ☑ Selection pressure is easily adjusted by changing the tournament size.
- ☑ If the tournament size is larger, weak individuals have a smaller chance to be selected.
- ☑ It is efficient to code, works on parallel architectures and allows the selection pressure to be easily adjusted.



## Selection Operator - Stochastic Universal Sampling (SUS) (1/2)

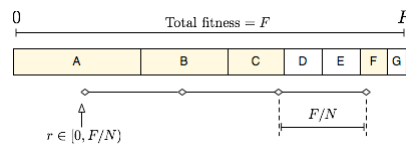
- ☑ SUS is an improvement of fitness proportionate selection (FPS). SUS exhibits no bias and minimal spread as compared with FPS.
- ☑ Where the fitness proportionate selection chooses several individuals from the population by repeated random sampling, SUS uses a **single random value to sample several individuals by choosing them at evenly spaced intervals**.
- ☑ This gives weaker members of the population (according to their fitness) a chance to be chosen and thus reduces the unfair nature of fitness-proportional selection methods.
- ☑ **Other methods like roulette wheel can have bad performance when an individual of the population has a really large fitness in comparison with other individuals.**



## Selection Operator

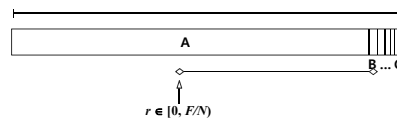
### - Stochastic Universal Sampling (SUS) (2/2)

- ☑ Using a comb-like ruler, SUS starts from a small random number, and chooses the next candidates from the rest of population remaining, not allowing the fittest members to saturate the candidate space.



Where,  
 $r$ : Random value  
 $N$ : Number of individuals to select. In this figure,  $N = 4$ .

Example for  $N = 2$



## Selection Operator

### - Reward-based Selection

- ☑ The probability of being selected for an individual is proportional to the cumulative reward from the fitness function, obtained by the individual.
- ☑ The cumulative reward can be computed as a **sum of the individual reward and the reward, inherited from parents.**

$$p_{selection}(i) = \frac{Ft(i)_i}{\sum_{j=1}^N Ft(i)_j} \Rightarrow p_{selection}(i) \propto Ft(i)$$

$$Ft(i) = Ft(i) + \sum Ft(parent_i)$$

where,  
 $Ft(i)$ : Fitness of individual  $i$  in the population  
 $Ft(parent_i)$ : Fitness of individual  $i$  in the population, inherited from parents  
 $N$ : Number of individuals in the population  
 $p_{selection}(i)$ : Probability of being selected of individual  $i$

## Crossover Operator

### - Order 1 Crossover (1/2)

- ☑ Order 1 crossover is a fairly simple permutation crossover.
- ☑ Basically, a swath of consecutive genes from Parent 1 drops down, and remaining values are placed in the child in the order which they appear in Parent 2.
- ☑ Performance
  - Order 1 crossover is perhaps the fastest of all crossover operators because it requires virtually no overhead operations.
  - On a generation by generation basis, edge recombination typically outperforms Order 1, but the fact that Order 1 runs between 100 and 1000 times faster usually allows the processing of more generations in a given time period.

## Crossover Operator

### - Order 1 Crossover (2/2)

- ☑ General Steps
  - Step 1: Select a random swath of consecutive genes from Parent 1. (red box)
  - Step 2: Drop the swath down to Child 1 and mark out these genes in Parent 2.
  - Step 3: Starting on the right side of the swath, grab genes from Parent 2 and insert them in Child 1 at the right edge of the swath. Since 5 is in that position in Parent 2, it is inserted into Child 1 first at the right edge of the swath. Notice that genes 7, 2, 1, and 6 are skipped because they are marked out and 8 is inserted into the 1<sup>st</sup> spot in Child 1.
  - Step 4: If you desire a second child from the two parents, flip Parent 1 and Parent 2 and go back to Step 1.

$P_1$  (3 5 7 2 1 6 4 8)  
 $P_2$  (~~2~~ 4 ~~1~~ ~~6~~ 8 3 5 ~~7~~)

$C_1$  (8 3 7 2 1 6 5 4)

Other variation:  $C_1$  (4 8 7 2 1 6 3 5) Following the orders of genes in Parent 2

### Crossover Operator - Order Multiple Crossover

This crossover is identical to the Order 1 Crossover, except **multiple swaths are participants** in the genetic exchange.

$P_1$  (3 5 7 2 1 6 4 8)  
 $P_2$  (~~2~~ ~~4~~ 1 ~~6~~ 8 3 ~~5~~ ~~7~~)

$C_1$  (1 5 7 2 8 6 4 3)  
 Other variation:  $C_1$  (1 5 7 2 8 6 4 3) Following the orders of genes in Parent 2

Topics in Ship Design Automation, Fall 2016, Myung-Il Roh **sydlab** 45

### Crossover Operator - Cycle Crossover (1/2)

The cycle crossover operator identifies a number of so-called cycles between two parent chromosomes. Then, to form Child 1, cycle one is copied from Parent 1, cycle 2 from Parent 2, cycle 3 from Parent 1, and so on. Two child can be made at the same time.

**General Steps**

- Step 1: Identify all cycles.

$P_1$  (3 5 7 2 1 6 4 8)  
 $P_2$  (2 4 1 6 8 3 5 7)

Cycle 1 values: **3 2 6**  
 Cycle 2 values: **5 4**  
 Cycle 3 values: **7 1 8**

Topics in Ship Design Automation, Fall 2016, Myung-Il Roh **sydlab** 46

## Crossover Operator - Cycle Crossover (2/2)

### ☑ General Steps (continued)

#### ■ Step 2: Copy alternate cycles to children.

$P_1$  (3 5 7 2 1 6 4 8)  
 $P_2$  (2 4 1 6 8 3 5 7)

Cycle 1 values: 3 2 6

Cycle 2 values: 5 4

Cycle 3 values: 7 1 8

$C_1$  (3 4 7 2 1 6 5 8)  
 $C_2$  (2 5 1 6 8 3 4 7)

Copy Cycle 1: Cycle 1 values from Parent 1 will be copied to Child 1, and values from Parent 2 will be copied to Child 2. Cycle 2 will be different.

Copy Cycle 2: Cycle 2 values from Parent 1 will be copied to Child 2, and values from Parent 2 will be copied to Child 1.

Copy Cycle 3: Cycle 3 is like Cycle 1, Parent 1 goes to Child 1, Parent 2 goes to Child 2.

## Crossover Operator - Edge Recombination (1/4)

- ☑ This is a crossover techniques for permutation (ordered) chromosomes.
- ☑ It strives to introduce the fewest paths possible.
- ☑ In problems such as the traveling salesman, introducing a stray edge between two nodes is usually very bad for a chromosome's fitness.
- ☑ The idea here is to use as many existing edges, or node-connections, as possible to generate children.
- ☑ Edge recombination typically **outperforms PMX and Ordered crossover**, **but usually takes longer time to compute**.



## Crossover Operator - Edge Recombination (2/4)

**General Steps**

■ **Step 1: Generate neighbor list.**

P<sub>1</sub> (A B F E D G C)  
P<sub>2</sub> (G F A B C D E)

Neighbor list of A: **B C F**

P<sub>1</sub> (A → B F E D G C)  
P<sub>2</sub> (G F ← A → B C D E)

Neighbor lists:

A: B C F      B: A F C      C: G A B D  
D: E G C      E: F D G      F: B E G A  
G: D C E F

## Crossover Operator - Edge Recombination (3/4)

P<sub>1</sub> (A B F E D G C)  
P<sub>2</sub> (G F A B C D E)

**General Steps (continued)**

■ **Step 2: Generate a child.**

- First, we randomly select the first node of a parent. Child C<sub>1</sub>: A
- Next, after crossing A out from all neighbor lists, we see that B is the least full list. So, Child C<sub>1</sub>: A B
- Next, after crossing B out from all neighbor lists, F and C both have only 2 neighbors, so we randomly choose between the two: Child C<sub>1</sub>: A B F
- Next, after crossing F out from all neighbor lists, E is the neighbor of F that has the fewest neighbors. Child C<sub>1</sub>: A B F E
- Next, after crossing E out from all neighbor lists, D and G both have only 2 neighbors, so we randomly choose between the two: Child C<sub>1</sub>: A B F E G
- Next, after crossing G out from all neighbor lists, D and C both have only 1 neighbor, so we randomly choose between the two: Child C<sub>1</sub>: A B F E G C
- Next, after crossing C out from all neighbor lists, D has only one neighbor and it has no neighbors left, so we randomly choose between the nodes that aren't yet included in the child. In this case, D is the only one left, so we're done: Child C<sub>1</sub>: A B F E G C D
- The child that we produced introduced only one new edge: A to D. This algorithm makes excellent use of existing edges and is much less likely to introduce stray edges during crossover.

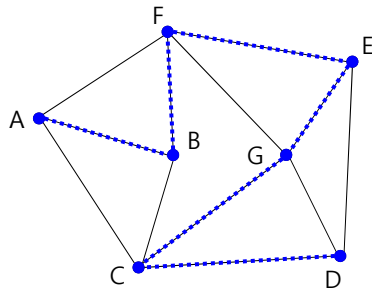
Neighbor lists:  
A: B C F / B: A F C / C: G A B D  
D: E G C / E: F D G / F: B E G A / G: D C E F

## Crossover Operator - Edge Recombination (4/4)

Neighbor lists:

A: B C F / B: A F C / C: G A B D  
D: E G C / E: F D G / F: B E G A / G: D C E F

Adjacency graph



Process 1: Child  $C_1$ : **A**

Process 2: Child  $C_1$ : **A B**

Process 3: Child  $C_1$ : **A B F**

Process 4: Child  $C_1$ : **A B F E**

Process 5: Child  $C_1$ : **A B F E G**

Process 6: Child  $C_1$ : **A B F E G C**

Process 7: Child  $C_1$ : **A B F E G C D**

$P_1$  (A B F E D G C)     $\rightarrow$      $C_1$  (A B F E G C D)  
 $P_2$  (G F A B C D E)

## Crossover Operator - PMX (Partially Mapped Crossover) (1/3)

- PMX crossover is a genetic algorithm operator for crossover.
- For some problems it offers better performance than most other crossover techniques.
- Basically, Parent 1 donates a swath genetic material and the corresponding swath from the other parent is sprinkled about in the child.
- Once that is done, the remaining genes are copied direct from Parent 2.



## Mutation Operator - Inversion Mutation Operator

### ☑ General Steps

- Step 1: Determine start and stop of swath.
- Step 2: Put the values of the swath back in reverse order.
- Step 3: Copy the values which are not in the swath.

$C_1$  (4 8 7 2 1 6 3 5)

$C_1'$  (4 8 6 1 2 7 3 5)

## Mutation Operator - Insertion Mutation Operator

### ☑ General Steps

- Step 1: Select some genes for insertion.
- Step 2: Put the values of the genes in order.
- Step 3: Copy the values which are not selected for insertion.

$C_1$  (4 8 7 2 1 6 3 5)

$C_1'$  (4 8 7 1 3 2 6 5)

## Mutation Operator - Single Swap (or Exchange) Operator

### General Steps

- Step 1: Select two genes to be swapped.
- Step 2: Swap two genes and put them.
- Step 3: Copy the values which are not selected for swap.

$C_1$  (4 8 7 2 1 6 3 5)

$C_1'$  (4 8 1 2 7 6 3 5)

## Mutation Operator - Random Swap Operator

- This is like single swap, but swaps a random string of consecutive values instead.

### General Steps

- Step 1: Select two swaths to be swapped.
- Step 2: Swap two swaths and put them.
- Step 3: Copy the values which are not selected for swap.

$C_1$  (4 8 7 2 1 6 3 5)

$C_1'$  (4 1 6 2 8 7 3 5)

## Mutation Operator - Scramble Mutation Operator

### ☑ General Steps

- Step 1: Determine start and stop of swath.
- Step 2: Scramble and put the values of the swath back.
- Step 3: Copy the values which are not in the swath.

$C_1$  (4 8 7 2 1 6 3 5)

$C_1'$  (4 8 2 6 7 1 3 5)

## Mutation Operator - Random Slide (or Displacement) Mutation Operator

### ☑ General Steps

- Step 1: Determine start and stop of swath.
- Step 2: Determine the length of the slide.
- Step 3: Calculate new location of swath after slide.
- Step 4: Put the values of the swath back.
- Step 5: Copy the values which are not in the swath.

$C_1$  (4 8 7 2 1 6 3 5)

$C_1'$  (4 8 6 3 5 7 2 1)

→ Slide length = 3

## Mutation Operator

### - Displaced Inversion Mutation Operator

☑ This mutation operator is a combination of “Inversion” and “Random Slide” operators.

☑ General Steps

- Step 1: Determine start and stop of swath.
- Step 2: Determine the length of the slide.
- Step 3: Calculate new location of swath after slide.
- Step 4: Put the values of the swath back **in reverse order**.
- Step 5: Copy the values which are not in the swath.

$C_1$  (4 8 7 2 1 6 3 5)  
→ Slide length = 3  
 $C_1'$  (4 8 6 3 5 1 2 7)

## 7.5 Example of Traveling Salesman Problem

## Traveling Salesman Problem (TSP)

- ☑ A problem for **finding a tour of a given set of cities** so that
  - each city is visited only once
  - the total distance traveled is minimized

## Representation of Chromosome for TSP

- ☑ Representation is an **ordered list of city numbers** known as an order-based GA.
- ☑ **Example of City Numbers**
  - 1: Seoul, 2: Tokyo, 3: Beijing, 4: Washington, 5: London, 6: Ottawa, 7: Canberra, 8: New Delhi

CityList1 (3 5 7 2 1 6 4 8)  
CityList2 (2 4 1 6 8 3 5 7)



## Crossover Operator for TSP


- ☑ Crossover combines inversion and recombination.
- ☑ "Order 1" crossover can be used in this example.

$P_1$  (3 5 7 2 1 6 4 8)  
 $P_2$  (~~2~~ 4 ~~1~~ ~~6~~ 8 3 5 ~~7~~)  
 $C_1$  (8 3 7 2 1 6 5 4)

## Mutation Operator for TSP

- ☑ Mutation involves reordering of the list.

**Before**  $C_1$  (8 3 7 2 1 6 5 4)  
**After**  $C_1'$  (8 3 6 2 1 7 5 4)



## 7.6 Multi-Objective Genetic Algorithms

### Classification of Optimization Problems According to Number of Objective Functions

- ☑ **Single-objective Optimization Problem**
  - The problem has only one objective function.
  - The problem has only a single (global) optimum.
  - The optimum can be maximum or minimum according to the type of objective function.
  
- ☑ **Multi-objective Optimization Problem**
  - The problem has at least two or more objective functions
  - The problem has not a single optimum but multiple optima called **Pareto optimal set**.
  - The optima can be subdivided into dominant solution and non-dominant solution.
    - Dominant solution: When objective function value of the solution is better than those of others
    - Non-dominant solution: When objective function value of the solution can not be improved without the increase of those of others. It is also called **Pareto optimal set** or Pareto front or Pareto frontier.

## General Solving Method of Multi-Objective Problem - Weighting Method (1/2)

- ☑ A method for finding optimum after **transforming multi-objective problem into single problem by multiplying objective functions by weight factors**
- ☑ Various optima (**Pareto optimal set**) can be found according to weight factors for objective functions.

$$\text{Minimize } f(x) = w_1 f_1(x) + w_2 f_2(x) + w_3 f_3(x) + \dots$$

$w_n$ : weight factors  
 $f_n$ : Objective functions

- ☑ **Limitations**
  - Optimum can change according to weight factors.
  - In some cases for the selected weight factors, optimum can not be found.
  - It requires much computing resources to find optima for all weight factors.
  - It is difficult to find a suitable combination of weight factors.

## General Solving Method of Multi-Objective Problem - Weighting Method (2/2)

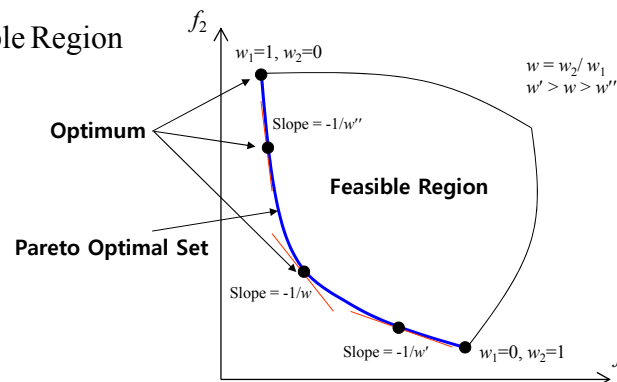
- ☑ Example of Multi-objective Problem Having 2 Objective Functions

*Minimize*

$$f(x_1, \dots, x_n; w) = w_1 f_1(x_1, \dots, x_n) + w_2 f_2(x_1, \dots, x_n)$$

*Subject to*

$$(x_1, \dots, x_n) \in \text{Feasible Region}$$



## General Solving Method of Multi-Objective Problem - Constraint Method (1/2)

- ☑ A method for performing optimization by regarding the other objective functions except for one as additional constraints of the problem to be solved
- ☑ Various optima (Pareto optimal set) can be found according to limiting values for the additional constraints.
- ☑ Limitations
  - It is difficult to find optimum when the problem has many objective functions due to the increase of additional constraints.

## General Solving Method of Multi-Objective Problem - Constraint Method (2/2)

- ☑ Example of Multi-objective Problem Having 2 Objective Functions

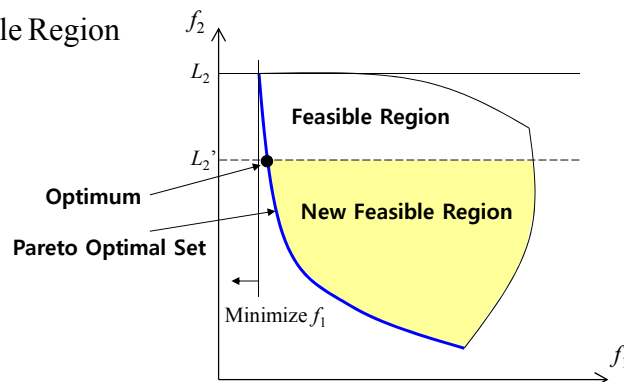
*Minimize*

$$f(x_1, \dots, x_n) = f_1(x_1, \dots, x_n)$$

*Subject to*

$$(x_1, \dots, x_n) \in \text{Feasible Region}$$

$$f_2(x_1, \dots, x_n) \leq L_2$$



## Multi-Objective GA (MOGA) (1/3)

- ☑ Most optimization problems in real world can not be represented with one objective function, but with **various objective functions which are opposed to each other** (e.g., Maximize strength vs. Minimize cost).
  - ➡ Multi-objective optimization problem
- ☑ It is called **genetic algorithms for solving multi-objective optimization problems**.
- ☑ In 1985, it was developed by Schaffer and is being widely used in various fields.

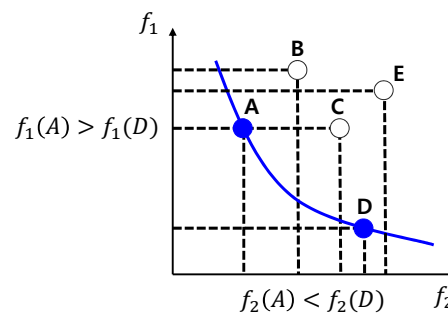
## Multi-Objective GA (MOGA) (2/3)

### Dominant Solution

- When objective function value of the solution is better than those of others
- $A \Leftrightarrow B, C, E$  or  $D \Leftrightarrow E$

### Non-dominant Solution

- When objective function value of the solution can not be improved without the increase of those of others
- It is also called Pareto optimal set or Pareto front or Pareto frontier.
- $A \Leftrightarrow D$



- ✓ Point B is dominated by point A. (The values of  $f_1$  and  $f_2$  are all small.)
- ✓ Point C is dominated by point A. (The value of  $f_1$  is same and the value of  $f_2$  is small.)
- ✓ Point E is not on the Pareto Frontier because it is dominated by both point A and point D.
- ✓ Points A and D are not strictly dominated by any other, and hence do lie on the frontier.
- ✓ A and D are called 'Pareto Optimal Set'.

## Multi-Objective GA (MOGA) (3/3)

- ☑ **MOGA** = **Multi-Objective** optimization problem + **Genetic Algorithms**
- ☑ Various methods have been developed to find Pareto optimal set based on genetic algorithms.

