APPENDIX A

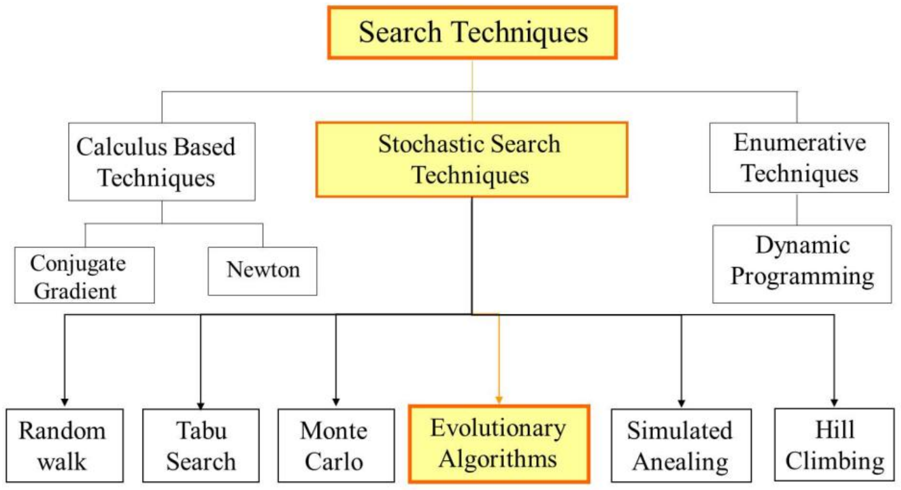
MULTI-OBJECTIVE EVOLUTIONARY ALGORITHM (MOEA)

At present, there are two main optimization approaches for multi-objective optimization problems (MOOPs) which are the derivative-based and derivative-free methods. The derivative-based schemes, such as Newton’s and conjugate gradient (CG) methods, have long been used in engineering applications. Meanwhile, the derivative-free optimization, such as, Evolutionary Algorithm (EA), Monte Carlo methods, Tabu search (TS) and Simulated Annealing (SA) are mostly employed in the field of control, machine intelligence and CAD/CAM (Marion, Michel, & Faye, 2006; Masoum, Ladjevardi, Jafarian, & Fuchs, 2004). Compared to the derivative-based approach, the functional derivative of a given objective is not necessary for derivative-free approach since it relies on repeated evaluation of the objective functions and identifies the search direction under nature-inspired heuristic guidelines. Although the derivative-free methods are generally slower than derivative-based methods in term of computer execution, the former are much more effective for complicated objective functions and combinatorial problems than the latter.

EA is a stochastic search technique, whose search mechanisms mimic the natural evolution and the Darwinian concept of survival of the fittest (Goldberg, 1989). EA embodies the techniques of genetic algorithms (GAs), evolutionary strategies (ES) and evolutionary programming (EP). Originally, EA is proposed to solve the single-objective optimization problems (Goldberg, 1989; Holland, 1975); subsequently it was developed for MOOPs.

MOEAs deal with a set of possible solutions, the so-called "population", simultaneously.

Figure A.1 Family of search techniques



The search process of MOEAs takes place between two spaces, namely, decision variable space and objective function space, simultaneously. MOEAs search for solution vectors $\mathbf{x}$ within the feasible region S and evaluate the objective function vectors $\mathbf{z} \in Z$ based on Pareto dominance relation. Therefore, many Pareto optimal solutions are found within a single run of the algorithm instead of performing a series of separate runs as in the case of the traditional mathematical programming techniques (Coello, 1999)**.** Additionally, MOEAs can easily deal with discontinuous Pareto fronts, whereas these are the real concern for single-objective approach and aggregating approach. MOEAs also have ability to search the partially ordered spaces for several alternatives trade-off. In the past decades, many researchers successfully used MOEAs to find good solutions for the complex MOOPs in medical, scientific and engineering field of research.

A.1 THE STRUCTURE OF EVOLUTIONARY ALGORITHMS

In the EAs, the evolution process is executed using evolutionary operators. Firstly, a solution vector (individual), i.e. solution **x**, is coded using strings of characters which are referred to as “chromosome”. A chromosome **x** is made up of N bit partitions. One bit partition corresponds to one decision variable of the problem, i.e. x_i. Due to natural selection process of EA, good solutions (fit individual) of the current generation, i.e. iteration, will be selected and reproduced by performing the genetic operators in order to create the better solutions for the next generation. The reproduction processes are operated by slightly changing some values in the bit partition of good solutions (fit individual) in order to explore the neighboring search space and to promote the propagation of desired characteristics from generation to generation. The standard procedure of EAs is displayed in Figure A.2 and explained as follows:

Step 1 Create individuals for initial population

Step 2 Decode chromosomes of individuals to obtain solution vectors of the problem

Step 3 Calculate the objective value of solution vectors

Step 4 Calculate the fitness of individuals using the obtained objective values.

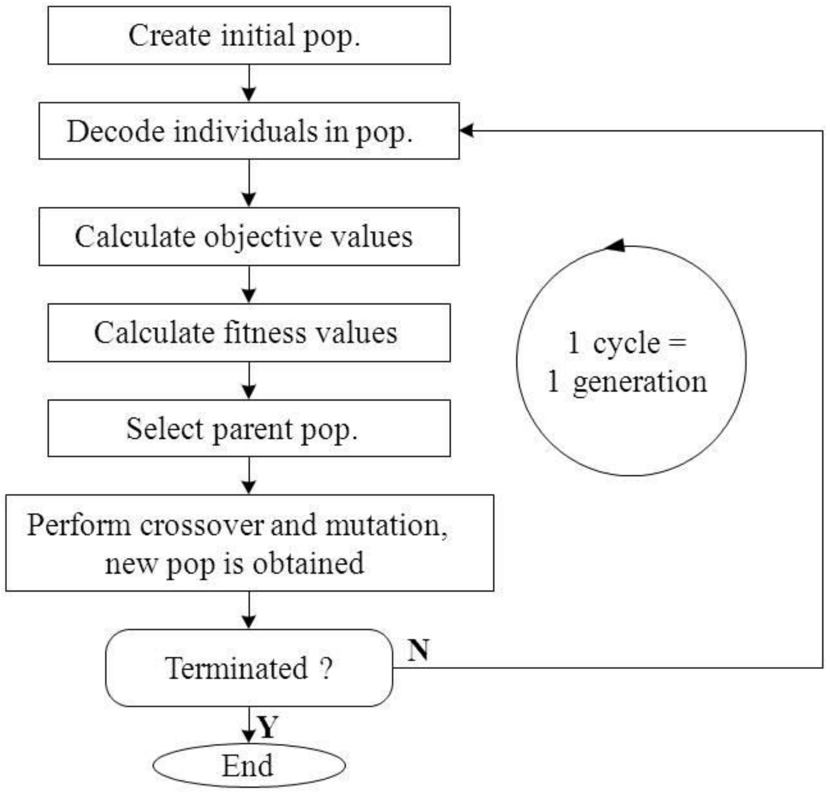
Step 5 Select a parent population from the current population based on the fitness value of individuals

Step 6 Produce an offspring population from a parent population using crossover operator and mutation operator

Step 7 Go back to step 2 until a termination condition, i.e. number of iteration, is satisfied.

Completion of one loop from steps 2 to 6 is accounted for one generation of algorithm run. The detailed explanation for genetic operators is given in the next section.

Figure A.2 Standard procedure of EAs

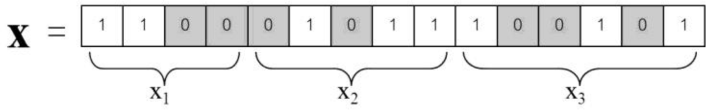

Create initial pop.
Decode individuals in pop.
Calculate objective values
Calculate fitness values
Select parent pop.
Perform crossover and mutation, new pop is obtained
Terminated ?
N
Y
End
1 cycle = 1 generation

A.2 EVOLUTIONARY OPERATORS

(a) Population Initiation Operator

As previously stated, population is made up of a set of solution vectors which coded as the strings of characters or chromosomes. Many chromosome coding techniques has been proposed such as binary coding, integer coding, and real-number. This study employed the real-number coding which is the newly developed technique and found to improve the running time of algorithms. However, an explanation of the binary coding is provided to give some fundamental explanations of how evolutionary operators work.

Basically, a chromosome **x** is made up of *N* bit partitions. One bit partition corresponds to one decision variable of the problem, i.e. x_i. For example, we consider a solution vector **x** having 3 decision variables or 3 bit partitions, i.e. $\mathbf{x} = [x_1 \quad x_2 \quad x_3]^{\mathrm{T}}$. According to the binary coding, decision maker has to design bit partition's length. Suppose that the bit partitions' lengths are 4, 5 and 6 for partition 1, 2 and 3, respectively. And given that $x_1 \in [-1,1]$, $x_2 \in [0,1]$ and $x_3 \in [1,5]$. By randomly generating the binary values, a chromosome **x** can be illustrated as follows:



$v_1 = (1100)_2 = 1(2^3) + 1(2^2) + 0(2^1) + 0(2^0) = 12$ Given v_1, v_2 and v_3 are corresponding decoded decimal numbers of binary bit partitions of x_1, x_2 and x_3 respectively and are simply evaluated as follows:

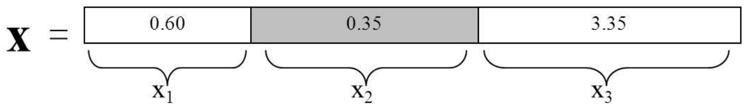
$$v_1 = (1100)_2 = 1(2^3) + 1(2^2) + 0(2^1) + 0(2^0) = 12$$

$$v_2 = (01011)_2 = 11 \text{ and } v_3 = (100101)_2 = 37$$

Then x_i can be simply decoded as the following equation:

$$x_i = x_i^l + v_i\left(x_i^u - x_i^l\right)/(2^{L_i} - 1) \qquad \text{(A.1)}$$

where x_i^l, x_i^u and L_i are the lower limit, upper limit and length of corresponding bit partition of x_i. Thus a solution vector **x** is decode as $\mathbf{x} = [0.6 \quad 0.3548 \quad 3.3492]^T$. Using the real-number coding, the solution **x** can be directly coded by the real-coded chromosome with three bit partition of which any bit *i* directly represents x_i. According to this example, a solution vector $\mathbf{x} = [0.6 \quad 0.3548 \quad 3.3492]^T$ is coded by 3 bit partitions chromosome as follows:



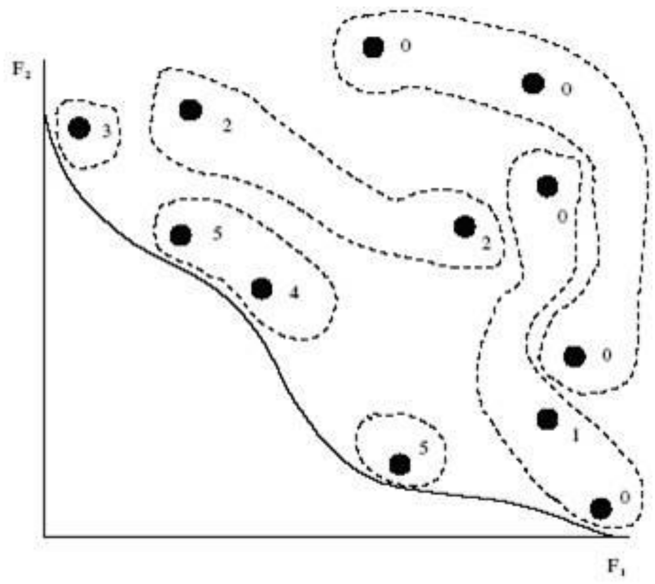
Next an encoded individual will be calculated its objective values and evaluated its fitness value.

(b) Fitness Evaluation Operator

In the case that the optimized objectives are expressed in mathematic form, the fitness of individual is firstly evaluated by its objective values. The fitness evaluation of single-objective EAs is quite easy because it depends on the type of problem, i.e. either maximization or minimization problems. Maximizing rate of returns function, for instance, the fitness of this objective has positive relation with its objective value. In the case of maximization problem, a fitness of an individual is equal to a constant plus its objective value. Meanwhile, for the minimization problem, the fitness of an individual is equal to a constant minus its objective value. The chosen values of the constant in both types must be able to guarantee that a fitness value of individuals is positive.

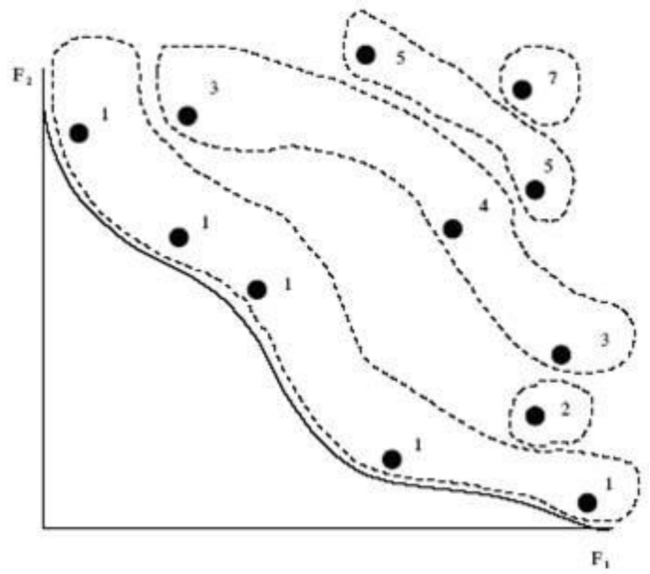
Pareto-based fitness evaluation, on the other hand, is performed in the objective function space based on the Pareto dominance relation. For example, considering two objectives optimization problem which both objectives will be minimized, i.e. Min-Min optimization, the objective values of individuals will be computed and projected on the two-dimension objective function diagram. Then, MOEAs compare objective values of individuals and assign ranking to individuals according to its dominance power. There are various ranking techniques proposed in the literature, such as, dominance rank, dominance count and dominance depth. Dominance count method, for instance, assigns rank to the considered individual relative to how many individuals in population does the considered individual dominate. Thus, the higher the assigned number the fitter the solution. Dominance depth technique assigns rank to the considered individual relative to which front or layer is the considered individual resided. Hence, the lower the front's number the fitter the individual. The graphical illustrations of dominance rank and dominance count method are displayed in Figure A.3 and Figure A.4, respectively.

Figure A.3 Dominance count method for Min-Min optimization problem



[Source: Coello, Lamont, and Van Veldhuizen (2007)]

Figure A.4. Dominance depth method for Min-Min optimization problem



[Source: Coello, Lamont, and Van Veldhuizen (2007)]

(c) Selection Operator

The selection operator is performed after the fitness assignment is done. The goal of the selection operator is to keep the fit individuals, i.e. good solutions, for either reproducing offspring solutions or representing the next generation solutions and to eliminate the weak individuals from the current population. There are several selection techniques that are usually implemented in MOEAs such as tournament selection, roulette wheel selection and stochastic universal sampling (SUS). For the tournament selection, which is widely used in the literature, the q solutions are randomly picked from the population. Then, the best solution among q solutions is selected. For example, in the Binary Tournament selection which is the most common used in MOEAs, two solutions (q = 2) are selected and compared. Then, the better solution is selected. The selection is repeated until the selection criterion is fulfilled.

(d) Crossover Operator

An EA explores the search space by reproducing the new individuals from the fit individuals. The crossover operator is used to guide the searching direction toward the true Pareto front. In crossover process, two fit individuals, referred to as parent individuals, are randomly picked out. Then, the elements in the chromosome of parents are passed onto two offspring individuals. Three widely implemented crossover techniques including n-point crossover, uniform crossover and simulated binary crossover (SBX) are illustrated here. The first two techniques are used for binary chromosomes, while the last one is used for real-number coded chromosomes.

n-point crossover

This crossover is operated on two randomly selected parent individuals. The crossover positions are randomly chosen from the numbers that range between 1 and $L - 1$ where L is the chromosome length. Figure A.5 and Figure A.6 illustrate the single-point crossover and the three-point crossover, respectively. For single-point crossover, the elements of chromosome behind the crossover point are split and swapped with the counterpart individual. In three-point crossover, split and swap processes are conducted three times to create a pair of offspring individuals.

Figure A.5 Single-point crossover

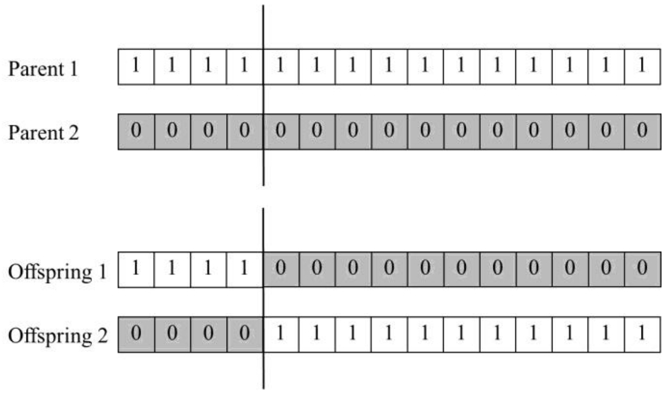
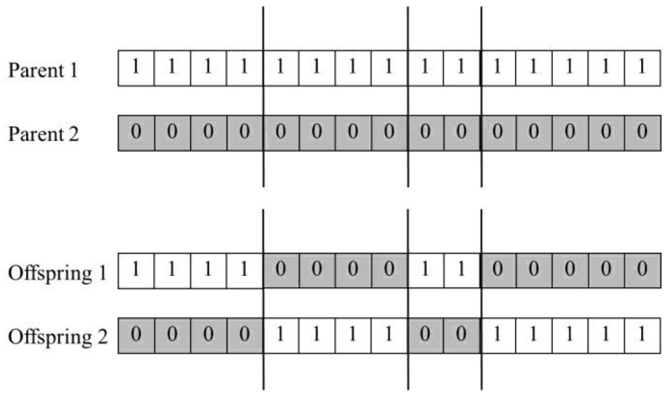


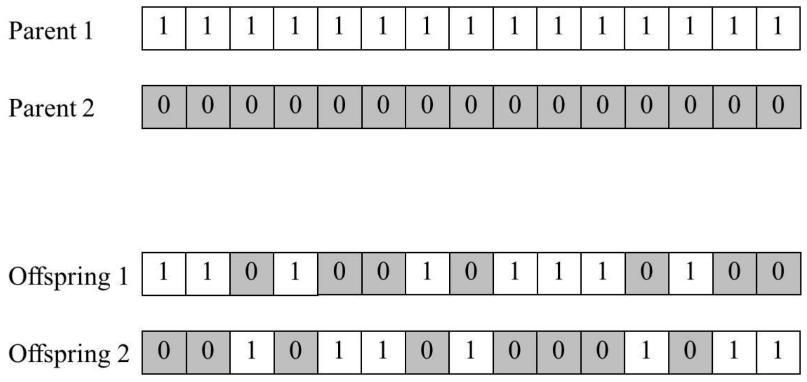
Figure A.6 Three-point crossover



Uniform crossover

In this technique, two parents are randomly selected and then assign “head” to one parent and “tail” to another parent. Next, randomly generate numbers between 0 and 1 for elements along the chromosome of the first offspring. By considering at each element, if the generated number of an element less than 0.5, the value of this element will be inherited from “head parent” and the value of this element of the second offspring will be transferred from “tail parent”. This assignment process is curried on until the last element of the first offspring is assessed. A schematic diagram describing uniform crossover is given in Figure A.7

Figure A.7 Uniform crossover



Simulated binary crossover

The simulated binary crossover (SBX) (Deb & Agrawal, 1995) adapts the one-point crossover on binary strings for real-number coded chromosomes. In this crossover, a probability distribution is used around parent solutions to create two offspring solutions. As an example for one-bit real-coded chromosome without lower and upper bound, offspring individuals y_i^1 and y_i^2 are computed from parent individuals x_i^1 and x_i^2 where $x_i^1 \leq x_i^2$. They are created by the use of a probability distribution around the parent individuals. The absolute difference values of offspring divided by that of the parent is a spread factor β_i and given by the following equation:

$$\beta_i = \left| \frac{y_i^2 - y_i^1}{x_i^2 - x_i^1} \right| \tag{A.2}$$

The probability distribution used to created offspring individuals has a similar search power as that of a single-point crossover in the binary-coded chromosomes. The probability distribution is given by the Equation (A.3) and its profile is shown in the Figure 3.6.

$$P(\beta_i) = \begin{cases} 0.5(\eta_c + 1)(\beta_i)^{\eta_c}, & \text{if } \beta_i \leq 1 \\ 0.5(\eta_c + 1)/(\beta_i)^{\eta_c + 2}, & \text{otherwise} \end{cases} \tag{A.3}$$

where η_c is the distribution index which can be any non-negative real number. The value of η_c has an effect on the distance between parent and offspring individuals which will be described later. From the Equation (A.3), $\int_0^\infty P(\beta_i) = 1$, $\int_0^1 \mathrm{P}(\beta_i) = 0.5$, and $\int_1^\infty \mathrm{P}(\beta_i) = 0.5$. In each crossover, a number *u* from 0 to 1 is randomly generated, then the parameter β_{u_i} is found so that the area under curve $\mathrm{P}(\beta_i)$ from 0 to β_{u_i} is equal to *u* or $\int_0^{\beta_{u_i}} \mathrm{P}(\beta_i) = u$. The β_{u_i} can be evaluated from the integral which is given by the following equation:

$$\beta_{u_i} = \begin{cases} \left(2u_i\right)^{1/(\eta_c+1)}, & \text{if } u_i \le 0.5 \\ \left(1/2\left(1-u_i\right)\right)^{1/(\eta_c+1)}, & \text{otherwise} \end{cases} \tag{A.4}$$

After β_{u_i} is obtained, the offspring individuals y_i^1 and y_i^2 can be computed by Equation (A.5) and Equation (A.6), respectively.

$$y_i^1 = 0.5\left[\left(1+\beta_{u_i}\right)x_i^1 + \left(1-\beta_{u_i}\right)x_i^2\right] \tag{A.5}$$

$$y_i^2 = 0.5\left[\left(1-\beta_{u_i}\right)x_i^1 + \left(1+\beta_{u_i}\right)x_i^2\right] \tag{A.6}$$

From Equation (A.5) and Equation (A.6), it can be noted that $\left|(y_i^2 - y_i^1)/(x_i^2 - x_i^1)\right| = \beta_{u_i}$ as in the Equation (A.2). Figure 3.7 shows a probability density distribution with η_c = 2 and 5 for creating offspring individuals from two parent individuals x_i^1 = 1.0 and x_i^2 = 3.0. A large value of η_c gives a higher probability for creating near parent individuals and a small value of η_c allows distant solutions to be selected as children individuals.

In the case that lower and upper boundaries (x^l and x^u) are specified, two parameter $\beta_{u_i}^1$ and $\beta_{u_i}^2$ are evaluated by Equation (A.7) and Equation (A.8), then, they are used to calculate y_i^1 and y_i^2 respectively.

$$\beta_{u_i}^1 = \begin{cases} \left(\alpha_1 u\right)^{1/(\eta_c+1)}, & u \le \dfrac{1}{\alpha_1} \\ \left(1/\left(2-\alpha_1 u\right)\right)^{1/(\eta_c+1)}, & \text{otherwise} \end{cases} \tag{A.7}$$

$$\beta_{u_i}^2 = \begin{cases} (\alpha_2 u)^{1/(\eta_c+1)}, & u \le \dfrac{1}{\alpha_2} \\ \left(1/(2-\alpha_2 u)\right)^{1/(\eta_c+1)}, & \text{otherwise} \end{cases} \tag{A.8}$$

where the parameters α_1 and α_2 are evaluated according to Equation (A.9) and Equation (A.10), respectively.

$$\alpha_1 = 2 - \beta_1^{-(\eta_c-1)},\ \beta_1 = 1 + 2\left(x_1 - x^l\right)/\left(x_2 - x_1\right) \tag{A.9}$$

$$\alpha_2 = 2 - \beta_2^{-(\eta_c-1)},\ \beta_2 = 1 + 2\left(x^u - x_2\right)/\left(x_2 - x_1\right) \tag{A.10}$$

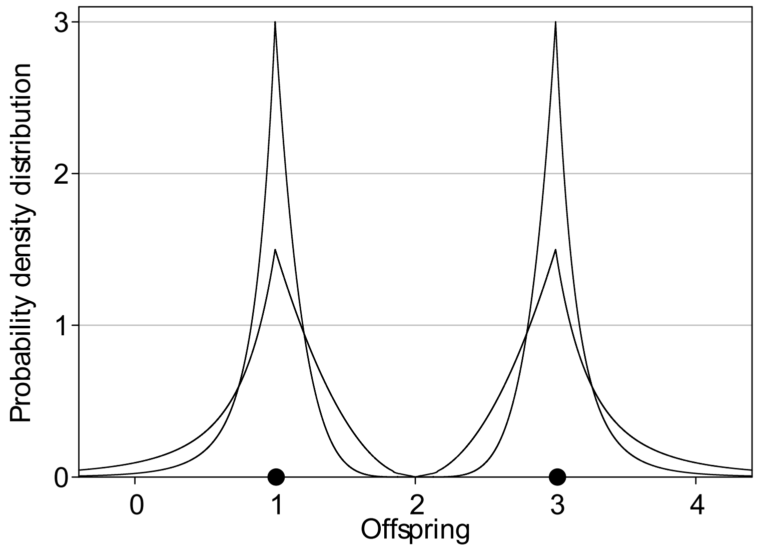
Thus, the offspring individuals y_i^1 and y_i^2 can be calculated by the Equation (A.11) and Equation (A.12), respectively.

$$y_i^1 = 0.5\left\{\left(x_i^1 + x_i^2\right) + \beta_{u_i}^1\left(x_i^2 - x_i^1\right)\right\} \tag{A.11}$$

$$y_i^2 = 0.5\left\{\left(x_i^1 + x_i^2\right) + \beta_{u_i}^2\left(x_i^2 - x_i^1\right)\right\} \tag{A.12}$$

The above description is the derivation of the SBX crossover for single-bit chromosomes. For multiple-bit chromosomes, similar to a uniform crossover in binary-code chromosome, two parent genes x_i^1 and x_i^2 at a same location are crossed by a probability 0.5. If the crossover is permitted, the parents are crossed to obtain offspring genes y_i^1 and y_i^2 as single-bit chromosomes, y_i^1 and y_i^2 are randomly switched to produce corresponding genes of offspring chromosomes.

Figure A.8 Probability distribution for creating offspring individuals



[Source: Deb & Agrawal (1995)]

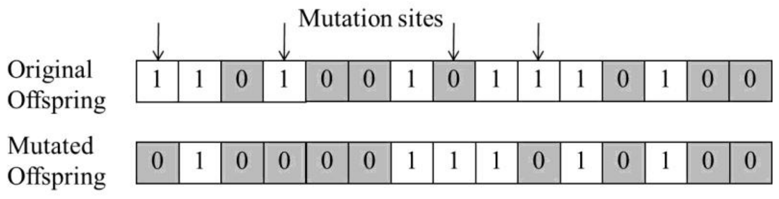
(e) Mutation Operator

A mutation operator is used to transform offspring individuals in order to promote a search at a neighboring area which has certain distance away from the original search point of parent individuals. It is used to maintain the diversity of individuals in a population resulting in prevention of the premature convergence of solution. Similar to crossover, the mutation of an offspring individual is also governed by the mutation probability. In the mutation, the randomly generated number having value between 0 and 1 is assigned to each element in an offspring's chromosome. In the case that the assigned number of a particular gene is less than or equal to the mutation probability, this gene is mutated, otherwise it remains unchanged. Two mutation methods for binary-coded and real-number coded chromosomes are presented as follows.

Bit-flipped mutation

For a binary chromosome, a mutation can be achieved by reversing the allele value of a gene. Figure A.9 illustrates bit-flipped mutation process which the mutation site is located at the element 4 of the chromosome. Thus the value at this element changes from 1 to 0. Similar to the crossover operation, mutation will not be operated to every elements of the chromosome. The mutation sites are chosen from the mutation probability of elements.

Figure A.9 Bit-flipped mutation for four mutation sites



Variable-Wise Polynomial Mutation

For real-coded chromosomes, a polynomial probability distribution is used to create a gene *y* in a mutated individual the vicinity of a corresponding gene *x* in an offspring individual (Deb, 1997). In the case of no lower and upper boundaries, firstly a random number *u* whose value is from 0 to 1 is generated. The parameter $\overline{\delta}$ is calculated as the following equation:

$$\bar{\delta} = \begin{cases} (2u)^{1/(\eta_m+1)} - 1, & \text{if } \; u \le 0.5, \\ 1 - \left[2(1-u)\right]^{1/(\eta_m+1)}, & \text{otherwise} \end{cases} \tag{A.13}$$

where η_m is the distribution index for mutation and takes any non-negative value. The gene y is evaluated as the following equation:

$$y = x + \bar{\delta}\Delta_{\max} \tag{A.14}$$

where $\Delta_{\max}$ is the maximum perturbation allowed in the considering gene in the offspring individual. For the gene whose lower and upper boundaries (x^l and x^u) are specified, the equation of parameter $\bar{\delta}$ is changed as following equation:

$$\bar{\delta} = \begin{cases} \left[2u + (1-2u)(1-\delta)^{1/(\eta_m+1)}\right] - 1, & \text{, if } \; u \le 0.5 \\ 1 - \left[2(1-u) + 2(u-0.5)(1-\delta)^{\eta_m+1}\right]^{1/(\eta_m+1)} & \text{, otherwise} \end{cases} \tag{A.15}$$

where $\delta = \min[(x - x^l),(x^u - x)]/(x^u - x^l)$ and $\Delta_{\max} = x^u - x^l$. The defined value of δ guarantees that no mutated genes are outside the range [x^l, x^u]. In Equation (A.14) and Equation (3.23), the mutated gene y is in the negative and positive sides of the corresponding gene x for u < 0.5 and u > 0.5, respectively. In addition, the value of normalized perturbation $(y-x)/(x^u-x^l)$ or $\bar{\delta}$ is the same order as that of $1/\eta_{\mathrm{m}}$ (Deb & Gulati, 2001). This implies that, for example, in order to get mutation effect of 5% perturbation in mutated genes, η_{m} should be set to 20.

A.3 THE NON-DOMINATED SORTING GENETIC ALGORITHM II (NSGA-II)

A fast elitist non-dominated sorting genetic algorithm (NSGA-II) purposed by Deb, et al. (2002) is quite different from the original algorithm, a non-dominated sorting genetic algorithm (NSGA) (Srinivas & Deb, 1994). Only the ranking technique of the previous version is remain unchanged and used in NSGA-II. This improved version does not require a niching parameter (σ_{share}) to maintain the diversity since it is found as a weak point when it is incorporated in a MOEA as in the NSGA. To maintain diversity of solutions, NSGA-II proposed a crowding distance sorting. Its procedure is as follows:

A.3.1 NSGA-II Main Algorithm

1. Create random initial population P_t of size N and set $t = 0$.

2. Compute fitness functions and evaluate fitness of individuals in the population

3. Assign rank for an individual in the population P_t.

4. Evaluate a crowding distance of an individual of each rank in P_t. The crowding distance of an individual indicates diversity of the individual.

5. Select parent population (A_t) from the current population P_t by binary selection which judges by rank and crowding distance value. For each time of binary selection, two solutions are randomly selected, if the ranks of the solutions are not equal, the solution with higher rank will be chosen. If their ranks are equal, for

diversity of parent population (A_t), the solution with more crowding distance value is chosen.

6. Apply crossover and mutation to A_t to form offspring population Q_t of size N.

7. Merge the current population P_t and offspring population Q_t to form a merged population R_t of size $2N$.

8. Compute fitness functions and evaluate fitness of individuals in the merged population R_t

9. Assign rank for an individual and evaluate crowding distance of each individual in the merged population R_t.

10. Obtain a new population P_{t+1} of size N from R_t of size $2N$ by using NSGA-II truncation method.

11. Check a termination condition. If the condition is satisfied, the algorithm is stop, the non-dominated solutions of final population P_{t+1} is the output of the algorithm, otherwise increase t to t+1 and go back to 3.

Three unique processes of NSGA-II including rank assignment, crowding distance evaluation and NSGA-II truncation will be described below.

A.3.2 NSGA-II Rank Assignment

As previously stated, a rank assignment of the non-dominated sorting genetic algorithm II (NSGA-II) is same as that of the original version. In the original algorithm, for a solution set *P* of size *N*, non-dominated solutions have the highest rank (rank = 1), they are then removed from the set. Subsequently, the non-dominated solutions of the remaining set are assigned the next rank or rank 2, and are removed from the current solution set again. The ranking process is repeated again until the remaining solution set is empty. The NSGA-II presents the fast non-dominated sorting for its ranking as the following pseudo-code in Figure A.10.

Figure A.10 Pseudo-code of fast-non-dominated sorting

```
Fast-non-dominated-sorting ($P$)
For each $p \in P$
  $S_p = \phi$
  $F_1 = \phi$
  $n_p = 0$
    For each $q \in P$
      If ($p \prec q$) then                    # If $p$ dominates $q$
        $S_p = S_p \cup [q]$                   # Add $q$ to the set of solutions dominated by $p$
      Else if ($q \prec p$) then               # $p$ belongs to the first rank
        $n_p = n_p+1$
    If $n_p = 0$ then
      rank$_p$ = 1
      $F_1 = F_1 \cup [p]$
$i$ =1                                         # Initialize the front counter
while $F_i \neq \phi$
  Q = $\phi$                                   # Used to store the members of the next rank
  For each $p \in Fi$
    For each $q \in Sp$
      $n_q = n_q$-1
      if $n_q = 0$ then                        # $q$ belongs to the next rank
        rank$_q$ = $i$+1
        Q = Q $\cup$ [$q$]
  $i = i + 1$

  $F_i$ = Q
```

[Source: Deb, et al. (2002)]

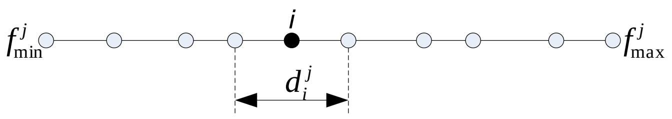
According to Deb (2005), the sorting procedure can be explained as follows. Firstly, for each solution, calculations for two entities are performed. The firs entity is domination count that is the number of solutions which dominate the solution *i*. The second entity is S_p which is a set of solutions which the solution *i* dominates. At the end of this procedure, all solutions in the first non-dominated front will have their domination count as zero. Now, for each of these solutions (each solution *i* with $n_i = 0$), we visit each member (*j*) of its set S_p, and reduce its domination count by one. In doing so, if for any member *j* the domination count becomes zero, we put it in a separate list *Q*. After such modifications on S_p are performed for each *i* with $n = 0$, all solutions of *Q* would belong to the second non-dominated front. The above procedure can be continued with each member of *Q* and the third non-dominated front can be identified. This process continues until all solutions are classified.

A.3.3 Crowding Distance Evaluation

NSGA-II proposed a crowding distance for diversity preservation of solutions in a population. The niching parameter (σ_{share}) of the original NSGA is replaced with a crowd comparison method, which sort the population according to the crowding distance of objective values. The crowd comparison method does not require any user-defined parameter such as sharing factor for maintaining diversity among population members. The crowding distance evaluation requires sorting the population according to each objective function. After that a crowding distance of a solution corresponding to each objective function is evaluated. The corresponding crowding distance of a boundary solution is equal to infinity, while, that of an interior solution is equal to the absolute normalized difference in the function values of two adjacent solutions. Figure A.11 shows the difference of two adjacent solutions of a non-extreme solution *i* in objective *j* (d_i^j), where $f_{\min}^j$ and $f_{\max}^j$ are the minimum and maximum values of objective function *j*. The corresponding crowding distance of the interior solution *i* for objective *j* (cd_i^j) is given by Equation (A.16).

$$cd_i^j = \frac{d_i^j}{\left(f_{\max}^j - f_{\min}^j\right)} \tag{A.16}$$

Figure A.11 Corresponding crowding distance to objective *j*



After the corresponding crowding distance of an individual to each objective is obtained, the overall crowding distance of the individual is computed as the sum of its corresponding crowding distance for each objective function. The crowding distance of a solution *i* (cd_i) for an *K*-objective optimization problem is given by the following equation:

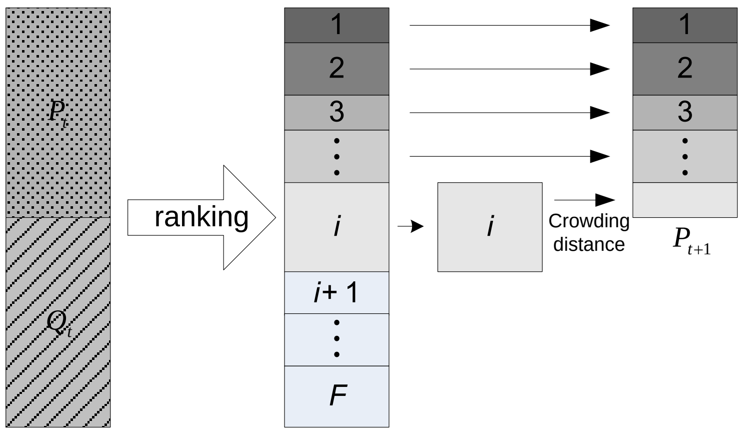
$$cd_i = \sum_{j=1}^{K} cd_i^j \tag{A.17}$$

The crowding distance is used not only in the binary selection but also in the truncation. The truncation of NSGA-II will be described in the following topic.

A.3.4 NSGA-II Truncation Method

After the merging of the current population P_t and offspring population Q_t, the new population P_{t+1} of size N is picked out from the merged population R_t of size $2N$ by NSGA-II truncation operator. At first, members of the merged population R_t is sorted according to their ranks from the highest rank (rank 1) to the lowest rank, afterward, P_{t+1} is formed. If the number of members in rank 1 is less than N, all members in the rank are put into the P_{t+1}, then, members in a next rank, rank 2, are considered to be chosen into P_{t+1}, the consideration for the rank is similar to that for the rank 1. If the number of members of rank 1 and rank 2 is less than N, all members in the rank are chosen, then, the members in the next rank are considered. The process is repeated until in the rank i such that the accumulated number of members of rank 1 to rank i is more than or equal to N. If the number is equal to N, the process is finished; otherwise if the number is more than N, the first members with the most crowding distances are fully filled P_{t+1}. The procedure of NSGA-II truncation is also described in Figure A.12.

Figure A.12 NSGA-II truncation



[Source: Deb, et al. (2002)]

A.4 IMPROVED STRENGTH PARETO EVOLUTIONARY ALGORITHM (SPEA-II)

An improved strength Pareto evolutionary algorithm (SPEA-II) (Zitzler, et al., 2002) is a upgraded version of the strength Pareto evolutionary algorithm (SPEA) (Zitzler & Thiele, 1999a). This new version has three major differences from the previous version. Firstly, it incorporates a fine-grained fitness assignment strategy which takes into account for each individual the number of individuals that dominate it and the number of individuals that it dominates. The second improvement is to use the nearest neighbor density estimation technique for guiding the more efficient search. The last point is to ensure the preservation of boundary solutions by using the enhanced archive truncation method. Given N^{pop} and N^{arc} is population size and archive size, unlike the original algorithm SPEA, the number of individuals in archive of SPEA-II is constant in every generation. The overall algorithm of SPEA-II is as the follows:

A.4.1 SPEA-II Main Algorithm

1. Create random initial population P_t of size N^{pop} and create the empty archive (external set) A_0, set $t = 0$.
2. Merge current population P_t and current archive A_t to form merged population R_t and then calculate fitness values of individuals in R_t.
3. Put all non-dominated solutions from R_t to the new archive A_{t+1}. If size of A_{t+1} is bigger than archive size N^{arc} then reduce A_{t+1} by using truncation operator, if size of A_{t+1} is equal to N^{arc} stop and go to the next step, otherwise, if size of A_{t+1} is smaller

than N^{arc} then fill A_{t+1} with the best $N^{arc} - |A_{t+1}|$ dominated individuals in R_t.

4. **4.** Check a termination condition. If the condition is satisfied, the algorithm is stop, the non-dominated solutions of final archive A_{t+1} is the output of the algorithm, otherwise go to the next step.
5. **5.** Perform binary tournament selection with replacement on the current archive A_{t+1} in order to fill the mating pool. In SPEA-II, only fitness values are consider in the binary selection.
6. **6.** Apply crossover and mutation operators to the mating pool and set P_{t+1} as the resulting population. Increase the generation counter by one, and then go back to Step 2.

The details of fitness assignment and archive truncation will be described below.

A.4.2 SPEA-II Fitness Assignment

At first, an individual *i* in merged population R_t is assigned its strength S_i as the number of solutions in R_t that it dominates. S_i is described as the following equation:

$$S_i = \left|\left\{ j \middle| j \in R_t \wedge i \prec j \right\}\right| \tag{A.18}$$

After strengths of all individuals in R_t are obtained, a raw fitness value of an individual *i* or RF_i is equal to the sum of strengths of individuals that dominate it. RF_i is computed by the following equation:

$$RF_i = \sum_{j \in R_t \ \wedge \ j \prec i} S_j \tag{A.19}$$

It is important to note that a raw fitness of a non-dominated individual is equal to zero. Although the raw fitness assignment provides a reasonable measurement based on the concept of Pareto domination, it may be fail when most individuals do not dominate each other. Therefore, a diversity value of an individual is incorporated to discriminate individuals with identical raw fitness values. The diversity value of an individual is evaluated from the density estimation technique, which is an adaptation of the *k*-th nearest neighbor method (Coello, 2001), where the density of a point *i* is inverse variation of its *k*-th nearest distance or d_i^k. Therefore, in SPEA-II, the density of an individual *i* or D_i is simply taken to the inverse of d_i^k, where *k* is equal to square root of the data size (Coello, 2001), i.e. k = $\sqrt{N^{pop} + N^{arc}}$. An individual with large density means that there are many neighboring point near it. Therefore, if the individual is selected, it contributes a little diversity to the mating pool. This implies that a density of an individual *i* or D_i is to be minimized as the raw fitness RF_i. The fitness or F_i is then equal to the sum of its raw fitness RF_i and its density D_i, hence the fitness F_i is also to be minimized. In addition, the evaluations of D_i and F_i can be described by Equation (A.20) and Equation (A.21), respectively.

The main goal of MOEAs developers is to upgrade the algorithms in order to provide the solutions which are close to the true Pareto front and well-diversified along the true Pareto front. The innovations of algorithms mostly deal with improving the fitness evaluation operator and the diversity preservation technique. For example, fitness assignment of NSGA-II and SPEA-II is evaluated based on dominance depth and dominance count method, respectively. For diversity preservation technique, NSGA-II uses the crowding distance technique, while SPEA-II employs nearest neighbor density estimation technique.

$$D_i = \frac{1}{d_i^k + 2} \tag{A.20}$$

$$F_i = RF_i + D_i \tag{A.21}$$

In Equation (A.20), value of two is added to d_i^k in order to ensure that the value of D_i is more than zero and less than one. In addition, the fitness of a non-dominated individual is less than one, while the fitness of dominated individual is more than or equal one.

A.4.3 SPEA-II Archive Truncation

Although Zitzler, et al., 2002 stated that the archive truncation of SPEA-II differs from that of its predecessor SPEA in the respect that it can prevent loss of extreme individuals which may happen by the clustering selection in SPEA. By detailed consideration, it prevents this loss only in a two objectives optimization problem, on the other hand, an extreme individual may be removed from the archive for a three-or-more objectives optimization problem. After non-dominated individuals of size more than the archive size N^{arc} in merged population R_t are put into the archive A_{t+1}, the archive truncation will iteratively removes individuals from the archive A_{t+1} until $|A_{t+1}| = N^{arc}$. For each iterative removal, an individual i is chosen to be removed from A_{t+1} if for all $j \in A_{t+1}$, $i \leq_d j$. The notation $i \leq_d j$ is described by the following equation:

$$\begin{aligned}&\forall k \in \left[1, |A_{t+1}| - 1\right] : d_i^k = d_j^k \quad \vee \\ &\exists k, k \in \left[1, |A_{t+1}| - 1\right] : \left[(\forall l \in [1, k-1] : d_i^l = d_j^l) \wedge d_i^k < d_j^k\right]\end{aligned} \tag{A.22}$$

where d_i^k denotes the k^{th} nearest distance from i to its neighbor in current A_{t+1}. The first condition in the above equation means that the objective vectors of i is the same as that of j. While the other condition can be described that the individual which has the minimum distance to another individual is chosen at each stage, if there are several individuals with minimum distance, the tie is broken by considering the second nearest distance and so on.