CHAPTER II

GENETIC ALGORITHM

2.1 Background to Genetic Algorithm (GA)

Genetic algorithm (GA) is an optimization and search technique based on the principles of genetics and natural selection. GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximize the "fitness" (i.e., minimizes the cost function). The method was developed by John Holland (1975) over the course of the 1960s and 1970s and finally popularized by one of his student, David Goldberg, who was able to solve a difficult problem involving the control of the gas-pipeline transmission for his dissertation (Goldberg, 1989). Holland's original work was summarized in his book. He was the first to try to develop a theoretical basis for GA through his schema theorem. The work of De Jong (1975) showed the usefulness of the GA for function optimization and made the first concerted effort to find optimized GA parameters. Goldberg has probably contributed the most fuel to the GA fire with his successful applications and excellent book (1989).

2.2 The Procedure of GA

Genetic algorithm (GA) searches the solution space of a function through the use of simulated evolution, i.e., the survival of the fitness strategy. In general, the fitness individuals of any population tend to reproduce and survive to the next generation, thus improving successive generations. However, inferior individuals can, by chance, survive and also reproduce. GA has been shown to solve linear and nonlinear problems by exploiting all regions of the state space and exponentially exploiting promising areas through selection, crossover, and mutation operations applied to individuals in the population. A more complete discussion of genetic algorithm, including extension and related topics, can be found in the books by Goldberg (1989), Michalewicz (1994), and Haupt (2004). A procedure of GA is summarized in figure 2.1, and each of the major components is discussed in detail below.

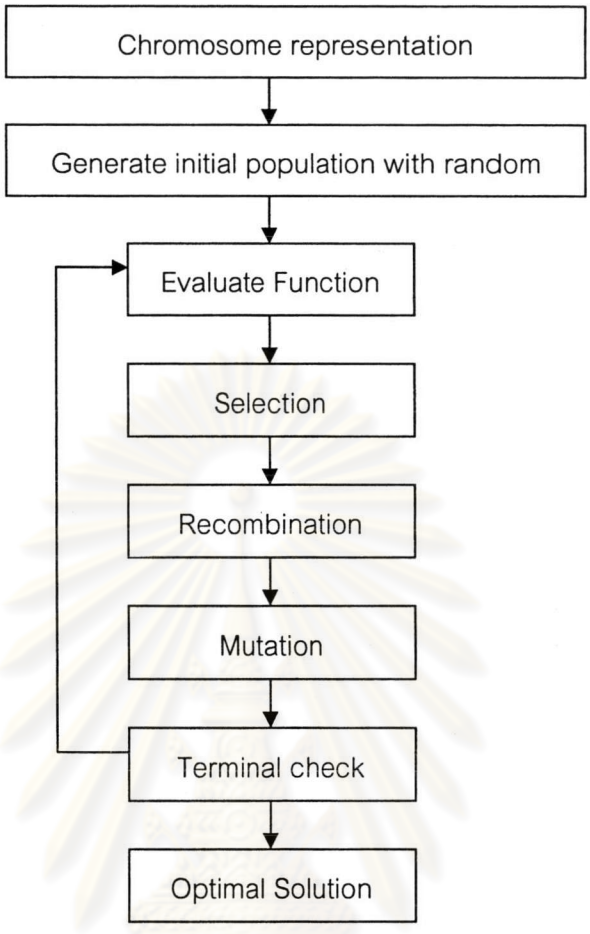

Figure 2.1 Procedure of GA

The use of genetic algorithm requires the determination of seven fundamental issues: chromosome representation, the creation of the initial population, the evaluation function, selection function, crossover operator, mutation operator, and the termination criteria. The rest of this section describes each of these issues.

2.2.1 Chromosome Representation

For GA, a chromosome representation is needed to describe each individual in the population of interest. The representation scheme determines how the problem is structured in the GA and also determines the genetic operators that are used. Each individual or chromosome is made up of a sequence of genes from a certain alphabets. An alphabet could consist of binary digits (0 and 1), floating point numbers

or real numbers, integers, symbols (i.e., A, B, C, D), matrices, and so on. In Holland's original design, the alphabet was limited to binary digits. Since then, problem representation has been the subject of much investigation. It has been shown that more natural representations are more efficient and produce better solutions (Michalewicz, 1994). One useful representation of an individual or chromosome for function optimization involves genes or variables from an alphabet of floating point numbers with values within the variables upper and lower bounds. Michalewicz has done extensive experimentation comparing real-valued and binary GA. He also shows that the real-value GA is an order of magnitude more efficient in terms of CPU time. Moreover, He presents a real-valued representation moves the problem closer to the problem representation which offers higher precision with more consistent results across replications. So, real-value GA is temped to use as a chromosome representation in this research.

2.2.2 Initialization

GA must be provided an initial population. The most common method is to randomly generate solutions for the entire population. However, since GA can iteratively improve existing solutions (i.e., solutions from other heuristics and/or current practices), the beginning population can be seeded with potentially good solutions, with the remainder of the population being randomly generated solutions.

2.2.3 Evaluation Function

Evaluation functions of many forms can be used in a GA, subject to the minimal requirement that the function can map the population into a partially ordered set. As started, the evaluation function is independent of the GA (i.e., stochastic decision rules).

2.2.4 Selection Function

The selection of individuals to produce successive generations plays an extremely important role in GA. A probabilistic selection is performed based upon the

individual's fitness such that the better individuals have an increased chance of being selected. An individual in the population can be selected more than once with all individuals in the population having a chance of being selected to reproduce into the next generation. There are several schemes for the selection process: roulette wheel selection and its extensions, scaling techniques, tournament, elitist models, and ranking methods (Goldberg, 1989 and Michalewicz, 1994).

A common selection approach assigns a probability pf selection, P_j, to each individual, j based on its fitness value. A series of N random numbers is generated and compared against the cumulative probability, $C_i = \sum_{j=1}^{i} P_j$, of the population. The appropriate individual, i, is selected and copied into the new population if $C_{i-1} < U(0,1) \leq C_i$. Various methods exist to assign probabilities to individuals: roulette wheel, linear ranking, and geometric ranking.

Roulette wheel, developed by Holland (1975), was the first selection method. The probability, P_i, for each individual is defined by:

$$P\text{ [Individual } i \text{ is chosen]} = \frac{F_i}{\sum_{j=1}^{popsize} F_j}, \qquad (2.1)$$

Where F_i equals the fitness of individual i. The use of roulette wheel selection limits the genetic algorithm to maximization since the evaluation function must map the solutions to a fully ordered set of values on R^+. Extensions, such as windowing and scaling, have been proposed to allow for minimization and negativity.

Ranking methods only require the evaluation function to map the solutions to a partially ordered set, thus allowing for minimization and negativity. Ranking methods assign P_i based on the rank of solution i when all solutions are sorted. Normalized geometric ranking (Joines and Houck, 1994) defines P_i for each individual by:

$$P\text{ [Selection the } i^{\text{th}}\text{ individual]} = q'(1-q)^{r-1}; \quad (2.2)$$

Where:

q = the probability of selecting the best individual,

r = the rank of individual, where 1 is the best,

P = the population size,

$$q' = \frac{q}{1-(1-q)^p}$$

Tournament selection like ranking methods only requires the evaluation function to map solutions to partially ordered set; however, it dose not assign probabilities. Tournament selection works by selecting j individuals randomly, with replacement, from the population, and inserts the best of the j into the new population. This procedure is repeated until N individuals have been selected.

2.2.5 Crossover

Genetic operators provide the basic search mechanism of the GA. The operators are used to create new solutions based on existing solutions in the population. There are two basic types of operators: crossover and mutation. Crossover takes two individuals and produces two new individuals while mutation alters one individual to produce a single new solution. The application of these two basic types of operators and their derivatives depends on the chromosome representation used. In this topic, crossover operator is discussed while mutation operator is explained in the next section.

Let $\overline{X}$ and $\overline{Y}$ be two *m*-dimensional row vectors denoting individuals (parents) from the population. For $\overline{X}$ and $\overline{Y}$ binary, simple crossover operator is defined. While, if $\overline{X}$ and $\overline{Y}$ are real-value, arithmetic and heuristic crossover procedure is described.

Simple crossover generates a random number r from a uniform distribution from 1 to m and creates two new individuals ($\bar{X}'$ and $\bar{Y}'$) according to the following equation.

$$x_i' = \begin{cases} x_i, \ if \ \ i < r \\ y_i, \text{ otherwise} \end{cases} \tag{2.3}$$

$$y_i' = \begin{cases} y_i, \ if \ \ i < r \\ x_i, \text{ otherwise} \end{cases} \tag{2.4}$$

Arithmetic crossover produces two complimentary linear combinations of the parents, where $r = U(0,1)$:

$$\bar{X}' = r\bar{X} + (1-r)\bar{Y} \tag{2.5}$$

$$\bar{Y}' = (1-r)\bar{X} + r\bar{Y} \tag{2.6}$$

Heuristic crossover produces a linear extrapolation of the two individuals. This is the only operator that utilizes fitness information. A new individual, $\bar{X}'$, is created using following equation, where $r = U(0,1)$ and $\bar{X}$ is better than $\bar{Y}$ in term of fitness.

$$\bar{X}' = \bar{X} + r(\bar{X} - \bar{Y}) \tag{2.7}$$

$$\bar{Y}' = \bar{X} \tag{2.8}$$

$$feasibility = \begin{cases} 1, if \ \ x_i' \geq a_i, x_i' \leq b_i \ \ \forall i \\ 0, \text{ otherwise} \end{cases} \tag{2.9}$$

If $\bar{X}'$ is infeasible, i.e., *feasibility* equals 0 as given by equation (2.9), then generate a new random number r and create a new solution using equation (2.7) and (2.8), otherwise stop. To ensure halting, after t failures, let the children equal the parents and stop.

In fact, there are further crossover techniques. However, according to Haupt (2004), type of crossover is not of much importance in term of GA performance. So, other crossover operations are insignificant in study.

2.2.6 Mutation

Mutation operation alters one individual to produce a single new solution in order to avoid an overly fast convergence and to jump out off a local optimum. For the binary chromosome, Binary mutation is defined.

Binary mutation flips each bit in every individual in the population with probability p_m according to equation (2.10).

$$x_i' = \begin{cases} 1 - x_i, \ if \ \ U(0,1) < P_m \\ x_i, \text{ otherwise} \end{cases} \tag{2.10}$$

For real $\overline{X}$ and $\overline{Y}$, the following mutation operators are defined: uniform mutation, boundary mutation, and non-uniform mutation. Let a_i and b_i be the lower and upper bound, respectively, for each variable i

Uniform mutation randomly selects one variable, j, and sets it equal to an uniform random number $U(a_i, b_i)$:

$$x_i' = \begin{cases} U(a_i, b_i), \ if \ i = j \\ x_i, \text{ otherwise} \end{cases} \tag{2.11}$$

Boundary mutation randomly selects one variable, j, and sets it equal to either its lower or upper bound, where $r = U(0,1)$:

$$x_i' = \begin{cases} a_i, if \ i = j, r < 0.5 \\ b_i, \ if \ i = j, r \geq 0.5 \\ x_i, \text{ otherwise} \end{cases} \tag{2.12}$$

Non-uniform mutation randomly selects one variable, j, and sets it equal to an non-uniform random number:

$$x_i' = \begin{cases} x_i + (b_i - x_i) f(G), if\ r < 0.5 \\ x_i - (x_i + a_i) f(G), if\ r \geq 0.5 \\ x_i, \quad \text{otherwise} \end{cases} \tag{2.13}$$

Where:

$$f(G) = (r_2(1 - \frac{G}{G_{\max}}))^b,$$

r_1, r_2 = a uniform random number between (0,1),

G = the current generation,

$G_{\max}$ = the maximum number of generations,

b = a shape parameter.

2.2.7 Termination

The GA moves from generation to generation selecting and reproducing parents until a terminal criterion is met. The most frequently used stopping criterion is a specified maximum number of generations. Another termination strategy involves population convergence criteria. In general, GA will force much of the entire population to converge to a single solution. When the sum of the deviations among individuals becomes smaller than some specified threshold, the algorithm can be terminated. The algorithm can also be terminated due to a lack of improvement in the best solution over a specified number of generations. Alternatively, a target value for the evaluation measure can be established based on some arbitrarily acceptable threshold. Several strategies can be used in conjugation with each other.

2.3 Example of Solving GA by Hand

In order to improve understanding of GA and their operators, GA is applied to a particular optimization problem step by step. Refer to Goldberg (1989), he consider the problem of maximizing the function $f(x) = x^2$, where x is permitted to vary between 0 and 31, a function displayed in figure 2.1

In solving GA by hand, binary chromosome is more appropriate than real-value string because crossover and mutation of real-value chromosome depend on randomness by computer. So, before we proceed with the simulation, let's briefly review the notion of a binary integer.

In base 2 arithmetic, we of course only have two digits to work with, 0 and 1; and as an example the number 10011 decodes to the base 10 number

$$1\times 2^4 + 0\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 = 19.$$

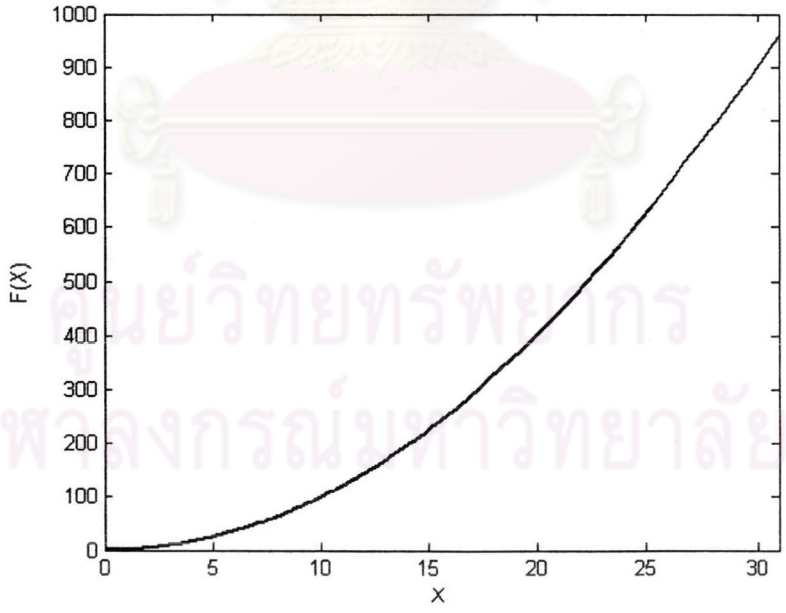

Figure 2.2 A simple function optimization example, the function $f(x) = x^2$ on the interval [0, 31].

With a five-bit (binary digit) unsigned integer we can obtain numbers between 0 (000000) and 31 (11111). With a well-defined objective function and coding, genetic algorithm is simulated a single generation through selection, crossover and mutation. The operators used in this example are defined in the table 2.1

The start off, an initial population is selected at random as shown in table 2.2. In the table, it was shown that the decoded x values are presented along with the fitness or objective function value $f(x)$.

Table 2.1 Operators used in each step for example of solving GA by hand

Step	Operator
Chromosome representation	Binary chromosome
Selection	Roulette wheel selection
Crossover	Simple crossover
Mutation	Binary mutation

Table 2.2 Initial population and its $f(x)$ for example of solving GA by hand

String No.	Initial Population	x value	$f(x) = x^2$
1	0 1 1 0 1	13	169
2	1 1 0 0 0	24	576
3	0 1 0 0 0	8	64
4	1 0 0 1 1	19	361

A generation of the GA begins with reproduction. Mating pool of the next generation is selected by spinning the weighted roulette wheel four times. Actual simulation of this process using coin tosses (Goldberg, 1989) has resulted in string 1

and string 4 receiving one copy in the mating pool, string 2 receiving two copies, and string 3 receiving no copy, as shown in table 2.3. Comparing this with the expected number of copies ($n \times \text{psection}_i$) we have obtained what we should expected: the best get more copies, the average stay even, and the worst die off.

With an active pool of strings looking for mates, simple crossover proceeds in two steps: (1) strings are mated randomly, using coin tosses to pair off the happy couples, and (2) mated string couples crossover, using coin tosses to select the crossing sites. Random choice of mates has selected the second string in the mating pool to be mated with the first. With a crossing site of 4, the two strings 01101 and 11000 cross and yield two new strings 01100 and 11001. The remaining two strings in the mating pool are crossed at site 2; the resulting strings may be checked in the table 2.4.

The last operator, mutation, is performed on a bit-by-bit basis. We assume that the probability of mutation in this test is 0.001. With 20 transferred bit positions we should expect $20 \times 0.001 = 0.02$ bits to undergo mutation during a given generation. Simulation of this process indicates that no bits undergo mutation for this probability value. As a result, no bit positions are changed from 0 to 1 or vice versa during this generation.

Following selection, crossover, and mutation, the new population is ready to be tested. To do this, we simply decode the new strings created by the GA and calculate the fitness function values from the x values thus decoded. The results of a single generation of the simulation are shown in table 2.4. In the table, note how both the maximal and average performance have improved in the new population. The population average fitness has improved from 293 to 493 in one generation. The maximum fitness has increased from 576 to 729 during that same period.

Table 2.3 Roulette wheel selection for example of solving GA by hand

String No.	$f(x) = x^2$	pselect $_i$ $\frac{f_i}{\sum f}$	Expect count $\frac{f_i}{\overline{f}}$	Actual Count (Roulette wheel)
1	169	0.14	0.58	1
2	576	0.49	1.97	2
3	64	0.06	0.22	0
4	361	0.31	1.23	1
Sum	1170	1.00	4.00	4.0
Average	293	0.25	1.00	1.0
Max	576	0.49	1.97	2.0

Table 2.4 Reproduction to the first generation for example of solving GA by hand

Parent	Mate	Crossover site	New population	x value	$f(x) = x^2$
0 1 1 0 | 1	2	4	0 1 1 0 0	12	144
1 1 0 0 | 0	1	4	1 1 0 0 1	25	625
1 1 | 0 0 0	4	2	1 1 0 1 1	27	729
1 0 | 0 1 1	3	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

In order to improve to the next generation, the best string of the first generation (11000) receives two copies because of its high, above-average performance. When this combines at random with the next highest string (10011) and is crossed at location 2 (again at random), one of the resulting strings (11011) proves to be a very good choice indeed. Then, if mutation operation occurs in string 11011 at location 3, it yields new string 11111. The string gives $f(x) = 961$, and this point is a global optimum.

2.4 Advantages of GA

GA has several advantages that enable GA to dominate other optimization techniques. Some of the advantages of GA include that it

- optimizes with continuous or discrete variable,
- doesn't require derivative information,
- simultaneously searches from a wide sampling of the cost surface (objective function),
- deals with a large number of variables,
- is well suited for parallel computers,
- optimizes variables with extremely complex objective surface (they can jump out of a local optimum),
- provides a list of optimum variables, not just a single solution,
- may encode the variables so that the optimization is done with the coded variables, and
- works with numerically generated data, experimental data, or analytical functions.

These advantages are intriguing and produce stunning results when traditional optimization approaches fail miserably.

2.5 Applications of GA

Because of GA's advantages, GA has been widely applied in various fields, such as engineering, computer science, biology, and so on. Table 2.5 lists some of the applications of GA in the field of engineering, especially chemical engineering. For other fields, various applications are listed in table 2.6.

Table 2.5 GA application in the field of chemical engineering

Year	Authors	Description
1983	Goldberg	Steady-state and transient optimization of gas pipeline using GA
1991	Androulajis and Venkatasubramanian	Synthesis heat exchanger networks via GA
1998	Garrard and Fraga	Improvement of mass transfer networks in order to reduce the wasted materials generated in the process
1998	Wang, Qian, Yuan, and Yao	Synthesis of distillation sequence by using GA
1999	Tayal, Fu, and Diwekar	Synthesis heat exchanger networks with GA
2004	Majumdar and Mitra	Maximize the main product of complex reaction networks via GA
2004	Son, Lee, Kim, and Choi	The use of GA to select the optimal architecture of neural network in the hot rolling process
2005	Lavric, Iancu, and Plesu	Optimization of water consumption and wastewater network topology via GA
2005	Kordabadi and Jahanmiri	The use of GA to optimization of methanol synthesis reactor in order to enhance overall production

Table 2.6 GA application in other fields

Year	Authors	Description
1981	Smith and De Jong	Calibration of population migration model using GA search
2005	Haldenbilen & Ceylan	Improvement transport energy demand estimation efficiency for future projections by using GA
2005	Topcuoglua, Coruta, Ermisb and Yilmaza	Design the hub location via GA
2005	Tan and Bhanu	Fingerprint matching by GA
2005	Bo, Hua, and Yu	The use of GA to optimize process route sequencing being key technology of the computer aided process planning (CAPP) system

2.6 Summary

In this chapter, theory of Genetic algorithm (GA) is explained. Section 2.1 presents the background to GA. Second 2.2 explains the procedure of GA, and their fundamental issues are discussed. In order to improve understanding the theory of GA, the example of solving GA by hand is illustrated. Section 2.3 shows several advantages of GA, and section 2.4 presents various applications of GA in the field of chemical engineering and other fields.

Further reading in area of GA and its application can be found in the book of Goldgerg (1989), Back, Fogel, and Michalewicz (2000), and Haupt and Haupt (2004).