Korean J. Chem. Eng., **28**(1), 32-40 (2011)
DOI: 10.1007/s11814-010-0323-3

Solving mixed-integer nonlinear programming problems using improved genetic algorithms

Tawan Wasanapradit*, Nalinee Mukdasanit, Nachol Chaiyaratana***, and Thongchai Srinophakun**,****,†**

*Department of Chemical Engineering, Faculty of Engineering,
King Mongkut's University of Technology Thonburi, Bangkok 10140, Thailand
**Department of Chemical Engineering, Faculty of Engineering,
Kasetsart University, Bangkean, Bangkok 10900, Thailand
***Department of Chemical Engineering, Faculty of Engineering,
King Mongkut's University of Technology North Bangkok, Bang Sue, Bangkok 10800, Thailand
****National Center of Excellence for Petroleum, Petrochemicals, and Advanced Materials,
Pathumwan, Bangkok 10330, Thailand
(*Received 6 March 2010 • accepted 13 May 2010*)

Abstract−This paper proposes a method for solving mixed-integer nonlinear programming problems to achieve or approach the optimal solution by using modified genetic algorithms. The representation scheme covers both integer and real variables for solving mixed-integer nonlinear programming, nonlinear programming, and nonlinear integer programming. The repairing strategy, a secant method incorporated with a bisection method, plays an important role in converting infeasible chromosomes to feasible chromosomes at the constraint boundary. To prevent premature convergence, the appropriate diversity of the structures in the population must be controlled. A cross-generational probabilistic survival selection method (CPSS) is modified for real number representation corresponding to the representation scheme. The efficiency of the proposed method was validated with several numerical test problems and showed good agreement.

Key words: Genetic Algorithms, Mixed Integer Nonlinear Programming, Repairing Strategy, CPSS, Modified Genetic Algorithms

INTRODUCTION

Genetic algorithms (GAs) are relatively well-established tools in the field of artificial intelligence (AI). Developed in the 1960s by John Holland, at the University of Michigan, they were initially designed as computer-based models which were exhibited and used to describe adaptive processes associated with natural genetics. Many researchers have since proven that GAs both theoretically and empirically provide a robust mathematical search mechanism. For this reason, GAs have subsequently become a mathematical technique, rather than merely a biological model [1]. GAs are used for solving optimization problems, and are classified as a direct method technique in the optimization field. The GAs are more robust than indirect methods because of their discontinuous and multimodal objective functions. A direct method has a greater chance of success when the nonlinear function is optimized. It can avoid getting stuck in local optima and does not require any derivative. On the other hand, indirect methods are more efficient than the direct method for such small problems. For instance, quadratic programming problems can be solved by an indirect method within minutes, while the conventional direct method may take several hours or even a day to find the solution. However, GAs differ from other direct approaches since they have an efficient search algorithm. Today GAs are used to solve the "hard-to-solve" problems such as mixed-integer nonlinear programming (MINLP). The performance of GAs depends on evolutionary strategies. Development is still continuing, and presently GAs are used to solve many types of problems, examples of which follow.

Murata et al. [2] applied a GA to flow-shop scheduling problems, and examined two hybridizations of GAs with other search algorithms: the two-point crossover and the shift change mutation. These were compared to other search algorithms such as local search, taboo search, and simulated annealing. However, the result showed that the GA was a bit inferior to the others. To improve the performance of the GA, they applied two hybrid GAs with a conventional algorithm: genetic local search and genetic simulated annealing. These hybrid GAs improved the performance of the original GA. Nazario [3] solved a mixed-integer programming problem of production planning by using a heuristic algorithm. First, optimal and suboptimal continuous solutions were identified. Then, an integer solution was found in the neighborhood of each suboptimal and optimal point. If the integer point provided an infeasible solution, then the dual simplex method was used to derive a feasible integer solution. The suggested algorithm was derived under the framework of an integer exploratory search principle. Once the integer solution was found at each optimal and suboptimal point, the best point was addressed by the heuristic algorithm. Yokota et al. [4] proposed a penalty function to evaluate infeasible chromosomes generated during genetic reproduction. This was applied to solve reliability problems and is classified as mixed-integer nonlinear programming.

Although GAs can attain the global optimum without getting stuck

†To whom correspondence should be addressed.
E-mail: fengtcs@ku.ac.th

at local optima, traditional GAs have limited power for multimodal functions. Shimodaira [5] proposed a new algorithm - a diversity control oriented genetic algorithm (DCGA) - to correct this problem. In DCGA, the structure of the population for the next generation comes from a merged population of parents and their children to eliminate duplication of chromosomes. The selection is based on probability, which is a function of a hamming distance between the candidate structure and the structure with the best fitness values. The hamming distance might be the number of the similar bit strings between a chromosome and the reference chromosome. DCGA will spread the chromosomes in the search space to ensure reaching the global optimum without converging to local optima.

In this paper, the algorithm for solving a mixed-integer nonlinear problem was constructed by improving on the work of Yokota et al. [4]. The improvements consist of mutation operators, constraint handling, and the selection process. The efficiency of the proposed method will be compared to the previous work. Other parts, such as representation and crossover operator, remain the same as in the previous method.

METHODOLOGY

1. Mixed-integer Nonlinear Optimization

This class of optimization problem arises from a variety of applications which involve integers or discrete variables in addition to continuous variables. The integer variables can be used to model, for instance, sequences of events, alternative candidates, and existence or nonexistence of units (0-1 representation), while discrete variables can be modeled for different equipment sizes. The continuous variables are used to model the input-output and interaction relationships among individual units/operations and different interconnected systems.

The general formulation of MINLP problems which maximize a nonlinear objective function f(**m**, **r**) can be stated as:

$$\text{Maximize } f(\mathbf{m},\mathbf{r}) \quad \text{Subject to} \begin{cases} g(\mathbf{m},\mathbf{r}) \le 0 \\ \mathbf{m} \in \text{M integer} \\ \mathbf{r} \in \text{R real number} \end{cases} \tag{1}$$

Many algorithms have already been developed for solving MINLP. Floudas [6] summarized the methods for MINLP:

1. Generalized Benders Decomposition
2. Branch and Bound
3. Outer Approximation
4. Feasibility Approach
5. Outer Approximation with Equality Relaxation
6. Outer Approximation with Equality Relaxation and Augmented Penalty
7. Generalized Outer Approximation
8. Generalized Cross Decomposition

However, these methods use conventional optimization techniques. In the subsequent section, the genetic algorithms for solving MINLP will be explained.

2. Improved Genetic Algorithms

Genetic algorithms (GAs) are stochastic search techniques based on the mechanism of natural selection and natural genetics. The process starts with an initial set of random solutions (population). Each population is called a chromosome. The chromosomes then evolve through continuous iterations (generations). After each iteration, the chromosomes are evaluated, using a measure of fitness. The offspring are the new generation of chromosomes that are formed by using genetic operators (crossover, mutation and selection). By selection of parents, offspring will be rejected or selected to regulate population size. The fitter chromosomes have higher probabilities of being selected. Finally, after many generations, the algorithms converge to the best chromosome, which is expected to present the optimal or suboptimal solution to the problem. In the following subsections, we will introduce the concepts of representation, crossover operator, mutation operator, constraint handling and selection process for solving MINLP, which will also cover nonlinear integer programming and nonlinear programming.

3. Representation

Normally, binary representation is used in GAs, but this has some disadvantages when applied to multidimensional, high-precision numerical problems. For example, for 100 variables with domain range [−500, 500] where the required precision after the decimal point is six, the length of the binary solution vector is 3,000.

The major inefficiency of GAs is the huge amount of computing time required for each solution. Michalewicz [7] used real numbers to represent the solution, and conducted many experiments to compare the efficiency of real number and binary string representation. The results showed that real number representation was superior, providing greater precision, especially in the case of large domains where binary coding would require prohibitively long representation. The strategy was straightforward; for example, if a solution has three variables - x_1=−10.32, x_2=5.47, and x_3=6.14 - the real number solution vector is [−10.32, 5.47, 6.14].

For the case of an integer variable, however, the real number of that variable will be rounded to a maximum integer less than that of the real number. For example, $\mathbf{v}_j$=[−10.32, 5.47, 6.14] will be converted to $\mathbf{v}_j$=[−11, 5, 6].

4. Crossover

In this work, arithmetical crossover is defined as a linear combination of two vectors. It was modified in order to apply to real number representation. For example, if $\mathbf{v}_1$ and $\mathbf{v}_2$ need to be crossed, the offspring are $\mathbf{v}_1'$=a•$\mathbf{v}_1$+(1−a)•$\mathbf{v}_2$ and $\mathbf{v}_2'$=a•$\mathbf{v}_2$+(1−a)•$\mathbf{v}_1$. This operator uses two random values of a ∈ [0, 1].

$\mathbf{v}_1$=[7.32, 5.28]; $\mathbf{v}_2$=[4.61, 9.23]; a=0.6;
$\mathbf{v}'_1$=0.6[7.32, 5.38]+(1−0.4)[4.61, 9.23]=[6.23, 6.86]; $\mathbf{v}'_2$=[5.69, 7.65]

5. Mutation

The procedure of uniform mutation is the selection of chromosomes for undergoing mutation with the same strategy as simple GAs. This operator requires a single parent **v** and produces a single offspring **v′**. The selection is a random component k∈(1, ..., q) of vector **v**=[v_1, …, v_k …, v_q] to produce **v′**=[v_1, …, v_k' …, v_q]. v_k' is a random value from the range [v_k^L, v_k^U] where [v_k^L, v_k^U] are the lower and upper bounds of variable v_k, respectively; and 'a' is a random number in {0, 1} from uniform probability distribution.

$$V_k' = v_k^L + a(v_k^U - v_k^L) \tag{2}$$

Using the same method, Yokota et al. [4] and Dhingra [8] incorporated a local search scheme for the mutation operator to find a

better position near the current one during the mutation stage. The general procedure is to randomly select a chromosome v_k for mutation, and to select a gene in this chromosome for mutation with v_k. The replacement of the gene with a value in its domain $[v_k^L, v_k^U]$ makes the fitness of offspring greater than other alternatives. Thus, $v_k^U - v_k^L$ iteration times are needed for an integer variable undergoing mutation for the best fitness value. This method can search a small boundary range, but this is a drawback for problems with a large boundary range. Supposing v_k^U=1000 and v_k^L=10: GA takes 990 times to perform a local search for just one chromosome. The algorithm speed will considerably decrease. In this research, to boost the mutation performance with local search, a new mutation operator, multiposition mutation, is proposed. This is expected to reduce the lag time of the local search. The mutated chromosome will be set to the target genes at the mutation. Consequently, these genes are replaced with new values using the same method as uniform mutation. If any variable is an integer, that gene value will be rounded to the integer value.

6. Handling Constraint

To handle optimization problems, a genetic algorithm operator which generates offspring often yields an infeasible chromosome. There are many methods to relax this constraint problem. The penalty method is presently widely used. One disadvantage, however, is that the unsuitable penalty function of GAs may produce an infeasible solution, since it cannot restrict the GAs to search only in feasible areas. The new method is introduced for constraint problems. It uses the repairing strategy. Repairing technique involves generating a feasible chromosome from an infeasible one through a repairing procedure. Consider the mathematical model of constraint selection, as follows:

$$g_i(x) \le b_i \text{ or}$$
$$S(x) = g_i(x) - b_i \le 0 \quad (3)$$

where $g_i(x)$ is the inequality constraint, b_i is the right hand side of the inequality constraint, and S(x) is the distance between chromosome and constraint.

Eq. (3) shows that S(x) must be equal to or less than zero for the feasible chromosome, and more than zero for the infeasible one. S(x)=0 means the chromosome is lying on the constraint boundary and is feasible. S(x)<0 means the chromosome is under the constraint; and vice versa, if S(x)>0 the chromosome exceeds the constraint and falls into the infeasible zone, as shown in Fig. 1. From this concept, S(x) can be used to indicate the feasibility of the chromosome.

The principle of the repairing strategy is to convert the infeasible (S(x)>0) into the feasible chromosome (S(x)≤0). The secant method is employed to convert the infeasible chromosome to the new one with S(x)=0. Therefore, the repaired chromosomes are located on the constraint boundary. The advantage of searching performance on the constraint boundary is that solutions such as linear programming problems will be clearly addressed.

The secant method in the following equation is similar to the Newton-Raphson technique in the sense that the root estimation is predicted by extrapolating a tangent of the function to the x-axis. However, the secant method uses a difference rather than a derivative to estimate the slope in a so-called direct method [9].

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(X_{i-1}) - f(x_i)} \quad (4)$$

However, the application of the secant method with GAs requires some additional steps for the calculation of f(x). Since each chromosome will give an amount of g(x) equal to the number of inequality constraints, the most suitable one that gives the maximum value set of $g_i(x) - b_i$ must be nominated to be f(x).

Even though the secant method is an effective method, it may not converge on some initial guesses. Hence, the bisection method is supported by the root finding of S(x) (S(x) is f(x) in secant method procedure) in this repairing procedure. Whenever the secant method cannot derive an infeasible solution to a feasible solution, the best solution at that stage is passed to the bisection method to find a rough solution. After that, the last two best solutions from bisection are then returned to the secant method to address more accurate root searching. The repairing method with bisection follows the work of Wasanapadit [10]. In practice, there are still some infeasible chromosomes that cannot be repaired even when passed through this repairing procedure. In such cases, they will be rejected from the population.

7. Selection Process

According to Yokota's algorithm from 1996, the selection strategy is an elitist strategy. It is suitable and can solve the general problem. However, the elitist strategy may give a small diverse chromosome in the next generation. Hence, the selection is only based

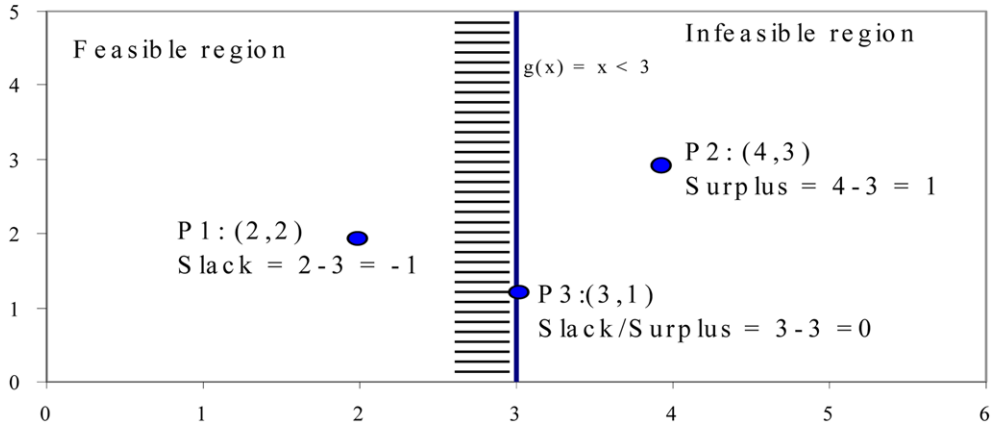


Fig. 1. Possible location of chromosome in search space.

on fitness value. Many researchers have observed that premature convergence (getting stuck in local optimum) often occurs after an individual or a small group of individuals contributes a large number of offspring to the next generation. To prevent this problem, Shimodaira [5] developed diversity control oriented genetic algorithms (DCGA) to handle the premature convergence problem of traditional genetic algorithms. These techniques are a cross-generational deterministic survival selection method (CDSS) and a cross-generational probabilistic survival selection method (CPSS). In this work, we selected the CPSS method as the DCGA.

In the CPSS method, chromosomes from offspring and the old population are selected using random numbers based on a selection probability using the following equation:

$$p_s = \left\{ (1-m)\frac{h}{L} + m \right\}^{\alpha} \tag{5}$$

where:

h=the hamming distance between the candidate chromosome and the chromosome with the best fitness value;

L=the length of the string representing the chromosome;

m=the shape coefficient; and

α=the exponent.

The original method of DCGA was invented to use with binary or gray code representation. The hamming distance (h) in Eq. (5) is calculated from the number of similar bit strings between the candidate chromosome and the fittest chromosome. The length of chromosome, L, can be calculated from the number of bit strings in the chromosome. In this paper, the calculation method for h and L from real number representation must be developed from the same basic concept.

Since the binary representation counts the number of bit strings as a distance, it is an analogy of Euclidean distance for real number representation. The Euclidean distance can be calculated by this equation:

$$h = \sqrt{(v_1 - v_1')^2 + \ldots + (v_k - v_k')^2} \tag{6}$$

where v_i is the i^{th} gene of the candidate chromosome, and v_i' is i^{th} gene of the fittest chromosome.

For example:

$\mathbf{v}=[3, 4]$, $\mathbf{v}'=[1.2, 2]$; $h = \sqrt{(3\text{-}1.2)^2 + (4\text{-}2)^2} = 2.6907$

L is the maximum distance between the fittest chromosome and any chromosome in that generation. It can be expressed as follows:

$$L = \max(h_j); \quad j = 1, \ldots, pop\_size \tag{7}$$

With this approach, we can apply CPSS with real number representation. Nevertheless, it must be further improved for mixed-integer problems, as it has two types of variables: real number variables and integer variables. Therefore, the distance must be calculated separately between real and integer variables. After that, their distances are combined together with a weight factor of distance. Eq. (8) is used to calculate the hamming distance of mixed integer variables:

$$h = w \bullet h_r + (1-w) \bullet h_i \tag{8}$$

where h_r is the hamming distance of real variables, h_i is hamming distance of integer variables, and w is the weight factor: [0, 1].

The value of the weight factor depends upon the degree of importance of each variable. In this work a weight factor of 0.5 was used. L can be calculated by the same procedure as hamming distance using a weight factor as follows:

$$L = w \bullet L_r + (1-w) \bullet L_i \tag{9}$$

where L_r and L_i are maximum Euclidean distances of real and integer variables, respectively. The procedure of survival selection for real number representation is:

1. N chromosomes are selected from M(t).
2. Eliminate duplicate structures in M(t).
3. The chromosome with the best fitness value always survives intact to the next generation.
4. Other chromosomes are selected by CPSS method [by Eq. (5)].
5. A random number is generated for all other chromosomes. If the generated random number is smaller than the p_s calculated for the structure, then the chromosome is selected; otherwise it is deleted. The selection process is performed in the order of fitness values, where M(t) is the combined population of parents and offspring.
6. If the number of the current population is smaller than N after the procedure of (2) or the CPSS method in (4-5), new structures are introduced by the insufficient number. These new structures are generated using random numbers.

RESULTS

The performance of the proposed algorithm and the based algorithm are compared separately. Both algorithms are also used to solve the test problems. The proposed algorithm consists of the new method described in section 3: transformation-based mutation, repair-

Table 1. Genetic parameters and results of mutation test

Run	1	2
Crossover	Arithmetical crossover	Arithmetical crossover
Mutation	Mutation with local search	Transform based mutation
Handling constraint strategy	Penalty	Penalty
Selection	Elitist	Elitist
Population size	15	15
Probability of crossover, Pc	0.3	0.3
Probability of Mutation, Pm	0.1	0.1
Maximum generation	1000	1000
The best solution	0.99956	0.99994

ing strategy, and CPSS. The based algorithm still uses a similar genetic operator as in Yokota's work.

1. Performance of Transformation-based Mutation and Mutation with Local Search

To compare the performance of these operators, the based algorithm was performed two times. First, the original based algorithm was used; after that, mutation with local search was changed to transformation-based mutation. The other details and genetic parameters are shown in Table 1.

Test problem 1 Class: MINLP

$$R(m,r)=\prod_{j=1}^{t}\{1-(1-r_j)^{m_j}\}$$

$$\text{s.t } g_1(m)=\sum_{j=1}^{4}v_j\cdot m_j^2\le v_Q$$

$$g_2(m,r)=\sum_{j=1}^{4}C(r_j)\cdot\left(m_j+\exp\left(\frac{m_j}{4}\right)\right)\le c_Q$$

$$g_3(m)=\sum_{j=1}^{4}w_j\cdot m_j\cdot\exp\left(\frac{m_j}{4}\right)\le w_Q$$

$$C(r_j)=\alpha_j\cdot\left(\frac{-T}{\ln(r_j)}\right)^{\beta_j}\quad j=1,\ldots,4$$

$1\le m_j\le 10$; int j=1, 2, … t

$0.5\le r_j\le 1-10^{-6}$; real number

where

c_Q	400
w_Q	500
v_Q	250
T	1000

J	α_j*10^5	b_j	v_j	w_j
1	1	1.5	1	6
2	2.3	1.5	2	6
3	0.3	1.5	3	8
4	2.3	1.5	2	7

The convergence of the two methods was plotted and is shown in Fig. 2. The plot shows that the proposed mutation operator allows the genetic algorithms to converge faster and achieve a better final solution than the based algorithm. (The average final solutions from 10 runs are 0.999389 and 0.999937 for the search-based algorithm and the transformation-based mutation method, respectively.)

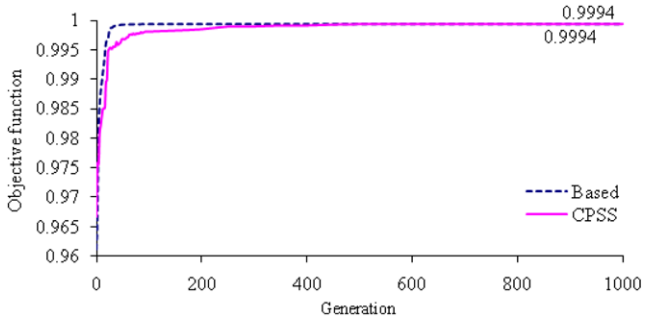


Fig. 2. Convergence between search-based mutation and transformation-based mutation.

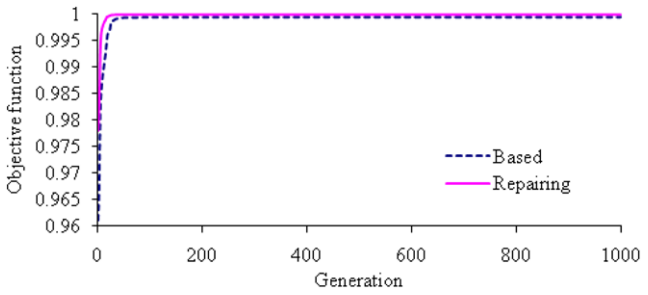


Fig. 3. Convergence between penalty method and repairing method.

2. Performance of Repairing and Penalty Strategy

This test was performed to investigate the performance of methods for solving constraint problems as well as repairing and penalty strategy. The genetic operators and genetic parameters used in this test are shown in Table 2.

The convergence of both methods is plotted in Fig. 3. The proposed method improves the genetic algorithms and provides a better final solution than the based algorithm (0.997832 and 0.999928, respectively).

The performance of repairing strategy obviously affects the problems that give optimum solutions near the constraint boundary.

Consider the penalty strategy: an unsuitable penalty function may cause the penalty to be "too low" or "too high." In the case of a penalty which is too low, GAs will diverge from the feasible region and give an infeasible final solution. However, for a penalty which is too high, GAs will act as a rejection strategy. All infeasible chro-

Table 2. Genetic parameters and results of handling constraint strategy test

Run	1	2
Crossover	Arithmetical crossover	Arithmetical crossover
Mutation	Mutation with local search	Mutation with local search
Handling constraint strategy	Penalty	Repairing
Selection	Elitist	Elitist
Population size	15	15
Probability of crossover (Pc)	0.3	0.3
Probability of Mutation (Pm)	0.1	0.1
Maximum generation	1000	1000
The best solution	0.99956	0.99995

Table 3. Genetic parameters and results of selection strategy test

Run	1	2
Crossover	Arithmetical crossover	Arithmetical crossover
Mutation	Mutation with local search	Mutation with local search
Handling constraint strategy	Penalty	Penalty
Selection	Elitist	CPSS
Population size	15	15
Probability of crossover (Pc)	0.3	0.3
Probability of Mutation (Pm)	0.1	0.1
Maximum generation	1000	1000
The best solution	0.997832	0.997009

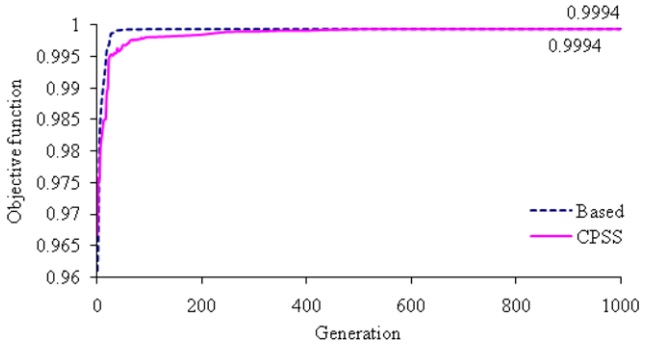


Fig. 4. Convergence between elitist selection and CPSS.

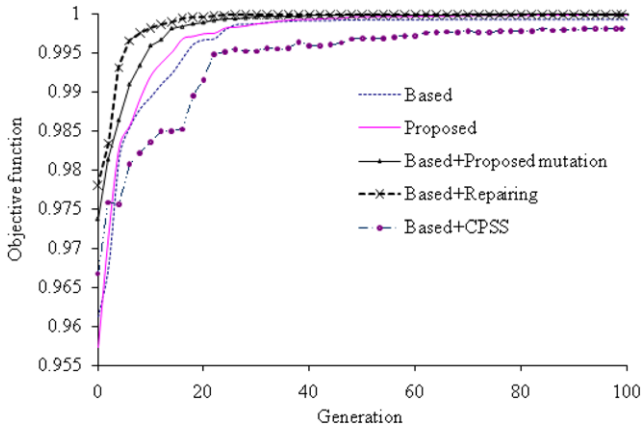


Fig. 5. Comparison of convergence of each modified algorithm.

mosomes are eliminated from the population, and therefore it becomes a simple GA.

3. Performance of CPSS and Elitist Selection

To study the performance of the proposed selection strategy, all parameters of GA were set in the same manner except selection strategy. The first run used elitist selection while the second used the CPSS method. The same test problems were still used. The genetic parameters in this experiment are shown in Table 3.

From Fig. 4, the results between CPSS and elitist selection show a slight difference. The convergence of CPSS seems to be slower than the elitist selection in this experiment. However, the result which should be considered is the diversity of each chromosome. CPSS functions to control the diversity of populations to prevent premature convergence by standard deviation of each variable in the population from the last generation.

Table 4 shows that the standard deviation of chromosomes obtained by the CPSS method is dramatically larger than the elitist selection. This means that the diversity of chromosomes is controlled by CPSS. However, it did not search widely enough to converge to the optimum solution, as shown in Fig. 5.

4. Performance of Proposed Algorithm and Based Algorithm

The performance of the proposed genetic operator was investigated in previous sections. In this section, the proposed algorithm which used all proposed operators was tested and compared to the performance of the based algorithm. The genetic parameters and the operator used in both algorithms are summarized in Table 5.

In considering the convergence of each algorithm that was modified with a transformation-based algorithm, a repairing method, or CPSS, Fig. 5 shows that a based algorithm modified with a repairing method gives the best result and the fastest convergence. Although the proposed algorithm is slower than a based algorithm with a repairing method, it ensures the prevention of premature convergence and has a convergence rate higher than the algorithm which was modified with CPSS.

5. Additional Test Problems

Various problems were selected to test and confirm the performance. These problems are of mixed-integer nonlinear programming type. All problems are maximization problems; their mathematical models are shown in the Appendix. They were solved using both the based algorithm and the proposed algorithm. The results are compared in Table 6.

According to this table, the based algorithm cannot find the optimal solution and searches away from the feasible region. The results

Table 4. Standard deviation of variables in the last generation

	x1	x2	x3	x4	x5	x6	x7	x8
CPSS	0.1954	0.3846	0.4039	0.4726	0.0445	0.0480	0.0396	0.0367
Elitist	0	0	0	0	0.0005	2.17E-05	2.88E-05	9.92E-05

Table 5. Genetic parameters and results of proposed algorithm and based algorithm

Run	1	2
Crossover	Arithmetical crossover	Arithmetical crossover
Mutation	Mutation with local search	Transform based mutation
Handling constraint strategy	Penalty	Repairing
Selection	Elitist	CPSS
Population size	15	15
Probability of crossover (Pc)	0.3	0.3
Probability of Mutation (Pm)	0.1	0.1
Maximum generation	1000	1000
The best solution	0.997832	0.999932

Table 6. Results of various other problems

Problem	Based algorithm		Proposed algorithm	
	Result	Status	Result	Status
2 (MINLP)	−2.29185	Infeasible	-2.20098	Feasible
3 (MINLP)	−2.31847	Infeasible	-2.12445	Feasible
4 (MINLP)	−2.09270	Infeasible	-2.00000	Feasible
5 (MINLP)	−2.37520	Infeasible	-2.12447	Feasible
6 (MINLP)	−0.33378	Infeasible	1.45476	Feasible
7 (LP)	263190.54	Infeasible	286758.54	Feasible
8 (NLP)	38.83	Feasible	38.85	Feasible

of the based algorithm were the last feasible solutions, except in the case of problem 8. This problem arose from the unsuitable penalty function model. Too low a penalty multiplier obstructs the algorithm from forcing a population search in the feasible region. This problem was corrected by replacing the penalty method with the repairing method. With this approach, target chromosomes are forced into the feasible region; therefore, all solutions can be achieved. Additionally, the proposed algorithm can also propose better solutions, as in problem 8.

DISCUSSION

Up to the present time, many researchers have focused on the modified genetic algorithms, especially the based algorithm. This study shows the advantages and drawbacks of that based algorithm. Hence, many parts of the based algorithm can be improved with newly created methods. These proposed methods are added to the based algorithm and generate the new approach. The genetic operators that were improved from the based algorithm are:

1. Mutation Operator

As the conventional method for performing uniform mutation for real variables and local search for integer variables, the program required a great deal of time for local search where there was a wide range domain of integer variables. The proposed method no longer uses local search, but the integer variable is treated as a real number and is subsequently rounded to an integer number. It gives a better solution than the previous method.

2. Handling Constraint Strategy

Based on a numerical method, the secant method incorporated with bisection was modified to repair the infeasible chromosome to the new one on the constraint boundary. This method can solve the constraint problem instead of penalty strategy in order to fix the problem of "too low" or "too high" penalty multipliers.

Because this repairing method considers the infeasible chromosome as a slack or surplus variable, the performance of the repairing method can be easily observed when the optimal solution is located near the constraint boundary.

3. Selection Operator

CPSS was used as a selection method to prevent premature convergence or getting stuck in local optimum. It controls the diversity of population by selecting the chromosome for the next generation based on chromosome structure, not on fitness value.

CONCLUSION

In this work, an improved genetic algorithm is proposed to solve a mixed-integer nonlinear programming problem. The proposed GA was developed in MATLAB programming language, and involves a detailed algorithm that improves a repairing method by incorporating secant and bisection methods. In addition, CPSS was modified for both integer and real numbers in order to prevent premature convergence. The proposed algorithm using these new methods was tested with various problems; the results show solutions superior to those obtained by previous methods.

ACKNOWLEDGEMENTS

This study was supported by the National Center of Excellence for Petroleum, Petrochemicals and Advanced Materials and Graduate School, Kasetsart University.

NOMENCLATURE

a : random number from uniform probability distribution
CDSS : cross-generational deterministic survival selection
CPSS : cross-generational probabilistic survival selection method
DCGA : diversity control oriented genetic algorithm
f : the set of equality constraints representing the model
g : the set of inequality constraints presented in the process
GA : genetic algorithms
h : hamming distance
L : the length of the string representing the chromosome
m : shape factor of CPSS

MGA : modified genetic algorithms
MINLP : mixed-integer nonlinear programming
M (t) : the set of current chromosomes combined with offspring at time t
P_s : survival probability
pop_size : population size
S(x) : the distance between chromosome and constraint
V : individual chromosome
x : true value of measured variable
α : shape factor of CPSS

Superscripts
L : lower bound of variable
U : upper bound of variable

Subscripts
i, j, t : variable index

REFERENCES

1. M. Mitchell, *An introduction to genetic algorithms*, MIT Press/Bradford, Cambridge, MA, 224 (1996).
2. T. Murata, H. Ishibuchi and H. Tanaka, *Comput. Ind. Eng.*, **30**(4), 1061 (1996).
3. N. D. Ramírez-Beltrán and K. Aguilar-Ruggiero, *Comput. Ind. Eng.*, **33**(1-2), 43 (1997).
4. T. Yokota, M. Gen and Y. Li, *Comput. Ind. Eng.*, **30**(4), 905 (1996).
5. H. Shimodaira, DCGA: a diversity control oriented genetic algorithm, *Proceedings of the IEEE International Conference on Genetic Algorithms in Engineering Systems*, 444 (1997).
6. C. A. Floudas, *Nonlinear and mixed-integer optimization: Fundamentals and applications*, Oxford University Press, New York, 478 (1995).
7. Z. Michalewicz, *Genetic algorithms+data structures=evolution programs*, Springer-Verlag, New York, 156 (1996).
8. A. K. Dhingra, *IEEE Trans. Reliab.*, **41**(4), 576 (1992).
9. S. C. Chapra, *Numerical methods for engineers*, McGraw-Hill, Singapore, 812 (1990).
10. T. Wasanapradit, *Solving nonlinear mixed integer programming using genetic algorithms*, Master's thesis, Chemical Engineering Practice School, King Mongkut's University of Technology Thonburi, 90 (2000).

APPENDIX TEST PROBLEM

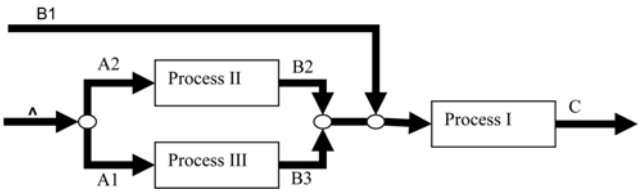
- Test problem 2 [1]
  Class: MINLP
  Maximize: $y_1+y_2+y_3+5x^2$
  Subject to: $3x-y_1-y_2\le 0$
  $-x+0.1y_2+0.25y_3\le 0$
  $y_1+y_2+y_3\ge 2$
  $y_1+y_2+2(y_3-1)\ge 0$
  $0.2\le x\le 1$
  $y_1, y_2, y_3=0.1$

- Test problem 3 [1]
  Class: MINLP
  Maximize: $y-2x_1+\ln(0.5x_1)$
  Subject to: $-x_1-\ln(0.5x_1)+y\le 0$
  $0.5\le x\le 1.4$
  $y=0, 1$

- Test problem 4 [1]
  Class: MINLP
  Maximize: $-(2x+y)$
  Subject to: $1.25-x^2-y\le 0$
  $x+y\le 1.6$
  $x\ge 0$
  $y=0, 1$

- Test problem 5 [1]
  Class: MINLP
  Maximize: $y-2x_1-x_2$
  Subject to: $x_1-2e^{-x_2}=0$
  $-x_1+x_2+y\le 0$
  $0.5\le x1\le 1.4$
  $y=0, 1$

- Test problem 6 [1]
  Class: MINLP



obj=revenue−cost

Investment cost:

process I: $3.5y_1+2C$
process II: $1.0y_2+1.0B_2$
process III: $1.5y_3+1.2B_3$

Revenue:

Re venue$=13C-1.8(A_2+A_3)-7B_1$

Material balance:

process I: $C-0.9(B_1+B_2+B_3)=0$
process II: $B_2-\ln(1+A_2)=0$
process I: $B_3-1.2\ln(1+A_3)=0$

Capacity of process:

process I: $C\le 1$

process II: $B_2\le\frac{1}{0.9}$

process I: $B_3\le\frac{1}{0.9}$

Existence variable:

$y_1, y_2, y_3=0, 1$

Processes II and III cannot operate simultaneously.

From the definition of the problem, a mathematical model can be set as below:

Maximize: $11C-7B_1-B_2-1.2B_3-1.8A_2-1.8A_3-3.5y_1-y_2-1.5y_3$
Subject to: $B_2-y_2\ln(1+A_2)=0$
$B_3-1.2y_3\ln(1+A_3)=0$
$C-0.9y_1(B_1+B_2+B_3)=0$
$C-y_1\leq 0$
$B_2-\frac{1}{0.9}y_2\leq 0$
$B_3-\frac{1}{0.9}y_3\leq 0$
$y_2+y_3\leq 1$
$C, B_1, B_2, B_3, A_2, A_3\geq 0$
$y_1, y_2, y_3=0, 1$

- Test problem 7 [4]
  Class: LP

Maximize: 8.1x1+10.8x2
Subject to: 0.8x1+0.44x2≤24000
0.05x1+0.10x2≤2000
0.10x1+0.36x2≤6000

- Test problem 8 [4]
  Class: NLP

Maximize: $x_1\sin(4\pi x_1)+x_2\sin(20\pi x_2)+21.5;$
$8\leq x_1\leq 12.1$
$5\leq x_2\leq 5.8$