Ind. Eng. Chem. Res., 35 (10), 3560 -3566, 1996. ie9506633 S0888-5885(95)00663-4

Optimal Design of Multiproduct Batch Chemical Process Using Genetic Algorithms

Chunfeng Wang, Hongyin Quan, and Xien Xu*

Chemical Engineering Department, Tianjin University, Tianjin 300072, People's Republic of China

Received for review November 3, 1995

Revised manuscript received June 14, 1996

Accepted June 14, 1996⊗

Abstract:

For the first time, genetic algorithms (GAs) are applied to the optimal design of the multiproduct batch chemical process (MBCP) successfully. An effective multiparameter crossed binary coding method is developed. The GAs have the advantages of no special demand for initial values of decision variables, lower computer storage, and less CPU time for computation. Better results are obtained in comparison with the results of mathematical programming (MP) and simulated annealing (SA). The effectiveness of GAs in solving the complex design problem of batch chemical process is demonstrated.

Introduction

Batch processes are widely used in the chemical process industry and are of increasing industrial importance due to a great emphasis on low-volume, high-value-added chemicals and the need for flexibility in a market-driven environment. In the optimal design of a multiproduct batch chemical process (MBCP), the production requirement of each product and the total production time available for all products are specified. The number and size of parallel equipment units in each stage as well as the location and size of intermediate storage are to be determined in order to minimize the investment.

The common approach used by previous research in solving the design problem of batch plant has been to formulate it as an MINLP problem and then employ optimization techniques to solve it. Mathematical programming (MP) (Grossmann and Sargent, 1979; Knopf et al., 1982; Takamatsu et al., 1982), and heuristics (Yeh et al., 1987; Modi and Karimi, 1989; Xu et al., 1993) are commonly used. Because of the NP-hard nature of the design problem of batch plant, unbearable long computational time will be induced by the use of MP when the design problem is somewhat complicated. Severe initial values for the optimization variables are also necessary. Moreover, with the increasing size of the design problem, MP will be futile. Heuristics needs less computational time, and severe initial values for optimization variables are not necessary, but it may end up with a local optimum due to its greedy nature. Also, it is not a general method with respect to the fact that special heuristic rules will be needed for a special problem. Patel et al. (1991) and Tricoire and Malone (1991) applied simulated annealing (SA) to solve the design of MBCP. SA performs effectively and gives a solution within 0.5% of the global optimum. However, SA has the disadvantage of long searching time and so needs more CPU time than heuristics. In order to speed up the convergence of SA, Wang et al. (1994) combined SA with heuristics to solve the design problem of MBCP, and satisfactory results were obtained.

To solve the proposed problem more effectively, we apply GAs, an intelligent problem-solving method that has demonstrated its effectiveness in solving combinatorial optimization problem and the "combinatorial explosion" associate with it in many areas to achieve this goal. Some modifications to traditional GAs, mainly an effective multiparameter crossed binary coding method, is developed, and satisfactory results are obtained.

The rest of this paper is organized as follows. Section 2 presents the mathematical model of MBCP. Section 3 gives a detailed account of the genetic algorithms as well as the way of implementation. To demonstrate the effectiveness of GAs in solving the proposed problem, section 4 presents two problems adopted from Patel et al. (1991) and their computation results using GAs. Comparisons with MP and SA are also given. Finally, section 5 presents the summary and conclusions.

Mathematical Model of MBCP

The optimal design for multiproduct batch processes can all be induced to a MINLP model. This paper employs Modi's model modified by Xu et al. (1993). It has the following assumptions:

(1) The processes operate in the way of overlay;

(2) The devices in the same production line cannot be reused by the same product;

(3) The long campaign and the single product campaign are considered;

(4) The type and size of parallel items in- or out-of-phase are the same in one batch stage;

(5) All intermediate tanks are finite ones;

(6) The operation between stages can be of zero wait or no intermediate tank when there is no storage;

(7) There is no limitation for utility;

(8) The cleaning time of the batch item can be neglected or included in processing time;

(9) The size of the devices can change continuously in its own range.

Assuming that there are J batch stages, K semicotinuous stages, and I products to be manufactured; that there are m_{oj} out-of-phase groups of parallel units in each batch stage in which there are m_{pj} in-phase parallel units of which the sizes are all V_j; that there are R_k parallel units in-phase in each semicontinuous stage, the operating rates of which are all R_k; that there are $S - 1$ intermediate tanks that divide the whole process into S subsystems; and that let

$$J_s = (j|\,\text{batch stage belonging to subprocess } s),\ s = 1, S$$

$$T_s = (t|\,\text{semicontinuous substrain belonging to subprocess } s),\ s = 1, S$$

$$U_t = (k|\,\text{semicontinuous stage } k \text{ belonging to semicontinuous substrain } t),\ t = 1, T$$

and using the equipment investment as a criterion of optimization, which is expressed as a power function of characteristic dimension of equipment, the following mathematical model could be obtained:

Missing Equation

subject to the following:

(1) Dimension constraints: every equipment alters in its allowable range:

$$V_j^{min} \le V_j \le V_j^{max} j = 1, J \tag{2}$$

$$R_k^{min} \le R_k \le R_k^{max} k = 1, K \tag{3}$$

(2) Time constraints: the summation of available production time for all products is not more than net total time for production:

$$H \ge \sum_{i=1}^{I} H_i = \sum_{i=1}^{I} \frac{Q_i}{P_i} \tag{4}$$

where all the following are true: (a) the productivity for product i:

$$p_i = \frac{B_{is}}{T_{is}^L} i = 1, I;\ s = 1, S \tag{5}$$

(b) the limiting cycle time for product i in subprocess s:

$$T_{is}^L = \max_{j \in J_s, t \in T_s} [T_{ij}, \theta_{ij}]\, i = 1, I;\ s = 1, S \tag{6}$$

(c) the cycling time for product i in batch stage j:

$$T_{ij} = \frac{\theta_{iu} + p_{ij} + \theta_{i(u+1)}}{m_{oj}} i = 1, I;\ j = 1, J \tag{7}$$

(d) the processing time for product i in batch stage j:

$$p_{ij} = p_{ij}^o + g_{ij} \left(\frac{B_{is}}{m_{pi}} \right) d_{ij} i = 1, I;\ j = 1, J;\ j \in J_s \tag{8}$$

(e) the operating time for product i in substrain t:

$$\theta_{it} = \max_{k \in U_t} \left[\frac{B_{is} D_{ik}}{R_k n_k} \right] i = 1, I;\ t = 1, T;\ t \in T_s \tag{9}$$

(f) the batch size for product i in subprocess s:

$$B_{is} = \min_{j \in J_s} \left(\frac{m_{pi}\nu_i}{S_{ij}}\right) i = 1, I; t = 1, T; t \in T_s \tag{10}$$

(3) The constraints of product quantity: the same product in different subprocess posses the same productivity.

(4) The dimension of intermediate storage is the maximum value of what is needed by all products:

$$V_s^* = \max_i \left[P_i S_{is}^* \left(T^{L}_{is} - \theta_{iu} + T^{L}_{i(s+1)} - \theta_{i(u+1)}\right)\right] i = 1, I; s = 1, S - 1 \tag{11}$$

Using the mathematical model to optimize a design for a given product demand, the size and number for each kind of equipment must be calculated to minimize the equipment investment.

Genetic Algorithms

Genetic Algorithms. Genetic algorithms (GAs), proposed by Holland (1975) and his colleagues, are search algorithms based on the mechanics of natural selection and natural genetics. They combine the survival of the fittest among string structures with a structured yet randomized information exchange to form search algorithms with some of the innovative flair of human search. In every generation, a new set of artificial creation (strings) is created using bits and pieces of the fittest of the old; an occasional new point is tried for good measure. While randomized, GAs are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance.

The canonical steps of the GAs can be described as follows:

(1) The problem to be addressed is defined and captured in an objective function that indicated the fitness of any potential solution.

(2) A population of candidate solutions is initialized subject to certain constraints. Typically, each trial solution is coded as a vector **x**, termed a chromosome, with elements being described as solutions represented by binary strings. The desired degree of precision would indicate the appropriate length of the binary coding.

(3) Each chromosome, x_I, i = 1, ..., P, in the population is decoded into a form appropriate for evaluation and is then assigned a fitness score, $\mu(x_I)$ according to the objective.

(4) Selection in genetic algorithms is often accomplished via differential reproduction according to fitness. In a typical approach, each chromosome is assigned a probability of reproduction, p_I, i = 1, ..., P, so that its likelihood of being selected is proportional to its fitness relative to the other chromosomes in the population. If the fitness of each chromosome is a strictly positive number to be maximized, this is often accomplished using roulette wheel selection (Goldberg, 1989). Successive trials are conducted in which a chromosome is selected, until all available positions are filled. Those chromosomes with above-average fitness will tend to generate more copies than those with below-

average fitness.

(5) According to the assigned probabilities of reproduction, p_i, $i = 1, \ldots, P$, a new population of chromosomes is generated by probabilistically selecting strings from the current population. The selected chromosomes generate "offspring" via the use of specific genetic operators, such as crossover and bit mutation. Crossover is applied to two chromosomes (parents) and creates two new chromosomes (offspring) by selecting a random position along the coding and splicing the section that appears before the selected position in the first string with the section that appears after the selected position in the second string and vice versa (see Figure 1). Bit mutation simply offers the chance to flip each bit in the coding of a new solution.

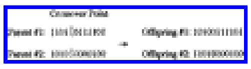

Figure 1 One-point crossover operator.

(6) The process is halted if a suitable solution has been found or if the available computing time has expired; otherwise, the process proceeds to step 3 where the new chromosomes are scored, and the cycle is repeated.

Implementation and Empirical Tuning Methods. (1) Mapping Objective Functions to Fitness Form. In many problems, the objective is more naturally stated as the minimization of some cost function $g(x)$ rather than the maximization of some utility or profit function $u(x)$. Even if the problem is naturally stated in maximization form, this alone does not guarantee that the utility function will be nonnegative for all x as we require in fitness function (a fitness function must be a nonnegative figure of merit; Goldberg, 1989). As a result, it is often necessary to map the underlying natural objective function to a fitness function form through one or more mappings.

The duality of cost minimization and profit maximization is well known. In normal operations research work, to transform a minimization problem to a maximization problem we simply multiply the cost function by a minus one. In genetic algorithm work, this operation alone is insufficient because the measure thus obtained is not guaranteed to be nonnegative in all instances. With GAs, the following cost-to-fitness transformation is commonly used:

$$f(x) = C_{\max} - g(x) \text{ when } g(x) < C_{\max}$$

$$= 0 \text{otherwise}$$

$C_{\max}$ may be taken as the largest g value observed thus far. For the problem of optimal design of MBCP in this paper, we take this transformation form.

(2) Fitness Scaling. In order to achieve the best results of GAs, it is necessary to regulate the level of competition among members of the population. This is precisely what we do when we perform fitness scaling.

Regulation of the number of copies is especially important in small population genetic algorithms. At the start of GA runs, it is common to have a few extraordinary individuals in a population of mediocre colleagues. If left to the normal selection rule ($pselect_i = f_i/\sum f$), the extraordinary individuals would take over a significant proportion of the finite population in a single generation, and this is undesirable, a leading cause of premature convergence. Later on during a run, we have a very different problem. Late in a run, there may still be significant diversity within the population; however, the population average fitness may be close to the population best fitness. If this situation is left alone, average members and best members get nearly the same number of copies in future generations, and the survival of the fittest necessary for improvement becomes a random walk among the mediocre. In both cases, at the beginning of the run and as the run matures, fitness scaling can help.

There are several methods of fitness scaling, i.e., ranking method (Baker, 1985), and linear scaling (Goldberg, 1989). We tried both of them and find that the linear scaling is simple and efficient to our problem.

Let us define the raw fitness f and the scaled fitness f'. Linear scaling requires a linear relationship between f' and f as the following: $f' = af + b$. It is required that $f'_{av} = f_{av}$ and $f'_{max} = Cf_{av}$ in which C is the number of expected copies desired for the best population member. In the problem, we tried different values of C ranging from 1.1 to 2.5. To our experiences, C ranging from 1.2 to 2 has been used successfully. The best results were obtained with C = 1.3 for example 1 and C = 1.5 for example 2.

From the equation above, we can give

$$a = (C - 1)f_{av}/f_{max} - f_{av}$$

$$b = (f_{max} - C)/f_{max} - f_{av}$$

In this way, simple scaling helps prevent the early domination of extraordinary individuals, while later on it encourages a healthy competition among near equals. It improve the performance of GAs in practice.

(3) Constraints. We deal with the dimension constraints by coding as eq 4 and deal with time constraints this way: a genetic algorithm generates a sequence of parameters to be tested using the system model, objective function, and the constraints. We simply run the model, evaluate the objective function, and check to see if any constraints are violated. If not, the parameter set is assigned the fitness value corresponding to the objective function evaluation. If constraints are violated, the solution is infeasible and thus has no fitness.

(4) Codings. When GAs manage a practical problem, the parameters of the problem are always coded into bit strings. In fact, coding designs for a special problem is the key to using GAs effectively. There are two basic principles for designing a GAs coding (Goldberg, 1989): (1) The user should select a coding so that short, low-order schemata are relevant to the underlying problem and relatively unrelated to schemata over other fixed positions. (2) The user should select the smallest alphabet that permits a natural expression of the problem.

Based on the characteristic and structure of MBCP, instead of choosing the concatenated, multiparamerted, mapped, fixed-point coding, a multiparameter crossed binary coding is designed according to the two principles above. The coding method of a MCBC is as follows: following the order-the numbers of out-of-phase groups in each bath stages, in-phase parallel units in each of the groups, semicontinuous parallel units in each semicontinuous stages, the size of batch stages, semicontinuous stages, each parameter is encoded independently in usual binary codings (local strings). Then we place the highest bit of each local string at the site from 1st to nth in MCBC chromosome and place the second highest bit of each local string at the site from $(n + 1)$th to $2n$th, and so on. Then we can obtain a MCBC chromosome (Figure 2).

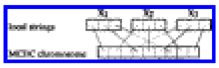

Figure 2 Multiparameter crossed binary codings.

The reason for using crossed coding can be analyzed in theory as follows: (1) Because of the strong relationship among the parameters, the highest bit in each local string in binary codings determines the basic structure (relation) among every parameter, and the second highest bit in each local string determines finer structure among every parameter, and so on for the third, the forth, etc. (2) The schema defining length under crossed coding (n) is shorter than the length under concatenated, mapped, fixed-point coding ($nL - L + 1$). According to the schema theorem: short schemata cannot be disturbed with high frequency, the schema under crossed coding has a greater chance to be reproduced in the next generation. Due to its combining the characteristics of function optimization with schema theorem and successful binary alphabet table, crossed coding demonstrates greater effectiveness than the ordinary coding method in our implementation.

Local string formation is achieved this way: for a parameter $x \in [x_{\min}, x_{\max}]$ that needs to be coded, transform it to a binary coding $X \in [0, 2^L]$ first (appropriate length L is determined by the desired degree of precision) and then map it to the specified interval $[x_{\min}, x_{\max}]$. In this way, the precision of this mapped coding may be calculated as $\delta = (x_{\max} - x_{\min})/(2^L - 1)$. In fact, this means that the interval from $x_{\min}$ to $x_{\max}$ is divided into $2^L - 1$ parts (because the biggest binary string that has a length of L equals the decimal number $2^0 + 2^1 + 2^2 + ... + 2^{L-1} = 2^L - 1$). Then, we can obtain $x = x_{\min} + \delta X$, and a local string for parameter x with a length of L is obtained.

To illustrate the coding scheme more clearly, we also want to give a simple example. For the minimization problem: min $z = f(x, y)$ in which $x \in [100, 500]$ and $y \in [500, 1000]$, if we adopt a string length of 5 for each local string and X: 10110, Y: 01101 is an initial solution, we will get the chromosome 1 0 0 1 1 1 0 0 0 1 (Figure 3) and obtain

$$x = x_{\min} + \delta_x X = 100 + [(500 - 100)/(2^5 - 1)](2^4 \times 1 + 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0) = 10$$

$$y = y_{\min} + \delta_y Y = 500 + [(1000 - 500)/(2^5 - 1)](2^4 \times 0 + 2^3 \times 1 + 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 1) = 5$$

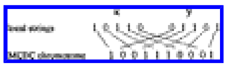

Figure 3 Multiparameter crossed binary codings.

(5) Reproduction. The reproduction operator may be implemented in algorithmic form in a number of ways. In this paper, we take the easiest method- Roulette wheel (Goldberg, 1989).

(6) Crossover. Crossover operator can take various forms, i.e., one-point crossover, multi-point crossover (Frantz, 1972). It is commonly believed that multi-point crossover has better performance. The number of crossover points in a multi-points crossover operator is determined by the string structure. In this paper, a two-point crossover operator is adopted, and the crossover length is no more than one-third of the total string length. The crossover rate plays a key role in GA implementation. Different values for crossover rate ranging from 0.4 to 1.0 were tried, and the results demonstrate that the values ranging from 0.6 to 0.95 are successful for examples 1 and 2, while 0.62 and 0.64 are the best choices for examples 1 and 2, respectively.

(7) Mutation. The mutation rate is also an important factor in GA implementation. We tried different values varying from 0.001 to 0.2. Small mutation rates prevent new solution spaces from being explored, while large mutation rate results in the search process being at random. According to our experiences, the values ranging from 0.001 to 0.1 are found to be successful for a problem as complicated as examples 1 and 2. The best results were obtained when we adopted a mutation rate of 0.001.

(8) Population-Related Factors. (a) Population Size. The GA performance is influenced heavily by population size. Various values ranging from 20 to 200 population size were tested. Small populations run the risk of seriously undercovering the solution space, which results in local optimum, while large populations incur severe computational penalties. According to our experience, a population size range from 50 to 150 is enough for a problem as complicated as examples 1 and 2 in this paper. The size of 100 for example 1 and 150 for example 2 were found to be appropriate.

(b) Initial Population. It is demonstrated that a high-quality initial value obtained from another heuristic technique can help GA find better solutions rather more quickly than it can from a random start. However, there is possible disadvantage in that the chance of premature convergence may be increased. In this paper, the initial population is simply chosen by random.

(9) Termination Criteria. It should be pointed out that there are no general termination criteria for GA. Several heuristic criteria are employed in GA, i.e., computing time (number of generations), no improvement for search process, or comparing the fitness of the best-so-far solution with average fitness of all the solutions. All types of termination criteria above were tried; the criteria of computing time is proven to be simple and efficient in our problem. In our experience, 5-15 generations simulation is enough for a problem as complicated as examples 1 and 2 in this paper. The best results were obtained when the number of generations were taken as 10 for example 1 and 15 for example 2.

Examples and Analysis

Two examples (Patel et al., 1991) are given here to demonstrate the effectiveness of GAs. The data for examples 1 and 2 are presented in Tables 1 and 2, respectively. The results are presented in Tables 3 and 4. From these results, we can see that better results are obtained in comparison with MP and SA. In addition, GAs result in a faster convergence. Patel et al. (1991) pointed out that, due to its complicity, example 2 cannot be solved using any existing method other than SA. However, GAs deal with it successfully, and the computing time is less than that of SA. This demonstrates the effectiveness of GAs in solving the complicated design problem of MBCP, which is due to GAs searching from population (not a single point) and its parallel computing nature. In fact, the structure of GA algorithms is well fit for and makes it very convenient to apply to the optimal design problem of MBCP.

Now, several words about some important aspects in our implication of GAs and some problems in practice. The most important of all is the method of coding. Because of the characteristics and inner structure of the design problem of MBCP, the commonly adopted concatenated, multiparamerted, mapped, fixed-point coding is not effective in searching for the global optimum. And it is also the inner structure of the design problem of MBCP that gives us some clues for designing the above multiparamerter crossed binary coding method. As is evident from the results of application, this coding method is well fit for the proposed problem.

Another aspect that affects the effectiveness of a genetic procedure considerably is crossover. Corresponding to the proposed coding method, we adopted a two-point crossover method. It is commonly believed that multi-point crossover is more effective than the traditional one-point crossover method. However, we find that it is not the case that the more points to crossover, the better. It is also important to note that the selection of crossover points as well as the way to carry out the crossover should take in account the bit-string structure, as is the case in our implication.

Despite the demonstrated advantages of GAs algorithms, the feeling persists that there is much to learn about effectively implementing a genetic algorithm. One problem in practice is the premature loss of diversity in the population, which results in premature convergence. Because premature convergence is so often the case in the implementation of GAs according to our calculation experience, something has to be done to prevent it. As stated above, fitness scaling is exactly what we do to achieve this goal. Our experience makes it clear that fitness scaling can solve the premature problem effectively and conveniently, although there might be some other methods that are applicable to this problem.

At first glance, it would seem that premature convergence is best treated by an increase in the mutation rate. As De Jong (1975) pointed out, however, higher mutation rates will disrupt the proliferation of high-performance schemata as well as poor ones. Our experience validated this.

Conclusions

For the first time, GAs are applied to the optimal design problem of MBCP. Satisfactory results are obtained. An effective multiparameter crossed binary coding method is developed. GAs are well fit for the proposed optimization problem and demonstrate the following advantages in application:

(1) GAs have no special demand for initial values of decision variables. The initial population of strings is chosen randomly as long as it does not violate the constraints for the problem.

(2) As is evident from the computation results, GAs yield highly satisfactory global optimum.

(3) Due to the parallel computing nature, GAs result in faster convergence in comparison with MP and SA.

(4) GAs are simple in structure and are convenient for implementation, with no more complicated mathematical calculation than such simple operators as encoding, decoding, testing constraints, and computing values of objective.

a_j = cost coefficient for bath stage j

b_k = cost coefficient for semicontinuous stage k

B_{is} = bath size for product i in subprocess s, kg

c_s = cost coefficient for intermediate storage

D_{ik} = duty factor for semicontinous stage k for product i

d_{ij} = power coefficient for processing time on stage j for product i

g_{ij} = coefficient for processing time on stage j for product i

H = horizon, h

H_i = production time of product i, h

i = index for product

I = total number of products

j = index for bath stage

J = total number of bath stages

k = index for semicontinous stage

K = total number of semicontinous stages

m_{oj} = number of out-of-phase groups in bath stage j

m_{pj} = number of in-phase parallel units in each of the out-of-phase groups in bath stage j

n_k = number of parallel unites in semicontinous stage k

P_i = productivity of product i, kg/h

p_{ij} = processing time for product i in stage j, h

p_{ij}^0 = constant in processing time equation for product i in stage j

Q_i = demand for product i, kg

R_k = processing rate for semicontinous unit k, L/h

$R_k^{\max}$ = maximum feasible size of semicontinuous stage k

$R_k^{\min}$ = minimum feasible size of semicontinuous stage k

S = total number of subprocesses

S_{ij} = size factor for bath stage j for product i

S^*_{is} = size factor for storage s for product I

t = index for substrain

T = number of substrains

t_{ij} = recycling time for product i in bath stage j

t^L_{is} = limiting cycle time of product i in subprocess s

V_j = size of bath stage j, L

$V_j^{\max}$ = maximum feasible size of bath stage j, L

$V_j^{\min}$ = minimum feasible size of bath stage j, L

V^*_s = size of intermediate storage s, L

α_j = cost coefficient for bath stage j

β_k = cost coefficient for semicontinous stage k

γ_s = cost coefficient for storage s

θ_{it} = operating time of substrain t for product i

* E-mail: xexu@tju.edu.cn.

⊗ Abstract published in *Advance ACS Abstracts,* September 1, 1996.

Baker, J. E. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and Their Application*; Grefenstette, J. J., Eds.; Lawrence Erlbaum Associates: Hillsdale, NJ, 1985; p 101.

De Jong, K. A. An analysis of the behavior of a class of genetic adaptive systems. *Diss. Abstr. Int. B* 1975, *36* (10), 5140.

Frantz, D. R. Non-linearities in genetic adaptive search. *Diss. Abstr. Int. B* 1972, *33* (11), 5240.

Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*; Addison-Wesley Publishing Company Inc.: Reading, MA, 1989.

Grossmann, I. E.; Sargent, W. E. Optimal design of multipurpose chemical plants. *Ind. Eng. Chem. Process Des. Dev.* 1979, *18*, 343. [ChemPort]

Holland, J. H. *Adaptation in Natural and Artificial Systems*; University of Michigan Press: Ann Arbor, MI, 1975.

Knopf, F. C.; Okos, M. R.; Reklaitis, G. V. Optimum design of batch/semicontinuous processes. *Ind. Eng. Chem. Process Des. Dev.* 1982, *21*, 79. [ChemPort]

Modi, A. K.; Karimi, I. A. Design of multiproduct bath processes with finite intermediate storage. *Comput. Chem. Eng.* 1989, *13*, 127. [ChemPort]

Patel, A. N.; Mah, R. S. H.; Karimi, I. A. Preliminary design of multiproduct non-continuous plants using simulating annealing. *Comput. Chem. Eng.* 1991, *15*, 451. [ChemPort]

Schrandolph, N. N.; Belew, R. K. Dynamic Parameter Encoding for Genetic Algorithms. *Mach. Learn.* 1992, *9*, 9.

Takamatsu, T.; Hashimoto, I.; Hasebe, S. Optimal design and operation of a bath process with intermediate storage tanks. *Ind. Eng. Chem. Process Des. Dev.* 1982, *21*, 431. [ChemPort]

Tricoire, B.; Malone, M. A new approach for the design of multiproduct bath processes. Presented at the AIChE Annual Meeting, Los Angeles, 1991.

Wang, C.; Quan, H.; Xu, X. Optimal design of multiproduct bath chemical process-mixed simulated annealing. *J. Chem. Eng.* 1996, *41*, 184 (in Chinese).

Xu, X.; Zheng, G.; Cheng, S. Optimized design of multiproduct bath chemical process-a heuristic approach. *J. Chem. Eng.* 1993, *44,* 442 (in Chinese).

Yeh, N. C.; Reklaitis, G. V. Synthesis and sizing of batch/semi-continuous processes: single product plants. *Comput. Chem. Eng.* 1987, *11*, 639. [ChemPort]

Table 1. Data for Example 1[a]

	SC1	B1	SC2	T	SC3	B2	SC4	B3	SC5	B4	SC6
a or b or c	370	250	370	278	370	250	370	250	370	250	370
α or β or γ	0.22	0.60	0.22	0.49	0.22	0.60	0.22	0.60	0.22	0.60	0.22
i = 1 S or D	1.0	8.28	1.0	1.0	1.0	9.70	1.0	2.95	1.0	6.57	1.0
P^0		1.15				9.86		5.28		1.20	
g		0.20				0.24		0.40		0.50	
d		0.40				0.33		0.30		0.20	
i = 2 S or D	1.0	5.58	1.0	1.0	1.0	8.09	1.0	3.27	1.0	6.17	1.0
P^0		5.95				7.01		7.00		1.08	
g		0.15				0.35		0.70		0.42	
d		0.40				0.33		0.30		0.20	
i = 3 S or D	1.0	2.34	1.0	1.0	1.0	10.3	1.0	5.70	1.0	5.98	1.0
P^0		3.96				6.01		5.13		0.66	
g		0.34				0.50		0.85		0.30	
d		0.40				0.33		0.30		0.20	

[a] H = 6000 h; J = 4; I = 3; Q = [437000, 324000, 258000]; $250 \leq v_i \leq 10000$, $300 \leq R_k \leq 10000$.

Table 2. Data for Example 2[a]

a or b or c	SC1	B1	SC2	SC3	B2	SC4	T	SC5	SC6	B3	SC7
α or β or γ	370	592	250	210	582	250	200	250	200	1200	600
g(i = 1, 15)	–	0	–	–	0	–	–	–	–	0	–
i = 1 S or D	1.2	1.2	1.2	1.2	1.4	1.4	1.0	1.4	1.4	1.0	1.0
P^0	–	3.0	–	–	1.0	–	–	–	–	4.0	–
i = 2 S or D	1.5	1.5	1.5	1.5	0.0	0.0	1.0	1.5	1.5	1.0	1.0
P^0	–	6.0	–	–	0.0	–	–	–	–	8.0	–
i = 3 S or D	1.1	1.1	1.1	1.1	1.2	1.2	1.0	1.2	1.2	1.0	1.0
P^0	–	2.0	–	–	2.0	–	–	–	–	4.0	–
i = 4 S or D	1.5	1.5	1.5	1.5	1.8	1.8	1.0	1.8	1.8	1.0	1.0
P^0	–	2.0	–	–	1.5	–	–	–	–	3.0	–
i = 5 S or D	1.3	1.3	1.3	1.3	3.0	3.0	1.0	3.0	3.0	1.0	1.0
P^0	–	1.0	–	–	2.0	–	–	–	–	2.5	–
i = 6 S or D	1.4	1.4	1.4	1.4	2.1	2.1	1.0	2.1	2.1	1.0	1.0
P^0	–	2.0	–	–	2.5	–	–	–	–	5.0	–

i = 7 S or D	1.2	1.2	1.2	1.2	5.2	5.2	1.0	5.2	5.2	1.0	1.0
P^0	–	1.0	–	–	0.5	–	–	–	–	7.0	–
i = 8 S or D	1.1	1.1	1.1	1.1	2.1	2.1	1.0	2.1	2.1	1.0	1.0
P^0	–	4.0	–	–	3.5	–	–	–	–	3.0	–
i = 9 S or D	1.3	1.3	1.3	1.3	1.1	1.1	1.0	1.1	1.1	1.0	1.0
P^0	–	2.0	–	–	3.0	–	–	–	–	2.0	–
i = 10 S or D	1.4	1.4	1.4	1.4	1.5	1.5	1.0	1.5	1.5	1.0	1.0
P^0	–	2.5	–	–	2.5	–	–	–	–	4.0	–
i = 11 S or D	1.5	1.5	1.5	1.5	1.7	1.7	1.0	1.7	1.7	1.0	1.0
P^0	–	3.0	–	–	2.0	–	–	–	–	4.0	–
i = 12 S or D	1.2	1.2	1.2	1.2	1.9	1.9	1.0	1.9	1.9	1.0	1.0
P^0	–	3.5	–	–	4.5	–	–	–	–	6.5	–
i = 13 S or D	1.5	1.5	1.5	1.5	3.7	3.7	1.0	3.7	3.7	1.0	1.0
P^0	–	5.0	–	–	7.0	–	–	–	–	9.0	–
i = 14 S or D	1.8	1.8	1.8	1.8	2.2	2.2	1.0	2.2	2.2	1.0	1.0
P^0	–	4.5	–	–	3.0	–	–	–	–	4.0	–
i = 15 S or D	1.5	1.5	1.5	1.5	2.7	2.7	1.0	2.7	2.8	1.0	1.0
P^0	–	3.0	–	–	2.0	–	–	–	–	6.0	–

[a] Q(m'000) = [40, 30, 10, 35, 33, 27, 25, 22, 20, 19, 15, 12, 9, 7, 5]; H = 8000; I = 15; $300 \leq V_i \leqslant 2400$; $300 \leqslant R_m \leqslant 2400$. C indicates batch stage; B indicates semicontinuous stage, T indicates intermediate storage.

Table 3. Results of Example 1[a]

MP objective function 369728		
	GAs	SA
objective function	362130	368883
V_1	6907	4290
		4290
V_2	9918	9930
	9918	9930
V_3	5724	5534
	5724	5534
V_4	9466	7627
R_1	7717, 7717	9252
R_2	2189	10000

R_3	6537	9675
R_4	9466, 9466	10000
R_5	9926	9000
R_6	5212	390
T	2946	1997

[a] Population size, 100; number of generations, 10; crossover rate, 0.62; mutation rate, 0.001.

Table 4. Results of Example 2[a]

	GAs	SA
objective function	425676	450983
V_1	1148	1590
	1148	1780
V_2	1585, 1585	2400, 896
	1585, 1585	1934, 756
V_3	1644	1897
	1644	1871
R_1	1805	2050, 1645
R_2	2061	1512
R_3	2268, 2268	1512
R_4	2362, 2362	1564, 559
R_5	2266	918, 300
R_6	1248	1185
R_7	1735	2046
T	2172	5131
CPU (min)	4[b]	10[c]

[a] Population size, 150; number of generations, 15; crossover rate, 0.64; mutation rate, 0.001. [b] On an IBM PS/VP 486 DOS6.1. [c] On a Sun Sparc station.