Constraint handling strategies in Genetic Algorithms application to optimal batch plant design

A. Ponsich, C. Azzaro-Pantel *, S. Domenech, L. Pibouleau

Laboratoire de Génie Chimique de Toulouse, 5 rue Paulin Talabot BP 1301, 31106 Toulouse Cedex 1, France

Abstract

Optimal batch plant design is a recurrent issue in Process Engineering, which can be formulated as a Mixed Integer Non-Linear Programming (MINLP) optimisation problem involving specific constraints, which can be, typically, the respect of a time horizon for the synthesis of various products. Genetic Algorithms constitute a common option for the solution of these problems, but their basic operating mode is not always well-suited to any kind of constraint treatment: if those cannot be integrated in variable encoding or accounted for through adapted genetic operators, their handling turns to be a thorny issue. The point of this study is thus to test a few constraint handling techniques on a mid-size example in order to determine which one is the best fitted, in the framework of one particular problem formulation. The investigated methods are the elimination of infeasible individuals, the use of a penalty term added in the minimized criterion, the relaxation of the discrete variables upper bounds, dominance-based tournaments and, finally, a multiobjective strategy. The numerical computations, analysed in terms of result quality and of computational time, show the superiority of elimination technique for the former criterion only when the latter one does not become a bottleneck. Besides, when the problem complexity makes the random location of feasible space too difficult, a single tournament technique proves to be the most efficient one.

Keywords: MINLP optimisation; Genetic Algorithms; Constraint handling; Batch plant design

1. Introduction

Due to the growing interest for batch operating mode, a wide range of studies deals with the batch plant design issue. Actually, the problem was already modelled under various forms for which assumptions are more or less simplistic, and generally, the objective consists in the minimization of a function representing the plant investment cost.

Kocis and Grossmann [1] proposed a simple posynomial formulation for multiproduct batch plant to validate the good behaviour of the Outer Approximation algorithm implemented in *DICOPT*++, a solver available in the *GAMS* modelling environment [2]. The model involved only batch stages and was submitted to a constraint on the total production time. Modi and Karimi [3] slightly modified this MINLP model by taking into account, in addition, semi-continuous stages and intermediate finite storage, for which location is previously set. They solved small size examples (up to two products and to three batch stages and five semi-continuous stages) with heuristics. The same formulation was used again by Patel et al. [4] who treated larger size examples with Simulated Annealing and by Wang et al. [5–7] who tackled successively Genetic Algorithms, Tabu Search and an Ants Foraging Method. Then, Montagna et al. [8] studied a quite similar formulation for the design of a protein production plant, with the location of the storage tanks being an optimisation variable: the solution, obtained with *DICOPT*++ [2], proves to be a more viable alternative thanks to storage availability.

Thus, most of time, the resulting problems are MINLP problems which prove to be NP-hard. Since they are known to be efficient-solving and easy-adaptive optimisation methods, metaheuristics are widely used for the above-mentioned kind of problems. This class of optimisation techniques can be divided into two classes. The first one consists of neighbourhood methods, among which the most famous instances are Tabu Search and Simulated Annealing. A neighbourhood method typically proceeds by starting with an initial configuration and iteratively replacing the actual solution by one of its neighbours, according to an appropriate evolution of the objective function. Evolutive Algorithms constitute the second class of stochastic methods. Based on the principles of natural evolution stated by Darwin,

* Corresponding author. Tel.: +33 5 34 61 52 52; fax: +33 5 34 61 52 53.
E-mail address: Catherine.AzzaroPantel@ensiacet.fr (C. Azzaro-Pantel).

they all involve three essential factors: (i) a population of solutions of the considered problem; (ii) an adaptation evaluation technique of the individuals; (iii) an evolution process made up of operators reproducing elimination of some individuals (selection) and creation of new ones from the survivors (crossover or mutation). This latter feature brings to an increase in the average quality of the solutions. The most used techniques are Genetic Algorithms (GAs), Evolutionary Strategies and Evolutionary Programming.

The framework of this study is the application of Genetic Algorithms to a particular formulation of optimal batch plant design problems. The point is that the initial version of GAs such as devised by Holland [9] did not account for constraint handling, except when these ones could be integrated in the codings, and then handled through appropriate genetic operators. But it appears that most applications, being purely mathematical problems or applications drawn from real world, are modelled as problems are subjected to more or less severe constraints. This is more particularly the case of problems from the Process System Engineering area and, obviously, from the above-mentioned batch plant design formulations. Consequently, the Evolutive Algorithms community devoted its energies to propose constraint handling techniques, in particular for GAs. A large variety of methods, more or less specific to the nature of the studied problems, were thus developed. Some of them are tested in this work, in order to determine which one is the best fitted for the treatment of problems, in the framework of the formulation of interest.

This paper is divided into six sections. An overview on constraint handling techniques is presented in Section 2. Section 3 is devoted to the model development of the optimal batch plant design problem. Section 4 describes the Genetic Algorithm implemented throughout the study and focuses on the specific constraint handling methods that were investigated. Some typical results are then analysed in Section 5 and finally, conclusions and perspectives are given in Section 6.

2. Overview of constraint handling techniques

Various papers reviewing the various existing techniques are available, for instance in [10,11]. They usually involve a distinction between the following classes:

(i) elimination of infeasible individuals;
(ii) penalisation of the objective function;
(iii) dominance concepts;
(iv) preservation of feasibility;
(v) infeasible individuals repairing;
(vi) hybrid methods.

2.1. Elimination

This method, also called "death penalty method", consists in rejecting infeasible individuals. The most common way to implement this strategy is to set their fitness equal to 0, which prevents infeasible solutions to pass the selection step. This method is very simply implemented, but encounters problems for harshly constrained problems. In addition, its second weakness is that no information is taken from the infeasible space, which could help to guide the search towards the global optimum. Nevertheless, this technique constitutes a first valid approach when no feature allows to previously determine a specific problem-fitted method.

2.2. Penalty functions

This second class is certainly the most popular one, because of its understanding and implementation simplicity. The constrained problem is transformed into an unconstrained one by introducing the constraints in the objective function via penalty terms. Then, it is possible to formulate this penalty term according to a wide diversity of techniques. Firstly, it is of common knowledge that the penalisation will be more efficient if its expression is related to the amount of constraint violation than to the violated constraint number [12].

Let us consider the classical optimisation problem formulation:

$$\text{Min } f(x) \quad \text{s.t. } g_j(x) \leq 0, \quad j = 1, m \tag{1}$$

Then, with the unconstrained formulation including the penalty term, the new criterion F to minimize can be generally written as follows:

$$F(x) = f(x) + \sum_{j=1}^{m} R_j \max[0, g_j(x)]^{\beta} \tag{2}$$

Most of time, the penalty is expressed under a quadratic form, corresponding to β equal to 2. Equality constraints such as $h_k(x) = 0$ can be reformulated as $|h_k(x)| - e \leq 0$, where ε is a very small value. Then, the R_j factor can be expressed in many ways, showing various complexity and solution efficiency for the tackled problem. General principles can however be stated in order to guide the development of performing penalisation strategies.

The first one points out that, in most problems, the global optimum is located on the feasible space boundary. So, on the one hand, if the influence of the penalty factor is too important, the pressure exerted to push the individuals inside the feasible space will be too strong, preventing them from heading for more promising regions. Furthermore, in case of disjointed feasible spaces, a too high penalty factor can confine the population to one feasible region without allowing individuals to cross infeasible zones and head for other feasible regions (where the global optimum could be located) [13]. On the other hand, a too low penalty factor can lead to an exhaustive search in the infeasible space, visiting regions where the objective function is very low but that are strongly infeasible (see Fig. 1).

In addition, it is commonly admitted that the penalty term should be preferentially pretty low at the beginning of the search, in order to explore a wide region of the search space. At the end of the run, promising regions should be determined yet. It is then more relevant to have a high penalty term, to intensify the search on these zones by forcing the individuals to satisfy the constraints.

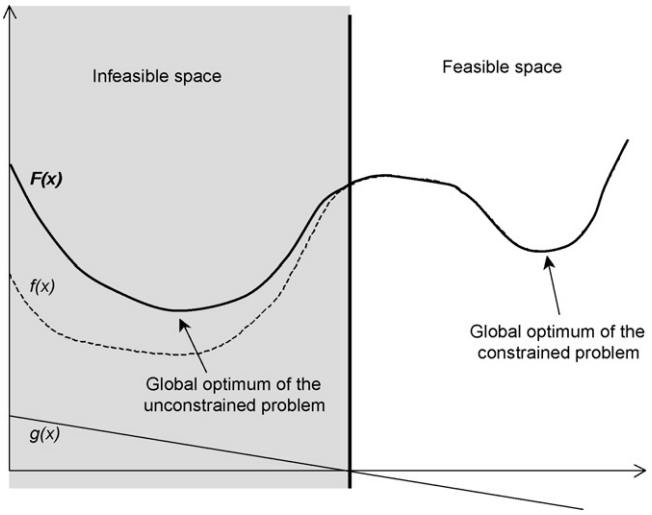

Fig. 1. Too weak penalty factor.

According to these principles, a great variety of penalisation methods were implemented, some of them are recalled in [14]. The simplest is the static penalty: a numerical value, that will not vary during the whole search, is allocated to each factor R_j. Obviously, the drawback is that as many parameters as existing constraints have to be tuned without any known methodology. Normalizing the constraints enables however to reduce the number of parameters to be chosen from m to 1.

A modified static penalty technique is proposed in [15], in which violation levels are set for each constraint. So considering l levels in a problem with m constraints, it was shown that the method needs the tuning of $m(2l+1)$ parameters (see [14] for details).

Another proposal is a dynamic penalty strategy, for which R_j is written as $(C \times t)^a$ where t is the generation number. Here, two parameters must be tuned, i.e. C and a. Common values are 0.5 and 2, respectively. Thus, this method enables to increase the pressure on infeasible solutions along the search. A similar effect can be obtained with a method presenting an analogy with Simulated Annealing:

$$R_j = \frac{1}{2t} \tag{3}$$

where τ is a decreasing temperature. It is necessary to determine initial and final temperatures, τ_i and τ_f, as well as a cooling scheme for τ. This technique has two special features. First, it involves a difference between linear and non-linear constraints. Feasibility as regard with the former is maintained by specific operators, so that only the latter has to be included in the annealing penalty term. In addition, the initial population is composed of clones of a same feasible individual that respects linear constraints [14].

Different approaches, called adaptive penalties, are based on learning from the population behaviour in the previous generations. In [16], the penalty factor decreases (resp. increases) if the best individual was always feasible (resp. infeasible) during the k last generations. For undeterminated cases, the factor value is kept unchanged. This methodology imposes the tuning of the initial value for the penalty factor and of the number of learning generations k.

New techniques now rest on self-adaptive penalty approaches, which also learn from the current run, without any parameter tuning. In [13], the constraints and the objective function are first normalized. Then, the method consists in computing the penalty factor for constraint j at generation q as the product of the factor at generation $q-1$ with a coefficient depending on the ratio of individuals violating constraint j at generation q. If this ratio is fewer to 50%, then the coefficient is inferior to 1 in order to favour individuals located in the infeasible side of the boundary. On the contrary, if the feasible individuals number is weak, the value increases up to 1 to have the population heading for the inside part of the feasible region. This operating mode enables to concentrate the search on the boundary built by each constraint, i.e. where the global optimum is likely to be located. The initial value is the ratio of the interquartile range of the objective function by the interquartile range of the considered constraint at first generation, which implicitly carries out normalization. No parameter is thus necessary in this method.

Another kind of self-adaptive penalty is proposed by Coello Colleo [17], but this one is based on the principle of co-evolution. In addition to the classical population P1 coding the tackled problem, the method considers a population P2 representing two penalty coefficients that enable to evaluate population P1 (w_1 for the amount of violation of all constraints and w_2 for the number of violated constraints). Thus, each individual of P1 is evaluated as many times as there are individuals in P2. Then, P1 evolves during a fixed number of generations and each individual of P2, i.e. each set of two penalty factors, is evaluated. This mechanism is depicted in Fig. 2. Basically, the evaluation is calculated as the average of all objective functions of P1 evaluated by each individual of P2. Then P2 evolves like in any GA process, given that one generation for P2 is equivalent to a complete evolution of P1. The evident drawback is the huge number of objective function evaluations, making this method computationally expensive. In addition, co-evolution involves the introduction and tuning of a new GAs parameters set: population size, maximum generation number, etc.

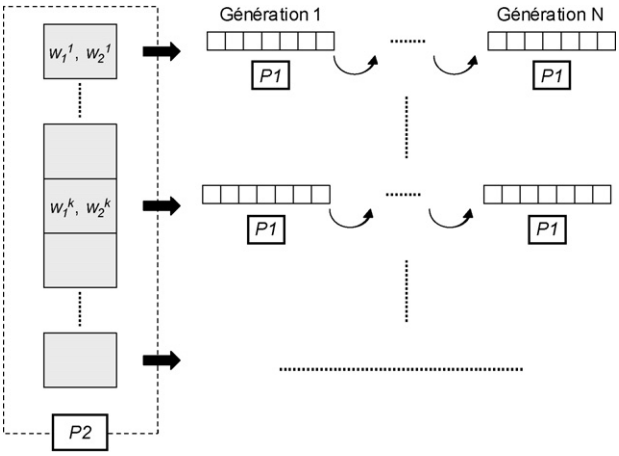

Fig. 2. Self-adaptive penalty by co-evolution [17].

Finally, the technique proposed by Deb [18] is half-way between classical penalisation and dominance-based methods. Superiority of feasible individuals on infeasible ones is expressed by the following penalised criterion:

$$\begin{cases} F(x) = f(x) & \text{if } g_j(x) \le 0, \quad j = 1, \ldots, m \\ F(x) = f_{\max} + \sum_{j}^{m} g_j(x) & \text{else} \end{cases} \tag{4}$$

$f_{\max}$ is the worst objective function value of all feasible solutions in the current population. The selection step is made out through a tournament process, but it could have been a Goldberg's roulette wheel as well [19]. Most individuals are infeasible at the beginning of the search, hence the selection exerts a pressure exclusively towards the feasible space until enough feasible solutions are located. Without the stochastic effect of both tournament or roulette wheel, the minimization of the objective function would only occur when the feasible individual number exceeds the survivor number. In spite of the random effect introduced, efficient mutation procedures and sometimes niching methods are necessary to maintain diversity in the population and to prevent the search from being trapped in a local optimum.

Let us recall that niching helps to avoid that two solutions characterized by a close set of variables both survive. Metric considerations (usually, the euclidean distance) help to estimate how close an individual is from another one. In [18], tournament between two feasible individuals is authorized only if the distance between them is lower than a constant threshold.

Among this profusion of techniques, some of the most classical methods were tested and evaluated in [20] for some benchmark examples. Some methods are really adapted to particular problems, but the authors finally chose the static penalty technique, which is the simplest and the most generic one.

2.3. Dominance-based methods

This class of constraint handling techniques is based on principles drawn from multiobjective optimisation and, in particular, on the dominance concept. The first idea is thus to transform a constrained mono-objective optimisation problem into an unconstrained multiobjective problem, where each constraint represents a new criterion to be minimized. Sorting procedures based on the domination in the sense of Pareto (x dominates y if and only if it is better than y for at least one criterion and as good as y for the other ones) leads toward the ideal solution x^*: $g_j(x^*) \le 0$ for $j = 1, \ldots, m$ and $f(x^*) \le f(y)$ for all feasible y.

These concepts are used again by Coello Coello [21] in the framework of mono-objective constrained optimisation in order to state particular dominance rules setting the superiority of feasible solutions on infeasible ones:

1. an infeasible solution is dominated by a feasible one;
2. if both individuals are feasible, the one with the worst objective function is dominated;
3. if both individuals are infeasible, the one with greatest constraint violation is dominated.

These rules are implemented in a tournament: it is to note that this technique is finally exactly identical to Deb's one [18], who just formalizes the rules as a penalty term added in the objective function.

Silva and Biscaia [22] use a quite similar methodology for multiobjective optimisation by setting additional domination rankings. The constraints are normalized and four domination levels are created and defined according to the range of constraint violation amount. The union of the range of all levels is 1. Each individual is classified in these levels according to its largest constraint violation amount. Then, a positive integer is assigned to each domination level and added as a penalty term to the normalized objective functions. The selection is carried out through successive Pareto sorting rounds, during which non-dominated individuals are chosen until obtaining enough surviving individuals.

Thus, the two above-mentioned examples highlight how tenuous the boundary is between this constraint handling mode and some kinds of penalisation techniques.

2.4. Other techniques

The description of other techniques is merged in this section because they are usually applicable only with some assumptions, or even defined exclusively for a particular problem. Firstly, in many cases, an adapted encoding method may enable to handle some constraints. A good is example is presented in [23], in which the number of 0-valued and 1-valued bits are coded instead of the bits themselves.

Besides, methods preserving solutions feasibility are usually based on specific crossover and mutation operators that are able to build, from feasible individual(s), one or several individuals that are feasible too. The GENOCOP algorithm provides a good example [24] for linear problems. Equality constraints are removed by substitution of an equal number of variables, so that the feasible space is then a convex set defined by linear inequalities. Due to this property, genetic operators consisting of linear combinations can ensure the feasibility of the created solutions. Maintaining the feasibility can also be carried out through the use of decoders, i.e., instructions contained in the chromosome that state rules for building a feasible solution.

Moreover, repairing infeasible chromosomes is a quite famous method. Indeed, in many cases of combinatorial optimisation, it is easy to create rules that, starting from an infeasible individual, enable to modify its structure to get a feasible one. In [25] for instance, repair procedures are implemented to act on individuals whose chromosome, resulting from crossover or mutation, has no physical meaning with regard to the used encoding method. However, repair rules are always devoted to the particular case of the studied problem and there is no existing heuristic for a general perspective. The particularity of the repair methods is also the possibility to replace in the population the infeasible individual by its repaired version or, on the contrary, to use this version only for the solution evaluation.

A generalized repair method proposed in [26] involves the first order development of the constraint violation vector ΔV, according to Δx, which represents a tiny variation in the

optimisation variables x:

$$\Delta V = \nabla_x V \times \Delta x, \quad \text{so } \Delta x = \nabla_x V^{-1} \times \Delta V \tag{5}$$

where matrix $\nabla_x V$ is the constraint violation gradient according to variables x. So, if the constraint violation amount is known and by approximating numerically its gradient, it is theoretically possible to determine the repair vector Δx for the considered infeasible individual. Since $\nabla_x V$ is usually not a square matrix, pseudoinverse computations provide an approximate inverse that can be used in (5). Despite its genericity ambition, it is predictable that such a method will only be applicable in some cases for which the functions and the nature of the involved variables are quite favourable.

This last technique can also be classified in the hybrid methods, just like the integration of Lagrange parameters in a penalty function, or the application of concepts drawn from fuzzy logic, etc. It should be referred to [10] for more details.

3. Batch plant design problems

Within the Process Engineering framework, batch processes are of growing industrial importance because of their flexibility and their ability to produce low-volumes and high added-value products. The Optimal Batch Plant Design (OBPD) problems have already been modelled and solved with a lot of approaches.

3.1. Problem presentation

This kind of problems is composed of items operating in a batch way. This means that a product batch has to be charged in each equipment item by an operator. This batch is subjected to given treatments in various equipment items, following a synthesis sequence, also named production recipe. Since a plant has to support the production of different products, the units must be cleaned after each batch has passed into it. In this study, we will only consider multiproduct plants, i.e. all the products follow the same operating sequence.

The aim of the OBPD problems is to minimize the investment cost for all items included in the plant by optimising the number and size of parallel equipment units in each stage. The production requirements of each product, the data related to each item (processing times and cost coefficients) and a fixed global production time are specified.

3.2. Assumptions

To model the OBPD problems, this paper uses Modi's formulation [3], modified by Xu et al. [27]. It considers not only treatment in batch stages, which usually appears in all kinds of formulation, but also represents semi-continuous units that are part of the whole process (pumps, heat exchangers, etc.). Let us recall that a semi-continuous unit is defined as a continuous unit working by alternating low-activity and normal activity periods.

Besides, this formulation takes into account short-term or mid-term intermediate storage tanks. They are used to divide the whole process into sub-processes, in order to store materials

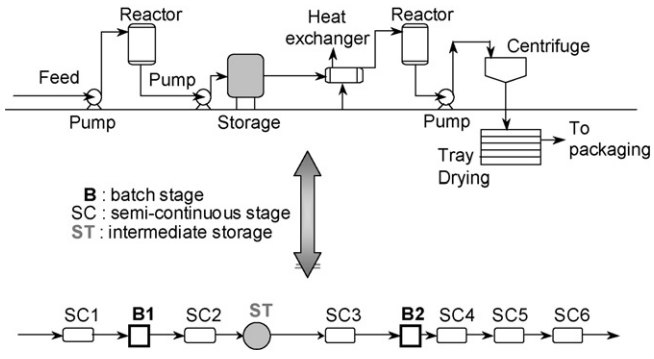

Fig. 3. Typical batch plant and modelling.

corresponding to the difference of each sub-process productivity. This representation mode confers to the plant a major flexibility for numerical resolution, by preventing the whole process production from being paralysed by one bottleneck stage. So, a batch plant is finally represented by series of batch stages (B), semi-continuous stages (SC) and storage tanks (T) as shown in Fig. 3 for the sake of illustration.

The model is based on the following assumptions:

(i) The devices used in a same production line cannot be used twice by one same batch.
(ii) The plant works according to a series of single product campaigns.
(iii) The units of the same batch or semi-continuous stage have the same type and size.
(iv) All intermediate tanks sizes are finite.
(v) If a storage tanks exists between two stages, the operation mode is "Finite Intermediate Storage". Otherwise, the "Zero-Wait" policy is applied.
(vi) There is no limitation for utility.
(vii) The cleaning time of the batch items is included into the processing time.
(viii) The item sizes are continuous bounded variables.

3.3. Model formulation

The model considers the synthesis of I products treated in J batch stages and K semi-continuous stages. Each batch stage consists of m_j out-of-phase parallel items of same size V_j. Each semi-continuous stage consists of n_k out-of-phase parallel items of same processing rate R_k. The item size (continuous variables) and equipment number per stage (discrete variables) are bounded. The *S-1* storage tanks, of size V_s^*, divide the whole process into S sub-processes.

According to the previously stated notations, a Mixed Integer Non-Linear Programming (MINLP) problem can be formulated, which minimizes an economic criterion representing the investment cost for all items. This cost is written as a power function of the unit size:

$$\text{Min Cost} = \sum_{j=1}^{J} m_j a_j V_j^{\alpha_j} + \sum_{k=1}^{K} n_k b_k R_k^{\beta_k} + \sum_{s=1}^{S-1} c_s V_s^{\gamma_s} \tag{6}$$

where a_j and a_j, β_k and b_k, γ_s and c_s are the classical cost coefficients (a complete nomenclature is in provided in Appendix A). Eq. (1) shows that no fixed cost coefficient was set for any item, which would be formulated, for instance: $a_j^1 + a_j^2 V_j^{aj}$. This may be few realistic and will not make optimisation to tend towards the minimization of the equipment number per stage. However, this formulation was kept unchanged in order to be able to compare the results with those found in the dedicated literature.

This problem is subjected to three kinds of constraints:

(i) Variables bounding:

$$V_{\min} \leq V_j \leq V_{\max}, \qquad \forall\, j \in \{1, \ldots, J\} \tag{7}$$

$$R_{\min} \leq R_k \leq R_{\max}, \quad \forall\, k \in \{1, \ldots, K\} \tag{8}$$

For the treated example, $V_{\min} = 250\,\mathrm{L}$ and $R_{\min} = 300\ \mathrm{L\,h^{-1}}$; $V_{\max}$ and $R_{\max}$ were both taken equal to 10,000 (L and $\mathrm{L\,h^{-1}}$, respectively). Moreover, for either batch or semi-continuous stages, the maximum item number per stage is limited to 3 ($m_{\max} = n_{\max} = 3$).

(ii) *Time constraint*: the total production time for all products must be lower than time horizon H, given in the problem data set.

$$\sum_{i=1}^{I} H_i = \sum_{i=1}^{I} \frac{Q_i}{\mathrm{Prod}_i} \leq H \tag{9}$$

where Q_i is the demand for product i.

(iii) *Constraint on productivities*: the global productivity for product i (of the whole process) is equal to the lowest local productivity (of each sub-process s).

$$\mathrm{Prod}_i = \underset{s \in S}{\mathrm{Min}}[\mathrm{Prodloc}_{\mathrm{is}}], \quad \forall\, i \in \{1, \ldots, I\} \tag{10}$$

These local productivities are calculated from the following equations:

(a) Local productivities for product i in sub-process s:

$$\mathrm{Prodloc}_{\mathrm{is}} = \frac{B_{\mathrm{is}}}{T_{\mathrm{is}}^{\mathrm{L}}}, \quad \forall\, i \in \{1, \ldots, I\}; \quad \forall\, s \in \{1, \ldots, S\} \tag{11}$$

(b) Limiting cycle time for product i in sub-process s:

$$T_{\mathrm{is}}^{\mathrm{L}} = \underset{j \in Js, k \in Ks}{\mathrm{Max}} [T_{ij}, \Theta_{ik}], \quad \forall\, i \in \{1, \ldots, I\}; \quad \forall\, s \in \{1, \ldots, S\} \tag{12}$$

J_s and K_s are respectively the sets of batch and semi-continuous stages in sub-process s.

(c) Cycle time for product I in batch stage j:

$$T_{ij} = \frac{\Theta_{ik} + \Theta_{i(k+1)} + p_{ij}}{m_j}, \quad \forall\, i \in \{1, \ldots, I\}; \quad \forall\, j \in \{1, \ldots, J\} \tag{13}$$

(k and $k+1$ represent the semi-continuous stages before and after batch stage j).

(d) Processing time of product i in batch stage j:

$$p_{ij} = p_{ij}^0 + g_{ij} B_{\mathrm{is}}^{d_{ij}}, \quad \forall\, i \in \{1, \ldots, I\}; \quad \forall\, j \in \{1, \ldots, J_s\}; \quad \forall\, s \in \{1, \ldots, S\} \tag{14}$$

(e) Operating time for product i in semi-continuous stage k:

$$\Theta_{ik} = \frac{B_{\mathrm{is}} D_{ik}}{R_k n_k}, \quad \forall\, i \in \{1, \ldots, I\}; \quad \forall\, k \in \{1, \ldots, K_s\}; \quad \forall\, s \in \{1, \ldots, S\} \tag{15}$$

(f) Batch size of product i in sub-process s:

$$B_{\mathrm{is}} = \underset{j \in J_s}{\mathrm{Min}} \left[\frac{V_j}{S_{ij}} \right], \quad \forall\, i \in \{1, \ldots, I\}; \quad \forall\, s \in \{1, \ldots, S\} \tag{16}$$

Finally, the size of intermediate storage tanks is estimated as the highest difference between the batch sizes treated by two successive sub-processes:

$$V^s = \underset{i \in I}{\mathrm{Max}}[\mathrm{Prod}_i S_{\mathrm{is}} (T_{\mathrm{is}}^{\mathrm{L}} + T_{i(s+1)}^{\mathrm{L}} - \Theta_{\mathrm{it}} - \Theta_{i(t+1)})], \quad \forall\, s \in \{1, \ldots, S-1\} \tag{17}$$

Then, the aim of OBPD problems is to find the plant structure that respects the production requirements within the time horizon while minimizing the economic criterion. The resulting MINLP problem proves to be non-convex and NP-Hard [5].

To solve this problem by a GA, the constraint on variable bounds is intrinsically handled by the coding and the constraint on productivities is implemented in the model. As a consequence, the only constraint to be handled explicitly by GA is the time constraint, which imposes the I products to be synthesized before a time horizon H. The constraint handling techniques presented in Section 4.3 are applied only on this constraint.

3.4. Studied example

One instance of optimal batch plant design problem will be used as a reference. In order to test GA efficiency, its size was chosen so as the GA faces some solution difficulties without being penalized too heavily by computational times. So, as depicted in Fig. 4, the considered plant is composed of six batch stages, eight semi-continuous stages and one intermediate storage tank. The plant has to support the synthesis of three products. Table 1 sums up all data associated to this problem.

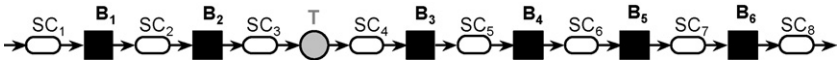

Fig. 4. Structure of the plant used as benchmark.

Table 1
Studied example data

		B1	B2	B3	B4	B5	*B6*
S_{ij}	$i=1$	8.28	6.92	9.70	2.95	6.57	10.60
	$i=2$	5.58	8.03	8.09	3.27	6.17	6.57
	$i=3$	2.34	9.19	10.30	5.70	5.98	3.14
p^0_{ij}	$i=1$	1.15	3.98	9.86	5.28	1.20	3.57
	$i=2$	5.95	7.52	7.01	7.00	1.08	5.78
	$i=3$	3.96	5.07	6.01	5.13	0.66	4.37
g_{ij}	$i=1$	0.20	0.36	0.24	0.40	0.50	0.40
	$i=2$	0.15	0.50	0.35	0.70	0.42	0.38
	$i=3$	0.34	0.64	0.50	0.85	0.30	0.22
d_{ij}	$i=1$	0.40	0.29	0.33	0.30	0.20	0.35
	$i=2$	0.40	0.29	0.33	0.30	0.20	0.35
	$i=3$	0.40	0.29	0.33	0.30	0.20	0.35

$I=3$; $J=6$; $K=8$; $S=2$; $H=6000$ h, $a_j=250$; $b_k=370$; $c_s=278$; $\alpha_j=0.60$; $\beta_k=0.22$; $\gamma_s=0.49$, $Q=[437\times10^3, 324\times10^3, 258\times10^3]$ kg; $S_{is}=1$ L kg^{-1}; $D_{ik}=1$ L kg^{-1} h^{-1}.

The optimisation variables are, for each (batch or semi-continuous) stage, the item size (V_j or R_k) and number (m_j or n_k). Thus, the numbers of continuous and integer variables are both 14. The corresponding combinatorial effect can be computed, considering that each stage may have at most three items (recall that $m_{max}=n_{max}=3$, see Section 3.3), and is equal to 4.8×10^6. It is to underline that the continuous variables do not participate in the previous calculation since the desired precision would a have a strong influence on the resulting combinatorial value.

The problem was solved to optimality by the *DICOPT++* solver, implementing the Outer Approximation algorithm [28] and available within the *GAMS* modelling environment [2]. The optimal cost is equal to 620638 and will be helpful to provide a reference, with respect to which the GA result quality can be evaluated. The corresponding variables are neither shown here nor in the computational results section, since they did not prove to be relevant for any performance evaluation.

4. Specific comments on the used Genetic Algorithm

The aim of this section is not to describe into detail the optimisation technique developed by Holland from 1975 [9]. We will just recall its basic principles and our comments are focused on the specific parameters used in this study.

4.1. General principles

The principles of GAs just lie on the analogy made between a population of individuals and a set of solutions of an optimisation problem. Just like the former, the latter evolves towards a good quality, or adaptation, according to the rules of natural selection stated by Darwin: the weakest individuals will disappear while the best ones will survive and be able to reproduce themselves. By way of genetic inheritance, the characteristics that make these individuals "stronger" will be preserved generation after generation.

The mechanisms implemented in the GAs mimic this natural behaviour. By creating selection rules that will express that the individuals are adapted to the considered problem, the best solutions are settled. Crossover and mutation operators then enable to get, at the end of the algorithm run, a population of good quality solutions. This heuristics set is mixed with a strongly stochastic aspect, leading to the compromise between exploration and intensification in the search space, which contributes to GAs efficiency.

The GA presented in this study is adapted from a classical, previously developed GA. The major difficulty for GAs use lies in its parameters tuning. The quality of this tuning greatly depends on user's experience and on his knowledge of the problem. Concerning the parameters such as population size, maximal number of computed generations or survival and mutation rates, a sensitivity analysis was performed to tend to an appropriate choice.

4.2. Main parameters

4.2.1. Variables encoding

A great number of studies proposed strategies coding simultaneously integer and continuous variables in the chromosome. It is clear that the way the variables are encoded is essential for the GAs efficiency.

The array representing the complete set of all variables is called a chromosome. It is composed of genes, each one encoding a variable by means of one or several locus. Obviously, we will make in this work a difference between the genes encoding a continuous variable from those encoding a discrete one. Since the formers are bounded, they can be written in a reduced form, like a real number a bounded within 0 and 1. Besides, each integer variable is coded directly in a single-locus gene, containing the variable value. Since an item number in a stage may be equal to 1, 2 or 3, so does a gene coding an integer variable.

The encoding method for continuous variables consists in discretising them, i.e. discretising the above-mentioned reduced variables a. According to the required precision, a given number of decimals of a is coded. This will logically have an effect on the string length, and so on the problem size. The so-called weighed box was used in this study: each decimal is coded by four bits b_1, b_2, b_3, b_4, weighing respectively 1, 2, 3, 3. As an example, the value of decimal d of a will be:

$$d = b_1 \times 1 + b_2 \times 2 + b_3 \times 3 + b_4 \times 3 \tag{18}$$

The coding used for the continuous the part of the chromosome is illustrated in Fig. 5. It is to admit that this encoding method presents drawbacks. Firstly, one same number can be represented in various ways (for instance, number 3 may be coded by the following strings: 1100, 0010 or 0001). This means that there exists some kind of ponderation of the probability to get a given number, which could cause higher computational times. Furthermore, the chosen encoding method does not integrate any Hamming distance concept, that would enable to have two numbers close one from another represented by two similar strings (i.e. differing by only a few values). Nevertheless, the chosen method offers the advantage of preventing from bias, since the sum of all weights is lower than ten. That is why it was adopted in the following study.

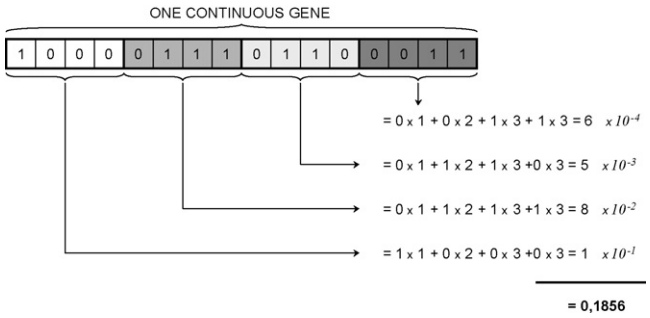

Fig. 5. Coding example with the weighed boxes.

Further works proved that floating-point encoding may be also adapted. However, since encoding techniques in Genetic Algorithms is not the point of this work, it is recommended to report to [29], providing a more detailed study about this issue.

Finally, considering that four decimals of the reduced variables are necessary to have the required precision on the initial continuous variables, and since each decimal is coded by 4 bits, a string for a continuous variable is 16 bits long. An integer variable is coded on one locus, so the size of the complete chromosome is equal to 17 multiplied by the stage number (here, 14), thus 238 loci.

4.2.2. Genetic operators

Obviously, the crossover and mutation operators are to be in accord with the encoding method logic. A wide variety of crossover techniques exists (1-point crossover, *k*-points crossover, uniform crossover, etc.). In a simplicity perspective and since a comparative study on crossover techniques does not constitutes the main issue of this work, a classical single cut-point procedure is implemented. Fig. 6 gives an example of the crossover technique, for a string involving 0–1 bits (item size) and integer loci (item number).

Besides, two distinct mutation operators must be used: (i) mutation by reversion of the bit value on the continuous part of the string; (ii) mutation by subtraction of one unit of a bit value on the discrete part (when possible). This technique leads towards minimization, but cannot prevent the algorithm from being trapped in a local optimum. Finally, elitism is carried out by copying the two best percents of the current population into the next generation.

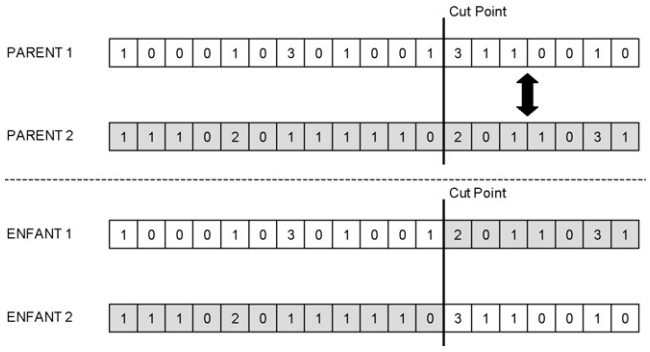

Fig. 6. One cut-point crossover.

4.3. Constraint handling

Obviously it is not possible to tackle all the previously mentioned constraint handling techniques. The followed methodology only considers methods that seem easy-implementing in GAs and adaptable to the framework of the treated problem, for which only the time constraint needs to be handled by the GA.

4.3.1. Elimination technique

The first technique tested in this study is elimination technique. As explained previously, it works by preventing the infeasible individuals to be able to take part in the following generation. This is carried out in the selection step of the algorithm, which involves the use of the Goldberg's roulette wheel [30].

This method can be seen like a random draw for which a sector of the wheel stands for each individual. The area of the sector depends on the relative fitness f_i of solution i related to the sum of all the other solution's fitness. In order to be fitted to the maximization effect of GAs, the fitness of each individual F_i is calculated as the difference between its objective function C_i and the worst objective function in the current population:

$$F_i = C_{\max} - C_i \tag{19}$$

So, setting the fitness of infeasible individuals to 0 implies that, there is no probability to have them selected by the roulette wheel. The consequence of this operating mode is that a completely feasible population has to be generated for the first generation. Indeed, without that condition, all fitness values would be equal to 0 and the GA would be equivalent to a simple random walk.

4.3.2. Penalisation of infeasible individuals

The most classical penalisation technique was used here, i.e. static penalty. Then, the criterion that is minimized by the GA is:

$$\begin{cases} F(x) = C(x) & \text{if } \sum_{i}^{I} H_i \leq H \\ F(x) = C(x) + \rho\left(H - \sum_{i}^{I} H_i\right)^2 & \text{else} \end{cases} \tag{20}$$

The selection method is kept unchanged as regard to the elimination technique, i.e. Goldberg's roulette wheel. Moreover, the initial population creation is not submitted any more to any feasibility constraint. Our aim is not to implement a really sophisticated method as the ones previously mentioned in the section devoted to the state of the art (dynamic, annealing or self-adaptive factor for instance). So, the ρ factor is set at the beginning of the search and does not vary in the whole run. Obviously, on the one hand, the greater is the value of the penalty factor, the higher is the weight of the second term, i.e. minimization of constraint violation. On the other hand, if ρ takes low values, the search may tend towards solutions minimizing strongly the objective function but violating widely the time

constraint. So, various values of this factor were tested in order to determine the best compromise between solution quality and feasibility.

4.3.3. Relaxation of upper discrete bounds

The basic idea of this technique came from the antagonism between the time constraint, which imposes to manufacture a given quantity of the I products within a time horizon H, and the upper limit of parallel items in a stage. Indeed, increasing the upper bounds of discrete variables, $m_{\max}$ and $n_{\max}$, means that the item number per stage can be higher, which allows the productivity to be higher. Thus, it makes it easier for the considered plant to respect the time constraint. As a consequence, if $m_{\max}$ and $n_{\max}$ are relaxed to a higher value, building a feasible initial population for the first roulette wheel step will become an easier issue. Afterwards, it is predictable that the minimization of the objective function by the GA will make the discrete variables tend towards lower values, thus within the initial feasible space. This one initially corresponded to discrete upper bounds equal to 3. In this study, $m_{\max} = n_{\max}$ were relaxed to the value $N_{\max}$, which is set to a constant value at the beginning of the run. Since the studied problem is not so difficult, values for $N_{\max}$ have been chosen equal to 4, 5 or 6.

4.3.4. Dominance-based tournament

This method proposed in [21] is based on the discrimination between feasible and infeasible solutions, and follows these dominance rules: (i) a feasible individual dominates an infeasible one; (ii) if both individuals are feasible, the one with the worst objective function is dominated; (iii) if both individuals are infeasible, the one with greatest constraint violation is dominated. These rules are implemented in a tournament that replaces the roulette wheel as the selection process. A tournament instance involves N_{Comp} competitors: each of them is compared to all the other ones according to the mentioned rules, in order to determine the N_{Win} winners (championship). For each selection step, the tournament is repeated as many times as necessary to get enough surviving individuals.

A previous intuitive understanding makes it able to foresee that the selection pressure will be all the more important when the difference between N_{Comp} and N_{Win} is greater. However, various combinations of competitors and winners were tested. A special case of this method, namely single tournament (referred as T.U. in the following sections, for Unique Tournament), occurs when the number of competitors is equal to the population size while the number of winners is the survivor number: then, all survivors are determined in one single tournament occurrence for each selection step.

4.3.5. Multiobjective strategy

The last option under investigation for an efficient constraint handling involves a multiobjective strategy. As mentioned in the literature analysis, this method simply considers the constraint violation as a second criterion that has to be minimized. So, the resulting problem is a bi-criteria unconstrained problem. The first objective function is the initial investment cost, while the second one is written as a quadratic form:

$$C'(x) = \left(H - \sum_{i}^{I} H_i \right)^2 \tag{21}$$

The use of a multiobjective strategy with the two considered criteria means a modification of the selection technique. The new method involves two roulette wheels. Each one is used to select, with the usual probabilistic effect, the best individuals according to each criterion. At the end of the search, a sorting process is carried out for all visited solutions in order to keep only Pareto's non-dominated individuals.

5. Computational results and interpretation

The parameters used to study the Genetic Algorithm performances were tuned by a sensitivity analysis. The algorithm was run with a 200 individuals population (*PopSize*) during 200 generations (*MaxGen*). Crossover and mutation rate were fixed to 0.6 and 0.4, respectively.

The results in this section are analysed in terms of computational time and quality. This quality is evaluated, of course, by the gap between the best found solution and the optimum. But, since GAs are a stochastic method, its results have to be analysed also in terms of repeatability. So, for each test, the GA was run 100 times and the best solution F^*_{GA}, is recorded. The criterion for repeatability evaluation is the dispersion of the runs around F^*_{GA}. The 2%-dispersion or 5%-dispersion are then defined as the percentage of runs providing a result lying respectively in the range $[F^*_{\text{GA}},\ F^*_{\text{GA}} + 2\%\ (= F^*_{\text{GA}} \times 1.02)]$ or $[F^*_{\text{GA}},\ F^*_{\text{GA}} + 5\%]$.

5.1. Elimination technique

Results for elimination technique are presented in Table 2. They show an unquestionable quality since both exhibit an optimal gap. Two percent and 5%-dispersions are also excellent since they are equal to, respectively, 82% and 100% of the runs. So this method will constitute a reference for the other constraint handling techniques. Besides, the computational time is equal to 35 s (with a Compaq Workstation W6000 with Xeon processor). This is quite high for such a medium-size example, since the *DICOPT++* module gave an optimal result in less than one second. It allows to foresee restrictive computational cost when treating more complex problems. This feature would also constitute a bottleneck for applications that would carry out the computation of the objective function by a simulator.

This low performance is due to the number of objective function evaluations. Actually, the theoretical evaluation number is the product MaxGen × PopSize, which is equal to 4.0×10^4

Table 2
Results for elimination technique

	Elimination technique
Best result	622,566
Optimal gap (%)	0.31
CPU time (s)	35

Table 3
Best found solution for various penalty factor values

	$\rho = 10^{-4}$	$\rho = 10^{-2}$	$\rho = 1$	$\rho = 10^{2}$	$\rho = 10^{4}$
Best result (including penalty)	257,024	544,547	620,137	622,799	622,245
Real investment (without pen.)	207,820	486,506	619,464	622,783	622,245
Optimal gap (%)	−66.52	−21.61	−0.19	0.35	0.26
Constraint violation (%)	369.7	40.15	0.43	0.01	0.0

(with systematic evaluations of identical individuals). But the real evaluation number is equal to 7.11×10^5, i.e. almost 18 times the theoretical value. The huge difference is due to the necessity of creating a completely feasible initial population, which turns to be a harsh task. The time spent in randomly finding 200 feasible individuals for the first generation takes almost the majority of computing resources.

5.2. *Penalisation technique*

First, as regard to the computational time, it is obvious that this technique is really more performing. Indeed, since no feasibility condition is imposed for the initial population generation, the number of objective function evaluations is here equal to MaxGen × PopSize: this allows the computational time to be reduced from 35 s with the elimination technique to 2 s with the penalisation method.

Concerning the solution quality, the results are presented in Table 3: in the first row, the value of the penalized objective function is given; in the second one, the corresponding investment cost represents the objective function without the penalty term; the optimal gap presented in the third row can have negative values since infeasible solution may be really "cheaper" than the feasible ones; finally, the constraint violation is the total production time related to the initial time horizon H.

These results agree with the predicted trends. For small values of the penalty factor, the found solution is widely infeasible, which explains the low investment cost, so far away from the constrained optimal value. As the penalty factor rises, the amount of constraint violation decreases. At the same time, the economic criterion logically increases until reaching near optimal values when the best found solution is feasible.

No feasible solution is found during a whole run for the two first cases (10^{-4} and 10^{-2}). Moreover, for the three highest values, the evolution of feasible individuals ratio in the population during a run is shown in Fig. 7. After fifty generation, the ratio stabilizes between 30% and 40% for $\rho = 10^2$ and $\rho = 10^4$, but drops down when ρ is equal to 1. The best feasible solution found for this last penalty factor is equal to 621,973 (this value does not appear in Table 3 since the best result found by the GA (620,137) is slightly infeasible) and is quite interesting since it is only 0.21% higher than the optimal value.

Finally, 2% and 5% dispersions increase markedly when the ρ factor decreases (Fig. 8) since it is easier to find low-valued individuals. For $\rho = 10^{-4}$, the dispersions exhibit equivalent results as those obtained with the elimination technique. For $\rho = 1$, it should be noted that the dispersions concerning only feasible solutions found in each run (which does not appear in Fig. 8) are quite high: 58% and 100% of the runs found a feasible solution closer than the above-mentioned 621,973 value plus respectively 2% and 5%.

So, the efficiency of this constraint handling mode is demonstrated in terms of computational time. Some values of the penalty factor even provide dispersions that turn out to be quite satisfactory. Nevertheless, the choice of an appropriate ρ value is disrupted by the antagonism between criterion quality and solution feasibility. The number of feasible solutions is not truly significant for this example since its medium-complexity does not prevent from finding, whatever the encountered case, near-optimal solutions.

However, this criterion will be much more important when, for larger size examples, the location of feasible solutions will be a challenging task which, thus increasing the probability of determining a good quality result. Higher values of the penalty factor should then be more attractive. From $\rho = 10^4$ to $\rho = 10^2$, the 2–5% dispersions saving is high without any extreme feasible

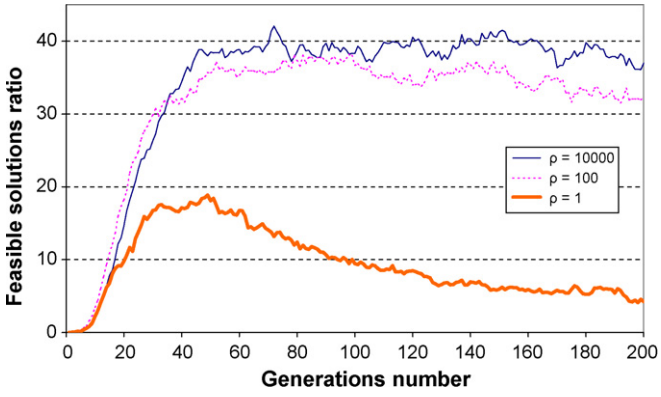

Fig. 7. Evolution of feasible solutions ratio.

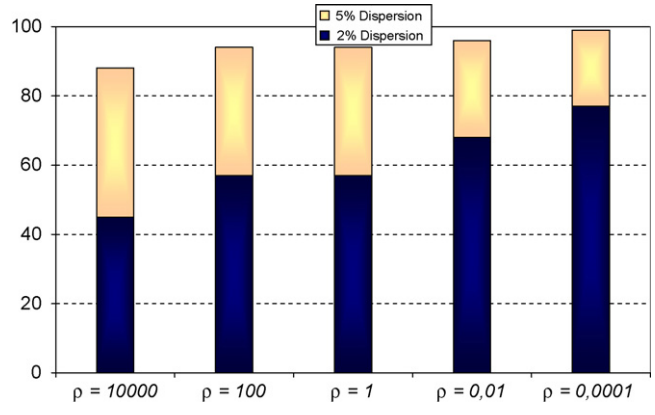

Fig. 8. Dispersions for different ρ values.

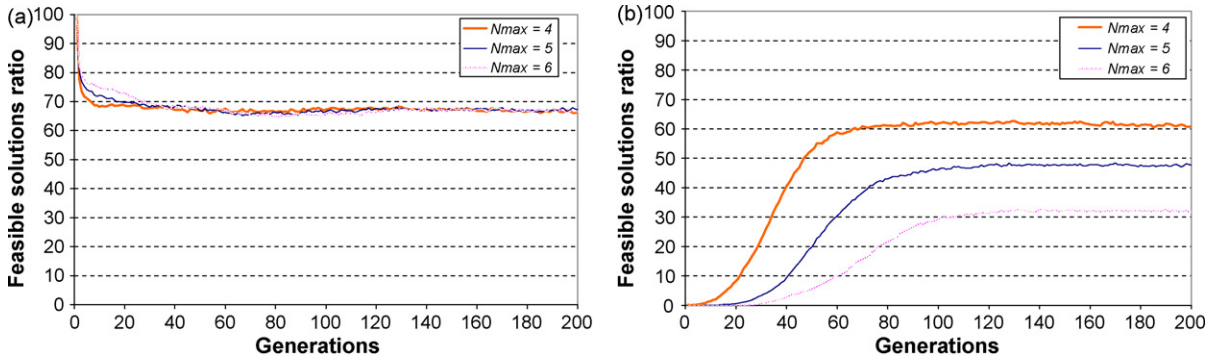

Fig. 9. Evolution of feasible solutions ratio: (a) strict sense; (b) broad sense.

ratio loss. So, the most appropriate compromise between these two possibilities turns to be the second one.

5.3. Discrete upper bound relaxation

The first results that have to be checked is the effect of GA on solution feasibility, in order to justify the good behaviour of the upper bound relaxation for discrete variables. Charts in Fig. 9 represent: (a) the evolution, generation after generation, of the feasible solution ratio in a broad sense, i.e. respecting the time constraint only; (b) the evolution of the feasible solutions ratio in a strict sense, respecting the initial upper bound on equipment number per stage (in addition to the constraint time, obviously). Trivially, the relaxed upper bound is necessarily respected since it is integrated in the variables encoding.

The Goldberg's roulette wheel, which involves the generation of a fully feasible initial population (in a broad sense) can be observed in Fig. 9(a). The curves that correspond to $N_{max} = 4$, 5 and 6 are almost merging together. The feasible solutions ratio in a broad sense then decreases inevitably as the stochastic crossover and mutation processes create infeasible individuals. However, for all the three cases, the rate stabilizes between 65% and 70% from generation 40.

In Fig. 9(b), the feasible solutions ratio in a strict sense follows a conflicting evolution. The initial population does not involve any solution respecting the three items per stage limit, which is the required effect. Then, the Genetic Algorithm seems to guide the search towards minimization of the equipment number since the three curves rise until reaching a plateau. The plateau value depends on the relaxation degree: the lower is N_{max}, then the higher will be the pressure towards (really) feasible regions. So, the values for the plateau are 32%, 48% and 62% for an upper bound respectively relaxed to 4, 5 and 6 items per operating stage.

It is to note that no failure is to be lamented (i.e. a run for which no one feasible solution is found during the whole search). This reinforces the GA ability to reduce the items number while respecting the time constraint.

With regard to results quality presented in Fig. 10, the first remark is that the best solutions found are all very close to the global optimum, thus no conclusion could be drawn from this values. Moreover, a better reliability of lower relaxations is found. This behaviour is not a surprise since their greater harshness makes them look like the elimination technique (which is also shown as a reference). Nevertheless, the 2% dispersion for $N_{max} = 4$ is still equal to the half-part of the elimination one: the efficiency loss is huge. For an extreme value ($N_{max} = 6$), the results are finally far from being satisfactory.

So, the most viable option seems to be a low relaxation of the discrete upper bound, but this one has a computational price: last row of Table 4 indicates that the CPU time is almost divided by two from $N_{max} = 4$ to $N_{max} = 6$. Admittedly, the 2 s absolute difference is weak, but it is easily predictable that the gap should be quite higher for a larger size example.

Finally, it must be kept in mind that the computational time with the elimination technique is 10 or 15 times greater (35 s). The CPU time differences between the various relaxations can be related to the number of visited solutions in order to be able to create a feasible initial population. As it is plotted in Fig. 11, the value shows a decreasing trend when the relaxation is larger. For the last case ($N_{max} = 6$), the number of infeasible individuals tested to generate the initial population is divided by a 100 factor regarding the elimination method. So, the additional cost of

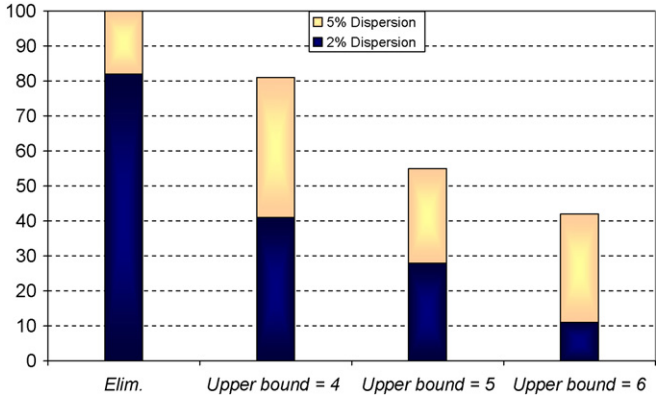

Fig. 10. Two percent to 5% dispersions for relaxation method.

Table 4
Results for the relaxation method

	$N_{max} = 4$	$N_{max} = 5$	$N_{max} = 6$
Best solution	622,433	621,322	621,906
Optimal gap (%)	0.29	0.11	0.20
CPU time (s)	4.5	2.7	2.3

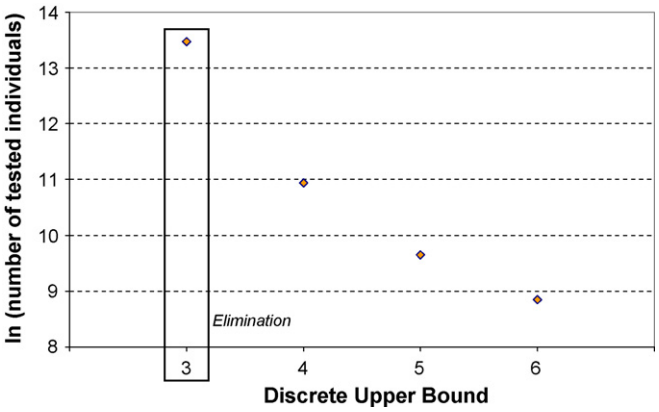

Fig. 11. Visited individual number to create the initial population.

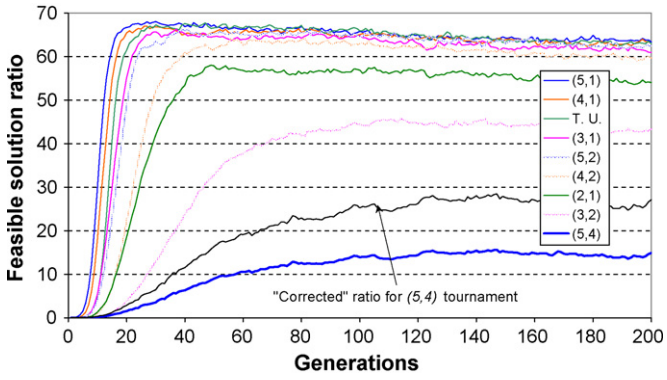

Fig. 12. Evolution of feasible solutions ratio (curves and legend written in the same order).

objective function evaluation becomes insignificant with regard to the total evaluation number.

5.4. Dominance-based tournament

The computational time for this method is logically identical for all tested tournament versions, i.e. 2 s. Indeed, just like in the penalisation technique, the real number of objective function evaluations is equal to the theoretical one (MaxGen × PopSize).

Results of Table 5(a) and (b) point out that, as for the relaxation case, the best found solutions for all tournament versions are all close to the optimal solution (from 0.23% to 0.65%). So this criterion does not allow drawing any significant conclusions about superiority of some combinations on other ones. Besides, the last row of Table 5, showing the failure ratio, highlights the weakness of the combinations that were pointed out to be, *a priori*, the less selective ones. Indeed, the failure number is higher than 0 for the (*3*, *2*) version and concerns almost a half-part of the runs for the (*5*, *4*) tournament. Since the tackled example complexity is only medium, this trend does not encourage to try to apply these versions to larger size examples.

So the two remaining criteria that should help to decide between the various options are now 2% and 5% dispersions and the feasible solutions ratio. Fig. 12 shows the evolution of feasible solutions ratio, generation after generation, and exhibits the bad performances of (*3*, *2*) and (*5*, *4*) tournaments. For both versions, the encountered feasible solutions ratio keeps being weak. This remark should nevertheless be moderated by the fact that the feasible solution number is actually an average calculated according to the 100 runs without taking into account the failed runs. This means that, for instance, in the (*5*, *4*) case showing a 45% failure rate, the feasible solutions ratio must be divided by (1–0.45) to get the average feasible individuals ratio in a successful run.

This "corrected" version for (*5*, *4*) tournament was plotted in Fig. 12 just for the sake of illustration, as it could have been done for the (*3*, *2*) combination. But the corresponding values still remain lower than 30%. This analysis highlighted a double unfitness of this tournament version, concerning both exploration and intensification. On the one hand, it is not reliable to push individuals towards the feasible region when this is not determined yet. On the other hand, when a feasible point is found, this kind of tournament is unable to exert pressure on the individuals towards the located feasible region in order to intensify the search in this promising zone.

Regarding the other combinations, all of them seem quite performing. They all show almost identical feasible solutions ratios, except the (*2*, *1*) and (*4*, *2*) versions which prove to be a bit lower. It is to note that the curves can be classified according to an increasing order of the ratio N_{Comp}/N_{Win}. As predicted, the higher is this ratio, the more feasible solutions are got in a run.

The dispersions plotted in Fig. 13 reinforce the low quality of (*5*, *4*) tournament. An analysis similar to the previous one can be carried out to correct the dispersions values according to

Table 5
Results for different tournament versions

	($N_{Comp}=2$, $N_{Win}=1$)	(*3*, *1*)	(*3*, *2*)	(*4*, *1*)
(a)				
Best result	622,269	622,921	622,246	624,671
Optimal gap (%)	0.26	0.37	0.26	0.65
Failure ratio (%)	0	0	7	0

	(*4*, *2*)	(*5*, *1*)	(*5*, *2*)	(*5*, *4*)	T.U.
(b)					
Best result	622,035	623,819	623,840	622,638	623,093
Optimal gap (%)	0.23	0.51	0.52	0.32	0.40
Failure ratio (%)	0	0		45	0

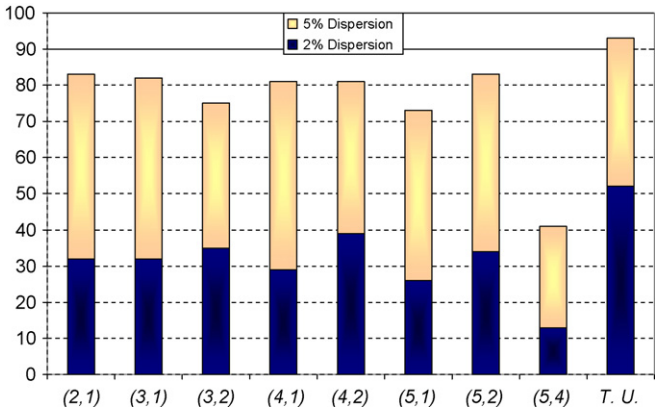

Fig. 13. Dispersions for different tournament versions.

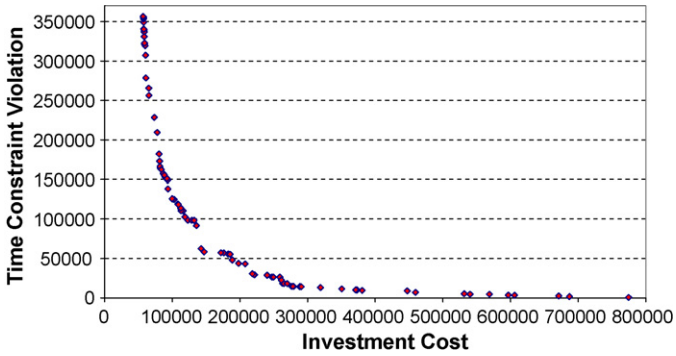

Fig. 15. Pareto non-dominated solutions.

the failure number. This allows some moderation but it does not change the negative conclusions about the mentioned versions. Then, it must be pointed out that the (*5*, *1*) combination, which was the best one in terms of solutions feasibility, is in the case of dispersions among the worst tested versions. This behaviour is disappointing and no explanation was found to account for this failing. But this trend has to be moderated since the results for the (*5*, *1*) tournament do not lie far from the average of other versions.

Considering the best combinations, the single tournament method shows a huge superiority on the other ones. Then comes the (*4*, *2*) version. Between the two above-mentioned extremes, the statistic qualities of the results provided by the other versions are quite identical. Finally, the most favourable compromise is, without any doubt, the single tournament method. Indeed, its performances for dispersions as well as number for feasible solutions found during the search belong to the best ones among the various tested combinations. However, the comparison with the elimination technique highlights the fact that even if this last one is really slower, it keeps being the best one in terms of solution quality.

Nevertheless, it is to note that the way the single tournament works involves the removal of the tournament stochastic character. This effect is usually obtained through the random choice of the competitors among the population. In the single tournament case, only the very best individuals are selected. Furthermore, its operating mode underlies that the pressure towards a minimization of the objective function is exerted only when all the selected surviving individuals are feasible. These two aspects may tend to deteriorate the diversity in the population, which could be penalising for more constrained problems.

To overcome this difficulty, it is recommended in [18] to use a niching technique such as the one described in the second section. In [21], another technique to preserve the individuals diversity consists in applying a tournament with a probability S_r. If a random number is higher than S_r, the tournament winner is nominated randomly. Such a method was adopted in this work. According to the above mentioned dominance rules, the best individuals among the population are determined. Then, each survivor is selected among the single tournament winners with a probability S_r. If a random number is higher than S_r, the survivor is randomly chosen in the whole population.

It is suggested in [21] to use S_r values between 0.8 and 1. Some computations were carried out to test this parameter influence, only for the single tournament case. As proven by Fig. 14, the obtained results are not convincing, for the dispersions as well as for the feasible solution number: the simplest version ($S_r = 1$) is the most efficient. However, it is not possible to generalize, since for larger size instances of the problem, this diversification method could be useful.

5.5. *Multiobjective strategy*

The Pareto front, i.e. the set of non-dominated individuals according to the Pareto sense is shown in Fig. 15. A first negative comment is that there is no feasible non-dominated

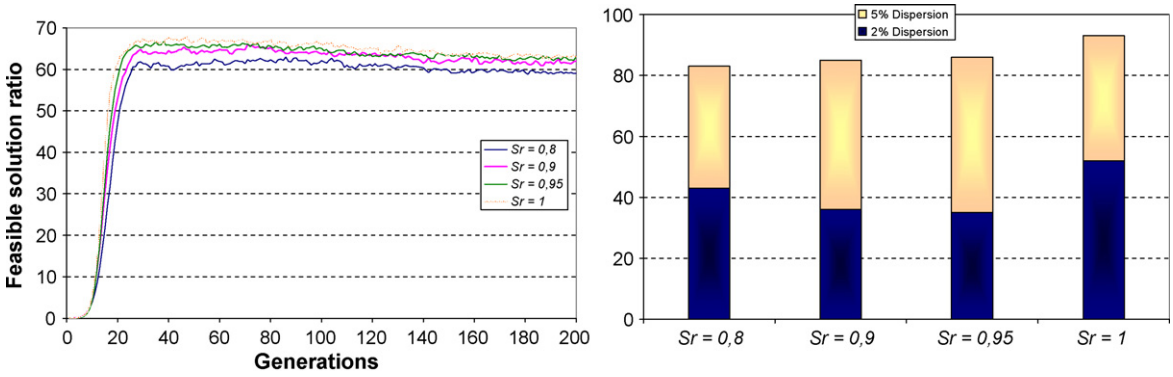

Fig. 14. Study on S_r influence.

solution. The solution characterized by the smallest amount of constraint violation (10%) lies at almost 25% of the global optimum of the mono-objective initial problem. The two solutions being the closest to the optimal value propose an investment cost improvement equal to 2.36% and 3.86%, but they show an amount of constraint violation of respectively 56% and 58%. These solutions may be helpful in a decision helping aim, but in the framework of mono-objective optimisation they are few interesting. In addition, one-run computational time, mainly due to the Pareto sorting procedure, is time expensive, 20 min. So the results are rather inconclusive for this method.

6. Conclusions

The aim of this study was to determine a technique that, in the framework of optimal batch plant design problems, should be competitive in terms of computational time and of result quality. So, a mid-size benchmark example was proposed and solved with one same GA implementing various constraint handling modes: elimination, penalisation, relaxation of discrete variables bounds, dominance-based tournament and multiobjective strategy.

The first remark is that among the five studied techniques, three of them need the tuning of one or several parameters: static penalty factor, value of the relaxed discrete upper bound, terms of the tournament, etc. This makes a drawback towards the elimination and multiobjective strategies. However, this latter turns out to be quite unadapted to the presented mono-objective optimisation framework. The elimination technique is yet interesting but computationally expensive. Actually, the time spent in generating a feasible initial population constitutes the bottleneck of the run. Indeed, methods that are not submitted to this restriction, i.e. penalisation and tournament, are attractive from the computational time viewpoint since the objective function evaluation number is equal to its theoretical value. Then, an intermediate technique is the relaxation of discrete upper bounds, for which the computational cost is quite limited. But it is predictable that, for larger size problems, an appropriate tuning of the upper bound should be necessary to find a satisfactory compromise between solution quality and time efficiency.

As regard to the result quality, it is clear that the elimination technique is the best one. Besides, the result quality is quite interesting for two factor values of the penalisation technique but the feasible solution number is not convincing. The same remark is valid for the relaxation method, which shows good solutions for the fewest relaxed cases. But the necessary parameter tuning to prevent the time cost from becoming prohibitive is not encouraging. Finally, concerning the dominance-based tournaments, the best option is the single tournament. The visited feasible solutions and the study of the result dispersions provided by various runs exhibit a similar behaviour to the elimination technique.

It can be concluded that the elimination constraint handling mode is attractive when the computational time is not a bottleneck for calculations. When the problem complexity involves high computational time, the single tournament may be an efficient alternative. These guidelines should be useful for selecting the most appropriate constraint handling technique for the treatment of similar problems of batch plant design.

Appendix A. Nomenclature

a_j	cost factor for batch stage j
b_k	cost factor for semi-continuous stage k
B_{is}	batch size for product i in batch stage s (kg)
c_s	cost factor for intermediate storage tanks
d_{ij}	power coefficient for processing time of product i in batch stage j
D_{ik}	duty factor for product i in semi-continuous stage k ($\mathrm{L\,kg^{-1}}$)
g_{ij}	coefficient for processing time of product i in batch stage j
H	time horizon (h)
H_i	production time of product i (h)
i	index for products
I	total number of products
j	index for batch stages
J	total number of batch stages
J_s	total number of batch stages in sub-process s
k	index for semi-continuous stages
K	total number of semi-continuous stages
K_s	total number of semi-continuous stages in sub-process s
m_j	number of parallel out-of-phase items in batch stage j
n_k	number of parallel out-of-phase items in semi-continuous stage k
p_{ij}	processing time of product i in batch stage j (h)
p^0_{ij}	constant for calculation of processing time of product i in batch stage j (h)
Prod_i	global productivity for product i (kg/h)
$\mathrm{Prodloc}_{is}$	local productivity for product i in sub-process s (kg/h)
Q_i	demand for product i
R_k	processing rate for semi-continuous stage k ($\mathrm{L\,h^{-1}}$)
R_{max}	maximum feasible processing rate for semi-continuous stage k ($\mathrm{L\,h^{-1}}$)
R_{min}	minimum feasible processing rate for semi-continuous stage k ($\mathrm{L\,h^{-1}}$)
S	total number of sub-processes
S_{ij}	size factor of product i n batch stage j ($\mathrm{L\,kg^{-1}}$)
S_{is}	size factor of product i in intermediate storage tanks ($\mathrm{L\,kg^{-1}}$)
T_{ij}	cycling time of product i in batch stage j (h)
T^{L}_{is}	limiting cycling time of product i in sub-process s (h)
V_j	size of batch stage j (L)
V_{max}	maximum feasible size of batch stage j (L)
V_{min}	minimum feasible size of batch stage j (L)
V^s	size of intermediate storage tank (L)

Greek letters

α_j	power cost coefficient for batch stage j
β_k	power cost coefficient for semi-continuous stage k
γ_s	power cost coeficient for intermediate storage
Θ_{ik}	operating time of product i in semi-continuous stage k

References

[1] G.R. Kocis, I.E. Grossmann, Global optimization of nonconvex mixed-integer non-linear programming (MINLP) problems in process synthesis, Ind. Eng. Chem. Res. 27 (1998) 1407–1421.

[2] A. Brooke, D. Kendrick, A. Meeraus, R. Raman, GAMS User's Guide, GAMS Development Corporation, 1998.

[3] A.K. Modi, I.A. Karimi, Design of multiproduct batch processes with finite intermediate storage, Comput. Chem. Eng. 13 (1989) 127–139.

[4] A.N. Patel, R.S.H. Mah, I.A. Karimi, Preliminary design of multiproduct non-continuous plants using simulated annealing, Comput. Chem. Eng. 15 (1991) 451–469.

[5] C. Wang, H. Quan, X. Xu, Optimal design of multiproduct batch chemical process using genetic algorithms, Ind. Eng. Chem. Res. 35 (1996) 3560–3566.

[6] C. Wang, H. Quan, X. Xu, Optimal design of multiproduct batch chemical process using tabu search, Comput. Chem. Eng. 23 (1999) 427–437.

[7] C. Wang, Z. Xin, Ants foraging mechanism in the design of batch chemical process, Ind. Eng. Chem. Res 41 (2002) 6678–6686.

[8] J.M. Montagna, A.R. Vecchietti, O.A. Irribarren, J.M. Pinto, J.A. Asenjo, Optimal design of protein production plants with time and size factor process models, Biotechnol. Prog. 16 (2000) 228–237.

[9] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.

[10] C.A. Coello Coello, Theoretical and numerical constraint-handling techniques uses with evolutionary algorithms: a survey of the state of the art, Comput. Meth. Appl. Mech. Eng. 191 (2002) 1245–1287.

[11] Z. Michalewicz, M. Schoenauer, Evolutionary algorithms for constrained parameters optimization problems, Evol. Comput. 4 (1996) 1–32.

[12] J.T. Richardson, M.R. Palmer, G. Liepins, M. Hilliard, Some guidelines for genetic algorithms with penalty functions, in: D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, Morgan Kaufmann Publishers, 1989, pp. 191–197.

[13] W.H. Wu, C.Y. Lin, The second-generation of self organizing adaptive penalty strategy for constrained genetic search, Adv. Eng. Softw. 35 (2004) 815–825.

[14] Z. Michalewicz, Genetic algorithms, numerical optimization, and constraints, in: L. Eshelman (Ed.), Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, 1995, pp. 151–158.

[15] A. Homaifar, S.H.Y. Lai, X. Qi, Constrained optimization via genetic algorithms, Simulation 62 (1994) 242–254.

[16] A.B. Hadj-Alouane, J.C. Bean, A genetic algorithm for the multiple-choice integer program, Operat. Res. 45 (1997) 92–101.

[17] C.A. Coello Coello, Use of a self-adaptive penalty approach for engineering optimization problems, Comput. Industr. 41 (2000) 113–127.

[18] K. Deb, An efficient constraint handling method for genetic algorithms, Comput. Meth. Appl. Mech. Eng. 186 (2000) 311–338.

[19] L. Costa, P. Oliveira, Evolutionary algorithms approach to the solution of mixed integer non-linear programming problems, Comput. Chem. Eng. 25 (2001) 257–266.

[20] Z. Michalewicz, D. Dasgupta, R.G. Le Riche, M. Schoenauer, Evolutionary algorithms for constrained engineering problems, Comput. Ind. Eng. 30 (1996) 851–870.

[21] C.A. Coello Coello, E. Mezura Montes, Constraint-handling in genetic algorithms through the use of dominance-based tournament selection, Adv. Eng. Informat. 16 (2002) 193–203.

[22] C.M. Silva, E.C. Biscaia Jr., Genetic algorithm development for multiobjective optimization of batch free-radical polymerization reactors, Comput. Chem. Eng. 27 (2003) 1329–1344.

[23] Y.C. Hou, Y.H. Chang, A new efficient encoding mode of genetic algorithms for the generalized plant allocation problem, J. Informat. Sci. Eng. 20 (2004) 1019–1034.

[24] Z. Michalewicz, N.F. Attia, Evolutionary optimization of constrained problems, in: A.V. Sebald (Ed.), Proceedings of Third Annual Conference of Evolutionary Programming, World Scientific, Singapore, 1994, pp. 98–108.

[25] A. Dietz, C. Azzaro-Pantel, L. Pibouleau, S. Domenech, A framework for multiproduct batch plant design with environmental considerations: application to protein production, Ind. Eng. Chem. Res. 44 (2005) 2191–2206.

[26] P. Chootinan, A. Chen, Constraint handling in genetic algorithmsising a gradient-based repair method, Comp. Operat. Res. 33 (2006) 2263–2281.

[27] X. Xu, G. Zheng, S. Cheng, Optimal design of multiproduct batch chemical process—a heuristic approach, Chem. Eng. J. 44 (1993) 442–450 (in Chinese).

[28] J. Viswanathan, I.E. Grossmann, A combined penalty function and outer-approximation method for MINLP optimisation, Comput. Chem. Eng. 14 (1990) 769–782.

[29] A. Ponsich, C. Azzaro-Pantel, L. Pibouleau, S. Domenech, in: Michalewicz, Z., Siarry (Eds.), Some guidelines for Genetic Algorithm implementation in MINLP Batch Plant Design problems, Springer volume Metaheuristics, in press.

[30] D.E. Golberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing Company Inc., MA, 1989.