Some guidelines for Genetic Algorithm implementation in MINLP Batch Plant Design problems

A Ponsich, Catherine Azzaro-Pantel, S Domenech, Luc Pibouleau

A. Ponsich, **C. Azzaro-Pantel,** S. Domenech, and L. Pibouleau
Some guidelines for Genetic Algorithm implementation in MINLP Batch Plant Design problems, sous presse dans Springer volume, intitulé "Advances in Metaheuristics for Hard Optimization". edited by Z. Michalewicz and P. Siarry in the Natural Computing series.

Some guidelines for Genetic Algorithm implementation in MINLP Batch Plant Design problems

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

Laboratoire de Génie Chimique de Toulouse, 5 rue Paulin Talabot BP 301, 31106 Touloues Cedex 1, France

Abstract

In the last decades, a novel class of optimisation techniques, namely metaheuristics, has been developed and essentially devoted to the solution of highly combinatorial discrete problems. The improvements provided by these methods were however extended to the continuous or mixed-integer optimisation area. This paper addresses the problem of adapting a Genetic Algorithm (GA) to a Mixed Integer Non Linear Programming (MINLP) problem. The support of the work is optimal batch plant design, which is of great interest in the framework of Process Engineering. This study deals with the two main issues for GAs, i.e. the treatment of continuous variables by specific encoding and the efficiently constraints handling in GA. Various techniques are tested for both topics and numerical results show that the use of a mixed real-discrete encoding and a specific domination based tournament method constitutes the most appropriate approach.

Keywords : Genetic Algorithms ; Variables encoding ; Constraint handling ; Batch plant design

1. Introduction

A large range of applications drawn from the Process Engineering framework can be expressed as optimisation problems. This application range consists of examples formulated as pure continuous problems – for instance the phase equilibrium calculation problem (see Teh and Rangaiah, 2003), as well as problems involving pure discrete variables – like the discrete job-shop batch plant design (Dedieu et al., 2003). Typically, for the former case, difficulties arise from non-linearities while they are due to the discontinuous nature of functions and search space for the latter one. Finally, a great variety of models from the Process Engineering area combines both kinds of problems and involves simultaneously (continuous) operation and (discrete) decision variables. Design problems represent good

* Author to whom correspondence should be addressed: Antonin.Ponsich@ensiacet.fr

examples of this complexity level, such as process network superstructure (Lee and Grossmann, 2000) or multiproduct batch plant design problems (Modi et Karimi, 1989).

Obviously, a significant investigation effort was carried out to develop efficient and robust optimisation methods, at the beginning especially in the Operational Research and Artificial Intelligence areas, but subsequently within the Process Engineering community. Among the diversity of optimisation techniques, two great classes can be distinguished. The former consists of deterministic methods, which are based on a harsh mathematical study (derivability, continuity…) of the objective function and of the constraints to ensure an optimum. However, despite their ability to handle non-linear models, their performances might be strongly affected by non-convexities. This implies a great effort to get a proper formulation of the model. Grossmann (2002) proposes a review of the various Mixed Integer Non Linear Programming (MINLP) existing techniques but it is commonly accepted that these methods might be heavily penalised by the NP-hard nature of the problems, and will be then unable to solve large size instances.

So, a rising interest was given to the development of methods of the second class, i.e. metaheuristics or stochastic methods. They work by evaluating the objective function at various points of the search space. These points are chosen by using a set of heuristics combined with the generation of random numbers. The stochastic techniques do not use any mathematical properties of the functions, so they cannot guarantee to obtain any optimum. Nevertheless, metaheuristics allow the solution of a large range of problems, particularly when the objective function is computed by a simulator embedded in an outer optimisation loop (Dietz et al., 2005). Furthermore, despite their computational greedy aspect, they are quite easily adaptable to highly combinatorial optimisation problems. A classification of metaheuristics and a survey of the main techniques is proposed in Hao et al. (1999).

Actually, this study deals with the treatment of a problem drawn from the Chemical Engineering literature, i.e. optimal design of batch plants usually dedicated to the production of chemicals. The model, involving both real and integer variables, is solved with a Genetic Algorithm. This technique has already shown its efficiency for this problem class, especially when the objective function computation is carried out through the use of Discrete Event Simulators (DES) Dietz et al. (2005). Then, the aim of the paper consists in the adaptation of the Genetic Algorithm to the studied problem and will particularly focus on the two main issues inherent to the model formulation: constraint handling and continuous variables

encoding. The efficient management of these two GAs internal procedures constitutes the key-point to obtain good quality results within acceptable computational time.

This paper is divided into six sections. The problem formulation and the methodology are presented in section 2, while 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. Some typical results are then analysed in section 5 and finally, conclusions and perspectives are given in section 6.

2. Outline of the problem

2.1. Outline on metaheuristics

In the two last decades, major advances were carried out in the optimisation area through the use of metaheuristics. These methods are defined as a set of fundamental concepts that lead to design heuristics rules dedicated to the solution of an optimisation problem (Hao et al., 1999). Basically, they can be divided into two classes : neighbourhood methods and evolutionary algorithms. The former is obviously based on the definition of neighbourhood notion.

Definition : considering a set X and a string $x = [x_1, x_2, \ldots, x_n] \in X$. Let also f be an application that from x leads to $y = [y_1, y_2, \ldots, y_n] \in X$. Then the neighbourhood $Y_x \subset X$ of x is the set of all possible images y of string x for the application f.

Then, 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. Consequently, neighbourhood methods differ one from another by the application that defines the neighbourhood of a configuration and by the strategy used to update the current solution.

A great variety of neighbourhood optimisation techniques were proposed, such as Simulated Annealing (SA : see Triki et al., 2005), Tabu Search (TS : see Hedar and Fukushima, 2006), threshold algorithms (Ducek and Scheuer, 1990) or GRASP methods (Ahmadi and Osman, 2005)… SA and TS are indeed the most representative examples. Simulated Annealing mimics the physical evolution of a solid to thermal equilibrium, slowly cooling it until this one reaches its lower energy state. Kirkpatrick et al. (1982) studied the analogy between this process and an optimisation procedure. A new state, or solution, is

accepted if the cost function decreases or if not, according to a probability depending on the cost increase and the current temperature.

Besides, Tabu Search tackles a group of neighbours of a configuration s and keeps the best one s' even if it deteriorates the objective function. Then, a tabu list of visited configurations is created and updated to avoid cycles like $s \longrightarrow s' \longrightarrow s \ldots$ Furthermore, specific procedures of intensification or diversification allow respectively to concentrate the search on most promising zones or either to guide the search towards unexplored regions.

The second class of metaheuristics consists of evolutionary algorithms. They are based on the principle of natural evolution as stated by Darwin and involve three essential factors : (i) a population of solutions to the considered problem ; (ii) an adaptation evaluation technique of the individuals ; (iii) an evolution process made up of operators reproducing elimination of some individuals and creation of new ones (through crossover or mutation). This leads to an increase in the average quality of the solutions in the last computed generations.

The most used techniques are Genetic Algorithms (GAs), Evolutionary Strategies and Evolutionary Programming. Section 4 presents in detail the GA adopted within this investigation. It just must be pointed out at this level that until recently, a large number of contributions shows how their efficiency can be improved (André et al., 2001). The second technique, commonly said $(\mu+\lambda)$-ES, generates λ children from μ parents and a selection step reduces the population to μ individuals for the following iteration (Beyer and Schwefel, 2002). Finally, Evolutionary Programming is based on an appropriate coding of the problem to be solved and on an adapted mutation operator (Yang et al., 2006).

To summarize, as regards to metaheuristics performances, their efficiency is generally balanced by two opposite considerations : on the one hand, their general procedures are powerful enough to search for an optimum without much specific information of the problem, like in a "black box" context. But the *No Free Lunch* theory shows that no general method can overtake performances of all the other ones and for all problems. So, on the other hand, metaheuristics performances can be improved by integrating particular knowledge of the studied problem and this specialization means, of course, an adaptation effort.

2.2. Optimal Batch Plant Design problems

Due to the growing interest for batch operating mode, a lot of studies deal with the batch plant design issue. Actually, the problem was already modelled under various forms for which

assumptions are more or less simplistic. Generally, the objective consists in the minimization of plant investment cost.

Grossmann and Sargent (1979) proposed a simple posynomial formulation for multiproduct batch plant. Kocis and Grossmann (1988) then used the same approach to validate the good behaviour of a modified version of the Outer Approximation algorithm. This model involved only batch stages and was subjected to a constraint on the total production time. Modi and Karimi (1989) modified this MINLP model by taking into account, in addition, semi-continuous stages and intermediate finite storage with fixed location. They solved small size examples (up to two products and to eight operating stages) with heuristics. The same model was used again by Patel et al. (1991) who treated larger size examples with Simulated Annealing and by Wang et al. (1996, 1999, 2002) who tackled successively Genetic Algorithms, Tabu Search and an Ants Foraging Method. Nevertheless, Ponsich et al. (2005) showed that, for this mixed continuous and discrete formulation, and independently from the size of the studied instance, a Branch-and-Bound technique proves to be the most efficient option. This Mathematical Programming (MP) technique is implemented in the *SBB* solver which is available in the *GAMS* modelling environment (Brooke et al., 1998).

Besides, the above mentioned formulations were further improved by taking into account continuous process variables (Pinto et al., 2001) or uncertainties on product demand modelled by normal probability distributions (Epperly et al., 1997) or by fuzzy arithmetic concepts embedded in a multiobjective GA (Aguilar-Lasserre et al., 2005). However, those sophistication levels were not considered in the framework of the presented study.

2.3. Methodology

This paper is dedicated to the treatment of MINLP problems by a Genetic Algorithm. The case study is a typical engineering problem, involving mixed integer variables and constraints. Even though the used stochastic technique is initially devoted to deal with discrete variables, it was applied yet to a large number of either continuous or mixed integer optimisation problems. But the crucial issue is the necessary adaptation effort to integrate the treatment of real variables and the efficient handling of the constraints.

The support of this work consists in several instances of the optimal batch plant design problem and this investigation aims at testing and evaluating various operating modes of the GA. As shown in previous works, (deterministic) MP methods proved to be the most efficient

for the considered model. Thus, the results are compared with the optimal solutions provided by the above mentioned *SBB* solver. The variables encoding issue is studied on three different size examples : the first one is a small size example but quite difficult to solve to global optimality. The two other ones are larger size instances. The constraint handling problem is analysed by tackling a medium size example in order to force the Genetic Algorithm to cope with a quite complex problem, without being restricted by computational time.

3. Optimal 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 high added-value products in low volumes.

3.1. Problem presentation

Basically, batch plants are composed of items operating in a discontinuous way. Each batch then visits a fixed number of equipment items, as required by a given synthesis sequence (so-called production recipe). Since a plant is flexible enough to carry out 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, which means that all the products follow the same operating steps. Only the operating times may be different from a recipe to another one.

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

3.2. Assumptions

The model formulation for OBPD problems adopted in this paper is based on Modi's approach (Modi and Karimi, 1989), modified by Xu et al. (1993). It considers not only treatment in batch stages, which usually appears in all types of formulation, but also represents semi-continuous units that are part of the whole process (pumps, heat

exchangers…). A semi-continuous unit is defined as a continuous unit working alternating idle times and normal activity periods.

Besides, this formulation takes into account mid-term intermediate storage tanks. They are just used to divide the whole process into sub-processes in order to store an amount of materials corresponding to the difference of each sub-process productivity. This representation mode confers to the plant a better flexibility for numerical resolution : it prevents the whole process production from being paralysed by one limiting 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 figure 1 for the sake of illustration.

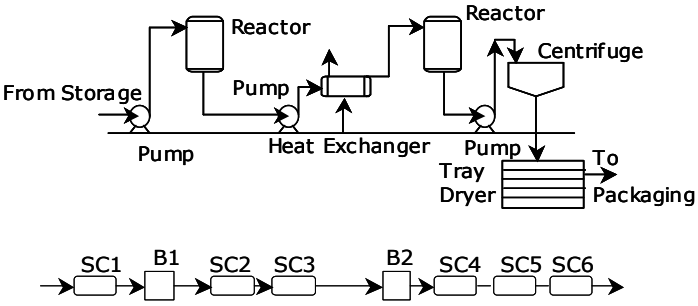

Figure 1. Typical batch plant and modelling

The model is based on the following assumptions:

(i) The devices used in a same production line cannot be used again by the same product ;

(ii) The production is achieved through 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 tank sizes are finite ;

(v) If a storage tank exists between two stages, the operation mode is "Finite Intermediate Storage". If not, the "Zero-Wait" policy is adopted ;

(vi) There is no limitation for utility ;

(vii) The cleaning time of the batch items is included into the processing time ;

(viii) The size of the items 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 with same size V_j. Each semi-continuous stage consists of n_k out-of-phase parallel items with same processing rate R_k (i.e. treatment capacity, measured in volume unit per time unit). The item

sizes (continuous variables) and equipment numbers per stage (discrete variables) are bounded. The *S-1* storage tanks, with size V_s*, divide the whole process into *S* sub-processes.

Following the above mentioned notations, a MINLP problem can be formulated, minimizing the investment cost for all items. This cost is written as an exponential function of the units size :

$$MinCost=\sum_{j=1}^{J} a_j m_j V_j^{\alpha_j} + \sum_{j=1}^{K} b_k n_k R_k^{\beta_k} + \sum_{j=1}^{S-1} c_s V_s^{\gamma_s} * \quad (1)$$

where α_j and a_j, β_k and b_k, γ_s and c_s are classical cost coefficients. Equation (1) shows that there is no fixed cost coefficient for any item. This may be few realistic and will not help to tend towards the minimization of the equipment number per stage. Nevertheless, this formulation was kept unchanged in order to compare our results with those found in literature (see table 1 in section 5.1).

This problem is subjected to three kinds of constraints :

(i) Variable bounding :

$$\forall j \in \{1,..,J\} \qquad V_{min} \le V_j \le V_{max} \quad (2)$$

$$\forall k \in \{1,..,K\} \qquad R_{min} \le R_k \le R_{max} \quad (3)$$

(ii) Time constraint : the total production time for all products must be lower than a given time horizon *H*.

$$H \ge \sum_{i=1}^{I} H_i = \sum_{i=1}^{I} \frac{Q_i}{Prod_i} \quad (4)$$

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*).

$$\forall i \in \{1,..,I\} \qquad Prod_i = \min_{s \in S} [Prodloc_{is}] \quad (5)$$

These local productivities are calculated from the following equations :

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

$$\forall i \in \{1,..,I\}; \forall s \in \{1,..,S\} \quad Prodloc_{is} = \frac{B_{is}}{T_{is}^L} \quad (6)$$

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

$$\forall i \in \{1,...,I\}; \forall s \in \{1,...,S\} \qquad T_{is}^L = \max_{j \in J_s} [T_{ij}, \theta_{ik}] \quad (7)$$

where 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* :

$$\forall i\in\{1,...,I\};\forall j\in\{1,...,J\} \qquad T_{ij}=\frac{\theta_{ik}+\theta_{i(k+1)}+p_{ij}}{m_j} \tag{8}$$

where 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 :

$$\forall i\in\{1,...,I\};\forall j\in\{1,...,J_s\};\forall s\in\{1,...,S\} \qquad p_{ij}=p_{ij}^{0}+g_{ij}B_{is}^{d_{ij}} \tag{9}$$

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

$$\forall i\in\{1,...,I\};\forall k\in\{1,...,K_s\};\forall s\in\{1,...,S\} \qquad \theta_{ik}=\frac{B_{is}D_{ik}}{R_k n_k} \tag{10}$$

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

$$\forall i\in\{1,..,I\};\forall s\in\{1,..,S\} \qquad B_{is}=\underset{j\in J_s}{Min}\left[\frac{V_j}{S_{ij}}\right] \tag{11}$$

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

$$\forall s\in\{1,..,S-1\} \qquad V^{*}=\underset{i\in I}{Max}\left[S_{is}\,Prod_i(T_{is}^{L}+T_{i(s+1)}^{L}-\theta_{ik}-\theta_{i(k+1)})\right] \tag{12}$$

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 (Wang et al., 1996).

4. Proposed Genetic Algorithm

The aim of this section is not to present in detail the stochastic optimisation technique initiated by Holland (1975). The following comments just recall its basic principles and then focus on the specific parameters used in this study.

4.1. General principles

The principles of GAs just lie on the analogy between a population of individuals and a set of solutions of any optimisation problem. The algorithm makes the solution set evolve towards a good quality, or adaptation, and mimics 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 features that make these individuals "stronger" will be preserved generation after generation.

The mechanisms implemented in the GAs reproduce this natural behaviour. Good solutions are settled by creating selection rules, that will state whether the individuals are

adapted or not to the considered problem. Crossover and mutation operators then contribute to the population evolution in order to obtain, at the end of the run, a population of good quality solutions. This heuristics set is mixed with a strong stochastic feature, leading to a compromise between exploration and intensification in the search space, which contributes to GAs efficiency.

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

As mentioned before, two main features of GAs implementation are still being a challenge for GAs performances : constraint handling and variables encoding. These two points are presented in the following sub-sections.

4.2. Constraint handling

Since constraints cannot be easily implemented just by additional equations, as in MP techniques, their handling is a key-point of GAs. Indeed, an efficient solution will widely depend on the correct choice of the constraint handling technique, in terms of both result quality and computational time.

In the framework of the studied problem, the constraint on variable bounds is intrinsically considered in the variable encoding while the constraint on productivities is implemented in the model. So the only constraint to be explicitly handled by the GA is the time constraint formulated in equation (4), which imposes the I products to be synthesized before a time horizon H.

Actually, the most obvious approach would be to lay down the limits of the feasible space through the elimination all the solutions violating any constraint. That means that only feasible solutions should be generated for the initial population. Then, the more severe the constraints are, the more difficult it is to randomly find one feasible solution. So, the effect of this technique on the computational time is strongly penalizing. Furthermore, the search efficiency would greatly benefit from getting information of the infeasible space. Then, allowing some infeasible solutions to survive the selection step should be considered.

This was performed in various alternative constraint handling modes. Thorough reviews are proposed by Coello Coello (2002a) and Michalewicz et al. (1996). The most famous technique is the penalisation of infeasible individuals, which is typically carried out by adding, in the objective function, the quadratic constraint violation weighted by a penalty factor. This factor can be either static (i.e. set to a fixed value along the whole search), dynamic (increasing with the generation number), or set by analogy with simulated annealing (for more details see Michalewicz and Schoenauer, 1996). The drawback due to the necessity of tuning at least one parameter (the penalty factor or its initial value) can be overcome with self-adaptive penalty approaches (Coello Coello, 2002b), yet associated with high computational costs. Some alternative options for constraint handling are based on domination concepts, drawn from multiobjective optimisation. They are implemented either within roulette wheel (Deb, 2000) or tournament Coello Coello and Mezura Montes (2002) selection methods.

According to these investigated literature references, the following methods were evaluated in this paper :

- Elimination as a reference.
- Penalisation of the feasible individuals objective function as shown in equation (13) :

$$F = CostFunction + \rho(H - \sum_i H_i)^2 \qquad (13)$$

  where ρ is a penalization factor. H and H_i are respectively the fixed horizon time and the production time for product i, from equation (4).

  A static penalty factor is used in this study, for implementation simplicity reasons. Obviously, this technique efficiency greatly depends on the value of the ρ factor in the added penalisation term. Its value was thus the object of a sensitivity analysis.
- Relaxation of the discrete variables range. By setting the discrete upper bounds to a greater value, this technique just means an enlargement of the feasible space : the minimization should anyway make the variables tend within their bounds.
- Tournament based on domination rules. This method, applied in the selection step of the GA, relies on domination rules stated in Coello Coello and Mezura Montes (2002) and Deb (2000). Basically, it is stated that : (i) a feasible individual dominates an infeasible one ; (ii) if two individuals are feasible, the one with the best objective function wins ; (iii) if two individuals are infeasible, the one with the smallest constraint violation wins. These rules enable the selection of the winners among some randomly chosen

competitors. Various combinations of competitors and winners were tested. A special case of this method, namely single tournament (ST), 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 realization for each selection step.

Specific selection procedures were implemented according to the above mentioned techniques. For the elimination, penalisation and relaxation techniques, the selection is performed via Golberg's roulette wheel (Goldberg, 1989). This procedure implies the evaluation of the adaptation or *strength* of each individual. This one is computed as being the difference between the highest value of the objective function in the current population and that of the considered individual i : $strength_i = fmax - f_i$. This method shows the advantage of being adapted to the operating way of GA, i.e. maximizing the criterion. Besides, the last constraint handling method is applied in the selection step itself, which is carried out by tournament.

4.3. Variables encoding

The way the variables are encoded is clearly essential for GAs efficiency. In what follows, three kinds of encoding, which show an increasing adaptation to the continuous context but share some general features, are presented. 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. A difference will be made between genes encoding continuous variables from those encoding discrete ones. Since the formers are bounded, they can be written in a reduced form, like a real number α bounded within 0 and 1. Each integer variable is coded directly in a single-locus gene, keeping it unchanged.

Therefore, the various encoding techniques differ one from another by the way the continuous variables are represented and by the gene location all along the chromosome.

4.3.1. Rough discrete coding

The first encoding method tested in this study consists in discretising the continuous variables, i.e. the above mentioned α. According to the required precision, a given number of decimals of α is coded. This will logically have an effect on the string length, and consequently on the problem size. The so-called weight box (Montastruc, 2003) was used in

this study (figure 2) : each decimal is coded by four bits b_1, b_2, b_3, b_4, weighting respectively 1 2, 3, 3. As an example, the value of decimal d of α is given by the following expression :

$$d=b_1*1+b_2*2+b_3*3+b_4*3 \quad (14)$$

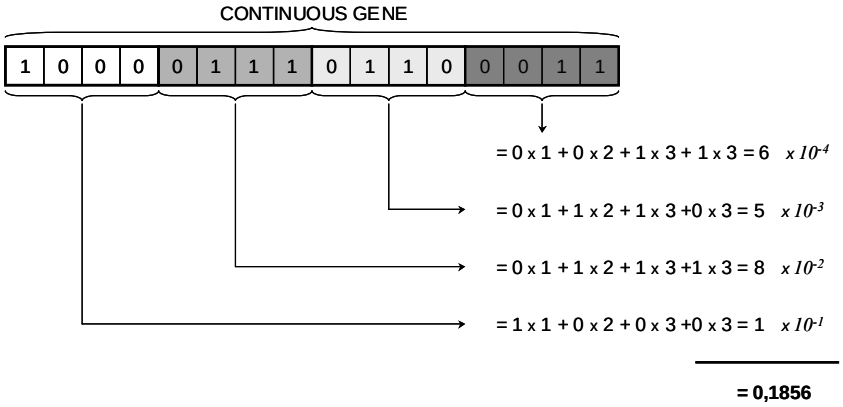

Figure2. Coding example with the weight box

This method enables to prevent from bias since the sum of all weights is lower than ten. Besides, the duplication of threes in the weighted box means that there exists various ways to encode a same number. This means that probabilities to get a given number have a weight. Concerning the variable position, all the continuous variables are located in the first part of the string, while the discrete genes are positioned at the bottom. The resulting configuration of a chromosome is shown in figure 3, for a small size example.

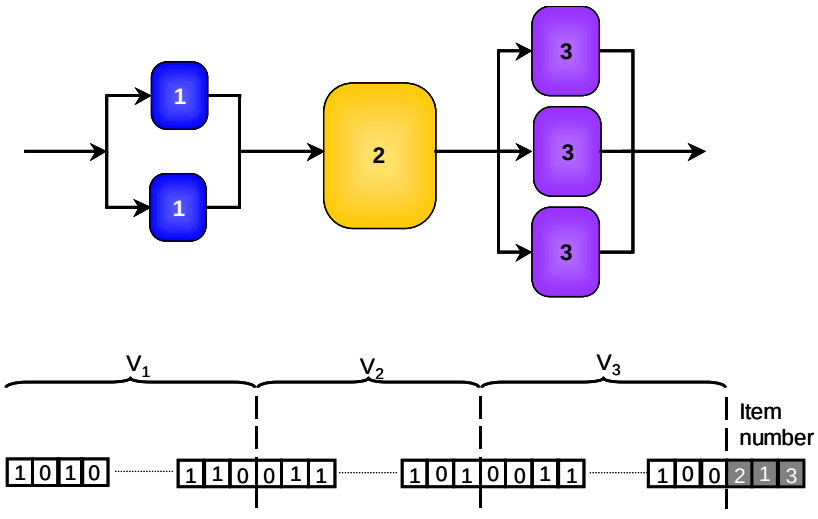

Figure 3. Variable location in the chromosome, Coding 1

Obviously, the crossover and mutation operators are to be adapted to the coding configuration. The crossover is implemented by a single cut-point procedure. But 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). The latter technique is not a symmetric mutation operator, thus it cannot prevent the algorithm from being trapped in some local optimum. However, it proved to lead efficiently towards minimization.

4.3.2. Crossed discrete coding

The discretisation of the continuous variables is unchanged with regard to the previous case, i.e. with the weight box method. The two methods only differ by the variables location. This change is suggested by the respective size of continuous and discrete parts of the chromosome. Indeed, because of the required precision for continuous variables, the former is much larger than the latter (in our case, the ratio equals 16 to 1). So, the crossover operator with a single cut-point procedure has it very difficult to act on the discrete variable, and, consequently, to allow a correct exploration of the search space.

In order to deal with this contingency, the continuous and discrete variables were mixed up inside the string : since each processing stage induces one continuous and one discrete variable (respectively the size and number of items), the variables are encoded respecting the order of the operating stages in the recipe, as shown in figure 4 for the same illustrative example.

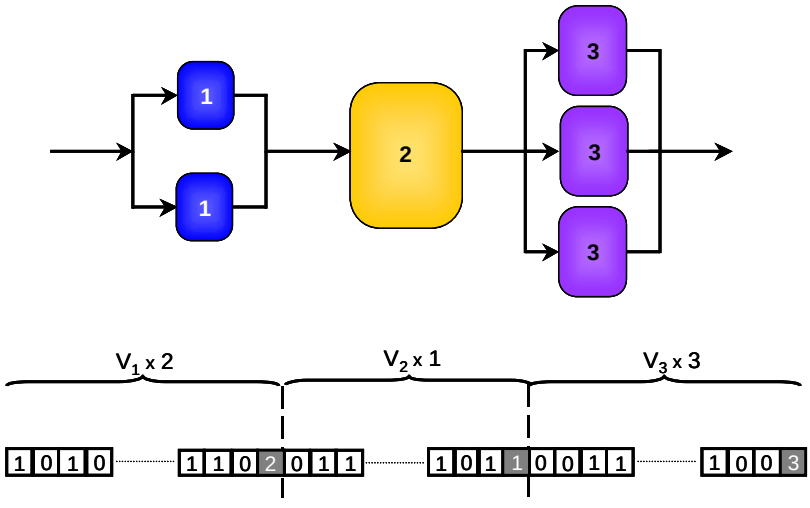

Figure 4. Variables position in the chromosome, Coding 2

Obviously, this new encoding method strengthens probabilities that discrete genes may be involved by the one cut-point crossover process. However, due to the size difference between continuous and integer genes, there is still much more opportunities that the former ones were directly concerned by crossover.

4.3.4. Mixed real-discrete coding

The last coding method seems to be the best fitted to the nature of the variables. The reduced form α of continuous variables is coded directly on a real-value locus while, as in the previous cases, the discrete variables are still kept unchanged for their coding. Therefore, both continuous and discrete genes have a one-locus size and occupy well-distributed lengths inside the chromosome (figure 5). A mixed real-discrete chromosome is obtained, that will require specific genetic operators.

Firstly, the crossover methods applied in real-coded GAs works on each gene and not on the global structure of the chromosome. The simplest method relies on arithmetical combinations of parent genes, such as it is presented in equation (15) :

$$\begin{cases} y_k^{(1)} = \alpha . x_k^{(1)} + (1-\alpha) . x_k^{(2)} \\ y_k^{(2)} = (1-\alpha) . x_k^{(1)} + \alpha . x_k^{(2)} \end{cases} \quad (15)$$

where $x_k^{(1)}$ and $x_k^{(2)}$ are genes k of both parents and $y_k^{(1)}$ et $y_k^{(2)}$ those of the resulting children. α is a fixed parameter within 0 and 1. Then, different methods are implemented according to the way of determining the α parameter (Michalewicz and Schoenauer, 1996). The chosen technique is a simulated binary crossover (SBX), proposed by Deb and Agrawal (1995). The method consists in generating a probability distribution around parent solutions to create two offspring.

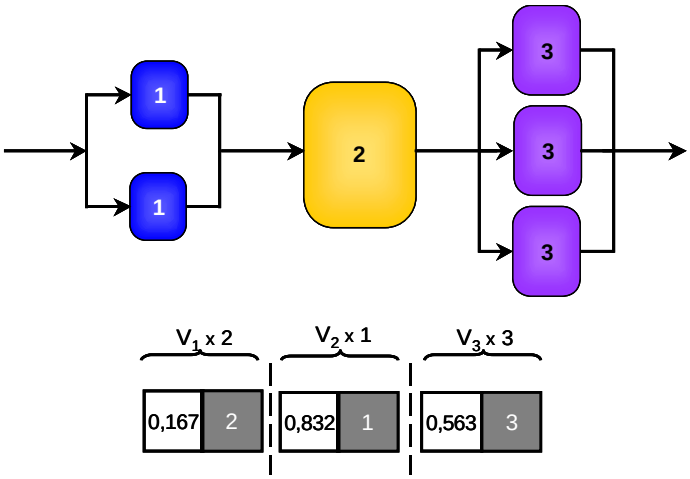

Figure 5. Configuration of the chromosome, Coding 3

This probability distribution is chosen in order to mimic single-point crossover behaviour in binary coded GAs, and mainly involves the two following features :

- The mean decoded parameter value of two parents strings is invariant among the resulting children strings.
- If the crossover is applied between two children strings at the same cross site as used to create the children strings, the same parents strings will result.

This feature generates higher probabilities to create offspring close to the parents than away from them, as illustrated in figure 6. The procedure for the generation of two children solutions from two parents solutions is fully explained in Deb and Agrawal (1995). It is to note that this crossover procedure is carried out for each locus of the chromosome and as a consequence, the disposal of the continuous and discrete along the string does not matter. However, even though the SBX crossover does not induce any problem for real variables, it may lead to a real value for the discrete genes of the resulting offspring. So, for the latter case, these real values were truncated in order to keep only their integer part.

With respect to mutation, on the one hand, the method corresponding to discrete variables was kept unchanged from the previous cases. On the other hand, for real-coded genes, an inventory of the variety of techniques is proposed in Michalewicz and Schoenauer (1996) or Raghuwanshi and Kakde (2005). They usually rely on a noise added on the initial muted gene value, according a specific probability distribution. For the technique used in this study, a uniform distribution probability was chosen.

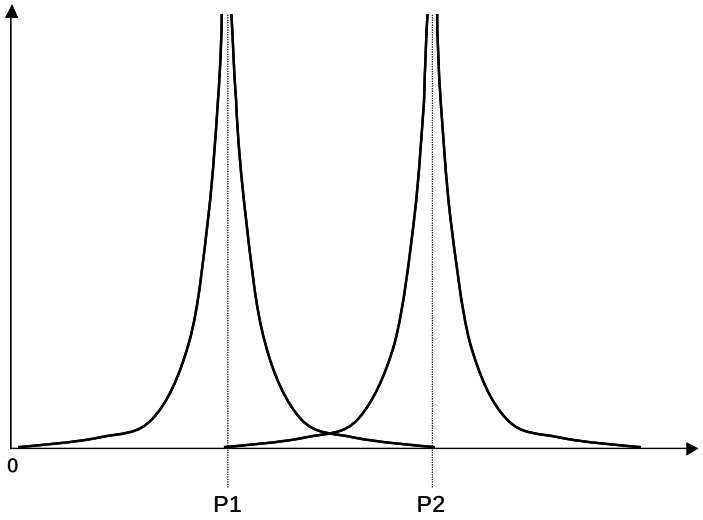

Figure 6. Probability distribution for the location of an offspring, SBX crossover

5. Numerical results and interpretation

In this section, the two main issues, i.e. variables encoding and constraint handling, are treated on different size instances of Optimal Batch Plant Design problems.

5.1 About variable encoding

The above mentioned procedures for variable encoding were tested on three examples. Problem 1 is a quite small size example. The plant has to synthesize three products and is constituted of four batch stages, six semi-continuous stages, and one storage tank. Thus, the problem involves ten continuous and ten integer optimisation variables. It was previously studied with various techniques as shown in table 1, that highlights the fact that Mathematical Programming locates the best solution, for which optimality is proved.

Despites its small size, this example turned out to be difficult to solve to optimality since some of the mentioned stochastic techniques were trapped in a local optimum, showing a set of discrete values different from the one of the global optimal solution. The optimal values of the discrete variables are actually $m_j = \{1, 2, 2, 1\}$ parallel items for batch stages, while those of the local optimum are $m_j = \{1, 3, 3, 1\}$.

The two other examples are larger size instances of batch plants, both manufacturing 3 products. Problems 2 and 3 are constituted of respectively 7 and 18 batch stages, 10 and 24 semi-continuous stages and 3 and 6 sub-processes. They were also solved to optimality by the *SBB* solver in this study.

Table 1 – Typical solutions obtained by various methods

Reference	Optimisation method	Best solution
Patel et al. (1991)	Simulated Annealing	368883
Wang et al. (1996)	GA	362130
Wang et al. (1999)	Tabu Search	362817
Wang et al. (2002	Ants Foraging Method	368858
Ponsich et al. (2005)	Mathematical Programming (*SBB-GAMS* environment)	356610

The parameters chosen for the Genetic Algorithm are the following ones : the survival (respectively mutation) rate is equal to 40 % (resp. mutation 30 %). The maximum generation number and the population size depend on the complexity of the example. Concerning the constraint handling, intuitive choices were adopted : elimination was chosen for problem 1 due to its small size (not much expensive in term of computational time). For the two larger examples, the single tournament method was selected. Table 2 sums up the main features of each problem.

The results were analysed in terms of quality and computational time. The number of function calls could also be studied, but the time criterion appeared to be more significant in order to check the influence of the variable encoding methods. The CPU time was measured for a Compaq Workstation W6000.

Table 2 - Characteristics of the problems and solution by GA

	Problem 1	Problem 2	Problem 3
Cont. / Discrete variables	10 / 10	17 / 17	42 / 42
Optimum	356610	766031	1925888
GA Parameters			
Population size	200	200	500
Generations number	200	500	500
Cst. Handling	Elim.	Single Tour.	Single Tour.

Quality is evaluated, of course, by the distance between the best found solution and the optimal value. But, since GAs is a stochastic method, its results have to be analysed also in terms of repeatability. So, for each test, the GA was run 100 times. The criterion for repeatability evaluation is the dispersion of the runs around GA best solution F^*_{GA}. The 2%-dispersion or 5%-dispersion are then defined as the percentage of runs providing a result lying respectively in the range $[F^*_{GA}, F^*_{GA}+2\%]$ or $[F^*_{GA}, F^*_{GA}+5\%]$.

The results for problem 1 are presented in table 3. Coding 1, 2 and 3 represent respectively rough discrete, crossed discrete and mixed real-discrete coding. Clearly, the solution obtained with the mixed real-coding is much better than the other ones : indeed, the optimal solution previously determined by the *SBB* solver is almost exactly located. GA with coding 1 stays trapped in a local optimum and was not able to find the set of discrete variables corresponding to the global optimum. Although it finds a bit lower solution, GA with coding 2 does not show much more efficiency. Besides, the 2% and 5%-dispersions seem to be really much superior for the two first encoding techniques but this trend is due to the high quality of the best solution found with coding 3.

Table 3 - Comparison of variables encoding for Problem 1

	Coding 1	Coding 2	Coding 3
GA best solution	371957	369774	356939
Gap to optimum (%)	4.30	3.69	0.09
2%-dispersion	68	67	1
5%-dispersion	100	98	79
CPU time (sec.)	3	3	1

Actually, the difference between relative and absolute results quality is stressed by the previous remark. A run set might show very good 2%- and 5%-dispersions, if the best result is far from the optimum, then we get a poor global quality of the runs. On the other hand, a run set with lower dispersions but a better final result may be more performing. This remark is illustrated in figure 7, in which case 1 shows a better relative quality than case 2, which is characterized by a better absolute quality.

Thus, by considering now a 5%-dispersion with the optimal solution as a reference (this means calculating an absolute quality), the value is equal to 50 %, 48 % and 77 %, for codings 1, 2 and 3 respectively. This clearly highlights the good global quality of the GA runs performed with coding 3.

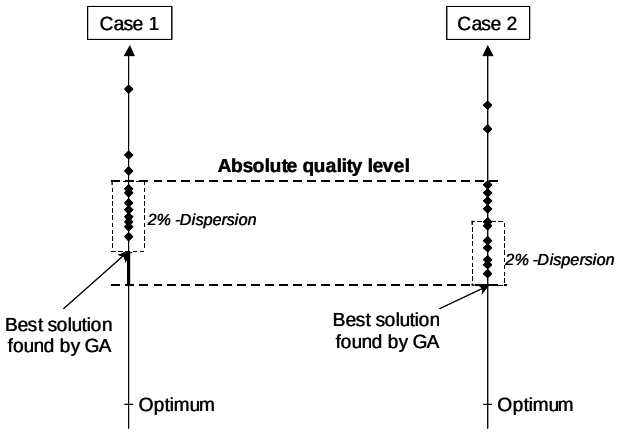

Figure 7. Example of absolute and relative quality of numerical results

The computational time is not significant on such a small example. The number of function evaluations lies around $5{,}5.10^4$ for all problems. The population size and generations number are both equal to 200, which would mean 4.10^4 evaluations per run, this shows that more than 25 % of the search is spent in randomly looking for an initial population of feasible solutions. Finally, the conclusion for this small size but complex problem is the superiority of mixed real-discrete coding.

The results for problems 2 and 3 are presented in table 4. It can be observed that the three coding methods find results very close to the optimum. With regard to dispersion evolution, figure 8 shows the increasing superiority of coding 3 from example 1 to example 3. This behaviour does not seem useful for such simple examples, since the final solutions are quite similar, but it may be interesting for more severely constrained problems, for which the feasible space is reduced : it might be assumed that better solutions could be found, or at least more easily. This would then mean a minor necessity to use large population sizes and generations number and consequently, a lower computational time.

For codings 1 and 2, the quality of all the runs can be related to the percentage of feasible solutions in the last generation and to the number of failures of the GA runs. It is considered that a GA run fails when no feasible solution was found during the whole search. It is clear that, however, this failure number compensates slightly the dispersion fall for encoding 1 and 2, since the percentage of runs finding a result close to the best found solution is based on the total number of runs and not on the number of unfailed runs.

Table 4 - Comparison of variables encoding for Problems 2 and 3

	Problem 2	Problem 3

	Cod. 1	Cod. 2	Cod. 3	Cod. 1	Cod. 2	Cod. 3
GA best solution	770837	771854	767788	1986950	1997768	1975027
Gap to optimum (%)	0.63	0.76	0.23	3.17	3.73	2.55
% feas. solutions (end search)	65	67	54	41	59	52
% failures	0	0	0	37	10	0
CPU time (sec.)	17	17	3	126	126	22

It is to note that for examples 1 and 2, coding 1 is more or less as performing as coding 2. With an increasing problem size, the difficulty to act on discrete variables – which are located at the bottom of the chromosome in coding 1 – increases but there is not a high difference between the results provided by the two encoding methods. Only the dispersions seem to slightly favour coding 2 for problem 3.

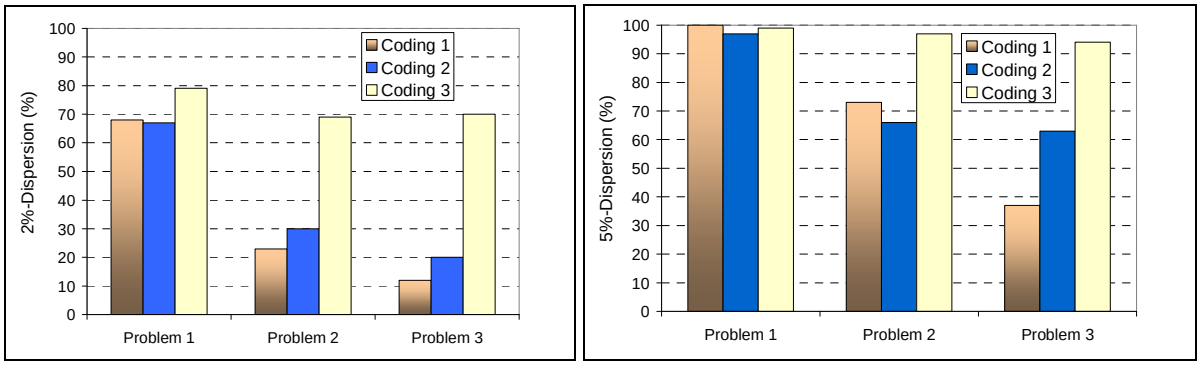

Figure 8. Dispersion evolution for the three problems, different coding methods

Finally, the comparison criterion based on CPU time highlights coding 3 performances, since a run of this GA version lasts almost 6 times fewer than coding 1 or 2. As a conclusion on variable encoding selection, numerical results prove the superiority of the mixed real-discrete encoding method. This one will be used in the following section.

5.2 About constraint handling

The constraint handling techniques described in Section 4 are now tested on problem 2.

5.2.1. Elimination of infeasible individuals

The results in table 5 present trends for GA with the elimination technique. These can be seen as a reference in order to evaluate the performance of different constraint handling methods : they show a very good quality in terms of both distance to the optimum and

dispersion of the runs. However, this technique looks really less performing considering the computational time : this feature could constitute a bottleneck for applications that would compute the objective function with a simulator.

Actually, the comparison with the behaviour of other techniques that do not need a feasible initial population, like tournament or penalisation, highlights that the GA with the elimination method has spent most of the computing time randomly searching for feasible solutions. Indeed, for a 1000 generation run, involving a population constituted by 200 individuals, the theoretical function evaluation number is 2.10^5, but it turns out to be approximately $2.35\ 10^6$. So, it can be deduced that less than 10 % of the computing time is used for the GA normal sequence while the remaining is devoted to the initial population generation. For all the other techniques, the computational time has an order of magnitude of six seconds.

Table 5 - Results for elimination technique

GA best solution	767182
Gap to optimum (%)	0.15
2%-dispersion	95
5%-dispersion	100
Function evaluation nbr	$2.346\ 10^6$
CPU time (sec.)	56

5.2.2. Penalisation of infeasible individuals

This study was carried out for different values of the penalisation factor ρ. The results presented in table 6 underline the logical results of the method. On the one hand, for small values of ρ, priority is assigned to the minimisation of the economic term while the time constraint is severely violated. The best solution found is then widely infeasible. On the other hand, for higher values of the penalisation factor, the result is feasible while the dispersion and optimality gap criteria keeps being satisfying.

Table 6 - Results for penalisation technique

ρ factor	100	1	0.01
GA best solution	769956	766520	592032
Gap to optimum (%)	0.51	0.06	-22.71
Constraint violation (%)	0	0.64	37.4
2%-dispersion	44	54	68
5%-dispersion	85	89	96

% feas. solutions (end search)	16	3	0

Finally, a compromise solution can be reached with intermediate values of ρ. Actually, by giving the same weight to respect of the time constraint and to the minimisation of the investment cost, the global performances can be improved with solutions slightly exceeding the time horizon. Anyway, for all cases, the number of feasible solutions keeps being quite low.

5.2.3. Relaxation of discrete upper bounds

In this section, the upper bounds of discrete variables, i.e. the maximal number of parallel items per stage, is doubled and fixed to six. As shown in table 7, the best result is similar to that previously obtained with the elimination technique. The dispersions fall with regard to the elimination technique but keep being acceptable. The ratio of feasible solutions at the end of the search proves the good behaviour of GA, which makes the discrete variables to tend within their initial bounds. Moreover, the CPU time is considerably reduced with regard to the elimination technique, showing the efficiency of relaxation to avoid the wasted time spent in generating the initial solution. Indeed, the function evaluation number is almost equal to the standard number *generation number* x *population size*.

So, the discrete upper bounds relaxation really appears to be well-suited technique for constraint handling. Actually, for larger size problems, the problem is how to determine the order of magnitude of the upper bound relaxation. On the one hand, the relaxation should be sufficient to easily create the initial population. On the other hand, a too high increase would cancel the necessary pressure that pushes the individuals towards the initial discrete feasible space, i.e. that leads to a minimization of the parallel item number.

Table 7 - Results for relaxation technique

GA best solution	767361
Gap to optimum (%)	0.17
2%-dispersion	58
5%-dispersion	95
% feas. solutions (end search)	53
Function evaluation nbr	$2.196\ 10^5$
CPU time (sec.)	7

5.2.4. Domination based tournament

This method was applied for various combinations of competitors *Ncomp* and survivors *Nsurv*, referred as (*Ncomp*, *Nsurv*)-tournaments in the following, except for the single tournament technique (ST). The tested combinations and their corresponding results are available in table 8. For all cases, the computational time is equal to 6 seconds, that is almost ten times faster than the elimination technique.

Table 8 - Results for various tournament techniques

Tournament version	(2,1)	(3,1)	(3,2)	(4,1)	(4,2)	(5,1)	(5,2)	(5,4)	ST
GA best solution	767450	767334	767900	768228	767587	768907	767981	767955	767788
Gap to optimum	0.19	0.17	0.24	0.29	0.20	0.38	0.25	0.25	0.25
% feas. solutions	34	47	23	53	42	54	48	10	55
% failures	0	0	1	0	0	0	0	15	0

The best results are few significant since they are similar and near-optimal for all kinds of tested tournaments. But, due to the low number of feasible solutions obtained at the end of the search, the (2,1), (3,2) and (5,4)-tournaments can be discarded. This behaviour can be easily explained by the following assumption : the smaller the difference between *Ncomp* and *Nsurv* is, the easier it is to pass the selection step for weak individuals, with a poor objective function. This underlines the efficiency of a more severe selection pressure, which is furthermore confirmed by the important number of failures of the (5,4) version.

To evaluate the other options, the remaining criteria are the evolution of feasible solutions ratio during the search and the dispersions of the runs. The corresponding numerical results are given respectively in figures 9 and 10. On the one hand, it can be deduced that the combinations visiting more feasible solutions are single tournament, (5,1) and (4,1)-tournaments.

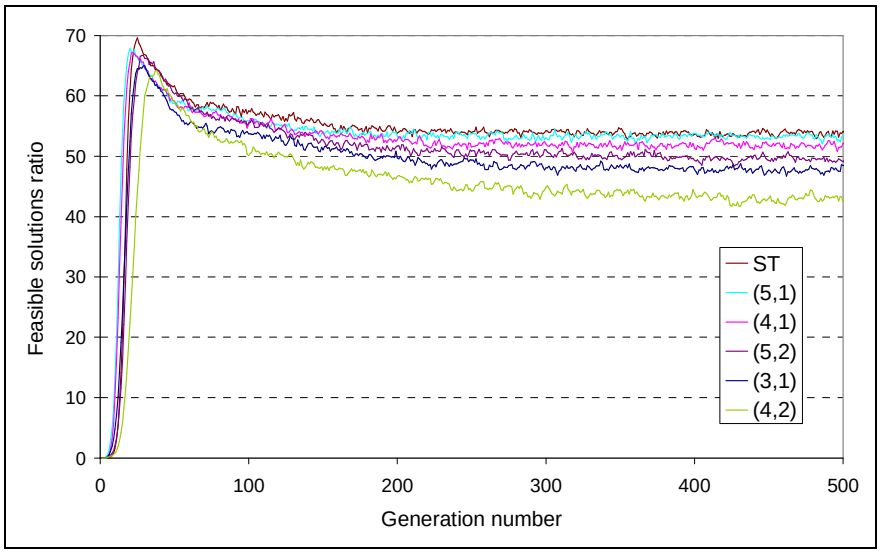

Figure 9. Evolution of the ratio of feasible solutions
(Curves in the order of legend)

On the other hand, figure 10 shows that (4,2) and (5,2)-tournaments as well as single tournament are the most performing combinations in terms of 2%- and 5%-dispersion. So, even though several combinations are very close in terms of result quality, the single tournament method proves to be the best compromise, closely followed by the (4,1) option.

Finally, for this example, the single tournament and relaxation methods are the most well-fitted constraint handling methods. It is must be yet pointed that the relaxation method needs the relaxed upper bound as a parameter that requires a relevant choice, achieved with some knowledge of the studied problem. So, to conclude this study and highlight the general trends, the single domination-based tournament technique appears to be the most efficient constraint handling technique for Genetic Algorithms.

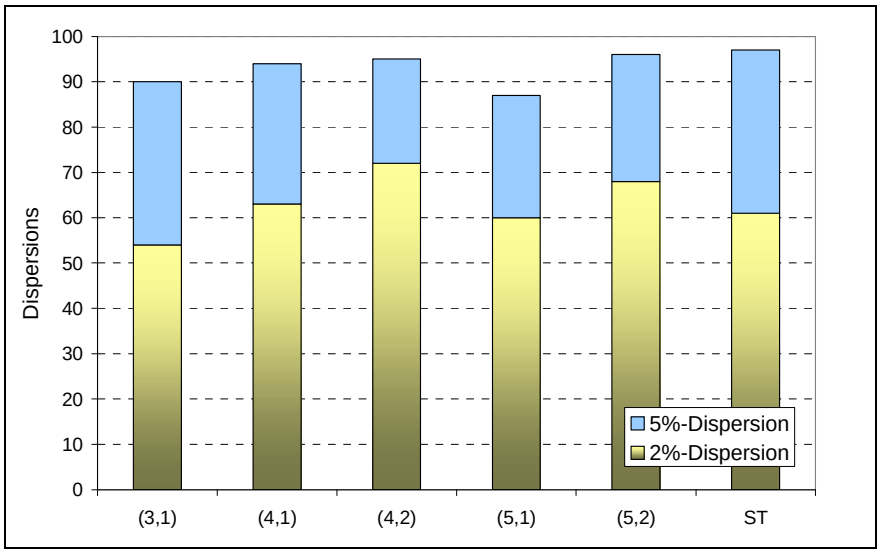

Figure 10. Dispersions of the run for various tournament versions

This conclusion is valid for medium and large size problems. Nevertheless, it is obvious that when the computational time is not a limiting factor, the preference will be given to an elimination technique.

6. Conclusions

A Genetic Algorithm was adapted to the solution of Optimal Batch Plant Design problems. This work considered several operating modes of classical GA operators in order to determine the best fitted to the studied problem.

Since the problem involves an MINLP formulation, the first main issue was the evaluation of different variable encoding techniques, that deals efficiently with both continuous and integer variables. Computing tests were carried out on three different sizes bench problems, in order to compare the behaviour of rough discrete, crossed discrete and mixed real-discrete coding methods. The result quality was evaluated with regard to the optimum found by a MP technique. The mixed real-discrete method proved to be the most relevant option, since it provided nearly optimal results for all examples with a very performing computational time. Moreover, it overcame some local optimal difficulties while the two other techniques were trapped in sub-optimal solutions.

The second investigation was devoted to constraint handling techniques, namely here, the production time constraint.. Four constraint handling methods were tested on a medium size problem, that are elimination, penalisation, relaxation of the discrete upper bounds and

dominance based tournament. Elimination, leading to very performing results in term of quality but fewer efficient in term of computational time, is recommended for small size example treatments. The use of a penalisation technique, severely depending on the appropriate choice of the penalisation factor, does not ensure feasible solutions in the whole search. Nevertheless, it could provide compromise solutions that improve the investment criterion, slightly violating the constraint.

Finally, numerical results showed that the relaxation and tournament methods were the most efficient procedures. However, the relaxation technique depends on the variable physical meaning and may be viewed as less general since it is a parameter-based technique. Dominance rules implemented in the selection step for the single tournament thus reveal to be the best constraint handling technique in case of severely constrained problems.

This contribution has thus proposed some guidelines to tackle the two operating mode issues that constitute limiting factors for GA. The use of these strategies should be appropriate to enable an efficient evolution of the algorithm on different instances of mixed continuous and integer problems.

References

Aguilar-Lasserre, A., Azzaro-Pantel, C., Domenech, S., Pibouleau, L., 2005. Modélisation des imprécisions de la demande en conception optimale multicritère d'ateliers discontinus. Proceedings of SFGP Congress in Toulouse (France), Setpember 20th-22th, Under Press.

Ahmadi, S., Osman, I. H, 2005. Greedy random adaptive memory programming search for the capacitated clustering problem. European Journal of Operational Research 162, 30-44.

Andre, J., Siarry, P., Dognon, P., 2001. An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization. Advances in Engineering Software 32 ,49-60.

Beyer, H.G., Schwefel, H.P., 2002. Evolution Strategies, a comprehensive introduction. Natural Computing : An International Journal 1, 3-52.

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

Coello Coello C.A., 2002a. Theoretical and numerical constraint-handling techniques uses with evolutionary algorithms : a survey of the state of the art. Computers Methods in Applied Mechanical Engineering 191, 1245-1287.

Coello Coello C.A., 2002b. Use of a self-adaptive penalty approach for engineering optimization problems. Computers in Industry 41, 113-127.

Coello Coello, C.A., Mezura Montes, E., 2002. Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. Advanced Engineering Informatics 16, 193-203.

Deb, K., Agrawal, R.B., 1995. Simulated binary crossover for continuous search space. Complex Systems 9, 115-148.

Deb, K., 2000. An efficient constraint handling method for genetic algorithms. Computers Methods in Applied Mecanics and Engineering 186, 311-338.

Dedieu, S., Pibouleau, L., Azzaro-Pantel, C., Domenech, S., 2003. Design and retrofit of multiobjective batch plants via multicriteria genetic algorithm. Computers and Chemical Engineering 27, 1723-1740.

Dietz, A., Azzaro-Pantel, C., Pibouleau, L., Domenech, S., 2005. A framework for multiproduct batch plant design with environmental considerations : application to protein production. Industrial and Engineering Chemistry Research 44, 2191-2206.

Ducek, G., Scheuer, T., 1990. Threshold accepting : a general purpose optimization algorithm. Journal of Computational Physics 90, 161-175.

Epperly, T.GW., Ierapetritou, M.G., Pistikopoulos, E.N., 1997. On the global and efficient solution of stochastic batch plant design problems. Computers and Chemical Engineering 21, 1411-1431.

Golberg, D.E., 1989. Genetic algorithms in search, optimization and machine learning. Addison-Wesley Publishing Company Inc., MA.

Grossmann, I.E., Sargent R.W.H., 1979. Optimum design of multipurpose plants. Industrial Engineering and Chemical Process Design and Development 18, 343-348.

Grossmann, I.E., 2002. Review of nonlinear mixed-integer and disjunctive programming techniques. Optimization and Engineering 3, 227-252.

Hao, J.K., Galinier, P., Habib, M., 1999. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contrainte. Revue d'Intelligence Artificielle.

Hedar, A.R., Fukushia, M., 2006. Tabu search directed by search methods for nonlinear global optimization. European Journal of Operational Research 170, 329-349.

Holland, J.H., 1975. Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, MI.

Kirkpatrick S., Gelatt Jr., C.D., Vecchi, M.P., 1982. Optimization by Simulated Annealing. IBM Research Report RC 9355.

Kocis, G.R., Grossmann, I. E., 1988. Global optimisation of nonconvex mixed-integer non linear programming (MINLP) problems in process synthesis. Industrial Engineering and Chemistry Research 27, 1407-1421.

Lee, S., Grossmann, I.E., 2000. New algorithms for nonlinear generalized disjunctive programming. Computers and Chemical Engineering 24, 2125-2141.

Michalewicz Z., Dasgupta D., Le Riche R.G., Schoenauer M., 1996. Evolutionary algorithms for constrained engineering problems. Computers and Industrial Engineering 30, 851-870.

Michalewicz Z., Schoenauer M., 1996. Evolutionary algorithms for constrained parameters optimization problems. Evolutionary Computation 4, 1-32.

Modi, A.K., Karimi, I. A., 1989. Design of multiproduct batch processes with finite intermediate storage. Computers and Chemical Engineering 13, 127-139.

Montastruc, L., 2003. Développement d'un pilote automatisé, fiable et sûr pour la dépollution d'effluents aqueux d'origine industrielle. PhD Thesis, INP Toulouse, France.

Patel, A.N., Mah, R.S.H., Karimi, I.A., 1991. Preliminary design of multiproduct non-continuous plants using simulated annealing. Computers and Chemical Engineering 15, 451-469.

Pinto, J.M., Montagna, J.M., Vecchietti, A.R., Irribarren, O.A., Asenjo, J.A., 2001. Process performance models in the optimization of multiproduct protein production plants. Biotechnology and Bioengineering 74, 451-465.

Ponsich, A., Azzaro-Pantel, C., Domenech, S,. Pibouleau, L., 2005. About the relevance of Mathematical Programming and stochastic optimisation methods : application to optimal batch plant design problems. Proceedings of ESCAPE Congress in Barcelona (Spain), May 30th - June 1st, Ed. Espuña and Puigjaner, 49-55.

Raghuwanshi, M.M., Kakde, O.G. 2005. Survey on multiobjective evolutionary and real coded genetic algorithms. Proceedings of the 7th International Conference on Adaptative and Natural Computing Algorithms, Coimbra (Portugal), March 21st-23rd.

Teh, Y.S., Rangaiah, G.P., 2003. Tabu search for global optimization of continuous functions with application to phase equilibrum calculations. Computers and Chemical Engineering 27, 1665-1679.

Triki, E., Collette, Y., Siarry, P., 2005. A theoretical study on the behaviour of simulated annealing leading to a new cooling schedule. European Journal of Operational Research 166, 77-92.

Wang, C., Quan, H.,. Xu, X, 1996. Optimal design of multiproduct batch chemical process using genetic algorithms. Industrial and Engineering Chemistry Research 35, 3560-3566.

Wang, C., Quan, H., Xu, X., 1999. Optimal design of multiproduct batch chemical process using tabu search. Computers and Chemical Engineering 23, 427-437.

Wang, C., Xin, Z., 2002. Ants foraging mechanism in the design of batch chemical process. Computers and Chemical Engineering 41, 6678-6686.

Xu, X., Zheng, G., Cheng, S., 1993. Optimized design of multiproduct batch chemical process – A heuristic approach. Journal of Chemical Engineering (in Chinese), 44, 442-453.

Yang, Y.W., Xu, J.F., Soh, C.K., 2006. An evolutionary programming algorithm for continuous global optimization. European Journal of Operational Research 168, 354-369.