Applied Soft Computing xxx (2009) xxx–xxx

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Differential Evolution performances for the solution of mixed-integer constrained process engineering problems

A. Ponsich*, C.A. Coello Coello

CINVESTAV-IPN (Evolutionary Computation Group), Departamento de Computación, Av. Instituto Politécnico Nacional No. 2508, Col. San Pedro Zacatenco, México, D.F. 07300, Mexico

ARTICLE INFO

Article history:
Received 21 February 2009
Received in revised form
15 September 2009
Accepted 17 November 2009
Available online xxx

Keywords:
Differential Evolution
Mixed-integer constrained optimization
Process engineering

ABSTRACT

An important number of publications deal with the computational efficiency of a novel Evolutionary Algorithm called Differential Evolution (DE). However, there is still a noticeable lack of studies on DE's performance on engineering problems, which combine large-size instances, constraint-handling and mixed-integer variables issues. This paper proposes the solution by DE of process engineering problems and compares its computational performance with an exact optimization method (Branch-and-Bound) and with a Genetic Algorithm. Two analytical formulations are used to model the batch plant design problem and a set of examples gathering the three above-mentioned issues are also provided.

The computational results obtained highlight the clear superiority of DE since its best found solutions always lie very close to the Branch-and-Bound optima. Moreover, for an equal number of objective function evaluations, the results repeatability was found to be much better for the DE method than for the Genetic Algorithm.

1. Introduction

It appears that many engineering issues, drawn from many application areas, may be formulated as optimization problems. These problems might differ, depending on the nature of the decision variables (continuous, discrete, etc.), on the mathematical characteristics of the functions involved (continuity, derivability, linearity, convexity, etc.), or on the existence/absence of constraints. Generally speaking, a global optimization problem is defined such as:

$$\begin{array}{ll} \text{Minimize} & f(x,y) \\ \text{Sujected to} & g(x,y) \leq 0 \\ & h(x,y) = 0 \end{array}$$

where x is a continuous variable vector and y is a discrete variable vector.

As an answer to the huge variety of optimization problems that exist, a wide variety of solution techniques have been developed. A great research effort was first dedicated to the implementation of exact methods, i.e., granting the optimality of the solution. However, facing the necessity to solve more and more complex problems, metaheuristic methods began to appear and were successfully used for the solution of a wide range of applications. Despite their inability to guarantee the generation of optimal solutions, the simplicity of their operating mode and, in some cases, their good computational efficiency have made them an attractive option as solution methods in many types of optimization problems.

Recently, Storn and Price [1] proposed a novel Evolutionary Algorithm called Differential Evolution (DE). Since then, an important number of publications have been produced, dealing with the computational performance of DE, evaluating the obtained results:

- through comparisons: in some cases, the optimal solution is known and this allows to compute a distance between this solution and the DE's best result. However, comparisons with other optimization methods are often used in order to assess the computational reliability of DE towards those techniques, whose quality was already demonstrated.
- according to two kinds of criteria: computational time (or, in order to enable fair comparisons in case different computer platforms are used, number of objective function evaluations) and solution quality. This latter one not only considers the best found solution, but also, since DE's stochastic nature does not ensure to find every time the same result, the repeatability of the method (mean value and standard deviation are typically computed for a given number of runs).

* Corresponding author.
E-mail address: aponsich@itam.mx (A. Ponsich).

1568-4946/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2009.11.030

- on mathematical benchmark problems, formulated as equation-oriented (usually non-linear) models having, to the best of our knowledge, up to 100 (continuous) variables and 9 constraints [1,2].

The computational results obtained in these studies usually highlight the good performances of DE solutions, in terms of quality as well as number of objective function evaluations [3–6]. However, apart from its efficiency for pure mathematical problems solving, it is also interesting to evaluate this metaheuristic method on applied problems, such as those drawn from an engineering framework. The treatment of industrial problems would indeed lead the technique to face various types of computational difficulties that could be absent in artificial test problems:

- *Size of the treated problems*. Industrial applications may provide instances that contain up to hundreds of decision variables.
- *Simultaneous handling of different types of variables.* The original DE method only considers continuous problems. Nevertheless, the design variables appearing in classical engineering problems, which, for example, represent the system configuration (existence, or number of treatment units), are generally of integer or discrete type. Conversely, operating conditions or levels, as well as equipment sizes or stream flows are commonly modeled by continuous variables. Thus, the handling of decision variables is required.
- Literature reviews already report integer variable handling in DE through very simple methods [7]. But the treatment of mixed continuous and discrete variables, at the same time, breeds an important issue for the application of DE to engineering optimization problems.
- *Handling constrained problems.* Even though this aspect has been already tackled in the dedicated literature and has generated the implementation or adaptation of many methods (cf. Section 2), physical or material engineering constraints sometimes allow a treatment through case-adapted techniques. Moreover, the constraint-handling issue combined with the former two points might be a challenging problem.

There exist some studies that have dealt with the application of Differential Evolution to engineering problems [5,6] but they represent a tiny minority of the literature devoted to DE's performance evaluation. There still exists a clear lack of analysis of DE's performance on problems combining the three above-mentioned issues, which may arise in real-world optimization problems. The aim of the present paper is thus to propose the solution by DE of problems drawn from the process engineering area: for that sake, batch plant design problems constitute a relevant choice. Furthermore, many optimization techniques have already been tested on these problems, and such results might be used as a reference against which the results produced by DE can be compared. Particularly, DE's computational performances will be compared to results obtained by the application of a stochastic method and an exact one [8]. Comparisons will be made with respect to:

- a classical Genetic Algorithm (this method was already applied to batch plant design problems and was found to be effective in many cases).
- Mathematical programming techniques implemented in the GAMS software [9], which provide optimal solutions.

So, two sets of computational experiments will be carried out on different (although relatively similar) models. The former (general) formulation enables the construction of an increasing complexity set of instances, leading to the treatment of large-size problems (see [8]); while the latter one considers a (medium size) protein production plant problem [10], involving an additional complexity arising from operating conditions variables integrated within the design problem.

The remainder of this paper is organized as follows. Section 2 presents an overview of Differential Evolution, with a particular emphasis on methods devoted to the treatment of constrained problems and integer variables handling. In Section 3, the two formulations used to model batch plant design problems are developed. Finally, computational results are provided in Section 4, while some conclusions and perspectives are drawn in Section 5.

2. Outline on Differential Evolution

2.1. DE basics

Differential Evolution (DE) is a very recent Evolutionary Algorithm (EA), developed by [1]. The method was initially proposed for unconstrained, continuous optimization problems. Its basic principle relies on the design of a simple mutation operator based on the linear combination of three different individuals and on a crossover step that mixes the initial and the mutated solutions. The mutation process included in DE is, not only simple, but also, as indicated in [11], fulfills the following requirements, which are important for obtaining an efficient mutation scheme:

- uses a zero-mean distribution for generating mutated vectors;
- dynamically scales the distribution to suit each variable "improvement interval";
- correlates mutations to ensure rotational invariance.

To a certain extent, the implementation of the DE mutation operator can be seen as a self-adaptive methodology, such as the German evolution strategy (ES), since it produces a noise on the involved individuals that gradually decreases when the population converges towards the optimum.

Various DE versions exist, depending on the choice of the three individuals used to build the mutated individual and depending on the combination of mutated and initial solutions that takes place during the crossover step. The version that will be presented next is the most popular one, and is called DE/rand/bin, meaning that the three individuals used for the mutation process are randomly chosen; and that the crossover test (assigning either the initial or the mutated individual gene) is performed for each gene independently from what happened to the previous ones. The other existing versions of DE are outlined in [1].

As in most EAs, a selection process is carried out by comparing the trial solutions (obtained by the mutation and crossover steps) to the initial ones. Two interesting points concerning the DE selection scheme deserve, nevertheless, to be underlined [12]:

- a trial vector is not compared against the whole population, but only against its counterpart in the current population;
- all the individuals of the next generation are as good or better than their counterparts in the current generation (greedy aspect) since the mechanism excludes stochastic operations.

The canonical steps of the algorithm are defined as follows:

Step 1: Random generation of *NP* initial individuals.
Step 2: Evaluation of the objective function *f* for each individual.
Step 3: For each individual x_{ij}^G in the current population (*i* being the individual index, *j* the variable index and *G* the current generation number), creation of a mutated solution u_{ij}^G according to:

$$u_{ij}^G = x_{1j}^G + F(x_{2j}^G - x_{3j}^G), \quad \forall j = \{1, \ldots, N\} \tag{1}$$

Please cite this article in press as: A. Ponsich, C.A.C. Coello, Differential Evolution performances for the solution of mixed-integer constrained process engineering problems, Appl. Soft Comput. J. (2009), doi:10.1016/j.asoc.2009.11.030

where N is the number of decision variables, F is a scalar ($F \in [0,1]$) and $x_{1j}^G, x_{2j}^G, x_{3j}^G$ are randomly selected individuals in the current population ($x_{1j}^G \neq x_{2j}^G \neq x_{3j}^G \neq x_{ij}^G$).

Step 4: Recombination of the initial and mutated individual according to a crossover rate CR ($CR \in [0,1]$), to create an offspring y_{ij}:

$$u_{ij}^{G+1} = \begin{cases} u_{ij}^G & \text{if } rand(j) \leq CR \text{ or if } j = rnbr(i) \\ x_{ij}^G & \text{if } rand(j) > CR \text{ and if } j \neq rnbr(i) \end{cases}, \quad (2)$$

$$\forall j = \{1, \ldots, N\}, \forall i = \{1, \ldots, NP\}$$

where $rnbr(i)$ is a randomly chosen index ($rnbr(i) \in \{1, \ldots, N\}$), which ensures that u_{ij}^{G+1} gets at least one variable from u_{ij}^G.

Step 5: Evaluation of the objective function of the offspring u_{ij}^{G+1} and application of a deterministic selection process by comparing the parent and offspring objective values:

$$x_{ij}^{G+1} = \begin{cases} u_{ij}^{G+1} & \text{if } f(u_{ij}^{G+1}) \leq f(x_{ij}) \\ x_{ij} & \text{if } f(u_{ij}^{G+1}) > f(x_{ij}) \end{cases}, \quad (3)$$

$$\forall j = \{1, \ldots, N\}, \forall i = \{1, \ldots, NP\}$$

Step 6: If the termination criterion is met, output the best solution found so far and its objective value; else, go back to *Step 3*.

The typical termination criterion is to reach a given number of generations NG. F (amplification factor) and CR (crossover rate) are control parameters that, with NP (population size), must be tuned in an appropriate way. Lampinen and Zelinka [12] indicate that a fine tuning of these parameters might improve the convergence velocity and the robustness of the search process. Their optimal values depend on the tackled objective function and on the problem characteristics; suitable values are commonly found by a trial and error process. However, Price [11] provides some general recommendations on how to tune these factors: he suggests to adopt very high values for CR (close to 1), values for NP between 20N and 100N, and random values between 0.5 and 1 for F.

2.2. Treatment of constrained problem

2.2.1. Boundary constraints

From its intrinsic operating mode (which combines three randomly selected individuals), the DE process is very likely to produce solutions that, in the case of bounded variables, lie outside their allowed range of variation. Thus, some techniques were developed to make these solutions feasible and repair the variables violating their bounds.

Price [11] mentions several options to reset the variable to an allowable value:

(i) Setting the decision variable value to the violated bound. This technique, however, may lead to a degradation of the solution's diversity.
(ii) A completely opposite strategy consists in (randomly) reinitializing the considered decision variable within its allowable bounds.
(iii) As a compromise between the two previous methods, setting the decision variable midway between its initial value (i.e., the parent's decision variable value) and the violated bound, as formulated in Eq. (4). This approach is also applied in [13].

$$u_{ij}^G = \begin{cases} (x_{ij}^G + x_j^{(lo)})/2 & \text{if } u_{ij}^G < x_j^{(lo)} \\ (x_{ij}^G + x_j^{(up)})/2 & \text{if } u_{ij}^G > x_j^{(up)} \\ u_{ij}^G & \text{otherwise} \end{cases} \quad (4)$$

Finally, in [3] is proposed the use of the violated bound as a symmetry center to send the considered variable to the feasible side of the boundary (i.e., inside the allowed variation range), with a distance to the violated bound equal to the initial constraint violation. This strategy is expressed as follows:

$$u_{ij}^G = \begin{cases} 2x_j^{(lo)} - u_{ij}^G & \text{if } u_{ij}^G < x_j^{(lo)} \\ 2x_j^{(up)} - u_{ij}^G & \text{if } u_{ij}^G > x_j^{(up)} \\ u_{ij}^G & \text{otherwise} \end{cases} \quad (5)$$

2.2.2. Constraint functions

As in the case of many other EAs, the canonical DE scheme does not integrate the treatment of constrained problems. Consequently, several researchers have proposed schemes to add constraint-handling methods to DE. In many cases, techniques inspired from other EAs (Genetic Algorithms, particularly) have been directly transferred to the DE algorithm. Next, we will briefly review the most significant efforts of this sort, reported in the specialized literature.

First, a static penalty approach (named soft-constraint by the authors) is proposed in [4], showing the drawbacks (just as for any other EA) involved in the fine tuning of the penalty factors associated to each constraint violation. A similar technique is used in [14], not by considering a number of penalty terms equal to the number of constraints, but adopting only one penalty term for the normalized sum of constraint violations (this obviously reduces the number of parameters to be tuned).

Another kind of penalty method is introduced in [15] by dynamically tuning a penalty factor for the sum of inequality constraint violations and another one for the sum of equality constraint violations. The general expression of each penalty factor is defined through a NFT factor representing the threshold distance from the feasible region in which the search is promoted:

$$\frac{1}{NFT} = \frac{1 + \lambda \cdot t}{NFT_0} \quad (6)$$

where NFT_0 is an upper bound for NFT, λ is a user-defined positive parameter and t is the generation counter. The penalty factor thus increases with the iteration number, pushing more drastically the solutions towards the feasible region of the search space.

The penalty factors may also be considered as strategy parameters than can be evolved at the same time as the decision variables, like a "second population". In the resulting co-evolutionary optimization algorithm [5], inspired on the self-adaptive penalty approach proposed in [16], one generation for the set of strategy parameters represents a global evolution of the decision variables, evaluated with the mentioned set of penalty factors. This set is then updated and another variable evolution is carried out. The amount of computational effort is thus huge compared to more classical algorithms.

Apart from the penalization of the infeasible solution fitness, another relevant method is based on feasibility rules [5,13] adapted from the Genetics Algorithms framework. According to Deb [17], these rules were initially suggested by David Goldberg in 1992. Subsequently implemented by Deb [18], the methodology adopts the following comparison criteria:

- between two feasible solutions, the one with the higher fitness value wins;
- if one solution is feasible and the other one is infeasible, the feasible individual wins;
- if both solutions are infeasible, the one with the lower sum of constraint violation wins.

Please cite this article in press as: A. Ponsich, C.A.C. Coello, Differential Evolution performances for the solution of mixed-integer constrained process engineering problems, Appl. Soft Comput. J. (2009), doi:10.1016/j.asoc.2009.11.030

Another version is, in case of two non-feasible solutions, to choose the Pareto non-dominated one in constraints search space [4,19].

The two above-mentioned techniques lack a mechanism to maintain diversity. Mezura-Montes et al. [3] proposed to allow each parent in the current population to generate more than one offspring. Furthermore, to introduce some diversity from regions away from the feasible boundaries, an infeasible solution with a good fitness value, regardless of feasibility is, in some cases (determined according to a probability S_r) allowed to remain in the population, i.e., to win a feasible solution. The expected behavior is that the population will include a proportion S_r (at most) of infeasible solutions, with competitive values of the objective function: this mechanism is aimed to promote diversity in the population. This process is also applied for various mathematical and engineering problems and was found to be more efficient than more traditional DE approaches in [6].

A similar way to handle constraints by making a difference between feasible and infeasible solutions is presented by [20], with the ε-level comparison. Considering $\phi(x)$ a constraint violation function for solution x (ϕ represents either the maximum constraint violation or the sum of constraint violations), the classical order relationship is replaced by the following comparison, subject to level ε:

$$(f_1, \phi_1) <_\varepsilon (f_2, \phi_2) \Leftrightarrow \begin{cases} f_1 < f_2 & \text{if } \phi_1, \phi_2 \leq \varepsilon \\ f_1 < f_2 & \text{if } \phi_1 = \phi_2 \\ \phi_1 < \phi_2 & \text{otherwise} \end{cases} \quad (7)$$

where ε is a control parameter that is continuously decreased during the search process, according to the following equation:

$$\varepsilon(0) = \phi(x_0) \quad (8)$$

$$\varepsilon(t) = \begin{cases} \varepsilon(0)\left(1 - \dfrac{t}{T_c}\right)^{cp} & \text{if } t < T_c \\ 0 & \text{if } t \geq T_c \end{cases}$$

where x_0 is the top θth individual with $\theta = 0.2$N and cp is a control parameter. This technique enables to integrate, at the beginning of the run, slightly infeasible solutions that bring an additional diversity to the population. The pressure towards the feasible region is then enforced by decreasing the ε parameter down to a generation threshold T_c after which no constraint violation is allowed. At the end of the search, the process is similar to the feasibility rules used in [3].

2.3. Integer and discrete variables

DE is by nature a floating-point optimization method. A common method to overcome the issue of integer or discrete variables is to treat them all as if they were continuous. Once the problem has been optimized, the integer or discrete variables can be rounded-off to the nearest available value. However, in practice, this method gives less than optimal results since no attempt is made, during the optimization process, to evaluate only realizable systems [11].

In spite of the great variety of problems involving integer or discrete variables, there are very few methods available to transform the canonical DE algorithms for the treatment of this type of variables. Basically, the most common approach is that proposed by [7], which only requires, for the objective evaluation, the following modification on integer variables y_i:

$$y_i = INT(x_i) \quad (9)$$

where $INT(\cdot)$ is a function converting a real value into an integer value, by truncation or rounding-off, for instance. It is important to carry out the conversion into integer values only in the evaluation context, and not to replace the initial continuous-valued variables in the considered solution. This would lead us to a biased search process and would likely prevent us from reaching the optimum. Regarding discrete variables, the same process is applied but, this time, by encoding the variables indexes instead of their values: this process takes us back to the treatment of integer variables, performed according to Eq. (9).

3. Optimal batch plant design problems

3.1. Generalities

Due to the current interest for batch operating mode, a lot of studies deal with the batch plant design issue. Actually, the problem has already been modeled under various forms for which assumptions are more or less simplistic. Generally, the objective consists in the minimization of the plant investment cost. Grossmann and Sargent [21] were the first authors to propose a simple posynomial formulation for multiproduct batch plants. This model, which involves only batch stages and is subject to a constraint on the total production time, was used again in [22]. Modi and Karimi [23] modified this approach 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 up to eight operating stages, i.e., 16 decision variables) with heuristics.

The same model was adopted Patel et al. [24] who treated larger size examples with Simulated Annealing and by Wang et al. [25–27], who adopted, successively, Genetic Algorithms, Tabu Search and an Ants Foraging Method. Nevertheless, Ponsich et al. [8] showed that, for this mixed continuous and discrete formulation, and independently from the size of the studied instance, a Branch-and-Bound technique (implemented in the SBB solver, available in the GAMS modeling environment [9]) is the most efficient option. The same analytical formulation is used as an initial test suite within the study reported in this paper.

Besides, the above-mentioned formulation was further improved by taking into account continuous process variables [28,10]. Thus, the optimization not only considers the plant configuration, but also integrates the operating conditions in each unit of a process manufacturing four kinds of proteins. The associated, additional variables generate second-order interactions with the design variables that make the solution process much more difficult for non-derivative supported methods, such as stochastic methods [29]. This approach provides the second application example involved in the following computations.

3.2. Generic model for batch plant design problems

Batch plants must be flexible enough to manufacture different products, which are processed through various stages in the form of limited size batches. Basically, each stage is composed of parallel items operating in a discontinuous way, meaning that the product batches are successively loaded in and discharged from the items, along with their synthesis sequence (the so-called production recipe). In this study, we will only consider multiproduct plants: all the products follow the same operating steps. Only the operating times may be different from a recipe to another one.

The objective of optimal batch plant design (OBPD) problems is typically the minimization of the investment cost for all the items involved in the plant, by optimizing the number and size of parallel equipment units in each stage. The model formulation for OBPD problems adopted in this paper is based on Modi's approach [23]. It considers not only treatment in batch stages, but also represents semi-continuous units that are part of the whole process

Please cite this article in press as: A. Ponsich, C.A.C. Coello, Differential Evolution performances for the solution of mixed-integer constrained process engineering problems, Appl. Soft Comput. J. (2009), doi:10.1016/j.asoc.2009.11.030

Table 1
Characteristics of the first set of instances.

Instance	Stage number	Intermediate storage	Product number	Combinatorial effect
A	10	1	3	5.905×10^4
B	14	1	3	4.783×10^6
C	21	2	3	1.046×10^{10}
D	42	5	3	1.094×10^{20}
E	63	8	3	1.145×10^{30}
F	84	11	3	1.197×10^{40}
G	105	14	3	1.252×10^{50}

(pumps, heat exchangers, etc.). 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.

Thus, 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 the same size V_j. Each semi-continuous stage consists of n_k out-of-phase parallel items with the 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 these notations, a Mixed-Integer Non-Linear Programming (MINLP) problem is formulated in order to minimize the investment cost:

$$MinCost = \sum_{j=1}^{J} a_j m_j V_j^{\alpha_j} + \sum_{j=1}^{K} b_k n_k R_k^{\beta_k} + \sum_{j=1}^{S-1} c_s V_s^{\gamma_s *} \tag{10}$$

where α_j and a_j, β_k and b_k, γ_s and c_s are classical cost coefficients.

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. This problem is subject to two kinds of boundary constraints and one constraint function. The former is expressed as follows:

$$\forall j \in \{1, \ldots, J\} \quad V_{\min} \leq V_j \leq V_{\max} \tag{11}$$

$$\forall k \in \{1, \ldots, K\} \quad R_{\min} \leq R_k \leq R_{\max} \tag{12}$$

$$\forall j \in \{1, \ldots, J\} \quad 1 \leq m_j \leq 3 \tag{13}$$

$$\forall k \in \{1, \ldots, K\} \quad 1 \leq n_k \leq 3 \tag{14}$$

The second constraint enforces the respect of a deadline for product orders, involving that the total production time must be lower than a given time horizon H:

$$H \geq \sum_{i=1}^{I} H_i = \sum_{i=1}^{I} \frac{Q_i}{Prod_i} \tag{15}$$

where Q_i is the demand for product i and $Prod_i$ is the process productivity for product i. The computation of the productivity derives from an analytical model provided in Appendix A. The aim of batch plant design problems is finally to find the plant structure that respects the production requirements within the time horizon, while minimizing the economic criterion. The resulting problem proves to be non-convex and NP-hard [25].

The benchmark of this work is constituted by seven instances, some of which are treated in [8]. The set of examples shows, for purposes of evaluation of optimization method performances, an increasing complexity that leads to the solution of huge size problems (see Table 1).

3.3. Protein production plant

The considered example, proposed [28] and then extended [10], deals with the production of four kinds of proteins through an 8-stages production process: initially manufactured in a fermentation stage, the products are separated in two microfiltration, one homogenization, two ultrafiltration steps, one extraction and one chromatographic separation steps (see Fig. 1). The associated formulation is quite similar to that presented in the previous section but shows differences related to the product recipes, to the plant structure and to the processing unit models.

Concerning the former, the four manufactured products can be divided into extracellular and intracellular proteins, having processing sequences that differ a bit from one type to another (the extracellular products do not need treatment in all the 8 stages). With regard to the plant structure, the model only accounts for batch stages. These ones are, just like in the previous model, characterized by the item volume V_j and (out-of-phase) number m_j; but additional in-phase item numbers are also taken into account (this allows that, in each stage, a batch can be divided into smaller size

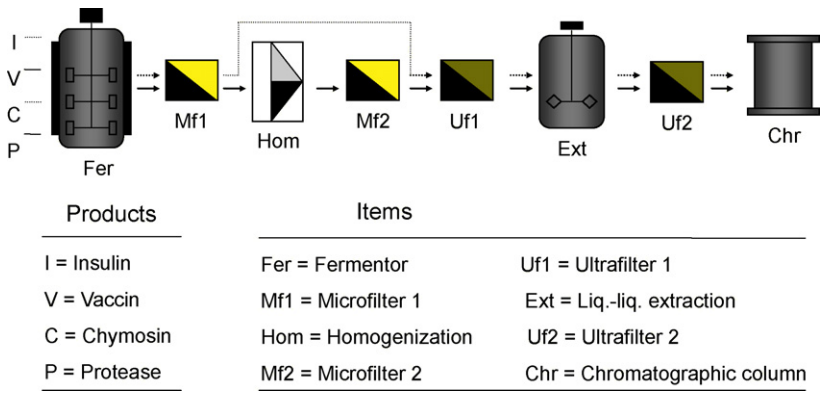

Fig. 1. Functional representation of the protein production plant.

Please cite this article in press as: A. Ponsich, C.A.C. Coello, Differential Evolution performances for the solution of mixed-integer constrained process engineering problems, Appl. Soft Comput. J. (2009), doi:10.1016/j.asoc.2009.11.030

Table 2
Analysis of *NG* and *NP* for instance B.

NP	*NG*	Best sol.	Mean value	Std. dev.
50	800	620,658.8	621,478.8	2251.6
80	**500**	**620,819.5**	**621,158.8**	**895.5**
100	400	621,034.5	621,503.7	1048.3
150	268	621,812.7	622,536.9	448.6
200	200	622,978.6	624,748.6	942.9

batches). A new integer variable n_j now appears in the model, Eq. (16) replacing Eq. (A.7) as follows:

$$\forall i \in \{1, \ldots, I\}; \quad B_i = \underset{j \in J}{Min} \left[\frac{V_j n_j}{S_{ij}} \right] \tag{16}$$

The objective function also changes, as indicated in Eq. (17):

$$MinCost = \sum_{j=1}^{J} n_j m_j a_j V_j^{\alpha_j} \tag{17}$$

It is to note that storage tanks do not appear anymore in the new formalism. Besides, most processing stages are composed of several equipment units (some of which might be seen as semi-continuous items). Table B1 in Appendix B sums up the design variables for each stage.

Finally, the most important difference is due to the integration of variables accounting for the operating conditions (or treatment parameters) in each processing stage. These variables are allowed in the formulation of performance models that represent the physical and chemical phenomena involved in the treatment steps. The operation efficiency (or yield) can then be calculated to finally deduce the size factors and processing times that will influence the plant design. Thus, the global decision variable vector can be written $X = [X_{des}, X_{proc}]^T$, where X_{des} is the design variable vector and $X_{proc} = [X_{i,Fer}, X_{i,Mf1}, W_{i,Mf1}, W_{i,Mf2}, NP_i, R_i]^T$ the process variable vector. The process variables are available in Appendix B while the performance model equations can be found in [10].

In conclusion, the resulting optimization has 16 continuous and 16 integer design variables, and 18 process variables involved in a strongly non-linear model. The coexistence of design and process variables within the same equations makes the search process harder than in the previous section formulation.

4. Computational results

4.1. Study operating mode

The operating mode of the following computations involves the definition of:

- the optimization methods selected to stand the comparison with Differential Evolution;
- the choice of the internal procedures that drive DE's search process (constraint handling, variable encoding, etc.);
- the criteria on which the comparison of the methods performances are based.

First, to evaluate the computational performance of DE, the obtained results are compared, for both formulations (generic model and protein production plant), to two other optimization techniques, namely a mathematical programming technique and Genetic Algorithms. These methods, often reported in the specialized literature, are widely used for the solution of batch plant design problems. Their applicability and performances were already studied in [8] and [29].

The first (exact) technique is represented by a Branch-and-Bound technique. Adapted for the solution of non-linear problems, the latter is implemented in the SBB module, available in the GAMS environment. Details on the method are not provided here since the main features are presented in [8].

With regards to the other stochastic method, i.e., Genetic Algorithms (GAs), we used its classical (or simple) version. The details of how GAs work are not described here but it is, however, worth describing the internal procedures that we adopted. After a random generation of the initial population, the selection step is carried out through a tournament that integrates, as a discrimination criteria, the domination rules stated by [18]. Considering the mixed continuous-integer nature of the tackled problem, a mixed encoding technique is implemented: real and integer loci coexist in the chromosome to represent, respectively, the item sizes and item numbers. Concerning the genetic operators, the adopted crossover technique is SBX [30] while mutation is performed by introducing a uniform distribution based noise [31].

Besides, the choice of the DE's internal procedures is made aiming for generality and for having the fairest comparison possible. Thus, no effort was made to add mechanisms that could improve performance of our DE algorithm. For instance, in spite of its proved efficiency, the multiple offspring generation technique proposed in [3] is not adopted here. However, mechanisms of this sort could be adopted in future extensions of the work reported here. The second indication encourages us to adopt procedures similar to those used in the other stochastic technique, i.e., the Genetic Algorithm (obviously, because of its completely different working mode, no similarity with the mathematical programming technique adopted is possible).

Thus, the main features chosen for this work are a randomly generated initial population and a selection process based, as proposed in [3], on Deb's domination rules (both procedures are identical to those used in the GA). A combination of two of the above-mentioned methods is chosen for boundary constraint handling: (i) setting the variable value to the violated bound; (ii) using the violated bound as a symmetry center to send the considered variable to the feasible side of the boundary. The application of these options depends on a probability P_{Bound}: for each boundary violation, the first scheme is adopted if a drawn random number is lower than P_{Bound}, and the second scheme elsewhere. With regards to the encoding technique, real loci are adopted (as proposed in [7]) and the item number variables are transformed to integer values only for the objective evaluation.

Some of the search parameters are also tuned aiming to allow a fair comparison: this is the case for the population size *NP* and generation number *NG*. These are chosen in such a way that the total number of evaluations is equal to the evaluation number required by the GA method (the GAMS environment does not provide this information for SBB runs). We will thus adopt $NP_{DE} \times NG_{DE} = NP_{GA} \times NG_{GA}$. Nevertheless, some sensitivity analysis is necessary for each example, in order to determine the values of these two parameters (imposing the total number of evaluations as a constraint), as well as for the amplification factor *F*, the crossover rate *CR* and probability for boundary constraint handling P_{Bound}. This issue will be specified in the next section.

Please cite this article in press as: A. Ponsich, C.A.C. Coello, Differential Evolution performances for the solution of mixed-integer constrained process engineering problems, Appl. Soft Comput. J. (2009), doi:10.1016/j.asoc.2009.11.030

Table 3
Analysis of *CR* for instance B.

CR	Best sol.	Mean value	Std. dev.
0.1	624,437.5	626,719.3	1324.4
0.3	621,662.7	622,439.1	512.3
0.5	621,259.9	621,571.6	204.1
0.7	621,110.2	621,416.1	185.8
0.8	621,033.1	621,286.6	148.9
1.0	621,728.0	627,929.0	6589.0
rand[0.8;1.0]	**620,809.9**	**620,966.6**	**113.5**

Finally, concerning the criteria used for the evaluation and comparison of the methods' performances, they are basically the same that the classical ones found in the literature: solution quality and number of objective evaluations. The former is based on the knowledge of the optimal solution provided by the Branch-and-Bound method. The quality is also evaluated in terms of repeatability of the result found, since the stochastic nature of GAs and DE cannot guarantee identical solutions for each of their runs. Thus, each metaheuristic is run 100 times and mean values and standard deviations are computed for the best solution obtained for each instance. Furthermore, in addition to these statistics, another quality measure proposed in [8] is considered here: the 1%-dispersion and 2%-dispersion of the results with respect to the best found solution f^*_{DE} within the 100 runs set (i.e., the percentage of runs providing a result lying respectively in the range $[f^*_{DE}, f^*_{DE} + 1\%]$ and $[f^*_{DE}, f^*_{DE} + 2\%]$). These dispersion measures enable us to know if the solution set is well-distributed, and close to the best found solution f^*_{DE}, instead of evaluating the distance to a mean value through standard deviation.

4.2. Sensitivity analysis

As mentioned in the above section, the first issue of the study is to tune the DE search parameters, i.e., the population size *NP* and maximum number of generations *NG*, as well as the amplification factor *F*, the crossover rate *CR* and the probability for boundary constraint handling P_{Bounds}. Since the two first parameters (*NP* and *NG*) are constrained by the total number of objective evaluations allowable (which must be equal to the number of evaluations performed by the GA), their tuning involves sensitivity analysis for each treated instance. This is also the case for P_{Bounds}. On the other hand, *F* and *CR* are tuned for the first example and their values are kept unchanged for all the following computations.

Tables 2–4 present the sensitivity analysis results for instance B. Only 10 runs are calculated for each test computation. Dispersions do not appear in the tables since, for this example, no noticeable difference according to the different parameter sets can be observed. The number of objective function evaluations performed by the GA is, in this case, equal to 40,000. The values in boldface indicate the selected parameters.

It is worth noting that in example B (but this is also true for other instances), a compromise is necessary between parameters promoting the best found solution and those promoting mean value–standard deviation statistics: a parameter set may indeed present the best solution while another one provides the best statistics. Generally in this study, priority is granted to the parameter set showing the best statistic results, but also ensuring a reasonable quality of the best found solution.

Table 4
Analysis of *F* for instance B.

F	Best sol.	Mean value	Std. dev.
0.3	620,754.3	628,160.0	8486.6
0.6	621,080.9	621,453.3	219.5
0.7	622,909.0	624,271.6	942.8
0.9	636,428.2	641,643.4	2955.1
rand[0.3;0.9]	**620,869.2**	**621,323.3**	**1303.7**
rand[0.3;0.6]	620,645.9	621,535.0	3585.5

Table 5
Analysis of P_{Bounds} for instance A.

P_{Bounds}	Best sol.	Mean value	1%-dispersion
0.0	**356,615.3**	**367,801.8**	**25**
0.2	356,627.2	368,616.8	19
0.4	356,633.8	369,729.2	12
0.6	356,621.0	370,606.4	6
0.8	371,401.0	371,411.7	0
1.0	356,690.9	371,119.3	2

The former comment is especially true concerning the *NP* and *NG* parameters. It is observed that, for all instances, low values of *NP* lead to the best values for the best found solution, while the corresponding statistics (including the dispersions) are not really satisfactory. Slightly increasing *NP* (which is equivalent to decreasing *NG*) generated a slightly worse value of the solution but improved the statistical results. Then, for really high values of *NP*, both quality criteria decrease. So, for most of the tackled instances (even though there are exceptions), medium values are chosen.

A similar study is carried out for the P_{Bounds} parameter. Its value is tuned for each example, since the experience has shown that, although in most cases the optimum is located on a variable boundary, setting the variable value to the violated bound does not always provide the best results. In order to illustrate this point, the sensitivity analysis made for instance A is presented in Table 5. The standard deviation here is not significant because the 100 run solutions are basically divided between two local optima, showing different qualities. The standard deviation is thus only a function of the number of runs locating the correct optimum and dispersions are much more expressive in this case. Actually, since 1%-dispersion and 2%-dispersion were found to be always identical for this instance, only the former is presented in the table. The parameter $P_{Bounds} = 0$ therefore indicates that, in all cases of bounds violation, the variable is set to the bound value.

Not all the performed sensitivity analysis is shown in this paper, but the parameter set used for each instance is available in Table 6. Parameters *F* and *CR* do not appear since they are taken, for all the examples, equal to those selected in Tables 3 and 4. Regarding the used parameters set in Table 6, it should be finally highlighted that, in spite of very global trends (such as *NP* much lower than *NG*), no precise guidelines can be defined to make the tedious step of parameter tuning easier: like it is done in this work, only a harsh preliminary study based on sensitivity analysis may help.

4.3. Generic batch pant design model

Table 7 summarizes the results obtained for the seven instances formulated according to the generic model. In addition, Figs. 2 and 3 help to illustrate the global trends as well as the comparison with the solutions generated by the GA. The Branch-and-Bound method converged in all cases and the obtained solutions are only recalled in the first row of Table 7. As mentioned above, the evaluation number

Table 6
DE search parameters for all instances.

Instance	Obj. evaluation nbr (equal to GA's)	*NP*	*NG*	P_{Bounds}
A	4.0×10^4	150	268	0.0
B	4.0×10^4	80	500	1.0
C	2.0×10^5	200	1000	0.0
D	2.5×10^5	100	2500	0.0
E	2.25×10^6	300	7500	0.0
F	4.0×10^6	300	13,334	1.0
G	1.8×10^7	600	30,000	0.80

Please cite this article in press as: A. Ponsich, C.A.C. Coello, Differential Evolution performances for the solution of mixed-integer constrained process engineering problems, Appl. Soft Comput. J. (2009), doi:10.1016/j.asoc.2009.11.030

Table 7
General results for the generic formulation.

	Instance A		Instance B		Instance C	
	GA	DE	GA	DE	GA	DE
Optimum (SBB)	356,610.2		620,638.1		957,270.5	
Best result	360,831.8	356,648.9	621,333.1	620,726.8	958,778.3	957,413.7
Mean value	377,661.7	365,549.8	628,516.1	620,853.6	993,561.2	964,412.9
Std. dev.	6057.7	7268.8	8441.5	74.0	16,873.5	9600.0

	Instance D		Instance D		Instance F	
	GA	DE	GA	DE	GA	DE
Optimum (SBB)	1,925,887.8		1,925,887.8		3,865,106.5	
Best result	1,975,027.2	1,930,721.7	2,967,473.6	2,939,601.2	3,961,817.8	3,948,048.5
Mean value	2,010,648.8	1,971,325.2	3,012,587.2	2,965,096.1	4,046,061.5	3,952,854.13
Std. dev.	28,970.8	9267.9	35,884.9	15,549.7	42,324.0	9477.8

	Instance G	
	GA	DE
Optimum (SBB)	4,786,804.7	
Best result	4,955,658.6	4,934,896.5
Mean value	5,075,988.6	4,938,840.8
Std. dev.	58,806.4	8889.4

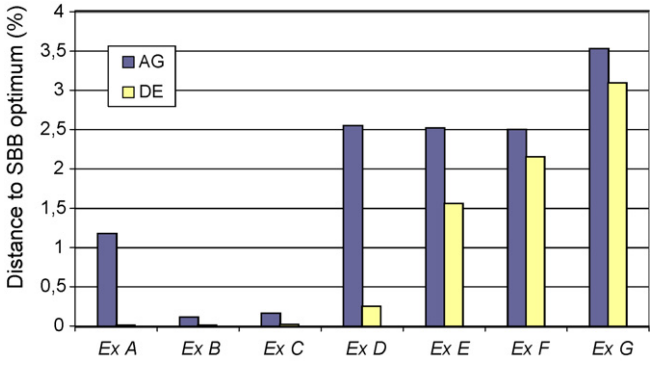

Fig. 2. Distance of both stochastic methods best found solution and SBB optimum.

of the exact method cannot be recorded but the convergence times were found to be quite similar to those of the Genetic Algorithm [8], or at least of the same order of magnitude.

The general trends are easy to find from the table and both figures: DE seems to provide better results than the GA, and with a better robustness, as can be inferred from the repeatability measures adopted. However, a more accurate description of the solutions for each instance is straightforward.

For instance A, it is clear that DE finds the SBB optimum while the GA is not able to do so. It should nevertheless be mentioned that, with another constraint-handling strategy (namely, elimination of infeasible individuals), the GA finds the optimum value, although with a weak repeatability [8]. Actually, among the solutions produced we can distinguish two local optima: the best one is that found by SBB (with an objective function equal to 356,610), while the other one has an objective lying at approximately 4% of the former. This gap is explained by a structural difference of the identified plant configurations, mainly for the item number in batch stages 2 and 3 (see Fig. 4).

The quality of the experiment is thus defined by the number of runs locating the best optimum; mean value and standard deviation are just functions of the ratio of successful runs within the whole set of results. Dispersions show that DE finds the best local optimum for 41% of the runs, while, except for one run, the GA only locates the second one. Both GA and DE find solutions very close to the SBB optimum of instance B, but DE's repeatability measures are unquestionably the best ones (the standard deviation, equal to 74, clearly indicates that DE locates almost always the correct solution).

Except for example D, results are similar for the following runs. DE is superior to GA in terms of distance to the optimum, mean value, standard deviation and dispersions. But it can be observed that the gap between both methods reduces as instances become more complex, especially regarding the distance to the optimum. However, the variability of the results produced by the GA increases, example after example, while DE's variability remains steady.

Concerning example D, the very weak value of 1%-dispersion clearly indicates that DE only finds once a solution lying at 0.25% of

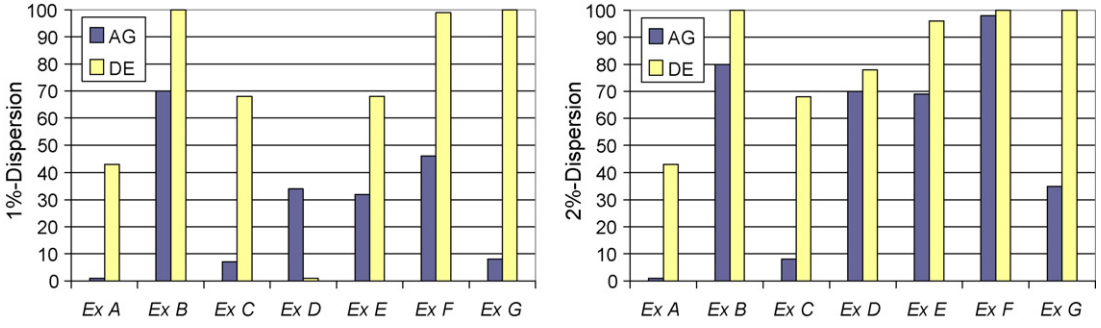

Fig. 3. 1%-dispersion and 2%-dispersion for both stochastic methods.

Please cite this article in press as: A. Ponsich, C.A.C. Coello, Differential Evolution performances for the solution of mixed-integer constrained process engineering problems, Appl. Soft Comput. J. (2009), doi:10.1016/j.asoc.2009.11.030

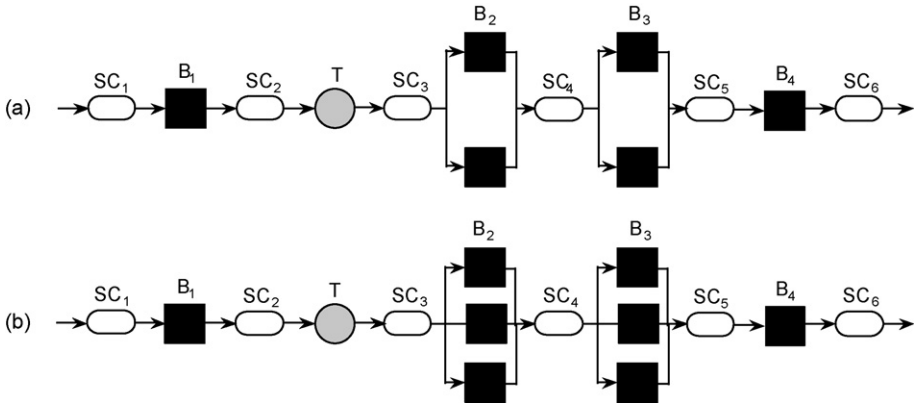

Fig. 4. Locally optimal solutions of instance A (a is the best one).

Table 8
Results for the protein production plant model.

	GA	DE
Optimum (SBB)	1,501,434.2	
Best result	1,516,559.1	1,501,939.5
Distance to optimum (%)	1.01	0.03
Mean value	1,564,022.2	1,508,547.6
Std. dev.	35,123.5	5444.3
1%-dispersion	10	94
2%-dispersion	34	100

the optimum. This good solution might thus be seen as an exception; nevertheless, the quality of the global run set is demonstrated by the 2%-dispersion, as well as the mean value.

4.4. Protein production plant

Similar conclusions can be drawn from the solutions obtained for the protein production plant model. Actually, the difference between the two stochastic techniques appears to be more drastic. The best found solution and statistics of both the GA and DE are available in Table 8. On the one hand, the DE method finds a value quite close to the optimum, and does so in almost all the runs. On the other hand, the GA solution lies at 1% of the optimum with a very poor repeatability. Furthermore, the DE's 1%-dispersion means that almost all its runs find a better solution that the best solution generated by the GA.

5. Conclusions

This work proposed the application of the Differential Evolution technique to problems drawn from the process engineering framework. Since the aim of the study is to provide general results for this stochastic method, very classical internal procedures were chosen, especially for the two most crucial issues: constraint-handling and mixed variables encoding. The method was evaluated on two (similar) formulations for the batch plant design problem and results were compared to those obtained by a Genetic Algorithm and a Branch-and-Bound technique. The latter provided optimal points that could be used as references to evaluate the metaheuristics efficiency. The fairness of comparison was enforced by ensuring an equal number of objective evaluations for these techniques.

The comparison of solutions turned very clearly to the advantage of Differential Evolution: the best found solution is always much closer to the Branch-and-Bound optimum than the one generated by the GA. Moreover, the results' repeatability is better for the DE method too: the used measures all support the fact that DE, compared to the widely used GAs, can find for both formulations of batch plant design problems better solutions, with a higher degree of confidence.

As part of our future work, it might be useful to extend our research interests towards two points. The first one is the treatment of integer or discrete variables: DE is a method designed to deal with decision variables provided as real numbers, and the technique that we used in this paper to handle integer variables is not really sophisticated. Trying to solve purely integer optimization problems might therefore result in few satisfactory solutions. A growing attention has been actually focused on the development of DE extensions, for the treatment of combinatorial optimization such as mentioned in the survey presented in [32] or for the case of permutation-based representations [33]. However, there still exists a lot of room for significant improvement in this research line. Another extension, which has already attracted interest among some researchers is the treatment of multi-objective problems with DE: in this regard, engineering problems would probably provide a wide range of application problems. Actually, in the process engineering area, various works already propose the application of Genetic Algorithms for the multi-objective optimization of chemical processes. We refer the reader to [34–40] for more details.

Acknowledgements

The first author acknowledges support from CONACyT through scholarship to pursue post-graduate studies in the Computer Science Department of CINVESTAV-IPN. The second author acknowledges support from CONACyT through research project No. 103570.

Appendix A. Generic batch plant design model

The global productivity ($Prod_i$) used in Eq. (15) is computed from local productivities (of each sub-process s):

$$\forall i \in \{1, \ldots, I\} \quad Prod_i = \min_{s \in S} [Prodloc_{is}] \tag{A.1}$$

These local productivities are calculated from the following equations:

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

$$\forall i \in \{1, \ldots, I\}; \forall s \in \{1, \ldots, S\} \quad Prodloc_{is} = \frac{B_{is}}{T_{is}^L} \tag{A.2}$$

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

$$\forall i \in \{1, \ldots, I\}; \forall s \in \{1, \ldots, S\} \quad T_{is}^L = \max_{j \in J_s} \left[T_{ij}, \theta_{ik}\right] \tag{A.3}$$

Please cite this article in press as: A. Ponsich, C.A.C. Coello, Differential Evolution performances for the solution of mixed-integer constrained process engineering problems, Appl. Soft Comput. J. (2009), doi:10.1016/j.asoc.2009.11.030

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:

$$\forall i \in \{1, \ldots, I\}; \forall j \in \{1, \ldots, J\} \quad T_{ij} = \frac{\theta_{ik} + \theta_{i(k+1)} + p_{ij}}{m_j} \tag{A.4}$$

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:

$$\forall i \in \{1, \ldots, I\}; \forall j \in \{1, \ldots, J_s\}; \forall s \in \{1, \ldots, S\} \quad p_{ij} = p_{ij}^0 + g_{ij} B_{is}^{d_{ij}} \tag{A.5}$$

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

$$\forall i \in \{1, \ldots, I\}; \forall k \in \{1, \ldots, K_s\}; \forall s \in \{1, \ldots, S\} \quad \theta_{ik} = \frac{B_{is} D_{ik}}{R_k n_k} \tag{A.6}$$

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

$$\forall i \in \{1, \ldots, I\}; \forall s \in \{1, \ldots, S\} \quad B_{is} = \underset{j \in J_s}{Min} \left[\frac{V_j}{S_{ij}} \right] \tag{A.7}$$

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

$$\forall s \in \{1, \ldots, S-1\}$$
$$V^* = \underset{i \in I}{Max} \left[S_{is} Prod_i (T_{is}^L + T_{i(s+1)}^L - \theta_{ik} - \theta_{i(k+1)}) \right] \tag{A.8}$$

Appendix B. Protein production plant model

Tables B1 and B2.

Table B1
Design variables for the protein production plant problem.

Variable	Description
V_{Fer}	Fermentation volume [m^3]
$V_{Mf1,ret}$	1st microfilter retentate volume [m^3]
$V_{Mf1,per}$	1st microfilter permeate volume [m^3]
$S_{Mf1,fil}$	1st microfilter filtration area [m^2]
V_{Hom}	Homogenizer tank volume [m^3]
Cap_{Hom}	Homogenizer capacity [m^3/h]
$V_{Mf2,ret}$	2nd microfilter retentate volume [m^3]
$V_{Mf2,per}$	2nd microfilter permeate volume [m^3]
$S_{Mf2,fil}$	2nd microfilter filtration area [m^2]
V_{Uf1}	1st ultrafilter retentate volume [m^3]
$S_{Uf1,fil}$	1st ultrafilter filtration area [m^2]
V_{Ext}	Extractor volume [m^3]
V_{Uf2}	2nd ultrafilter retentate volume [m^3]
$S_{Uf2,fil}$	2nd ultrafilter filtration area [m^2]
V_{Chr}	Chromatographic stage—tank volume [m^3]
$V_{Chr,col}$	Chromatographic stage—column volume [m^3]

Table B2
Process variables for the protein production plant problem.

Variable	Description
$X_{i,Fer}$	Fermentation output concentration for product i [kg m^{-3}]
$X_{i,Mf1}$	1st microfilter output concentration for product i [kg m^{-3}]
$W_{i,Mf1}$	Water added in the 1st microfilter for extracellular product i
$W_{i,Mf2}$	Water added in the 2nd microfilter for intracellular product i
NP	Pass number in homogenizer, intracellular product i
R_i	Phase ratio in extractor for product i

References

[1] R. Storn, K.V. Price, Differential Evolution—a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (1997) 341–359.
[2] E. Mezura-Montes, J. Velázquez-Reyes, C.A. Coello Coello, Modified Differential Evolution for constrained optimization, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006), Vancouver, 2006, pp. 25–32.
[3] E. Mezura-Montes, J. Velázquez-Reyes, C.A. Coello Coello, Promising infeasibility and multiple offspring incorporated to Differential Evolution for constrained optimization, in: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO'2005), Washington, DC, 2005, pp. 225–232.
[4] J. Lampinen, A constraint handling approach for the Differential Evolution algorithm, in: Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC 2002) (vol. 2), Piscataway, 2002, pp. 1468–1473.
[5] F.Z. Huang, L. Wang, Q. He, An effective co-evolutionary differential evolution for constrained optimization, Appl. Math. Comp. 186 (2007) 340–356.
[6] E. Mezura-Montes, C.A. Coello Coello, J. Velázquez-Reyes, L. Muñoz-Davila, Multiple trial vectors in differential evolution for engineering design, Eng. Optim. 39 (5) (2007) 567–589.
[7] J. Lampinen, I. Zelinka, Mixed variable non-linear optimization by Differential Evolution, in: I. Zelinka (Ed.), Proceedings of Nostradamus'99, 2nd International Prediction Conference, Zlin, 1999, pp. 45–55.
[8] A. Ponsich, C. Azzaro-Pantel, S. Domenech, L. Pibouleau, Mixed-Integer Non-Linear Programming optimisation strategies for batch plant design problems, Ind. Eng. Chem. Res. 46 (3) (2007) 854–863.
[9] A. Brooke, D. Kendrick, A. Meeraus, R. Raman, GAMS User's Guide, GAMS Development Corporation, 1998.
[10] J.M. Pinto, J.M. Montagna, A.R. Vechietti, O.A. Iribarren, J.A. Asenjo, Process performance models in the optimization of multiproduct protein production plants, Biotechnol. Bioeng. 74 (2001) 461–465.
[11] K.V. Price, An introduction to Differential Evolution, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, New York, 1999, pp. 79–108.
[12] J.J. Lampinen, I. Zelinka, On stagnation of the Differential Evolution algorithm, in: P. Ošmera (Ed.), Proceedings of MENDEL 2000, 6th International Mendel Conference on Soft Computing, Brno, 2000, pp. 76–83.
[13] K. Zielinski, L. Laur, Constrained single-objective optimization using Differential Evolution, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006), Vancouver, 2006, pp. 223–230.
[14] J. Brest, V. Žumer, M. Sepesy Maučec, Self-adaptive Differential Evolution algorithm in constrained real-parameter optimization, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006), Vancouver, 2006, pp. 215–222.
[15] M. Fatih Tasgetiren, P.N. Suganthan, A multi-populated Differential Evolution algorithm for solving constrained optimization problem, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006), Vancouver, 2006, pp. 33–40.
[16] C.A. Coello, C.A. Coello, Use of a self-adaptive penalty approach for engineering optimization problems, Comp. Ind. 41 (2002) 113–127.
[17] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms, John Wiley & Sons, Chichester, UK, 2001.
[18] K. Deb, An efficient constraint handling method for genetic algorithms, Comp. Meth. Appl. Mech. Eng. 186 (2000) 311–338.
[19] S. Kukkonen, J. Lampinen, Constrained real-parameter optimization with generalized Differential Evolution, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006), Vancouver, 2006, pp. 207–214.
[20] T. Takahama, S. Sakai, Constrained optimization by the ε constrained Differential Evolution with gradient-based mutation and feasible elites, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006), Vancouver, 2006, pp. 1–8.
[21] I.E. Grossmann, R.W.H. Sargent, Optimum design of multipurpose chemical plants, Ind. Eng. Chem. Proc. Des. Dev. 18 (1979) 343–348.
[22] G.R. Kocis, I.E. Grossmann, Global optimisation of nonconvex mixed-integer non linear programming (MINLP) problems in process synthesis, Ind. Eng. Chem. Res. 27 (1988) 1407–1421.
[23] A.K. Modi, I.A. Karimi, Design of multiproduct batch processes with finite intermediate storage, Comput. Chem. Eng. 13 (1989) 127–139.
[24] 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.
[25] C. Wang, H. Quan, X. Xu, Optimal design of multiproduct batch chemical process using genetic algorithms, Ind. Eng. Chem. Res. 35 (1996) 3560–3566.
[26] C. Wang, H. Quan, X. Xu, Optimal design of multiproduct batch chemical process using tabu search, Comput. Chem. Eng. 23 (1999) 427–437.
[27] C. Wang, Z. Xin, Ants foraging mechanisms in the design of batch chemical process, Comput. Chem. Eng. 41 (2002) 6678–6686.
[28] J.A. Asenjo, J.M. Montagna, A.R. Vecchietti, O.A. Iribarren, J.M. Pinto, Strategies for the simultaneous optimization of the structure and the process variables of a protein production plant, Comput. Chem. Eng. 24 (2000) 2277–2290.
[29] A. Ponsich, Stratégies d'optimisation mixte en Génie des Procédés—Application à la conception de procédés discontinus, Ph.D. Thesis, INP Toulouse, 2005.
[30] K. Deb, R.B. Agrawal, Simulated binary crossover for continuous search space, Complex Syst. 9 (1995) 115–148.

Please cite this article in press as: A. Ponsich, C.A.C. Coello, Differential Evolution performances for the solution of mixed-integer constrained process engineering problems, Appl. Soft Comput. J. (2009), doi:10.1016/j.asoc.2009.11.030

[31] K. Deb, M. Goyal, A combined adaptive search (GeneAS) for engineering design, Comput. Sci. Inf. 26 (1996) 30–45.
[32] R. Storn, Differential evolution research—trends and open questions, in: U.K. Chakraborty (Ed.), Advances in Differential Evolution, Studies in Computational Intelligence, vol. 143, Springer, 2008, pp. 1–31.
[33] G.C. Onwubolu, D. Davendra, Differential Evolution: A Handbook for Global Permutation-based Combinatorial Optimization, in: G.C. Onwubolu, D. Davendra (Eds.), Studies in Computational Intelligence, vol. 175, Springer, 2009.
[34] J. Hakanen, K. Miettinen, M.M. Mäkelä, J. Manninen, On interactive multiobjective optimization with NIMBUS in chemical process design, J. Multi-Crit. Dec. Anal. 13 (2–3) (2005) 125–134.
[35] J. Hakanen, Y. Kawajiri, K. Miettinen, L.T. Biegler, Interactive multi-objective optimization for simulated moving bed processes, Control Cybern. 36 (2) (2007) 282–320.
[36] N. Chakraborti, P. Mishra, A. Aggarwal, A. Banerjee, S.S. Mukherjee, The Williams Otto chemical plant re-evaluated using a Pareto-optimal formulation aided by genetic algorithms, Appl. Soft Comp. 6 (2) (2006) 189–197.
[37] A. Biswas, N. Chakraborti, P.K. Sen, Multiobjective optimization of manganese recovery from sea nodules using genetic algorithms, Mater. Manuf. Process. 24 (1) (2009) 22–30.
[38] A. Biswas, N. Chakraborti, P.K. Sen, A genetic algorithms based multi-objective optimization approach applied to a hydrometallurgical circuit for ocean nodules, Miner. Process. Extr. Metall. 30 (2) (2009) 163–189.
[39] C. Gutierrez-Antonio, A. Briones-Ramirez, Pareto front of ideal Petlyuk sequences using a multiobjective genetic algorithm with constraints, Comput. Chem. Eng. 33 (2) (2009) 454–464.
[40] K. Miettinen, J. Hakanen, Why use interactive multi-objective optimization in chemical process design? in: Rangaiah Gade Pandu (Ed.), Multi-Objective Optimization: Techniques and Applications in Chemical Engineering, World Scientific, Singapore, 2009, pp. 153–188 (Chapter 6).

A. Ponsich received a PhD in process engineering in 2005 at the Institut National Polytechnique de Toulouse (France). He is currently carrying out a postdoctoral project at the Computer Science Department at CINVESTAV-IPN in Mexico City. His research interests are continuous and combinatorial optimization using metaheuristics, operational research and applications to industrial processes.

C.A. Coello Coello received a PhD in computer science in 1996. He is currently full professor at CINVESTAV-IPN in Mexico City, Mexico. He has published over 200 papers in international peer-reviewed journals and conferences. He has also co-authored the book "*Evolutionary Algorithms for Solving Multi-Objective Problems*" (second edition, Springer, 2007). He received the 2007 National Research Award granted by the Mexican Academy of Science, in the area of exact sciences. His publications report over 2750 citations. His current research interests are: multi-objective optimization using metaheuristics, constraint-handling techniques for Evolutionary Algorithms and evolvable hardware.

Please cite this article in press as: A. Ponsich, C.A.C. Coello, Differential Evolution performances for the solution of mixed-integer constrained process engineering problems, Appl. Soft Comput. J. (2009), doi:10.1016/j.asoc.2009.11.030