Constraint-Handling Techniques within Differential Evolution for Solving Process Engineering Problems

Victor H. Cantú[a], Catherine Azzaro-Pantel[a,*], Antonin Ponsich[b]

[a] *Laboratoire de Génie Chimique, UMR 5503 CNRS/INP/UPS, Université de Toulouse, Toulouse, France*
[b] *Departamento de Sistemas, Universidad Autónoma Metropolitana Azcapotzalco, Mexico City, Mexico*

Abstract

Chemical engineering optimization problems are typically considered as difficult optimization problems as non-convexities and discontinuities arise, and as they usually present a high number of constraints to be fulfilled. Differential Evolution algorithm (DE) has proven to be robust for the solution of highly non-convex and mixed-integer problems; nevertheless, its performance greatly depends on the constraint-handling technique used. In this study, numerical comparisons of some state-of-the-art constraint-handling techniques are performed: static penalty function, stochastic ranking, feasibility rules, ε constrained method and gradient-based repair. The obtained results show that the gradient-based repair technique deserves a special attention when solving highly constrained problems. This technique enables to efficiently satisfy both inequality and equality constraints, which makes it particularly adapted for the solution of process engineering optimization problems.

1. Introduction

Complex optimization problems are ubiquitous in chemical engineering, and more generally, in process engineering (PE). Examples of optimization problems related to this area encompass batch process design [1, 2], phase equilibrium [3], distillation sequencing [4], heat exchanger networks [5, 6], reactor network design [7], supply chain design [8, 9], among others. All these real-world problems are typically represented by a mathematical model containing both binary and continuous variables, and a set of linear and non-linear constraints, i.e., leading to a mixed-integer non-linear programming (MINLP)

*Corresponding author
Email address: catherine.azzaropantel@toulouse-inp.fr (Catherine Azzaro-Pantel)

formulation. The single-objective formulation of these optimization problems can be stated as follows:

$$\begin{aligned}
\text{minimize} \quad & f(\mathbf{x}), \\
\text{subject to} \quad & g_i(\mathbf{x}) \leq 0, \quad i = \{1, \ldots, p\} \\
& h_j(\mathbf{x}) = 0, \quad j = \{1, \ldots, q\} \\
& \mathrm{l}_i \leq x_i \leq \mathrm{u}_i, \quad i = \{1, \ldots, n\}.
\end{aligned} \tag{1}$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ is a n-dimensional vector of decision variables (either discrete or continuous), $f(\mathbf{x})$ is the objective function to be minimized[1], p is the number of inequality constraints and q is the number of equality constraints. The functions g_i and h_j may be linear or non-linear, continuous or not, real-valued functions. Each variable x_i has upper and lower bounds, u_i and l_i, respectively, which define the search space $\mathcal{S} \subseteq \mathbb{R}^n$. Inequality and equality constraints define the feasible region $\mathcal{F} \subseteq \mathcal{S}$.

Throughout the years, mathematical programming techniques have been used to address the solution of these problems. Several algorithms such as Generalized Benders Decomposition (GBD), Outer-Approximation with the Equality-Relaxation strategy (OA/ER) and multiple Branch-and-Bound algorithms (BB) have been proposed to solve to ε-global optimality these problems (i.e. to a given gap between the lower and upper bounds) [10, 11]. It should be noted, however, that all these Newton's methods rely on the initial given solution and on convexity assumptions of the non-linear functions to guarantee the global optimum solution, and in fact, their performance may depend strongly on the problem's formulation, meaning that a different algebraic form of the same equations might lead to quite different algorithmic performance [12]. That is, the problem needs to satisfy some specific mathematical characteristics (e.g., convexity, derivability) so that a valid (automatic) reformulation can be generated, otherwise, the reformulated convex problem might miss the original global optimum and converge to a local optimum. For the convex relaxation to be performed, the problem has to be reduced to a *standard form*, in which nonlinear terms are linearized (by adding new variables and constraints), and then each non-convex term is replaced with a convex envelope obtained as the convex hull of the inequality constraints. Since one of the key factors to obtain the ε-global optimum depends on computing a tight convex lower bound, several methods have been proposed depending on the mathematical characteristics of the studied problems (e.g. McCormick's envelopes, piecewise-linear underestimators, etc.). Furthermore, the application of these deterministic techniques can be computationally expensive to obtain the rigorous global solution of large-scale problems.

[1]for the case of maximization, a negative sign is just added to the objective function

In that context, metaheuristics, and especially evolutionary algorithms techniques (EAs) have been proposed to solve highly non-convex optimization problems. Some examples of EAs are Evolutionary Strategies (ES), Genetic Algorithms (GA), Particle Swarm Optimization (PSO) and Differential Evolution (DE), among others [13]. Even if these techniques lack of any theoretical convergence proof, it is well recognized that good quality solutions can be obtained and several advantages can be highlighted : (i) these population-based algorithms do not require any mathematical property for the treated problem as they only use the evaluation of the objective function; (ii) due to their population-based nature, EAs are particularly suited to tackle problems in which more than one objective is to be optimized. This property is important in PE for which several kinds of objectives (for instance, environmental or social impacts) need to be minimized simultaneously with an economic criterion.

However, as EAs have been conceived as directed search engines to work over unconstrained spaces, their main drawback appears when tackling constrained problems and thus their applicability has been limited by a deficiency of general techniques to manage constraints. Consequently, much effort has been made to efficiently incorporate constraint-handling techniques into EAs, beginning, as in mathematical programming, by penalty functions. Nevertheless, though penalty functions may work well in some problems, they involve the tuning of some parameters that may affect significantly the quality of the final solutions found. For this reason, evolutionary computation researchers have proposed many other sophisticated approaches for handling constraints. In [14], the authors proposed a co-evolutionary algorithm for ensuring feasibility of individuals in problems containing only equality constraints. Equality constraints are eliminated by substituting an equal number of decision variables, the feasible space being then a convex set defined by linear inequalities. Thanks to this property, the genetic operators consist of linear combinations of individuals who ensure the feasibility of the solutions thus created. Maintaining feasibility can also be obtained through "decoders", i.e. instructions contained in the chromosome dictating a way to build a feasible solution. In [15] an approach for handling inequality constraints in (1+1)-ES uses an augmented Lagrangian technique (which involves additional quadratic penalty terms) and the performance of the algorithm is evaluated on sphere and ellipsoid functions with a single linear constraint. This approach was then extended by [16] to work with an adaptive covariance matrix adaptation evolution strategy (CMA-ES) exploring the performance of the algorithm on a set of linearly constrained functions, including convex quadratic and ill-conditioned objective functions, and observing linear convergence to the optimum. Diversely, in order to overcome the parameter-setting drawback when using penalty methods, stochastic ranking was proposed by [17] attempting to perform a useful comparisons between feasible and infeasible solutions, balancing the influence of the objective function and the penalty function without the need of any penalty factor; this method showed encouraging results when solving benchmark problems with multiple (active) constraints. Later, the same authors proposed in [18] a multiobjective constraint-handling method which conducts to a bias-free search, however, most of the search was spent in the infeasible region, finding poor feasible solutions. Another classical approach for handling-constraints in EAs, is to allow constraint relaxation at some extent at early generations, and then to stress the search to feasible regions in late generations. The ε-constrained method [19] stands in this framework, in which a polynomial function is employed for decreasing the relaxation level throughout the evolutionary process. The same authors also proposed some extensions of this method [20, 21], suggesting the preservation of elite solutions and the reparation of infeasible ones by using the constraint gradient information. In [22], another relaxation method is proposed, in which the relaxation level is computed at each generation considering the fraction of feasible solutions in the population and the median of the total violation of constraints; the authors considered in conjunction a self-adaptive differential evolution algorithm with tabu search, and employed a gradient-based local optimizer at the end of the search for ensuring convergence. In the context of multiobjective optimization, a modification of the ε-constrained method has been proposed [23] aiming to improve the original ε relaxation function, this time enabling the function to increase/decrease depending on the ratio of feasible solutions in the population. More recently, a two-stage procedure (called, push and pull) was proposed [24]: in the first stage (push stage), the algorithm explores the unconstrained search space which allows the population to get across infeasible disconnected regions, and then, in the pull stage, original constraints are considered along with the improved ε constrained method to gradually pull the population towards feasible regions. On the other hand, the solution of multiobjective problems containing equality constraints has been the question in [25], where authors proposed a two-phase hybrid algorithm; in the first phase, a rough approximation of the Pareto front is obtained via NSGA-II algorithm coupled with the ε-constrained method, and then, in the second phase, each obtained solution is refined using the so-called Pareto Tracer, which can be viewed as a gradient-based local optimizer capable of efficiently handle equality constraints. Nevertheless, exploiting the constraints' gradient information for solving constrained problems within EAs, was presented for the first time in [26]; the obtained results indicated a slight superiority of the method over stochastic ranking. It is important to note that this method does not constitute a local optimizer, since the search is still performed by the EA; that is, Newton's method is only employed on the constraints information for repairing, at some extent, infeasible solutions. In [27], the constraint gradient information has also been employed for repairing solutions that violate only equality constraints, the basic idea explored in this different method is that, for a given infeasible solution, some variables are set to a fixed value, while a number of variables equals to the number of equality constraints in the problem are to be repaired, by solving the system of equations (constraints).

Surprisingly enough, even if EAs have been widely used in literature for the solution of process engineering optimization problems, the same constraint-handling techniques are usually considered: static penalty function and feasibility rules, whereas the performance of more recent methods has not been investigated in the PE framework, at least not in a systematic manner. Further, in order to improve the performance of these EAs in constrained problems, a reformulation of the problem is often done to reduce the number of decision variables and to remove, as many as possible, equality constraints. However, despite its efficiency, such an approach may not work properly for large-scale real-world problems that usually contain a considerable number of constraints. In [28], authors tackled the solution of chemical engineering MINLP problems using an algorithm combining simulated annealing (for treating discrete variables) and the non-linear simplex method (for treating continuous variables), and then, concerning the handling of constraints, authors employed a dynamic penalizing scheme that considers the maximum extent of constraint violation in such a way that no penalty parameter is needed. Results showed that this method, though efficient for most of the treated problems, experiences difficulties for finding the global optimum in highly constrained problems. In [29], authors studied the solution of seven chemical engineering optimization problems, comparing the performance of both genetic algorithms and evolution strategies; equality constraints were eliminated by reducing the number of decision variables, whereas inequality constraints were treated using feasibility rules. It is noteworthy that only small problems were studied. A similar study was carried out in [30], where authors employed differential evolution algorithm as the search engine; constrains were handled using Deb's feasibility rules and, for problems presenting equality constraint, the reformulation of the model was done in order to eliminated equality constraints. However, conclusions about the efficiency of this methodology for the solution of real-world problems are not clear, since only small instances were studied (the biggest instance containing 7 continuous variables, 3 binary variables and none equality constraints). In [31], authors treated an important number of problems, implementing DE with tabu search to enhance the search; all equality constraints were removed by a model reformulation and the static penalty function was employed for handling inequality constraints, in fact, authors used a high value of the penalty factor for all problems, which can be viewed as using feasibility rules. In [32], an enhanced version of PSO algorithm was proposed; constraint handling was performed in two steps: (1) problem reformulation (reduction of decision variables) is done aiming to remove all equality constraints, and (2) Deb's feasibility rules are used for comparing two individuals, and hence to stress the search over feasible regions. As has been mentioned earlier, a gradient-based reparation procedure was proposed to efficiently treat equality constraints without the need of reformulating the model [27], however, the results obtained seem to indicate that this method needs an important number of function evaluations for converging (approx. 10^6 for a problem containing 22 variables). Finally, the optimization of dynamic chemical processes was addressed in [33] using DE with a modification in the mutation process, concerning constraint handling (both equality and inequality), they are handled by Deb's feasibility rules.

In reference to the performance comparison of several optimization algorithms, and by extension, the performance of several constraint-handling techniques, the well-known no-free lunch theorem (NFL) [34] needs to be mentioned, stating that no algorithm outperforms any other when its performance is averaged over all possible optimization problems. This means, for example, that no algorithm is better than random search. However, this can only be understood on the basis that *all possible* optimization problems are considered over a given search space [35]. In this study, only problems related to process engineering are considered, which are usually formulated as very difficult optimization problems and, besides, only those constraint-handling techniques that are usually found in literature are studied, with the addition of a repair method.

Taking into account all the above considerations, the scientific objective and the main contributions of this work is therefore to explore and compare the performance of some state-of-the art constraint-handling techniques for the solution of a selection of problems drawn from the process engineering framework. In particular, we investigate five constraint-handling techniques, namely penalty functions, stochastic ranking [17], feasibility rules [36], ε constrained method [19] and gradient-based repair [26]. It is worth mentioning that this diversity of methods is representative, by their respective working mode, of those that are typically used in most works found in the specialized literature. Since the aim is to provide a fair basis for constraint-handling methods comparison, the same search engine was used in all cases, i.e. a self-adaptive variant of DE, both because of its simplicity (it does not introduce any sophisticated operator nor any non-uniform probability distribution) and its claimed superiority over GAs and PSO in terms of its computational efficiency [37, 38]. The aforementioned constraint-handling strategies have thus been embedded within DE for the solution of 14 well-known chemical engineering problems, highlighting the superiority of the gradient-based repair strategy for most of the treated examples.

The remainder of this article is organized as follows. In the next section, DE is briefly described. Then, some state-of-the-art constraint-handling techniques in EAs are reviewed. After that, the methodology developed for the computational experiments is presented along with some information regarding the test problems considered. Then, the results are presented and discussed. The last section highlights the conclusions and perspectives of this work.

2. Differential Evolution

Differential Evolution (DE), proposed by Storn and Price [39], is a simple yet efficient population-based search engine. As typical evolutionary algorithms, it relies on the Darwinian principle of *survival of the fittest* to obtain good quality solutions through the *reproduction*, *mutation*, *competition* and *selection* processes. DE has been successfully applied to optimization problems including non-linear, non-differentiable, non-convex and multi-modal functions. It has been shown that DE is fast and robust for solving these kinds of functions [39].

In DE, an initial population is randomly generated within the search space. Each individual in the population represents a solution, i.e., a vector of n decision variables. At each generation, all individuals are selected as parents to generate new solutions. The reproduction and mutation processes are carried out as follows: a mutant vector $\mathbf{v}$ is produced choosing randomly $1 + 2n_d$ individuals from the population excluding the current target parent. The first individual is a base vector, whereas the n_d subsequent individuals are paired to create difference vectors. The difference vectors are scaled by a factor F and then added to the base vector. The most widely used mutation process only considers one difference term and involves random vectors, as shown in the following equation:

$$\mathbf{v}_i = \mathbf{x}_{r_1} + F \cdot (\mathbf{x}_{r_2} - \mathbf{x}_{r_3}) \tag{2}$$

where $r_1 \neq r_2 \neq r_3 \neq i \in \{1, \ldots, NP\}$ are random indexes. The resulting vector is then recombined with the parent with a probability CR, the crossover rate factor. This crossover process produces a trial vector $\mathbf{u}$ according to:

$$u_{i,j} = \begin{cases} v_{i,j} & \text{if } w \sim U(0,1) \leq CR \text{ or } j = j_{rand}, \\ x_{i,j} & \text{otherwise} \end{cases} \tag{3}$$

where $j \in \{1, \ldots, n\}$ and j_{rand} is a randomly chosen index $\in \{1, \ldots, n\}$ ensuring that $\mathbf{u}_i$ gets at least one variable from the mutant $\mathbf{v}_i$. Finally, the trial vector is compared with its parent, that obtaining the best fitness is selected and inserted into the next population.

Depending on the mutation and crossover scheme considered, several variants of DE exist. They can be denoted as $DE/x/y/z$, where x specifies the base vector which can be *rand* (randomly chosen) or *best* (the vector with better fitness from the current population), y denotes the number of difference vectors used, and z denotes the crossover scheme that can be *bin* (independent binomial experiments) or *exp* (exponential crossover, similar to 1-point crossover in GAs). In this work, the most popular version of DE is used, that is, $DE/rand/1/bin$ which considers one randomly difference from the population and performs binomial crossover, as represented in Eqs.(2) and (3), respectively.

Furthermore, some self-adaptive versions of DE were introduced in the literature, in which the control parameters F and CR are self-adapted throughout the evolutionary process, e.g. SaDE [40], JADE [41], jDE[42], SHADE [43], DEPSO [44]. In this work, the jDE algorithm because of its low computational complexity. jDE has shown good performance solving numerical benchmark problems and has proven to be robust.

Also, it must be mentioned that DE is a technique devoted to continuous optimization, although it can easily be extended for the treatment of mixed-integer problems. In this study, binary variables are encoded as continuous variables, and their values are rounded to the next integer only for the evaluation of the fitness function.

3. Constraint-handling techniques

In this section, five popular techniques for handling constraints in evolutionary computation are presented, namely, penalty functions, Deb's feasibility rules, stochastic ranking, ε constrained method and gradient-based repair. As has been indicated in the introduction part, several other techniques exist in the specialized literature [45, 46], however there is no evidence that they significantly outperform the techniques tackled here and, in addition, these more sophisticated techniques usually involve the setting of several control parameters [47, 48, 49, 50, 51].

3.1. Penalty functions

Historically, the most common approach to incorporate constraints (both in evolutionary algorithms and in mathematical programming) involves penalty functions, which were originally proposed in the 1940s and later expanded in many research studies mainly because of their simplicity and efficiency. With this method, the fitness landscape is modified as some penalty value is added to the objective value of each infeasible individual. In their general form, penalty functions can be represented as:

$$\psi(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{p} r_i \cdot \max\{0, g_i(\mathbf{x})\}^{\alpha} + \sum_{j=1}^{q} c_j \cdot |h_j(\mathbf{x})|^{\alpha} \tag{4}$$

where $\psi(\mathbf{x})$ is the new fitness function to be minimized, r_i and c_j are positive constants called *penalty factors*, and α normally takes values of 1 or 2.

As can be noted, this implementation, though quite simple, requires the use of a number of parameters to be tuned (equals to the number of constraints) which might be impractical in highly constrained problems. For this reason, the static penalty function, the simplest form of penalty function, has remained as the most popular one:

$$\psi(\mathbf{x}) = f(\mathbf{x}) + r \cdot \phi(\mathbf{x}) \tag{5}$$

where r is the penalty coefficient and $\phi(\mathbf{x})$ is the overall constraint violation:

$$\phi(\mathbf{x}) = \sum_{i=1}^{p} \max\{0, g_i(\mathbf{x})\}^{\alpha} + \sum_{j=1}^{q} |h_j(\mathbf{x})|^{\alpha} \tag{6}$$

Although the static penalty function only needs the tuning of one parameter, its value is not straightforward to set. On the one hand, if r is too low, the search will be directed towards regions where the objective function is minimized, but the final obtained solutions are likely to be infeasible. On the other hand, if r is too high, the minimization of the overall constraint violation will be prioritized, obtaining a feasible solution in early generations with the disadvantage that, if the search space is disjointed or highly constrained, it will be very difficult to escape from the first feasible region found, the process being thus stuck in a local optimum. Ideally, the penalty should be kept as low as possible, just above the limit where the found solutions are infeasible, that is called the *minimum penalty rule*.

Furthermore, dynamic penalty functions, in which the coefficient r varies throughout the evolutionary process, have been proposed [52, 53, 54]. The constant idea in dynamic penalty functions is that allowing low values of r at early generations enables to explore the regions where the objective function is minimized, whereas a high value of r is desired at final generations in order to push the search towards the feasible region. Such an idea would work well for problems for which the unconstrained global optimum is close to its constrained global optimum, but there is no guarantee that this strategy will be efficient in all cases. Besides, the additional parameters needed to define the penalty coefficient schedule make this method less attractive than the simple static penalty function.

Finally, since a good choice of the penalty coefficient is necessary to enable a good balance between the objective function and the overall constraint violation minimizations, adaptive strategies have been suggested where information gathered from the search process is used to control the amount of penalty added to infeasible individuals. Adaptive penalty functions are not difficult to implement and they usually do not require user-defined parameters. Nevertheless, the results found in literature are not very encouraging as adaptive penalty methods usually need a lot of iterations to find the optimal solution as illustrated in [54]. For a more complete review of adaptive penalty techniques the reader is referred to [55].

3.2. Stochastic ranking

Stochastic ranking (SR) has been proposed by Runarsson and Yao [17] as an attempt to balance the relative weights of the objective and the constraint violation that occurs in penalty functions. In this method, the population is sorted following a probabilistic procedure: two individuals are compared according to their objective function with a probability P_f, otherwise, the overall constraint violation is used for the comparison as indicated in the pseudo-code presented in Fig. 1. Once the population has been sorted by SR, a part of the population assigned with highest rank is selected for recombination, thus sharing its characteristics to the next generation. In this way, the search is directed by the minimization of the objective function and by feasibility concepts at the same time.

```
for $i = 1$ to $NP$
    for $j = 1$ to $NP - 1$
        sample $u \sim U(0,1)$
        if $\phi(I_j) = \phi(I_{j+1}) = 0$ or $u < P_f$
            if $f(I_j) > f(I_{j+1})$
                swap $(I_j, I_{j+1})$
            end
        else
            if $\phi(I_j) > \phi(I_{j+1})$
                swap $(I_j, I_{j+1})$
            end
        end
    end
    if no swap done then break end
end
```

Figure 1: Stochastic ranking procedure. I is an individual in the population, P_f is the probability of using only the objective function for comparisons. The initial ranking is always generated randomly.

Since stochastic ranking was originally designed to work with Evolution Strategies (ES), which indeed requires a ranking process in its replacement mechanism, its implementation within other search paradigms is not straightforward, even if some studies have extended its use to other EAs [56, 57, 58]. Considering DE, SR could be used in two different ways: for selecting a part of the population that would participate in the mutation process, or for selecting the individuals that would survive to the next generation (after the mutation and crossover processes). In [57], the authors proposed to rank the population according to SR procedure before the mutation process: they divide the population into two parts (that they call higher and lower parts), the upper part containing the *better* individuals, i.e., the individuals ranked higher after SR. Then, for every trial vector, $\mathbf{v}_i$, the upper part contributes with two *good* individuals, while the lower part provides only one *less good* individual. This procedure was initially considered in this study, but since the obtained results were not satisfactory, SR was implemented within the selection process as follows: the new population is generated normally by DE operator, i.e., using the entire population, and then both populations (parents and offspring) are ranked according to SR. Finally, each new individual is compared with his parent, and the one ranked higher survives to the next generation.

3.3. Feasibility rules

This constraint handling technique establishes the superiority of feasible solutions over infeasible ones, that is, as opposite to the penalty functions, feasibility rules do not merge both information from constraint violation and objective function, but consider them separately. Proposed by Deb in [36], feasibility rules (also called lexicographical order) consist in a binary tournament selection according to the following criteria:

1. Any feasible solution is preferred to any infeasible solution.
2. Among two feasible solutions, that with better objective function value is preferred.
3. Among two infeasible solutions, that with smaller constraint violation is preferred.

Deb's feasibility rules represent an easy-to-implement, parameter-free technique to handle constraints. Further, due to its simplicity and its overall good performance, feasibility rules are usually the first constraint-handling technique tested for treating a given problem with EAs. However, one of the main drawbacks of this method appears when dealing with problems with a reduced and disconnected feasible region (e.g., problems with one or several equality constraints). Because any feasible solution is preferred over an infeasible one, once the algorithm has converged to some feasible region, it may be very difficult to escape from there in order to explore other regions, i.e., once the constraints are fulfilled, the algorithm is very likely to get trapped prematurely in some subregion of the search space. Moreover, considering that there are high probabilities that the optimum lies close to the feasibility boundary, slightly infeasible solutions might be more useful to the search process than solutions wide inside the feasible region. However, the feasibility rules would prefer the latter solution to the former one. In fact, feasibility rules can be seen as a limiting case of static penalty function when the penalty value takes a very high value, in this way, when comparing two solutions, that with a lesser amount of overall constraint violation will be always preferred.

3.4. ε constrained method

In order to tackle the above-mentioned issues related to feasibility rules in severely constrained problems, the ε constrained method for evolutionary algorithms has been proposed by Takahama and Sakai in 2005 [19], where a relaxation of constraints is permitted to explore constrained regions. This tolerance level over the relaxation, called the ε level, indicates the limit under which solutions are considered as feasible. Once the feasibility of solutions has been identified by means of the ε level, the lexicographical order (i.e. Deb's feasibility rules) is used for selecting the surviving individuals for the next generation. This technique has proven to be especially efficient in highly constrained problems, such as those involving equality constraints, because this relaxation, allowed at the early generations within a certain level, promotes exploration of regions that would be impossible to reach by simple feasibility rules.

The main drawback of this method is the difficulty for setting the ε parameter. It has been remarked that ε level enables a good exploration of the search space in early generations but also it is clear that ε must be 0 at some point of the evolutionary process in order to obtain feasible solutions. In [20], the authors proposed a dynamic control of ε level, according to:

$$\begin{aligned} &\varepsilon(0) = \phi(\mathbf{x}_\theta) \\ &\varepsilon(t) = \begin{cases} \varepsilon(0)(1 - \frac{t}{T_c})^{cp}, & 0 < t < T_c, \\ 0, & t \geq T_c \end{cases} \end{aligned} \tag{7}$$

where $\mathbf{x}_\theta$ is the best θ-th individual (in terms of constraint violation) in the first generation, cp is a parameter to control the speed ε level decrease and T_c represents the generation after which the ε level is set to 0 (after that, Deb's feasibility rules are considered). According to the authors, the following parameter setting works well in many problems: $\theta = 0.2NP$, $cp = 5$, $T_c \in 0.2T_{max}$. However, their tuning still constitutes a disadvantage, as it might become a harsh task. Additionally, it is important to recall that the use of the ε level according to Eq. (7) is only recommended for highly constrained problems in which the feasibility rules do not work properly, otherwise ε constrained method may get worse results than the feasibility rules in terms of efficiency and efficacy.

It should be underlined that the ε constrained method obtained the first place in the competition on constrained optimization of the Congress of Evolutionary Computation (CEC 2006) and very competitive results in CEC 2010 [20, 21]. Due to this success, ε constrained method has also been embedded in a number of algorithms for multiobjective optimization [59, 23, 60]. However, the excellent results obtained by this method in the two above-mentioned competitions did not only depend on the use of the ε level relaxation strategy by itself, but also on a additional technique, namely gradient-based repair method (presented in the next section) [26]. It is difficult to determine which of both methods contribute the most in obtaining such excellent results. In this study, the ε constrained method has been implemented with and without the gradient-based repair, in this way, the performance of both algorithms is compared.

3.5. Gradient-based repair

The gradient-based repair method, proposed by Chootinan and Chen in 2006 [26], is a constraint-handling technique that uses the gradient information derived from the constraint set to systematically repair infeasible solutions. Basically, the gradient of constraint violation is used to direct infeasible solutions toward the feasible region. The vector of constraint violations $\Delta C(\mathbf{x})$ is defined as:

$$\begin{aligned}\Delta C(\mathbf{x}) = [&\Delta g_1(\mathbf{x}), \ldots, \Delta g_m(\mathbf{x}), \\ &\Delta h_1(\mathbf{x}), \ldots, \Delta h_p(\mathbf{x})]^T\end{aligned} \tag{8}$$

where $\Delta g_i(\mathbf{x}) = \max\{0, g_i(\mathbf{x})\}$ and $\Delta h_j(\mathbf{x}) = h_j(\mathbf{x})$. This information, additionally to the gradient of constraints $\nabla C(\mathbf{x})$, is used to determine the step $\Delta\mathbf{x}$ to be added to the solution $\mathbf{x}$, according to:

$$\nabla C(\mathbf{x})\Delta\mathbf{x} = -\Delta C(\mathbf{x}) \tag{9}$$

$$\Delta\mathbf{x} = -\nabla C(\mathbf{x})^{-1}\Delta C(\mathbf{x}) \tag{10}$$

Although the gradient matrix ∇C is not invertible in general, the Moore-Penrose inverse or pseudoinverse $\nabla C(\mathbf{x})^{+}$ [61], which gives an approximate or best (least square) solution to a system of linear equations, can be used instead in Eq.(10). Thus, once the step $\Delta\mathbf{x}$ has been computed, the infeasible point $\mathbf{x}$ is moved to a *less* infeasible point $\mathbf{x} + \Delta\mathbf{x}$. This repair operation is performed with a probability P_g and repeated R_g times while the point is infeasible.

In this work, the computation of the gradient $\nabla C(\mathbf{x})$ is done numerically using forward finite differences, for all problems. Also, it is worth noting that in the original article [26] only non-zero elements of $\Delta C(\mathbf{x})$ are repaired, i.e., the gradient is only computed for constraints that are actually being violated. On the contrary, in [20] all constraints are considered in the repair process, even those that are already satisfied. The former approach has the disadvantage that a given constraint may be fulfilled at one iteration but violated in the next one, nevertheless, this is usually more efficient compared to the latter approach, in terms of number of iterations needed to get to the feasible region. In this study, the former approach is considered, i.e., only non-zero elements of $\Delta C(\mathbf{x})$ are taken into account within the repair process. Note that this procedure can produce situations where some variables lie outside their allowed variation range, so that two inequality constraints may be added for each variable, accounting for their bounds. Due to the associated computational burden in real-world optimization problems, where the number of variables may be high, these additional constraints are not considered here. Instead, an additional repair process, performed at each iteration, sets the variable value to the violated bound if necessary. The pseudo-code of the gradient-based repair procedure used in this study is presented in Fig. 2.

```
1  for $i = 1$ to $NP$
2      $t = 0$
3      sample $u \sim U(0,1)$
4      while $t < R_g$ and $\phi(\mathbf{x}) > 0$ and
       $u < P_g$
5          compute $\nabla C(\mathbf{x})$ of violating
           constraints
6          compute $\nabla C(\mathbf{x})^+$
7          compute $\Delta\mathbf{x}$
8          $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}$
9          repair $\mathbf{x}$ to its bounds
10         compute $\Delta C(\mathbf{x})$
11         $t = t + 1$
12     end
13 end
```

Figure 2: Gradient-based repair method procedure.

Even if the gradient-based repair can be considered as a constraint-handling technique in itself, using it alone would be computationally expensive, since, in highly constrained spaces, this procedure might require many iterations to reach the feasible region, and in extreme cases, a feasible solution could be impossible to obtain and therefore, usually this technique is coupled with any other constraint-handling technique. In this work, gradient-based repair is coupled with ε constrained method because, by its working mode, it presents advantages over feasibility rules when dealing with equality constraints and, in addition, methods like penalty functions or stochastic ranking might spoil the efforts made to repair infeasible solutions, due to a bad-tuned penalty factor or to the stochastic ranking sorting procedure itself. Nevertheless, for illustrative purposes, a preliminary test experiment is carried out for one instance, by applying the gradient-based repair method to each of the four constraint-handling techniques. The results are shown in Table 2, highlighting that every constraint-handling technique, when coupled with gradient-based repair, performed equally good. The identical results observed for feasibility rules and ε constrained methods are due to the ε level, which is zero at the first generation.

3.6. Computational complexity

Before any computational experiment, the five constraint-handling methods described above can be compared in terms of their respective theoretical time complexity (in the worst case). Regarding the penalty function, the feasibility rules and the ε constrained method, some constant time operation should be performed for each individual, resulting

Table 1: Brief description of example problems

Example	Decision variables Binary	Decision variables Continuous	Constraints (active)	Description
1	0	6	5(5)	Reactor network design
2	0	10	6(6)	Flowsheeting
3	1	1	2(1)	Process synthesis
4	1	2	2(2)	Process synthesis
5	1	2	3(2)	Process synthesis
6	3	2	5(3)	Process synthesis
7	2	6	8(6)	Reactor network design
8	4	3	9(5)	Process synthesis
9	3	8	9(7)	Planning problem
10	5	7	13(7)	Batch plant design
11	5	7	13(8)	Batch plant design
12	12	16	61(15)	Batch plant design
13	14	19	85(18)	Batch plant design
14	16	20	97(16)	Batch plant design

in an upper bound $O(NP)$, where NP is the population size. For stochastic ranking, the procedure described in Figure 1 involves a two-pass sorting mechanism, which is of order $O(NP^2)$. Finally, the main operations in the gradient-based repair process are the computation of the Moore-Penrose pseudoinverse matrix and the solution of equation (10), with respective upper bounds $O\left((p+q)^3\right)$ and $O\left(n^3\right)$, which are constant for a given problem. These operations should be carried out several times (the number of iterations is, however, bounded) for each infeasible individual. So, the order of the gradient-based repair procedure is $O(k \cdot NP)$ where k is a constant depending of the instance size (number of variables and constraints) and the maximum number of iterations allowed for each repair process.

4. Computational experiments

To illustrate the benefits of the above-mentioned constraint-handling techniques in process engineering applications, 14 problems have been selected as representative in the specialized literature. These problems present some mathematical characteristics typically found in engineering, e.g., non-linearities, equality and inequality constraints, binary and continuous variables. Some characteristics of these examples are provided in Table 1. In Appendix A, the complete formulation of these problems is presented in details and additional information concerning local and global optimal solutions is also given.

The algorithms previously presented were implemented with MATLAB R2017b and all the following computational experiments were carried out with a processor Intel Xeon E3-1505M v6 at 3.00 GHz and 32 Go RAM.

4.1. Parameters settings

In order to perform a fair comparison of the different constraint-handling methods, the parameters tuning has been set constant for all the test problems, so that, for a given technique, the best overall performance is obtained, excepting, obviously, the static penalty function where the tuning of the parameter r for each problem is intrinsic to this method. The actual parameters used are:

- Static penalty function. Parameter r is tuned following the *minimum penalty rule.* The precision of the parameter is set according to $r = x \times 10^y$, where x and y are integer numbers.
- Stochastic ranking. $P_f = 0.45$.
- ε constrained method. $\theta = 0.2NP$, $cp = 5$, $T_c = 0.2T_{max}$.
- Gradient-based repair. Identical parameters as for the ε constrained method above. Additionally, $P_g = 1$, $R_g = 3$.

Regarding the jDE algorithm, the only parameter to be tuned is the population size (NP), as the scaling factor (F) and crossover rate (CR) are adjusted by the algorithm. The population size is calculated as $NP = \min(100, 10n)$ where n is the number of decision variables. The algorithm stops if the current best solution is as close as 0.0001% to the reported global optimal solution or if the number of function evaluations (NFEs) exceeds 200 000. Due to the stochastic nature of evolutionary algorithms, 50 independent executions are carried out for each problem and each method.

5. Results and discussion

The results obtained for the 14 optimization test problems are summarized in Tables 3 to 6. The results are analyzed through the best, median and worst objective function values, "–" means no feasible solution was found. Feasibility and success rates represent respectively the rates of feasible and optimal solutions found out of 50 independent runs (considering the best solution found in each run). Please note that the computational times in Tables 4 and 6 represent the overall elapsed time for the 50 runs. In Tables 3 and 4 the results related for problems 1 to 9 are shown. Since problems 10 to 14 are different size instances of the same problem (the optimal design of a multi-product batch plant), their results are shown together in Tables 5 and 6.

Table 2: Gradient-based repair coupled with each constraint-handling method. Experimental results for problem 1 in terms of NFEs needed to achieve convergence.

Problem	Constr-handling	Best	Median	Worst	Mean	Std	Feas. rate	Succ. rate	CPU time(s)
1	St. penalty fcn.	211	1884	4665	1943	1009	100	100	2.6
	SR	211	2063	6151	2295	1419	100	100	3.3
	Feas. rules	211	2142	5490	2176	1226	100	100	2.9
	ε-constrained	211	2142	5490	2176	1226	100	100	2.9

Problem 1 addresses the optimal design of a sequence of two CSTR reactors. It can be considered as a small size problem, however, from Table 4 it can be appreciated that only gradient-based repair technique achieved to find the optimum in each single run. Further, the few NFE needed to converge is to be highlighted. This problem was studied in [30, 31], but the problem needed to be reformulated removing all equality constraints and eliminating dependent variables, and then a static penalty function was used. In contrast, the results obtained by the gradient-based repair method suggest that this reformulation is not necessary since this method finds the global optimum in short CPU times (lower than 0.1 second per run).

Problem 2 constitutes a difficult case, containing 6 non-linear equality constraints that involve all decision variables. Not any one of the tested constraint-handling technique except gradient-based repair was able to found the optimum in any run. In [62] this problem was also addressed by DE, considering the constraints as a system of non-linear equations which is solved by an exact algorithm, so that the original problem is transformed into an unconstrained one. However, such repair process is computationally expensive, as it is performed for every individual at each generation. In this study, the same approach has been carried out for comparison purposes. The system of non-linear equations has been solved using the Levenberg-Marquardt algorithm embedded in the MATLAB software. This approach took approximately 40 seconds per run, i.e., about 20 times more than the gradient-based repair procedure.

Problems 3 and 5 consist in small and rather simple MINLP examples. All constraint-handling methods obtained an overall good performance in terms of success rate.

Regarding problem 4, this problem is modelled as a MINLP involving one non-linear equality constraint and one binary variable, which together, yield a rather high difficulty for the solution by feasibility rules, since this technique gets trapped in an "easy-to-access" local optimum. In addition, all the other techniques obtain a very good performance. It is noteworthy that in [29, 32], the problem is reformulated by reducing one continuous variable and thus eliminating the equality constraint. This approach, although efficient, is problem-devoted and may not be practical in highly constrained

Table 3: Experimental results in terms of objective function values.

Problem (`optimum`)	Constr-handling	Best	Median	Worst	Mean	Std
1 (`-0.38881`)	St. penalty fcn.	−0.38881	−0.38871	−0.38802	−0.38870	1.42e-04
	SR	−0.38881	−0.38871	−0.37867	−0.38813	1.87e-03
	Feas. rules	−0.38881	−0.38871	−0.38377	−0.38854	7.77e-04
	ε-constrained	−0.38881	−0.38871	−0.38720	−0.38848	4.24e-04
	Grad-based repair	−0.38881	−0.38876	−0.38872	−0.38878	3.92e-05
2 (`9490593`)	St. penalty fcn.	10 041 455	12 562 687	16 932 972	12 626 104	1.32e+06
	SR	—	—	—	—	—
	Feas. rules	10 148 136	12 421 765	18 043 019	12 667 483	1.40e+06
	ε-constrained	10 603 444	—	—	13 457 709	1.88e+06
	Grad-based repair	9 490 594	9 490 600	9 490 603	9 490 600	2.51e+00
3 (`2.000`)	St. penalty fcn.	2.000	2.000	2.236	2.005	3.34e-02
	SR	2.000	2.000	2.000	2.000	3.00e-05
	Feas. rules	2.000	2.000	2.236	2.019	6.47e-02
	ε-constrained	2.000	2.000	2.236	2.014	5.66e-02
	Grad-based repair	2.000	2.000	2.000	2.000	3.03e-05
4 (`2.124`)	St. penalty fcn.	2.124	2.124	2.558	2.168	1.31e-01
	SR	2.124	2.124	2.558	2.142	8.58e-02
	Feas. rules	2.124	2.558	2.558	2.549	6.13e-02
	ε-constrained	2.124	2.124	2.558	2.150	1.04e-01
	Grad-based repair	2.124	2.124	2.124	2.124	1.71e-05
5 (`1.0765`)	St. penalty fcn.	1.0766	1.0766	1.2500	1.0801	2.45e-02
	SR	1.0766	1.0766	1.2500	1.0835	3.43e-02
	Feas. rules	1.0766	1.0766	1.2500	1.0880	4.19e-02
	ε-constrained	1.0766	1.0766	1.0766	1.0766	1.80e-05
	Grad-based repair	1.0765	1.0766	1.0766	1.0766	3.14e-05
6 (`7.667`)	St. penalty fcn.	7.667	7.667	7.931	7.688	7.22e-02
	SR	7.667	7.667	7.931	7.693	7.99e-02
	Feas. rules	7.667	7.931	8.240	7.928	1.07e-01
	ε-constrained	7.667	7.931	7.931	7.846	1.24e-01
	Grad-based repair	7.667	7.667	7.667	7.667	1.49e-05
7 (`99.238`)	St. penalty fcn.	99.238	99.240	107.374	101.355	3.60e+00
	SR	99.238	99.239	107.374	100.703	3.16e+00
	Feas. rules	99.238	107.374	—	111.974	2.30e+01
	ε-constrained	99.238	107.374	107.374	103.795	4.08e+00
	Grad-based repair	99.238	99.240	99.240	99.239	2.69e-04
8 (`4.57958`)	St. penalty fcn.	4.57958	4.57962	4.57968	4.57962	3.18e-05
	SR	4.57958	4.57967	4.57968	4.57966	1.69e-05
	Feas. rules	4.57958	4.57966	4.57968	4.57966	1.86e-05
	ε-constrained	4.57958	4.57966	4.57968	4.57966	2.14e-05
	Grad-based repair	4.57958	4.57958	4.57964	4.57958	8.95e-06
9 (`-1.9231`)	St. penalty fcn.	−1.9231	−1.7236	−1.4125	−1.6925	1.95e-01
	SR	−1.9231	−1.7235	−0.2202	−1.5924	4.51e-01
	Feas. rules	−1.4125	−1.2138	0.7607	−0.8621	6.70e-01
	ε-constrained	−1.4099	−0.0011	0.7431	−0.0370	3.71e-01
	Grad-based repair	−1.9231	−1.9231	−1.9230	−1.9230	1.31e-04

Table 4: Experimental results in terms of NFEs needed to achieve convergence.

Problem	Constr-handling	Best	Median	Worst	Mean	Std	Feas. rate	Succ. rate	CPU time(s)
1	St. penalty fcn.	14 340	64 230	200 000	88 734	73 334	100	90	10.1
	SR	45 480	156 840	200 000	153 434	44 795	100	70	91.6
	Feas. rules	27 660	149 490	200 000	136 008	53 187	100	86	15.6
	ε-constrained	32 700	193 320	200 000	156 421	61 826	100	54	19.1
	Grad-based repair	211	2142	5490	2176	1226	100	100	2.9
2	St. penalty fcn.	200 000	200 000	200 000	200 000	0	100	0	19.8
	SR	200 000	200 000	200 000	200 000	0	0	0	142.2
	Feas. rules	200 000	200 000	200 000	200 000	0	100	0	19.3
	ε-constrained	200 000	200 000	200 000	200 000	0	20	0	21.0
	Grad-based repair	15 300	19 100	22 800	19 094	1366	100	100	90.3
3	St. penalty fcn.	500	870	200 000	4851	28 162	100	98	1.2
	SR	640	1350	2640	1423	444	100	100	1.2
	Feas. rules	520	1110	200 000	17 006	54 511	100	92	4.3
	ε-constrained	420	1550	200 000	13 583	47 582	100	94	3.8
	Grad-based repair	33	203	387	226	89	100	100	0.3
4	St. penalty fcn.	1680	2775	200 000	22 441	59 792	100	90	3.6
	SR	3540	6825	200 000	14 887	38 229	100	96	6.6
	Feas. rules	10 200	200 000	200 000	196 214	26 843	100	2	31.2
	ε-constrained	24 090	28 245	200 000	38 545	41 240	100	94	7.4
	Grad-based repair	100	108	676	197	151	100	100	0.2
5	St. penalty fcn.	1830	2895	200 000	6804	27 885	100	98	1.2
	SR	4620	6465	200 000	14 212	38 332	100	96	6.0
	Feas. rules	2610	4065	200 000	19 623	53 738	100	92	3.2
	ε-constrained	22 800	28 575	31 200	27 988	2182	100	100	6.0
	Grad-based repair	93	175	1046	244	173	100	100	0.3
6	St. penalty fcn.	2400	4125	200 000	19 780	53 686	100	92	2.5
	SR	10 150	14 500	200 000	33 204	56 229	100	90	15.8
	Feas. rules	3900	200 000	200 000	184 422	53 365	100	8	22.2
	ε-constrained	14 550	200 000	200 000	143 018	83 935	100	32	18.2
	Grad-based repair	195	198	360	207	37	100	100	0.3
7	St. penalty fcn.	16 880	22 280	200 000	68 891	78 739	100	74	6.1
	SR	39 120	69 600	200 000	92 301	52 791	100	82	60.9
	Feas. rules	92 720	200 000	200 000	190 869	23 914	86	22	16.4
	ε-constrained	35 280	200 000	200 000	128 550	81 429	100	44	11.8
	Grad-based repair	3761	9591	20 639	10 082	3574	100	100	10.3
8	St. penalty fcn.	5810	6720	8540	6891	714	100	100	0.9
	SR	8330	11 480	14 770	11 739	1485	100	100	8.2
	Feas. rules	7840	9415	11 270	9362	785	100	100	1.3
	ε-constrained	8120	25 760	30 730	23 408	6563	100	100	3.3
	Grad-based repair	407	2133	9879	3691	3276	100	100	3.3
9	St. penalty fcn.	47 800	200 000	200 000	159 822	65 129	100	28	16.0
	SR	149 200	200 000	200 000	197 052	9930	100	18	149.5
	Feas. rules	200 000	200 000	200 000	200 000	0	100	0	20.8
	ε-constrained	200 000	200 000	200 000	200 000	0	100	0	20.9
	Grad-based repair	16 197	41 450	77 610	41 982	10 260	100	100	66.3

real-world problems.

Problem 6 takes into account a MINLP problem with 3 binary variables and 2 equality constraints. Although this problem can be considered as a small one, its characteristics are not easy to overcome by feasibility rules, meaning that the first feasible solution is likely to be found far from the global minimum region. Further, the relaxation done by ε constrained method does not manage to obtain acceptable success rates, at least not with the parameters used here. Regarding stochastic ranking, static penalty function and gradient-based repair, they solve the problem efficiently, with much lower CPU times reported for the gradient-based repair technique. In [28, 31], the same problem was tackled, but the model was simplified by eliminating the continuous variables by means of the equality constraints.

For problem 7, feasibility rules and ε constrained method present a poor performance due to the existence of 2 binary variables and 4 equality constraints. Stochastic ranking and static penalty function present a fairly good performance. On the contrary, gradient-based repair method enables the algorithm to search in the whole search space before converging to an optimum. Again, this example was addressed in previous works [28, 29, 32, 31] by reformulating the problem in order to eliminate equality constraints and simultaneously, reducing the number of decision variables, and then, the remaining constraints are handled by a static penalty function.

For problem 8 all constraint-handling techniques performed excellently, finding the global optimum in all runs. For this problem the first feasible region found coincide with the region where the global optimal solution lies, even though the problem present some difficulties regarding its mathematical properties (4 binary variables with 9 constraints.

Problem 9 constitutes a planning problem in which several alternatives are proposed for obtaining one desired product. Since it contains 3 binary variables and 5 equality constraints involving all the continuous variables, the resulting problem results difficult if the global optimum is required. For SR technique, the balance between feasible and infeasible solutions is not enough to reach the global optimum region in most cases. With respect to Deb's feasibility rules, convergence to the global optimum is highly unlikely, since this technique always prefers feasible solutions over the infeasible ones, whatever the quality of the objective function value. For the ε constrained method, the relaxation conducted on the equality constraints seems not to be sufficient to reach the global optimum. On the other hand, when this relaxation is combined with the gradient-based repair, the algorithm is able to search over the entire space so that the global optimum is found.

Problems 10–14 consider the optimal design of a multi-product batch plant consisting of a given number of processing stages M through which several products N have to be manufactured. The objective is to minimize the investment cost and then, for each processing stage j, the number of parallel units N_j and their sizes V_j need to be determined, as well as the batch sizes B_i and cycle times $T_{\mathrm{L}i}$ for each product i. Thus, increasing the number of stages M, the number of products N and the possible number of parallel units N_j^{u}, results in a large non-convex MINLP. Note that the mathematical formulation of the problem implemented in this study presents multiple non-convexities in the objective function and in several inequality constraints (see Appendix A). Also, in order to explore the scalability of the studied constraint-handling techniques, problems 13 and 14 have been artificially created increasing the size of problem 12, both in number of variables and in number of constraints. Thus, since no global optimum is reported in the literature for these problems, a convexified formulation of the problem was solved using BARON solver in GAMS environment, which obtained solutions are reported in Table 5. In this way, once the optimum has been computed, a better comparison of the constraint-handling techniques can be performed.

Problem 10 and 11 are equivalent in size, their only difference is the quantity Q_i of product i that needs to be manufactured. Nevertheless, performance of constraints-handling techniques like ε constrained, feasibility rules and SR are significantly different for both problems. It seems that this slight modification makes problem 11 much more difficult for these techniques, the feasible region has been modified in such a way that the global optimum lies now in a region that is difficult to reach. Besides, the consistency of the static penalty function and gradient-based repair methods is observed in both problems, as their performance remained unchanged in terms of success rate.

Problem 12 can be viewed as a medium size instance, it contains 6 integer variables (processing stages) with 4 possible values each (the equivalent to 12 binary variables), so the problem size is equivalent to solving 4 096 NLP subproblems. Additionally, it has 16 continuous variables and 61 inequality constraints. The global optimum corresponds to an ill-conditioned point, since variations as small as 0.01% in any of the 16 continuous variables produce infeasibility. For this problem, no constraint-handling technique could obtain the reported optimal solution. However, stochastic ranking and gradient-based repair were able to locate the global optimum region: SR in 20% of the runs and 100% of the runs for gradient-based repair. It seems that once the global optimal region has been identified, new solutions generated by DE operator are very likely to be infeasible, and even if the repair process acts upon them, the direction in which constraint violation is minimized is not necessarily the same as the direction in which the objective function decreases, so that the optimization process gets very slow.

For problem 13, an additional processing stage and an additional product are considered with respect to problem 12. The pattern in performance observed in previous

Table 5: Experimental results for problems 10–14 in terms of objective function values.

Problem (`optimum`)	Constr-handling	Best	Median	Worst	Mean	Std
10 (`38499.5`)	St. penalty fcn.	38 499.5	38 500.1	38 500.2	38 500.1	5.33e-02
	SR	38 499.5	38 499.8	38 499.8	38 499.8	2.39e-02
	Feas. rules	38 499.5	38 500.1	38 500.2	38 500.1	5.84e-02
	ε-constrained	38 499.5	38 500.2	40 977.5	38 747.9	7.51e+02
	Grad-based repair	38 499.5	38 499.7	38 499.8	38 499.7	8.77e-02
11 (`106755.8`)	St. penalty fcn.	106 755.8	106 756.8	106 756.9	106 756.8	8.69e-02
	SR	106 755.8	106 755.9	112 947.6	107 009.4	1.23e+03
	Feas. rules	106 755.8	112 947.2	122 607.8	110 739.1	4.28e+03
	ε-constrained	106 755.8	122 607.8	136 009.7	126 123.0	1.02e+04
	Grad-based repair	106 755.8	106 755.8	106 755.9	106 755.8	1.67e-02
12 (`285506.5`)	St. penalty fcn.	304 660.5	310 155.0	311 349.9	308 282.6	2.66e+03
	SR	286 826.0	308 092.0	—	313 469.9	1.54e+04
	Feas. rules	310 350.1	322 711.5	332 793.1	322 466.2	7.04e+03
	ε-constrained	305 311.9	330 042.1	370 131.6	330 407.5	1.48e+04
	Grad-based repair	285 550.6	285 868.6	286 497.8	285 911.8	2.41e+02
13 (`430324.5`)	St. penalty fcn.	431 403.9	455 438.4	—	452 197.8	9,27E+03
	SR	450 347.2	—	—	476 650.0	3.40E+04
	Feas. rules	457 559.5	461 764.5	479 289.1	463 844.8	5.35E+03
	ε-constrained	444 914.7	467 524.0	576 092.0	470 245.2	1.99E+04
	Grad-based repair	430 418.2	431 809.4	444 650.7	432 786.3	2.99E+03
14 (`546998.6`)	St. penalty fcn.	550 965.5	564 520.4	—	564 746.7	3.72E+03
	SR	—	—	—	—	—
	Feas. rules	563 508.1	574 875.2	592 240.3	574 459.7	7.68E+03
	ε-constrained	562 878.8	584 552.9	663 862.1	585 503.9	1.64E+04
	Grad-based repair	549 496.6	553 317.3	562 335.7	555 301.5	4.53E+03

Table 6: Experimental results for problems 10–14 in terms of NFEs needed to achieve convergence.

Problem	Constr-handling	Best	Median	Worst	Mean	Std	Feas. rate	Succ. rate	CPU time(s)
10	St. penalty fcn.	41 600	47 300	59 100	48 120	3810	100	100	5.8
	SR	66 000	76 900	105 000	79 904	9492	100	100	66.1
	Feas. rules	54 900	62 650	84 600	63 884	5679	100	100	7.9
	ε-constrained	68 400	85 850	200 000	95 892	35 719	100	90	12.3
	Grad-based repair	31 684	48 641	63 401	47 392	7172	100	100	61.9
11	St. penalty fcn.	43 100	48 150	58 500	48 542	3222	100	100	6.2
	SR	136 000	183 700	200 000	178 034	21 151	100	68	153.1
	Feas. rules	69 000	200 000	200 000	144 760	60 730	100	46	17.6
	ε-constrained	87 900	200 000	200 000	191 724	28 443	100	8	24.2
	Grad-based repair	2744	18 337	32 626	18 142	9130	100	100	27.3
12	St. penalty fcn.	200 000	200 000	200 000	200 000	0	100	0	84.4
	SR	200 000	200 000	200 000	200 000	0	62	0	177.2
	Feas. rules	200 000	200 000	200 000	200 000	0	100	0	72.5
	ε-constrained	200 000	200 000	200 000	200 000	0	100	0	88.1
	Grad-based repair	200 000	200 000	200 000	200 000	0	100	0	1136.7
13	St. penalty fcn.	200 000	200 000	200 000	200 000	0	96	0	72.4
	SR	200 000	200 000	200 000	200 000	0	10	0	268.5
	Feas. rules	200 000	200 000	200 000	200 000	0	100	0	67.0
	ε-constrained	200 000	200 000	200 000	200 000	0	100	0	71.1
	Grad-based repair	200 000	200 000	200 000	200 000	0	100	0	944.3
14	St. penalty fcn.	200 000	200 000	200 000	200 000	0	88	0	70.6
	SR	200 000	200 000	200 000	200 000	0	0	0	269.3
	Feas. rules	200 000	200 000	200 000	200 000	0	100	0	76.3
	ε-constrained	200 000	200 000	200 000	200 000	0	100	0	74.4
	Grad-based repair	200 000	200 000	200 000	200 000	0	100	0	994.2

problem is highlighted here since this time 26 additional constraints need to be fulfilled and one integer variable has been added. Table 6 shows that no constraint-handling technique achieves the optimum in any run. However, in can be appreciated in Table 5 that static penalty function and gradient-based repair found near-optimal solutions, and besides, according to the median solution obtained for each constraint-handling technique those obtained with gradient-based repair can be considered as good quality solutions.

Finally, problem 14 is the biggest instance considered here. It considers 6 different products to be manufactured in 8 processing stages, the resulting problem contains 97 inequality constraints and is equivalent to solving 65 536 NLP subproblems. Considering the results from Table 5, it can be observed that SR was unable to find any single feasible solution in any run; considering both the best and median solutions obtained using the feasibility rules, it can be concluded that this technique does not allow to obtain near-optimal solutions. Besides, the performance of ε constrained method is quite similar to that of feasibility rules, and even more, if their mean solutions are compared, it is observed that ε constrained worsens the performance of Deb's rules. The static penalty function presents an overall good performance, even if the global optimum was not found in any run. Nevertheless, since, according to the minimum penalty rule, the best quality solutions are obtained using the lowest possible value for the penalty factor, this also implies that infeasible solutions are more likely to be obtained, as it was the case for this problem in 12% of executions. Again, gradient-based repair yields the best results according to the quality of solutions obtained (comparing for example the median solution in Table 5). However, due to the high number of constraints in the problem, the computation of the pseudoinverse of the gradient is very expensive, requiring approximately twice more computational time than a classical constraint-handling technique.

As it has been pointed out above, problems 12, 13 and 14 constitute difficult problems in part because of the high number of variables and constraints. For these problems, not one constraint-handling technique was able to find the global optimum in any run. However, some techniques obtained good-quality solutions, which are presumably located in the global optimum region. Thus, in order to further investigate the performance of these techniques and to evaluate their ability to identify sub-optimal solutions lying in the region of the global optimum, the use of an additional local search is explored. The local optimizer Successive Quadratic Programming (SQP) is applied with a probability $0.1/NP$ for each individual, in this way, one individual is improved on average every 10 generations. For ε constrained and gradient-based repair, this local search procedure is carried out only after the ε level is equal to 0, i.e., once the algorithm has likely identified the optimal region. The obtained results are displayed in Tables 7 and 8.

Table 7: Experimental results for problems 10–14 in terms of objective function values using a local search (SQP).

Problem (`optimum`)	Constr-handling	Best	Median	Worst	Mean	Std
	St. penalty fcn.	285 506.5	300 301.8	310 130.8	295 823.4	9.30e+03
	SR	285 506.5	285 506.5	—	288 383.8	1.41e+04
12	Feas. rules	285 506.5	304 660.0	329 222.0	308 645.8	1.03e+04
(`285506.5`)	ε-constrained	285 506.5	315 532.8	—	314 852.7	1.23e+04
	Grad-based repair	285 506.5	285 506.5	300 804.9	285 812.5	2.16e+03
	St. penalty fcn.	430 324.5	430 324.5	—	433 012.9	5,18E+03
	SR	430 324.5	430 324.5	575 159.2	443 431.2	2,64E+04
12	Feas. rules	430 324.5	454 395.7	458 842.3	452 870.8	7,67E+03
(`430324.5`)	ε-constrained	430 324.5	454 395.7	466 127.0	449 351.8	1,02E+04
	Grad-based repair	430 324.5	430 324.5	441 482.2	430 547.7	1,58E+03
	St. penalty fcn.	546 998.6	547 468.5	561 584.7	551 817.9	5,74E+03
	SR	546 998.6	—	—	617 325.4	7,46E+04
12	Feas. rules	558 256.9	563 403.0	576 658.5	563 573.9	4,04E+03
(`546998.6`)	ε-constrained	547 468.5	563 403.0	611 740.9	564 695.0	9,35E+03
	Grad-based repair	546 998.6	547 468.5	558 256.9	549 355.9	3,83E+03

Table 8: Experimental results for problems 10–14 in terms of NFEs needed to achieve convergence using a local search (SQP).

Problem	Constr-hand.	Best	Median	Worst	Mean	Std	Feas. rate	Succ. rate	SQP calls	CPU time(s)
12	St. penalty fcn.	31 165	200 000	200 000	153 508	60 053	100	42	128.5	148.0
	SR	4660	96 226	200 000	100 980	45 765	98	94	84.9	278.9
	Feas. rules	158 731	200 000	200 000	198 708	6665	100	6	162.8	198.4
	ε-constrained	135 830	200 000	200 000	198 782	9085	98	2	133.8	162.5
	Grad-based rep.	42 565	60 063	119 588	63 144	19 097	100	100	6.1	217.0
13	St. penalty fcn.	2140	84 187	200 123	98 076	64 780	100	78	76.0	198.8
	SR	130 502	200 000	200 000	195 854	14 052	36	10	167.0	1201.2
	Feas. rules	130 979	200 000	200 000	197 779	10 963	100	6	152.6	332.3
	ε-constrained	101 029	200 000	200 000	193 700	19 877	100	12	151.7	576.8
	Grad-based rep.	43 794	63 815	200 000	75 436	31 327	100	98	8.9	429.3
14	St. penalty fcn.	54 784	200 000	200 000	178 757	43 890	100	28	122.7	452.0
	SR	162 276	200 000	200 000	199 317	5346	14	2	171.1	1422.8
	Feas. rules	200 000	200 000	200 000	200 000	0	100	0	140.2	368.8
	ε-constrained	200 000	200 000	200 000	200 000	0	100	0	146.4	524.3
	Grad-based rep.	91 052	200 000	200 000	188 084	30 139	100	20	35.9	979.1

The NFEs reported in Table 8 take into account the evaluations of the objective function performed by both jDE and SQP. The column "SQP calls" represents the average number of individuals on which the local search is carried out before the algorithm stops. For example in problem 12, using gradient-based repair, the SQP solver needed to operate on average 6 individuals before finding the global optimum, whereas for feasibility rules, SQP was called on average 162 times before the algorithm attains the maximum NFE (more likely than finding the optimum, from table 8).

Outstanding results for problem 12 are obtained by stochastic ranking and gradient-based repair, obtaining success rates of 94% and 100%, respectively. Conversely, a poor performance considering the success rate is still observed for static penalty, feasibility rules and ε constrained methods. These results are in agreement with those of Table 5 where no local search is employed, that is, the local search is beneficial only if the global region has been identified. This feature is also observed for problems 13 and 14. For problem 13, those constraint-handling techniques that showed a poor performance before the use of SQP (SR, feasibility rules and ε constrained) present now the same tendency, with a significant additional CPU time, due to the use of local search. In addition, static penalty function obtains an acceptable success rate (78%) and gradient-based repair solved to optimality this problem in all runs, excepting one. On the contrary, problem 14 still constitutes a difficult problem, even if local search is applied. The high number of binary variables suggests that using a higher number of function evaluations might be necessary to solve this problem.

With respect to the use of the local search, it can be concluded that it can be advantageous only when the population-based algorithm succeeds to find promising regions, otherwise, significant improvements in performance will likely not be observed. Further, the computational costs associated to the use of a local search must also be taken into account.

Summarizing, the above empirical study highlighted the importance of using an efficient constraint-handling technique when solving PE optimization problems, which are in general highly constrained problems. The numerical results obtained point out that gradient-based repair method is the most robust and thus promising method among the techniques studied here. Also, as mentioned previously, one of the main motivations for using metaheuristics in PE area is their ability to simultaneously optimize multiple criteria. To illustrate this, in the next subsection, the gradient-based repair method is used to solve a biobjective version of the flow sheet example (Problem 2).

5.1. Additional example: Biobjective case study

This problem, presented in [62], considers the maximization of two objectives: the profit before taxes (PBT) and the net present value (NPW) for the Williams &

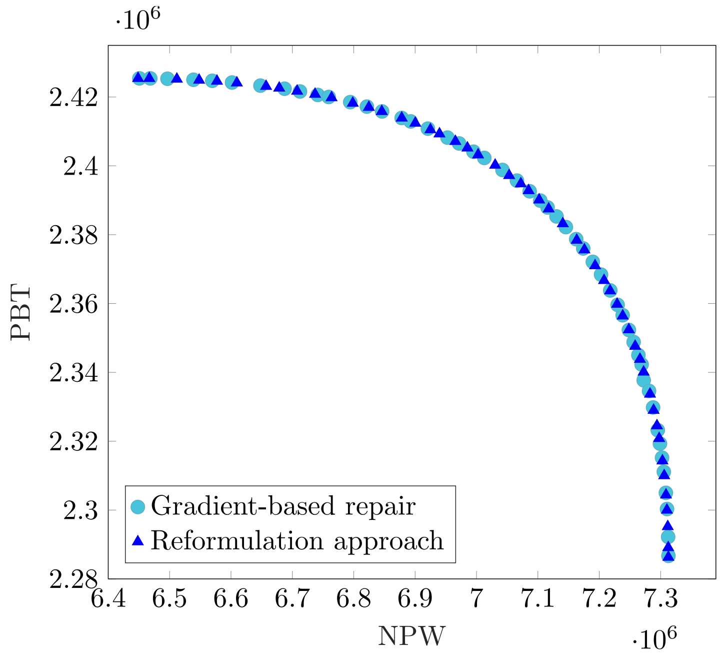

Figure 3: Approximation of the Pareto front of the biobjective Williams & Otto process example using NSGA-II with gradient-based repair method (about 1.5 s) and the reformulation approach (about 45 s).

Otto process problem. The nondominated sorting genetic algorithm II (NSGA-II)[63] coupled with gradient-based repair as constraint-handling technique is used as a solution technique. Also, for comparison purposes, the constraint-handling strategy presented in [62], in which all equality constraints are eliminated by means of solving a system of nonlinear equations, is explored. The obtained approximations to the Pareto front of this problems are presented in Figure 3.

The non-dominated solutions are obtained in one single run, unlike mathematical programming techniques in which multiple runs are needed to produce an approximation of the Pareto front. The importance of an efficient constraint-handling technique is also to be highlighted: no other constraint-handling technique studied in this work was able to find a non-dominated solution in the real Pareto front, actually no feasible solutions could be found except with the gradient-based repair procedure. Besides, the reformulation approach proposed in [62] was time consuming, taking approximately 45 seconds per run (since the solution of the system of nonlinear equations has to be performed for every evaluation of the objective function) while, in contrast, the gradient-based repair approach takes approximately 1.5 seconds per run. Also, it is worth mentioning that the Pareto front approximations obtained by both approaches are comparable, i.e., no approach outperforms the other.

6. Conclusions and perspectives

In this study, the performance of several constraint-handling techniques for EAs has been compared for the solution of a set of 12 test bench problems from the PE area. The empirical analysis conducted showed that the results' quality greatly depends on the constraint-handling technique used for the solution of problems with high number of constraints or binary variables.

The analysis of the dedicated literature has shown that the most widely used approach within EAs considers the reformulation of the model and the use of static penalty functions or feasibility rules as constraint-handling techniques. However, the results obtained in this work highlighted that the performance of this strategy, though acceptable in some cases, proved to be poor in others. Besides, among the constraint-handling techniques considered in this study, the gradient-based repair method deserves a special attention, as this constraint-handling technique was the only one capable of finding the global optimum region in all test problems. Coupled with ε constrained method, the search algorithm promotes the exploration of promising regions over the entire search space instead of getting trapped into a local optimum. It is worth emphasizing that, even if this method needs supplementary information (computation of constraints' gradient), its excellent results both in terms of computational time and solution quality encourage its use. In addition, the use of gradient-based repair method in highly constrained mixed-integer problems seems to be not only adequate, but necessary in order to obtain satisfactory results. Finally, this work highlighted the unquestionable benefits obtained using this constraint-handling method, usually under-estimated in the devoted literature. Therefore, these conclusions allow reconsidering evolutionary algorithms as a serious approach for solving highly-constrained real-world optimization problems.

Besides, the good performance exhibited in the solution of the biobjective case study, allows to contemplate the solution of bigger instances of PE multi-objective problems. Also, as the gradient-based repair method can be coupled with any multi-objective evolutionary algorithm (MOEA), the solution of multi-objective MINLP problems related to PE, using more sophisticated MOEAs is under the scope of future work.

Acknowledgments

The first author gratefully thanks the Mexican Council of Science and Technology (CONACyT) for scholarship support to pursue PhD studies.

Appendix A. Test problems

This appendix describes the 12 global optimization test problems considered in this study. For all problems, the global optimum solution is reported as found in the literature. Additional information related to local optima and active constraints is also given.

Example 1. Reactor network design. Proposed in [64], this problem involves the design of a sequence of two reactors of type CSTR, where the consecutive reactions A $\rightarrow$ B $\rightarrow$ C takes place. The objective is to maximize the concentration of product B (x_4) in the exit stream. The mathematical model is as follows:

$$\begin{aligned}
\min \quad & f(\boldsymbol{x}) = -x_4 \\
\text{s.t.} \quad & g_1(\mathbf{x}) = x_5^{0.5} + x_6^{0.5} - 4 \leq 0 \\
& h_1(\mathbf{x}) = x_1 + k_1 x_1 x_5 - 1 = 0 \\
& h_2(\mathbf{x}) = x_2 - x_1 + k_2 x_2 x_6 = 0 \\
& h_3(\mathbf{x}) = x_3 + x_1 + k_3 x_3 x_5 - 1 = 0 \\
& h_4(\mathbf{x}) = x_4 - x_3 + x_2 - x_1 + k_4 x_4 x_6 = 0 \\
& \quad 0 \leq x_i \leq 1, \quad i = \{1,2,3,4\} \\
& \quad 1\text{e}-5 \leq x_i \leq 16, \quad i = \{5,6\}
\end{aligned}$$

where $k_1 = 0.09755988$, $k_2 = 0.99k_1$, $k_3 = 0.0391908$, $k_4 = 0.9k_3$. The global optimum is at $\mathbf{x}^* = [0.771462,\ 0.516997,\ 0.204234,\ 0.388812,\ 3.036504,\ 5.096052]$, with $f(\mathbf{x}^*) = -0.388812$. Constraint g_1 is active. This example possesses a local minimum with an objective function value that is very close to that of the global solution. This local solution is at $\mathbf{x} = [1,\ 0.393,\ 0,\ 0.3881,\ 0,\ 16]$ with $f = -0.3881$. Interestingly, this solution utilizes only one of the two reactors whereas the global solution makes use of both reactors.

Example 2. Flowsheeting. This problems considers the optimization of a flow sheet example of the Williams & Otto process [65, 66]. Reactants A and B and the recycle stream enter the continuous-flow stirred-tank reactor, where the main product P is produced together with one by-product E and the waste product G, while C is an intermediate.

$$\begin{aligned}
\text{A} + \text{B} &\rightarrow \text{C} \\
\text{C} + \text{B} &\rightarrow \text{P} + \text{E} \\
\text{P} + \text{C} &\rightarrow \text{G}
\end{aligned}$$

In the decanter, component G is entirely removed from the other components. Product P is removed from the overhead of the distillation column, but some of the product is retained in the bottom due to the formation of an azeotrope. Part of the bottom stream is purged in order to avoid accumulation of the by-product, while most of it is recycled to the reactor. The purge stream has a substantial fuel value and can be sold on the market. The optimization variables account for the reactor volume, the reaction temperature, the purge fraction and the mass flow for each component, except for component P which is equal to 2160 kg/h. The objective is to minimize the total annual cost. The model is formulated as:

$$\begin{aligned}
\min \quad f(\mathbf{x}) = \frac{1}{0.453}\Big[&168x_5 + 252x_1 + 2.22\big[x_1 + x_5 + \sum_{i=6}^{8}(1-x_4)x_i + 1.1(1-x_4)x_9\big] \\
&+ 84x_{10} + 60x_2\rho\Big] + 1041.6
\end{aligned}$$

s.t.

$$\begin{aligned}
h_1(\mathbf{x}) &= x_5 + x_6(1-x_4) - \frac{k_1x_6x_7x_2\rho}{q_3^2} - x_6 = 0 \\
h_2(\mathbf{x}) &= x_1 + x_7(1-x_4) - \frac{(k_1x_6 + k_2x_8)x_7x_2\rho}{q_3^2} - x_7 = 0 \\
h_3(\mathbf{x}) &= x_8(1-x_4) + \frac{(2k_1x_6x_7 - 2k_2x_7x_8)x_2\rho}{q_3^2} \\
&\quad + \frac{(-k_3x_8(2160 + 0.1x_9))x_2\rho}{q_3^2} - x_8 = 0 \\
h_4(\mathbf{x}) &= x_9(1-x_4) + \frac{2k_2x_7x_8x_2\rho}{q_3^2} - x_9 = 0 \\
h_5(\mathbf{x}) &= \frac{x_9(1-x_4)}{10} + \frac{(k_2x_7 - 0.5k_3(2160 + 0.1x_9))x_8x_2\rho}{q_3^2} \\
&\quad - 2160 + 0.1x_9 = 0 \\
h_6(\mathbf{x}) &= \frac{1.5k_3(2160 + 0.1x_9)x_8x_2\rho}{q_3^2} - x_{10} = 0
\end{aligned}$$

$$1\text{e}4 \le x_1 \le 1.5\text{e}4$$

$$0.85 \le x_2 \le 10$$

$$322 \le x_3 \le 378$$

$$0 \le x_4 \le 0.99$$

$$0 \le x_i \le 1\text{e}5 \quad \forall i \in \{5, \dots, 10\}$$

where

$$\begin{aligned}
q_3 &= x_6 + x_7 + x_8 + 1.1x_9 + x_{10} + 2160 \\
k_1 &= 5.9755\text{e}9 \cdot \exp\left(\frac{-1.2\text{e}4}{x_3}\right) \\
k_2 &= 2.5962\text{e}12 \cdot \exp\left(\frac{-1.5\text{e}4}{x_3}\right) \\
k_3 &= 9.6283\text{e}15 \cdot \exp\left(\frac{-2\text{e}4}{x_3}\right) \\
\rho &= 801
\end{aligned}$$

The optimum lies at $\mathbf{x}^* = [10878.60,\ 7.90,\ 342.11,\ 0.102,\ 4807.37,\ 11122.40,\ 39668.61,\ 2874.52,\ 61925.59,\ 1101.336]$ with $f(\mathbf{x}^*) = 9490592.6$.

Example 3. Process synthesis MINLP. This is a little process synthesis problem with only two decision variables. It was proposed by [67], and also found in [64]:

$$\begin{aligned}
\min \quad & f(\mathbf{x}) = 2x_1 + x_2 \\
\text{s.t.} \quad & g_1(\mathbf{x}) = 1.25 - x_1^2 - x_2 \le 0 \\
& g_2(\mathbf{x}) = x_1 + x_2 - 1.6 \le 0 \\
& 0 \le x_1 \le 1.6 \\
& x_2 = \{0, 1\}
\end{aligned}$$

The global minimum is [0.5, 1] with $f = 2$. There is a local minimum at [1.118, 0] with $f = 2.236$. Constraint g_1 is active.

Example 4. MINLP. This example is taken from [68]:

$$\begin{aligned}
\min \quad & f(\mathbf{x}) = 2x_1 + x_2 - x_3 \\
\text{s.t.} \quad & g_1(\mathbf{x}) = -x_1 + x_2 + x_3 \le 0 \\
& h_1(\mathbf{x}) = x_1 - 2\exp(-x_2) = 0 \\
& 0.5 \le x_1 \le 1.4 \\
& 0 \le x_2 \le 2 \\
& x_3 = \{0, 1\}
\end{aligned}$$

There is one local optimum at [0.853, 0.853, 0] with $f = 2.558$. The global minimum is $\{\mathbf{x}^*; f(\mathbf{x}^*)\} = \{1.375,\ 0.375,\ 1;\ 2.124\}$. Constraint g_1 is active.

Example 5. MINLP. Problem taken from [69]:

$$\begin{aligned}
\min \quad & f(\mathbf{x}) = -0.7x_3 + 5(x_1 - 0.5)^2 + 0.8 \\
\text{s.t.} \quad & g_1(\mathbf{x}) = -\exp(x_1 - 0.2) - x_2 \leq 0 \\
& g_2(\mathbf{x}) = x_2 + 1.1x_3 + 1 \leq 0 \\
& g_3(\mathbf{x}) = x_1 - 1.2x_3 - 0.2 \leq 0 \\
& 0.2 \leq x_1 \leq 1 \\
& -2.22554 \leq x_2 \leq -1 \\
& x_3 = \{0, 1\}
\end{aligned}$$

The global minimum is at $[0.94194, -2.1, 1]$ where $f(\mathbf{x}^*) = 1.07654$. Constraints g_1 and g_2 are active.

Example 6. MINLP. Proposed in [67], and also reported in [70, 64, 28]:

$$\begin{aligned}
\min \quad & f(\mathbf{x}) = 2x_1 + 3x_2 + 1.5x_3 + 2x_4 - 0.5x_5 \\
\text{s.t.} \quad & g_1(\mathbf{x}) = x_1 + x_3 - 1.6 \leq 0 \\
& g_2(\mathbf{x}) = 1.333x_2 + x_4 - 3 \leq 0 \\
& g_3(\mathbf{x}) = -x_3 - x_4 + x_5 \leq 0 \\
& h_1(\mathbf{x}) = x_1^2 + x_3 - 1.25 = 0 \\
& h_2(\mathbf{x}) = x_2^{1.5} + 1.5x_4 - 3 = 0 \\
& 0 \leq x_1 \leq 1.5 \\
& 0 \leq x_2 \leq 2.2 \\
& x_i = \{0, 1\}, \quad i = \{3, 4, 5\}
\end{aligned}$$

There are 2^3 different combinations of the binary variables, of these only one combination is infeasible because it violates the pure integer constraint. The global solution is $\mathbf{x}^* = [1.118, 1.3310, 0, 1, 1]$ with $f(\mathbf{x}^*) = 7.667$. Constraint g_3 is active.

Example 7. Reactor network design. This problem, taken from [71] and also studied in [28], is a two-reactor problem, where selection is to be made among two candidate reactors the one that minimizes the cost of producing a desired product. The MINLP formulation is given as:

$$\min \quad f(\mathbf{x}) = 7.5x_5 + 5.5x_6 + 7x_1 + 6x_2 + 5(x_3 + x_4)$$

$$\begin{aligned}
\text{s.t.}\quad & g_1(\mathbf{x}) = x_1 - 10x_5 \le 0\\
& g_2(\mathbf{x}) = x_2 - 10x_6 \le 0\\
& g_3(\mathbf{x}) = x_3 - 20x_5 \le 0\\
& g_4(\mathbf{x}) = x_4 - 20x_6 \le 0\\
& h_1(\mathbf{x}) = x_5 + x_6 - 1 = 0\\
& h_2(\mathbf{x}) = x_7 - 0.9x_3(1 - \exp(-0.5x_1)) = 0\\
& h_3(\mathbf{x}) = x_8 - 0.8x_4(1 - \exp(-0.4x_2)) = 0\\
& h_4(\mathbf{x}) = x_7 + x_8 - 10 = 0\\
& x_i \ge 0, \quad i = \{1,2,3,4,7,8\}\\
& x_i = \{0,1\}, \quad i = \{5,6\}
\end{aligned}$$

The global minimum is $\mathbf{x}^* = [3.514,\ 0,\ 13.428,\ 0,\ 1,\ 0,\ 10,\ 0.0001]$ with $f = 99.238$. Constraints g_2 and g_4 are active.

Example 8. Process synthesis MINLP. This example is taken from [72], and is also found in [70, 64, 28, 32]:

$$\begin{aligned}
\min\quad & f(\mathbf{x}) = (x_4-1)^2 + (x_5-2)^2 + (x_6-1)^2\\
& -\ln(x_7+1) + (x_1-1)^2 + (x_2-2)^2 + (x_3-3)^2\\
\text{s.t.}\quad & g_1(\mathbf{x}) = \sum_{i=1}^{6} x_i - 5 \le 0\\
& g_2(\mathbf{x}) = \sum_{i=1}^{4} x_i^2 - 5.5 \le 0\\
& g_3(\mathbf{x}) = x_4 + x_1 - 1.2 \le 0\\
& g_4(\mathbf{x}) = x_5 + x_2 - 1.8 \le 0\\
& g_5(\mathbf{x}) = x_6 + x_3 - 2.5 \le 0\\
& g_6(\mathbf{x}) = x_7 + x_1 - 1.2 \le 0\\
& g_7(\mathbf{x}) = x_5^2 + x_2^2 - 1.64 \le 0\\
& g_8(\mathbf{x}) = x_6^2 + x_3^2 - 4.25 \le 0\\
& g_9(\mathbf{x}) = x_5^2 + x_3^2 - 4.64 \le 0\\
& x_i \ge 0, \quad i = \{1,2,3\}\\
& x_i = \{0,1\}, \quad i = \{4,5,6,7\}
\end{aligned}$$

The global minimum is $\{\mathbf{x}^*; f(\mathbf{x}^*)\} = \{0.2, 0.8, 1.9079, 1, 1, 0, 1; 4.579582\}$. Constraints g_3, g_4, g_6, g_7 and g_9 are active.

Example 9. Planning problem. First introduced in [67], this example represents a small planning problem, in which several alternatives are proposed for obtaining product C. The goal is to produce the profitable product C from B that is purchased from a market or produced from raw material A. There are also two paths to produce B from A. The problem is modelled as a MINLP:

$$
\begin{aligned}
\min \quad & f(\mathbf{x}) = 3.5x_1 + x_2 + 1.5x_3 + 7x_5 + x_6 \\
& \qquad\qquad + 1.2x_7 + 1.8x_8 - 11x_{11} \\
\text{s.t.} \quad & g_1(\mathbf{x}) = x_4 - 5x_1 \le 0 \\
& g_2(\mathbf{x}) = x_9 - 5x_2 \le 0 \\
& g_3(\mathbf{x}) = x_{10} - 5x_3 \le 0 \\
& g_4(\mathbf{x}) = x_{11} - 1 \le 0 \\
& h_1(\mathbf{x}) = x_6 - \ln(1 + x_9) = 0 \\
& h_2(\mathbf{x}) = x_7 - 1.2\ln(1 + x_{10}) = 0 \\
& h_3(\mathbf{x}) = x_{11} - 0.9x_4 = 0 \\
& h_4(\mathbf{x}) = -x_4 + \sum_{i=5}^{7} x_i = 0 \\
& h_5(\mathbf{x}) = -x_8 + x_9 + x_{10} = 0 \\
& x_i = \{0,1\}, \quad i = \{1,2,3\} \\
& x_i \ge 0, \quad \forall i \\
& x_6 \le 5 \\
& x_{11} \le 1
\end{aligned}
$$

The model contains three binary variables and five continuous variables. The global minimum is $\mathbf{x}^* = [1,\ 0,\ 1,\ 1.11111081,\ 0,\ 0,\ 1.11111081,\ 1.5242038,\ 0,\ 1.5242038,\ 0.99999978]$ and $f(\mathbf{x}^*) = -1.9231$. Constraints g_2 and g_4 are active. There is a local optimum at $\mathbf{x} = [1,\ 1,\ 1,\ 1.111,\ 0,\ 0.446744,\ 0.664156,\ 1.30208,\ 0.563058,\ 0.739121,\ 1]$ with $f(\mathbf{x}) = -1.41252645$.

Examples 10–14. Multi-product batch plant design. The multi-product batch plant consists of M processing stages in series where fixed amounts Q_i of N products have to be manufactured. The objective is to determine for each stage j the number of parallel units N_j and their sizes V_j and for each product i the corresponding batch sizes

B_i and cycle times $T_{\mathrm{L}i}$. The problem data are the horizon time H, the size factors S_{ij} and processing times t_{ij} of product i in stage j, the required productions Q_{ij}, and appropriate cost functions α_j and β_j. The mathematical formulation of this problem is as follows [67]:

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{M} \alpha_j N_j V_j^{\beta_j} \\
\text{s.t.} \quad & \sum_{i=1}^{N} \frac{Q_i T_{\mathrm{L}i}}{B_i} - H \leq 0 \\
& S_{ij} B_i - V_j \leq 0 \\
& t_{ij} - N_j T_{\mathrm{L}i} \leq 0 \\
& 1 \leq N_j \leq N_j^{\mathrm{u}} \\
& V_j^{\mathrm{l}} \leq V_j \leq V_j^{\mathrm{u}} \\
& T_{\mathrm{L}i}^{\mathrm{l}} \leq T_{Li} \leq T_{\mathrm{L}i}^{\mathrm{u}} \\
& B_j^{\mathrm{l}} \leq B_j \leq B_j^{\mathrm{u}} \\
& N_j \text{ integer}
\end{aligned}
$$

The bounds N_j^{u}, V_j^{l}, V_j^{u} are specified by the problem and appropriate bounds for $T_{\mathrm{L}i}$ and B_i can be determined as follows:

$$
\begin{aligned}
T_{\mathrm{L}i}^{\mathrm{l}} &= \max_j \frac{t_{ij}}{N_j^{\mathrm{u}}} \\
T_{\mathrm{L}i}^{\mathrm{u}} &= \max_j t_{ij} \\
B_i^{\mathrm{l}} &= \frac{Q_i}{H} T_{\mathrm{L}i}^{\mathrm{l}} \\
B_i^{\mathrm{u}} &= \min\left(Q_i, \min_j \frac{V_j^{\mathrm{u}}}{S_{ij}} \right)
\end{aligned}
$$

The number of inequality constraints for each problem depends on the number of products N and the number of processing stages M, according to $2MN + 1$. The data corresponding to these problems are presented in Table A.9. For all examples the parameters α_j, β_j and H are 250, 0.6 and 6000, respectively. For problems 10 and 11, the parameters V_j^{l} and V_j^{u} take values of 250 and 2500, respectively. For problems 12 to 14, the parameters V_j^{l} and V_j^{u} are 300 and 3000, respectively.

Table A.9: Input data for Examples 10–14.

Example	M	N	N_j^{u}	S_{ij}	t_{ij}	Q_i
10	3	2	3	$\begin{bmatrix} 2 & 3 & 4 \\ 4 & 6 & 3 \end{bmatrix}$	$\begin{bmatrix} 8 & 20 & 8 \\ 16 & 4 & 4 \end{bmatrix}$	$\begin{bmatrix} 40000 \\ 20000 \end{bmatrix}$
11	3	2	3	$\begin{bmatrix} 2 & 3 & 4 \\ 4 & 6 & 3 \end{bmatrix}$	$\begin{bmatrix} 8 & 20 & 8 \\ 16 & 4 & 4 \end{bmatrix}$	$\begin{bmatrix} 200000 \\ 100000 \end{bmatrix}$
12	6	5	4	$\begin{bmatrix} 7.9 & 2.0 & 5.2 & 4.9 & 6.1 & 4.2 \\ 0.7 & 0.8 & 0.9 & 3.4 & 2.1 & 2.5 \\ 0.7 & 2.6 & 1.6 & 3.6 & 3.2 & 2.9 \\ 4.7 & 2.3 & 1.6 & 2.7 & 1.2 & 2.5 \\ 1.2 & 3.6 & 2.4 & 4.5 & 1.6 & 2.1 \end{bmatrix}$	$\begin{bmatrix} 6.4 & 4.7 & 8.3 & 3.9 & 2.1 & 1.2 \\ 6.8 & 6.4 & 6.5 & 4.4 & 2.3 & 3.2 \\ 1.0 & 6.3 & 5.4 & 11.9 & 5.7 & 6.2 \\ 3.2 & 3.0 & 3.5 & 3.3 & 2.8 & 3.4 \\ 2.1 & 2.5 & 4.2 & 3.6 & 3.7 & 2.2 \end{bmatrix}$	$\begin{bmatrix} 250000 \\ 150000 \\ 180000 \\ 160000 \\ 120000 \end{bmatrix}$
13	7	6	4	$\begin{bmatrix} 7.9 & 2.0 & 5.2 & 4.9 & 6.1 & 4.2 & 3.6 \\ 0.7 & 0.8 & 0.9 & 3.4 & 2.1 & 2.5 & 0.6 \\ 0.7 & 2.6 & 1.6 & 3.6 & 3.2 & 2.9 & 3.8 \\ 4.7 & 2.3 & 1.6 & 2.7 & 1.2 & 2.5 & 3.5 \\ 1.2 & 3.6 & 2.4 & 4.5 & 1.6 & 2.1 & 3.6 \\ 5.2 & 3.0 & 1.8 & 4.2 & 4.0 & 2.4 & 1.6 \end{bmatrix}$	$\begin{bmatrix} 6.4 & 4.7 & 8.3 & 3.9 & 2.1 & 1.2 & 6.4 \\ 6.8 & 6.4 & 6.5 & 4.4 & 2.3 & 3.2 & 2.6 \\ 1.0 & 6.3 & 5.4 & 11.9 & 5.7 & 6.2 & 6.2 \\ 3.2 & 3.0 & 3.5 & 3.3 & 2.8 & 3.4 & 6.1 \\ 2.1 & 2.5 & 4.2 & 3.6 & 3.7 & 2.2 & 1.8 \\ 2.6 & 4.2 & 3.8 & 4.1 & 5.8 & 3.8 & 6.9 \end{bmatrix}$	$\begin{bmatrix} 250000 \\ 150000 \\ 180000 \\ 160000 \\ 120000 \\ 200000 \end{bmatrix}$
14	8	6	4	$\begin{bmatrix} 7.9 & 2.0 & 5.2 & 4.9 & 6.1 & 4.2 & 3.6 & 2.4 \\ 0.7 & 0.8 & 0.9 & 3.4 & 2.1 & 2.5 & 0.6 & 2.0 \\ 0.7 & 2.6 & 1.6 & 3.6 & 3.2 & 2.9 & 3.8 & 1.4 \\ 4.7 & 2.3 & 1.6 & 2.7 & 1.2 & 2.5 & 3.5 & 2.3 \\ 1.2 & 3.6 & 2.4 & 4.5 & 1.6 & 2.1 & 3.6 & 2.7 \\ 5.2 & 3.0 & 1.8 & 4.2 & 4.0 & 2.4 & 1.6 & 6.2 \end{bmatrix}$	$\begin{bmatrix} 6.4 & 4.7 & 8.3 & 3.9 & 2.1 & 1.2 & 6.4 & 5.2 \\ 6.8 & 6.4 & 6.5 & 4.4 & 2.3 & 3.2 & 2.6 & 8.0 \\ 1.0 & 6.3 & 5.4 & 11.9 & 5.7 & 6.2 & 6.2 & 7.1 \\ 3.2 & 3.0 & 3.5 & 3.3 & 2.8 & 3.4 & 6.1 & 8.2 \\ 2.1 & 2.5 & 4.2 & 3.6 & 3.7 & 2.2 & 1.8 & 1.4 \\ 2.6 & 4.2 & 3.8 & 4.1 & 5.8 & 3.8 & 6.9 & 4.6 \end{bmatrix}$	$\begin{bmatrix} 250000 \\ 150000 \\ 180000 \\ 160000 \\ 120000 \\ 200000 \end{bmatrix}$

References

[1] V. T. Voudouris, I. E. Grossmann, Mixed-integer linear programming reformulations for batch process design with discrete equipment sizes, Industrial & Engineering Chemistry Research 31 (5) (1992) 1315–1325.

[2] A. Ponsich, C. Azzaro-Pantel, S. Domenech, L. Pibouleau, Mixed-integer nonlinear programming optimization strategies for batch plant design problems, Industrial & engineering chemistry research 46 (3) (2007) 854–863.

[3] A. W. Dowling, L. T. Biegler, A framework for efficient large scale equation-oriented flowsheet optimization, Computers & Chemical Engineering 72 (2015) 3–20.

[4] Z. Zhu, D. Xu, X. Liu, Z. Zhang, Y. Wang, Separation of acetonitrile/methanol/benzene ternary azeotrope via triple column pressure-swing distillation, Separation and purification technology 169 (2016) 66–77.

[5] T. F. Yee, I. E. Grossmann, A screening and optimization approach for the retrofit of heat-exchanger networks, Industrial & Engineering Chemistry Research 30 (1) (1991) 146–162.

[6] H. V. H. Ayala, P. Keller, M. de Fátima Morais, V. C. Mariani, L. dos Santos Coelho, R. V. Rao, Design of heat exchangers using a novel multiobjective free search differential evolution paradigm, Applied Thermal Engineering 94 (2016) 170–177.

[7] N. M. Kaiser, R. J. Flassig, K. Sundmacher, Probabilistic reactor design in the framework of elementary process functions, Computers & Chemical Engineering 94 (2016) 45–59.

[8] S. D.-L. Almaraz, C. Azzaro-Pantel, L. Montastruc, M. Boix, Deployment of a hydrogen supply chain by multi-objective/multi-period optimisation at regional and national scales, Chemical Engineering Research and Design 104 (2015) 11–31.

[9] Y.-b. Woo, S. Cho, J. Kim, B. S. Kim, Optimization-based approach for strategic design and operation of a biomass-to-hydrogen supply chain, International Journal of Hydrogen Energy 41 (12) (2016) 5405–5418.

[10] M. Tawarmalani, N. V. Sahinidis, N. Sahinidis, Convexification and global optimization

in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications, Vol. 65, Springer Science & Business Media, 2002.

[11] C. A. Floudas, C. E. Gounaris, A review of recent advances in global optimization, Journal of Global Optimization 45 (1) (2009) 3–38.

[12] L. Liberti, Introduction to global optimization, Ecole Polytechnique.

[13] O. Bozorg-Haddad, M. Solgi, H. A. Loaiciga, Meta-heuristic and evolutionary algorithms for engineering optimization, Vol. 294, John Wiley & Sons, 2017.

[14] Z. Michalewicz, C. Z. Janikow, GENOCOP: A genetic algorithm for numerical optimization problems with linear constraints, Communications of the ACM 39 (12es) (1996) 175–es.

[15] D. V. Arnold, J. Porter, Towards an augmented Lagrangian constraint handling approach for the (1+1)-ES, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, 2015, pp. 249–256.

[16] A. Atamna, A. Auger, N. Hansen, Augmented Lagrangian constraint handling for CMA-ES—case of a single linear constraint, in: International Conference on Parallel Problem Solving from Nature, Springer, 2016, pp. 181–191.

[17] T. P. Runarsson, X. Yao, Stochastic ranking for constrained evolutionary optimization, IEEE Transactions on Evolutionary Computation 4 (3) (2000) 284–294.

[18] T. P. Runarsson, X. Yao, Search biases in constrained evolutionary optimization, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 35 (2) (2005) 233–243.

[19] T. Takahama, S. Sakai, Constrained optimization by ε constrained particle swarm optimizer with ε-level control, in: A. Abraham, Y. Dote, T. Furuhashi, M. Köppen, A. Ohuchi, Y. Ohsawa (Eds.), Soft Computing as Transdisciplinary Science and Technology, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 1019–1029.

[20] T. Takahama, S. Sakai, Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites, in: 2006 IEEE International Conference on Evolutionary Computation, IEEE, 2006, pp. 1–8.

[21] T. Takahama, S. Sakai, Constrained optimization by the ε constrained differential evolution with an archive and gradient-based mutation, in: Evolutionary Computation (CEC), 2010 IEEE Congress on, IEEE, 2010, pp. 1–9.

[22] H. Zhang, G. P. Rangaiah, An efficient constraint handling method with integrated differential evolution for numerical and engineering optimization, Computers & Chemical Engineering 37 (2012) 74–88.

[23] Z. Fan, H. Li, C. Wei, W. Li, H. Huang, X. Cai, Z. Cai, An improved epsilon constraint handling method embedded in MOEA/D for constrained multi-objective optimization problems, in: 2016 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2016, pp. 1–8.

[24] Z. Fan, W. Li, X. Cai, H. Li, C. Wei, Q. Zhang, K. Deb, E. Goodman, Push and pull search for solving constrained multi-objective optimization problems, Swarm and evolutionary computation 44 (2019) 665–679.

[25] O. Cuate, A. Ponsich, L. Uribe, S. Zapotecas-Martínez, A. Lara, O. Schütze, A new hybrid evolutionary algorithm for the treatment of equality constrained mops, Mathematics 8 (1) (2020) 7.

[26] P. Chootinan, A. Chen, Constraint handling in genetic algorithms using a gradient-based repair method, Computers & operations research 33 (8) (2006) 2263–2281.

[27] S. Kheawhom, Efficient constraint handling scheme for differential evolutionary algorithm in solving chemical engineering optimization problem, Journal of Industrial and Engineering Chemistry 16 (4) (2010) 620–628.

[28] M. Cardoso, R. Salcedo, S. F. de Azevedo, D. Barbosa, A simulated annealing approach to the

solution of MINLP problems, Computers & Chemical Engineering 21 (12) (1997) 1349–1364.

[29] L. Costa, P. Oliveira, Evolutionary algorithms approach to the solution of mixed integer non-linear programming problems, Computers & Chemical Engineering 25 (2-3) (2001) 257–266.

[30] B. Babu, R. Angira, Modified differential evolution (MDE) for optimization of non-linear chemical processes, Computers & Chemical Engineering 30 (6-7) (2006) 989–1002.

[31] M. Srinivas, G. Rangaiah, Differential evolution with tabu list for solving nonlinear and mixed-integer nonlinear programming problems, Industrial & Engineering Chemistry Research 46 (22) (2007) 7126–7135.

[32] L. Yiqing, Y. Xigang, L. Yongjian, An improved PSO algorithm for solving non-convex NLP/MINLP problems with equality constraints, Computers & chemical engineering 31 (3) (2007) 153–162.

[33] X. Chen, W. Du, F. Qian, Solving chemical dynamic optimization problems with ranking-based differential evolution algorithms, Chinese journal of chemical engineering 24 (11) (2016) 1600–1608.

[34] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE transactions on evolutionary computation 1 (1) (1997) 67–82.

[35] D. Corne, J. Knowles, Some multiobjective optimizers are better than others, in: The 2003 Congress on Evolutionary Computation, 2003. CEC'03., Vol. 4, IEEE, 2003, pp. 2506–2512.

[36] K. Deb, An efficient constraint handling method for genetic algorithms, Computer Methods in Applied Mechanics and Engineering 186 (2-4) (2000) 311–338.

[37] J. Vesterstrom, R. Thomsen, A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems, in: Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753), Vol. 2, IEEE, 2004, pp. 1980–1987.

[38] A. Ponsich, C. C. Coello, Differential evolution performances for the solution of mixed-integer constrained process engineering problems, Applied Soft Computing 11 (1) (2011) 399–409.

[39] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (4) (1997) 341–359.

[40] A. K. Qin, P. N. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in: Evolutionary Computation, 2005. The 2005 IEEE Congress on, Vol. 2, IEEE, 2005, pp. 1785–1791.

[41] J. Zhang, A. C. Sanderson, JADE: Adaptive differential evolution with optional external archive, IEEE Transactions on Evolutionary Computation 13 (5) (2009) 945–958. doi:10.1109/TEVC.2009.2014613.

[42] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems, IEEE Transactions on Evolutionary Computation 10 (6) (2006) 646–657.

[43] R. Tanabe, A. Fukunaga, Success-history based parameter adaptation for differential evolution, in: Evolutionary Computation (CEC), 2013 IEEE Congress on, IEEE, 2013, pp. 71–78.

[44] S. Wang, Y. Li, H. Yang, Self-adaptive mutation differential evolution algorithm based on particle swarm optimization, Applied Soft Computing 81 (2019) 105496. doi:https://doi.org/10.1016/j.asoc.2019.105496.
URL `http://www.sciencedirect.com/science/article/pii/S1568494619302662`

[45] E. Mezura-Montes, C. A. C. Coello, Constraint-handling in nature-inspired numerical optimization: past, present and future, Swarm and Evolutionary Computation 1 (4) (2011) 173–194.

[46] C. A. Coello Coello, Constraint-handling techniques used with evolutionary algorithms, in: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, ACM, 2016, pp. 563–587.

[47] E. Mezura-Montes, C. A. C. Coello, A simple multimembered evolution strategy to solve constrained optimization problems, IEEE Transactions on Evolutionary computation 9 (1) (2005) 1–17.

[48] R. Mallipeddi, P. N. Suganthan, Ensemble of constraint handling techniques, IEEE Transactions on Evolutionary Computation 14 (4) (2010) 561–579.

[49] N. Padhye, P. Mittal, K. Deb, Feasibility preserving constraint-handling strategies for real parameter evolutionary optimization, Computational Optimization and Applications 62 (3) (2015) 851–890.

[50] F. Samanipour, J. Jelovica, Adaptive repair method for constraint handling in multi-objective genetic algorithm based on relationship between constraints and variables, Applied Soft Computing 90 (2020) 106143. doi:https://doi.org/10.1016/j.asoc.2020.106143.
URL `http://www.sciencedirect.com/science/article/pii/S1568494620300831`

[51] Y. Yang, J. Liu, S. Tan, A constrained multi-objective evolutionary algorithm based on decomposition and dynamic constraint-handling mechanism, Applied Soft Computing 89 (2020) 106104. doi:https://doi.org/10.1016/j.asoc.2020.106104.

[52] C. A. C. Coello, Use of a self-adaptive penalty approach for engineering optimization problems, Computers in Industry 41 (2) (2000) 113–127.

[53] P. Nanakorn, K. Meesomklin, An adaptive penalty function in genetic algorithms for structural design optimization, Computers & Structures 79 (29-30) (2001) 2527–2539.

[54] B. Tessema, G. G. Yen, A self adaptive penalty function based algorithm for constrained optimization, in: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, IEEE, 2006, pp. 246–253.

[55] H. J. Barbosa, A. C. Lemonge, H. S. Bernardino, A critical review of adaptive penalty techniques in evolutionary computation, in: Evolutionary constrained optimization, Springer, 2015, pp. 1–27.

[56] M. Zhang, W. Luo, X. Wang, Differential evolution with dynamic stochastic selection for constrained optimization, Information Sciences 178 (15) (2008) 3043–3074.

[57] Z. Fan, J. Liu, T. Sorensen, P. Wang, Improved differential evolution based on stochastic ranking for robust layout synthesis of MEMS components, IEEE Transactions on Industrial Electronics 56 (4) (2009) 937–948.

[58] L. Ali, S. L. Sabat, S. K. Udgata, Particle swarm optimisation with stochastic ranking for constrained numerical and engineering benchmark problems, International Journal of Bio-Inspired Computation 4 (3) (2012) 155–166.

[59] Z. Yang, X. Cai, Z. Fan, Epsilon constrained method for constrained multiobjective optimization problems: some preliminary results, in: Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, ACM, 2014, pp. 1181–1186.

[60] Z. Fan, W. Li, X. Cai, H. Huang, Y. Fang, Y. You, J. Mo, C. Wei, E. Goodman, An improved epsilon constraint-handling method in MOEA/D for CMOPs with large infeasible regions, Soft Computing (2017) 1–20.

[61] S. L. Campbell, C. D. Meyer, Generalized inverses of linear transformations, SIAM, 2009.

[62] G. P. Rangaiah, Multi-objective optimization: techniques and applications in chemical engineering, Vol. 1, World Scientific, 2009.

[63] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197. doi:10.1109/4235.996017.

[64] H. S. Ryoo, N. V. Sahinidis, Global optimization of nonconvex NLPs and MINLPs with applications in process design, Computers & Chemical Engineering 19 (5) (1995) 551–566.

[65] L. T. Biegler, I. E. Grossmann, A. W. Westerberg, Systematic methods for chemical process design, Prentice Hall, Old Tappan, NJ (United States), 1997.

[66] Z. N. Pintaric, Z. Kravanja, Selection of the economic objective function for the optimization of

process flow sheets, Industrial & engineering chemistry research 45 (12) (2006) 4222–4232.

[67] G. R. Kocis, I. E. Grossmann, Global optimization of nonconvex mixed-integer nonlinear programming (MINLP) problems in process synthesis, Industrial & Engineering Chemistry Research 27 (8) (1988) 1407–1421. doi:10.1021/ie00080a013.

[68] G. R. Kocis, I. E. Grossmann, Relaxation strategy for the structural optimization of process flow sheets, Industrial & Engineering Chemistry Research 26 (9) (1987) 1869–1880.

[69] C. A. Floudas, Nonlinear and mixed-integer optimization: fundamentals and applications, Oxford University Press, 1995.

[70] C. Floudas, A. Aggarwal, A. Ciric, Global optimum search for nonconvex NLP and MINLP problems, Computers & Chemical Engineering 13 (10) (1989) 1117–1132.

[71] G. R. Kocis, I. E. Grossmann, A modelling and decomposition strategy for the MINLP optimization of process flowsheets, Computers & Chemical Engineering 13 (7) (1989) 797–819.

[72] X. Yuan, S. Zhang, L. Pibouleau, S. Domenech, Une methode d'optimization nonlineaire en variables mixtes pour la conception de procedes, RAIRO - Operations Research.