CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Open Archive Toulouse Archive Ouverte

MINLP optimisation strategies for batch plant design problems

Ponsich Antonin*, Azzaro-Pantel Catherine, Domenech Serge, Pibouleau Luc

Laboratoire de Génie Chimique UMR 5503 CNRS/INP/UPS

5 rue Paulin Talabot BP1301

31106 TOULOUSE Cedex 1

Authors to whom correspondence should be addressed :

{Catherine.AzzaroPantel@ensiacet.fr; Antonin.Ponsich@ensiacet.fr}

Abstract

Due to their large variety of applications, complex optimisation problems induced a great effort to develop efficient solution techniques, dealing with both continuous and discrete variables involved in non-linear functions. But among the diversity of those optimisation methods, the choice of the relevant technique for the treatment of a given problem keeps being a thorny issue.

Within the Process Engineering context, batch plant design problems provide a good framework to test the performances of various optimisation methods : on the one hand, two Mathematical Programming techniques – *DICOPT++* and *SBB*, implemented in the *GAMS* environment – and, on the other hand, one stochastic method, i.e. a genetic algorithm. Seven examples, showing an increasing complexity, were solved with these three techniques. The result comparison enables to evaluate their efficiency in order to highlight the most appropriate method for a given problem instance. It was proved that the best performing method is *SBB*, even if the GA also provides interesting solutions, in terms of quality as well as of computational time.

Keywords : optimisation methods, genetic algorithm, Mathematical Programming, batch plant design.

1. Introduction

A great variety of applications, drawn from a wide range of investigation areas, can be formulated as complex optimisation problems. It covers, for instance, the famous travelling man problem studied by Padberg and Rinaldi[1] as well as frequencies allocation for radio-mobile networks (Hao and Dorne[2]), process networks optimisation (Lee and Grossmann[3]), physicochemical equilibrium calculations (Teh and Rangaiah[4]), or hydrology computing (Jain and Srinivasalu[5]).

This large number of optimisation problems arises from models that have to enable, for industrial requirements, a truly realistic representation of the system they account for. Consequently, these models tend to show an increasing sophistication degree that derives into higher complexity and, thus, solution difficulties. The complexity of the formulated models is basically due to the nature of the functions and of the variables involved in the optimisation problem. The former ones may be not only non-linear, but moreover, they often prove to be non-convex, which is a strongly penalizing characteristic in the typical minimization case. Then, for constrained problem, determining the feasible space turns to be a really difficult task. With regard to variable nature, most of engineering problems consider both continuous and discrete variables, introducing discontinuities in the objective function and in the search space : those are called mixed-integer problems. Furthermore, the discrete variables induce an important combinatorial effect : this point is emphasized with NP-hard problems, for which no algorithm leading to polynomial solution times is known. Since industrial size problems have up to several thousands variables and constraints, the resulting computational times may easily become prohibitive.

In order to face these problems, a significant investigation effort has been carried out to develop efficient and robust optimisation methods. At the beginning, this aim was purchased especially in the Operational Research and Artificial Intelligence areas. But the trend was

subsequently followed by the Process System Engineering community, since this one provides a wide number of applications formulated as complex optimisation problems. A typical reference is constituted by design problems : heat or mass exchanger networks (Zamora and Grossmann[6]), supply chain design (Guillén et al.[7]), multiproduct (Ravemark and Rippin[8]) or multipurpose (Dedieu et al.[9]) batch plant design or retrofitting (Montagna and Vecchietti[10]).

As a consequence, a great diversity of optimisation methods was implemented to meet the industrial stakes and provide competitive results. But if they prove to be well fitted to the particular case they purchase, these techniques performance cannot be constant whatever the treated problem is. Actually, a method efficiency for a particular example is hardly predictable, and the only certainty we have is expressed by the *No Free Lunch* theory (Wolpert and Macready[11]) : there is no method that outdoes all the other ones for any considered problem. This feature generates a common lack of explanation concerning the use of a method for the solution of a particular example, and usually, no relevant justification for its choice is given *a priori*.

This lack of justification for the use of an optimisation method is the issue of the present study. The objective is then to propose some guidelines that may be useful for the choice of an appropriate optimisation technique. Obviously, the quoted *No Free Lunch* theory prevents from drawing any general conclusions, which could be extended to any class of problems. So, the framework of this paper is restricted to one particular Process Engineering problem, i.e. one typical batch plant design problem formulation. This Mixed-Integer Non Linear Programming (MINLP) problem provides indeed a good application aid to evaluate several methods efficiency.

The study is divided into five sections. The general aims and the adopted methodology are developed in section 2. Section 3 describes the investigated optimisation methods. Some

typical results are analysed in section 4. Finally, conclusions and perspectives are presented in section 5.

2. Problem position

The following section recalls some essential features firstly about the existing MINLP methods, and then about classical formulations for batch plant design. Then, the methodology considered in the study will be stated in detail.

2.1 A great diversity of optimisation methods

Among the diversity of optimisation techniques, two important classes have to be distinguished : deterministic methods and stochastic ones. Complete reviews are proposed by Grossmann[12] or Biegler and Grossmann[13] for the former class, and by Hao et al.[14] for the latter one.

The deterministic methods involve the verification of mathematical properties of the objective function and constraints, such as continuity or derivability. This working mode enables them to ensure to get an optimum, which is a great advantage. Among the deterministic class, the following ones stand out : the Branch & Bound methods (Gupta and Ravindran[15], Ryoo and Sahinidis[16] and Smith and Pantelides[17]) ; the Generalized Benders Decomposition (Geoffrion[18]) and the Outer Approximation (Duran and Grossmann[19]) algorithms ; the Extended Cutting Plane method (Westerlünd and Petterson[20]) ; Disjunctive Programming methods (Raman and Grossmann[21]). Even though most of the above-mentioned methods keep being at academic level, some (commercial or free) computational codes are available : the *SBB*, *BARON*, *DICOPT++* and *LOGMIP* solvers (within the *GAMS* modelling environment, see Brooke et al.[22]), *MINLP_BB* (Leyffer[24]) and *αECP* (Westerlünd and Lundqvist[25]).

The second class, namely metaheuristics or stochastic methods, is based on the evaluation of the objective function at different points of the search space. These points are chosen through the use of a set of heuristics, combined with generations of random numbers. Thus, metaheuristics cannot guarantee to obtain an optimum. They are divided into neighbourhood techniques (Simulated Annealing, Kirkpatrick et al.[26] ; Tabu Search, Teh and Rangaiah[4]) and evolutionary algorithms (genetic algorithms, Holland[27] ; evolutionary strategies, Beyer and Schwefel[28] ; evolutionary programming, Yang et al.[29]).

2.2 Batch plant design framework

Due to the growing interest for batch operating mode, a lot of studies deal with the batch plant design issue (Grossmann and Sargent[30], Kocis and Grossmann[31], Modi and Karimi[32], Patel et al.[33], Wang et al.[34,35,36]). Generally, the objective consists in the minimization of plant investment cost.

The model formulation for batch plant design problems adopted in this paper is based on Modi's approach (Modi and Karimi[32]). The formulation accounts for the synthesis of I products treated in J batch stages and K semi-continuous units (pumps, heat exchangers,...). The optimisation variables are the discrete number (m_j for the batch stages and n_k for the semi-continuous ones) and continuous size (V_j and R_k) of the items of each stage. Moreover, S-1 intermediate storage tanks, with size V_s*, divide the whole process into S sub-processes. The complete model will not be presented here, it is possible to report to Ponsich et al.[37] to get the detailed formulation. The main feature is the minimization of the investment cost for all the items of the plant :

$$MinCost=\sum_{j=1}^{J} a_j m_j V_j^{\alpha j}+\sum_{j=1}^{K} b_k n_k R_k^{\beta k}+\sum_{j=1}^{S-1} c_s (V_s^*)^{\gamma s} \qquad (1)$$

where α_j and a_j, β_k and b_k, γ_s and c_s are classical cost coefficients (a complete nomenclature is provided in Appendix A). This problem is submitted to one major constraint, forcing the total production time for all products to be lower than a given time horizon H :

$$H \geq \sum_{i=1}^{I} H_i = \sum_{i=1}^{I} \frac{Q_i}{Prod_i} \tag{2}$$

Then, the aim of batch plant design problems is to find the plant structure that respects the production requirements within the time horizon, while minimizing the economic criterion. The resulting MINLP problem proves to be non-convex and NP-hard (Wang et al.[34]).

2.3 Aims and methodology

As shown by the extent of literature devoted to optimisation methods, the performances have been substantially improved and the range of problems that can be solved is wider. However, the drawback is the lack of explanation concerning the use of a particular method. It is known that a method that is well-fitted to an example will give better results than a generic solution scheme. But most of times, the choice seems quite confused and is only justified by the good results obtained for particular cases. So, there still exists a deficiency of studies that would justify their use *a priori*.

Thus, the aim of this work is to evaluate the behaviour of optimisation techniques in order to provide guidelines helping to choose the most appropriate one. Obviously, it will not be possible to tackle all the existing MINLP methods, but in order to keep the conclusions as general as possible, techniques deriving from both deterministic and stochastic classes will be tested. Besides, another restriction is constituted by the considered problem : as it was stated by the above-mentioned *No Free Lunch* theory, the conclusions will neither be extended to another problem class, nor to another formulation.

So, these two remarks lead to develop the methodology basis for this work. To represent the deterministic class, solvers of the *GAMS* environment were chosen, since this optimisation

tool is widely used, and even stands as a reference for the solution of problems drawn from Process Engineering : the *SBB* and *DICOPT++* modules, that carry out, respectively, a Branch & Bound procedure and the Outer Approximation method, were adopted. To illustrate the stochastic methods, genetic algorithms will be used since they have already shown their efficiency for batch plant design problems treatment, especially in cases of multiobjective optimisation. Furthermore, by managing a population of individuals, they are able to provide a set of good feasible solutions at the end of the search : a decision-maker could then choose among them which is the most profitable option. Concerning the benchmark model for the study, the equation-oriented formulation proposed by Modi and Karimi[32] was adopted since it allows the use of Mathematical Programming methods.

2.4 Example set

Finally, the issue of the problem size has to be investigated too. Indeed, the combinatorial effect will make this point a crucial criterion within the objective of method evaluation. Thus, a set of increasing complexity examples, all formulated in the same way, will be used here as a bench for the three techniques. The comparison of the results obtained by every method enables to evaluate their performances and to judge the relevance of their use according to the studied problem instance.

This set was built on the basis of two existing problems drawn from the batch plant design literature. The first one, consisting of only three batch stages, was chosen by Kocis and Grossmann[31]. The second example was proposed by Modi and Karimi[32] : the plant, divided into two sub-processes, consists of four batch stages and six semi-continuous stages. This problem was already solved with various stochastic techniques : simulated annealing (Patel et al.[33]), genetic algorithm (Wang et al.[34]), tabu search (Wang et al.[35]) and finally ants foraging

method (Wang et al.[36]). The best solution found is equal to 362130, but slightly violates the time production constraint (Wang et al.[34]).

Due to the data similarities between these two problems, one intermediate size and four higher size examples were built to have the set complete. Table 1 sums up the studied examples and their respective complexity (computed according to the expression $3^{StageNumber}$). All data corresponding to the tackled examples are available in Appendix B.

Table 1. Examples set and associated complexity

Example	Stage number	Intermediate Storage	Product number	Combinatory
1 (Kocis and Grossmann[31])	3	0	2	$2.70\ 10^{1}$
2	7	0	2	$2.19\ 10^{3}$
3 (Modi and Karimi[32])	10	1	3	$5.90\ 10^{4}$
4	21	2	3	$1.05\ 10^{10}$
5	42	5	3	$1.09\ 10^{20}$
6	84	11	3	$1.20\ 10^{40}$
7	105	14	3	$1.25\ 10^{50}$

3. Investigated optimisation methods

3.1 Mathematical Programming techniques

This section will not present an exhaustive description of the methods implemented in the *GAMS* solvers, since they have been widely detailed in the devoted literature. An outline of the basic features and principles of the used algorithms will only be given.

3.1.1 DICOPT++

The *DICOPT++* solver relies on the Outer Approximation algorithm, initially developed by Duran and Grossmann[17]. This one works by decoupling the problem into one continuous NLP sub-problem and one MILP master problem. For the solution of the continuous sub-

problem, the discrete variables are set to fixed values, corresponding to the result of the previous master problem. If feasible, the solution of the NLP (obtained by means of solver *CONOPT3*) provides an upper bound Z_U of the initial problem. If the termination criterion is not met, then the integer master problem is built by linearizing the constraints at the NLP solution and adding them to all the linearizations computed in the previous iterations.

The resulting MILP problem is solved with the *CPLEX* solver (having the continuous variables set to the last NLP solution values) in order to get a new lower bound Z_L of the global problem. The accumulation of constraint linearizations enables to describe more and more precisely the feasible region. Consequently, the upper and lower bounds respectively decreases and increases monotonically, at each major iteration of the algorithm. The termination criterion is reached when a deterioration in the objective function is observed.

The drawback of the algorithm is that non-convexities could disrupt the process, either by cutting regions of the feasible space with an invalid constraint linearization or by generating invalid lower bounds. Thus, modifications were proposed through local and global convexity tests (Kocis and Grossmann[31]) and augmented penalty (Viswanathan and Grossmann[38]). In spite of these additional features, computational experience showed that avoiding to cutting off the global optimum cannot be ensured.

3.1.2 SBB

The *SBB* solver looks like a classical Branch and Bound algorithm, and the whole sequence will not be recalled here (see for instance Floudas[39]). This constructive technique is based on the principle of separation of the initial problem by successively branching on the discrete variables and tightening their respective bounds. The resulting sub-problems are represented by nodes and relaxed to get continuous problems. Solving those relaxations provides either a lower bound to the global problem or an upper bound when the solution is

mixed integer and continuous, i.e. feasible. The bounds intersection is the termination criterion.

The slight difference between a classical Branch & Bound method and *SBB* lies in the MINLP nature of the initial problem, which leads to non-linear relaxation for each node. These ones are solved with the NLP solver *CONOPT3*. It is to notice that, unlike *DICOPT++, SBB* does not need any MILP solver since the discrete variables are handled by the Branch & Bound procedure. Some *SBB* options are available to set the choice of the next variable to be branched on and of the next node to be treated.

3.2 Stochastic method : Genetic Algorithms

As previously, only the the genetic algorithms (GAs) basic principles (Holland[27]) are recalled and the following section just focuses on the specific parameters used in this study.

3.2.1 General principles

The principles of GAs lie on the analogy between a population of individuals and a set of solutions of some optimisation problem. The algorithm makes the solution set evolve towards a good quality, or adaptation, and mimics the rules of natural selection stated by Darwin : the weakest individuals will disappear while the best ones will survive and be able to reproduce themselves. By way of genetic inheritance, the characteristics that make these individuals "stronger" will be preserved generation after generation.

The mechanisms implemented in the GAs reproduce this natural behaviour. Good solutions are settled by creating selection rules, that will state whether the individuals are adapted or not to the considered problem. Crossover and mutation operators then lead the population to evolve in order to obtain, at the end of the run, a set of good quality solutions. The heuristics are mixed with a strong stochastic feature, leading to a compromise between exploration and intensification in the search space, which contributes to GAs efficiency.

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

3.2.2 Coding and associated genetic operators

The way the variables are encoded is clearly essential for GAs efficiency. The array representing the complete set of all variables is called a chromosome. It is composed of genes, each one encoding a variable by means of one or several locus/loci. A difference will be made between genes encoding discrete variables from those encoding continuous ones. The former ones are coded directly in a single-locus, integer-valued gene, containing the variable value. The associated loci are thus able to take the values 1, 2 and 3 (which are the only acceptable values for the item number of the stages).

Concerning the real variables encoding, binary or similar encoding techniques, based on variable discretisation, received much attention from the evolutionary algorithms community. But real-coded GAs have proved their efficiency and even superiority on binary coding in many cases (see for instance Deb[40]). Thus, real-value genes were adopted to represent the continuous variables. Since these ones are bounded, they can be written in a reduced form, as a real number α defined within 0 and 1 that is coded directly on a real-value locus. A mixed real-discrete chromosome is obtained, that will require specific genetic operators.

Firstly, unlike binary coding for which crossover procedures are carried out for the whole chromosome structure, the crossover methods for real-coded GAs are applied gene after gene. The most common methods usually rely on arithmetical or geometrical combinations of parent genes, such as it is presented in equations (3) and (4).

$$\begin{cases} y_k^{(1)} = \alpha.x_k^{(1)} + (1-\alpha).x_k^{(2)} \\ y_k^{(2)} = (1-\alpha).x_k^{(1)} + \alpha.x_k^{(2)} \end{cases} \quad (3)$$

$$\begin{cases} y_k^{(1)} = [x_k^{(1)}]^{\alpha}.[x_k^{(2)}]^{(1-\alpha)} \\ y_k^{(2)} = [x_k^{(1)}]^{(1-\alpha)}.[x_k^{(2)}]^{\alpha} \end{cases} \quad (4)$$

Where $x_k^{(1)}$ and $x_k^{(2)}$ represent genes k of both parents and $y_k^{(1)}$ and $y_k^{(2)}$ those of the resulting children. α is a fixed parameter within 0 and 1. Then, different methods are implemented according to the way of computing α (Michalewicz and Schoenauer[41]). Another kind of crossover techniques works by arithmetical combinations of parent offspring too, but also uses informations from the parent quality (see Raghuwanshi and Kakde[42]). This is the case of the simplex method (SPX) that builds a new individual by analogy with the simplex technique ; or of the direction based crossover (DBX) : if parent $x^{(1)}$ is better than parent $x^{(2)}$, i.e. $f(x^{(1)}) < f(x^{(2)})$ considering minimization, then one child offspring is created according to the following expression :

$$y = r.(x^{(1)} - x^{(2)}) + x^{(1)} \qquad (0 < r < 1) \quad (5)$$

The technique chosen for this study is a simulated binary crossover (SBX), proposed by Deb and Agrawal[43]. The method consists in generating a probability distribution around parent solutions to create two offspring. This probability distribution is chosen in order to mimic the single-point crossover behaviour in binary coded GAs, and mainly involves the two following features :

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

These assumptions generate higher probabilities to create offsprings close to the parents than away from them. The resulting probability distribution trend is illustrated in figure 1, and can be interpolated through a polynomial distribution. The addition of a random feature then enables to compute the weight factors for both parents $x^{(j)}$ in equation (3), to get the two children $y^{(j)}$.

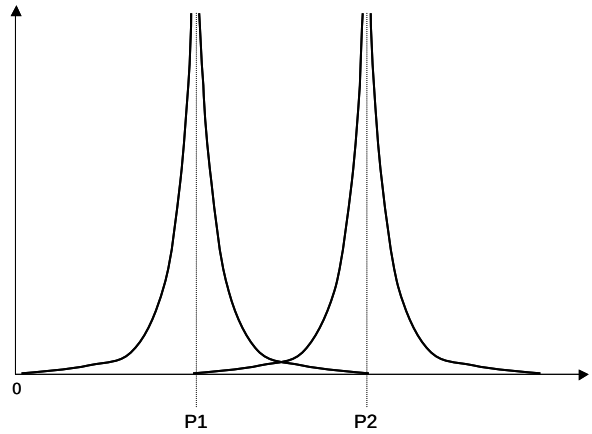

Figure 1. Probability distribution for the location of an offspring, SBX crossover

The procedure for the generation of two children from two parent offspring is fully explained in Appendix C. It is to note that this crossover procedure is carried out for each locus of the chromosome (with some probability measure) and, as a consequence, the gene position along the string does not matter. However, even though the SBX crossover does not induce any problem for real variables, it may lead to real values for the discrete genes of the resulting offspring. So, in this latter case, these real values were truncated in order to keep only their integer part.

With respect to mutation, specific operators had to be implemented according to variable nature. For the discrete ones, an intuitive subtraction of one unit to the bit value was adopted, when possible. This technique is not a symmetric operator, thus it cannot prevent the algorithm from being trapped in some local optimum ; but it proved to lead efficiently towards minimization.

On the other hand, for real-coded genes, an inventory of the variety of mutation methods is proposed in Michalewicz and Schoenauer[41]or Raghuwanshi and Kakde[42]. They usually rely on a noise added to the initial gene value, according to a specific probability distribution (uniform, normal, polynomial…). Some strategies allow important disturbances at the beginning of a run, and then reduce the possible range of variation around the initial gene when the generation number becomes higher (and, thus, when the located individuals are supposed to be close to the optimal solution). In this work, a uniform distribution probability was chosen (without any change during the run), fitting into the scheme of a diversification promoting strategy.

3.2.3 Constraint handling

Since constraints cannot be easily implemented just by additional equations, as in MP techniques, their handling is a key-point of GAs. Indeed, an efficient solution will widely depend on the correct choice of the constraint handling technique, in terms of both result quality and computational time. In the framework of the studied problem, the constraint on variable bounds is intrinsically considered in the variable encoding while the constraint on productivities is implemented in the model. So the only constraint to be explicitly handled by the GA is the time constraint formulated in equation (2), which imposes the I products to be synthesized before a time horizon H.

A previous work (Ponsich et al.[37]) proved that the choice of the most adequate constraint handling technique depends on the treated example size. Concerning small instances, that are not really severely constrained, an elimination method is adopted : the infeasible individuals should not be able to survive to the next generation. This is carried out by computing the fitness F_i of each individual i, according to its objective function f_i (for the minimization case) :

$$\begin{cases} Fi = f_{max} - fi & \text{if individual } i \text{ is feasible} \\ Fi = 0 & \text{elsewhere.} \end{cases} \quad (6)$$

f_{max} is the worst objective function in the current generation. Then, the implementation of the classical roulette wheel (Goldberg[44]) prevents infeasible individuals from passing the selection step.

But for more complex problems, the elimination technique will induce very expensive computational costs, essentially because of the need to randomly determine an initial population that must be feasible. So, for those cases, a tournament method based on domination rules was prefered. This method relies on rules proposed by Deb[40] and Coello Coello and Mezura Montes[45], by means of an analogy with multiobjective optimisation. Basically, they state that : (i) a feasible individual dominates an infeasible one ; (ii) if two individuals are feasible, the one with the best objective function wins ; (iii) if two individuals are infeasible, the one with the smallest constraint violation wins.

Then, these rules are implemented in a tournament : a fixed number of competitors is randomly taken in the population and, among them, those that will survive are chosen thanks to the above-mentioned rules. Here, a single tournament procedure was used, meaning that the competitor number is equal to the population size while the winner number is the survivor number : then, all the surviving individuals are determined in only one tournament instance for each selection step.

An additional stochastic feature was introduced for the largest size examples, in order to emphasize the population diversification. This was carried out through a probability S_r of either selecting an individual according to the domination rules, or of choosing it randomly. Finally, the elimination technique was implemented from example 1 to example 3, while the tournament based on domination rules was applied for example 4 to 7, with S_r=0.95 for examples 6 and 7.

4. Numerical results and interpretation

Computational results given by the three above-mentioned optimisation techniques are presented in this section. Then, from the perspective of comparing the techniques efficiency, they are analysed in terms of quality and computational time. The number of function calls could have been studied instead of the time criterion, but the former cannot be got from the output files of the *GAMS* solvers (Brooke et al.[22]). The CPU time was measured for a Compaq Workstation W6000.

4.1 Mathematical Programming techniques

The *GAMS* solvers were used without any initialisation. This presents a great advantage on solvers such as those available in the *IMSL* library. So, for the first relaxed MINLP problem, continuous and binary variables are automatically initialised, respectively, in the middle of their definition range and to 1. The results obtained with *DICOPT++* and *SBB* are provided in table 2.

The first obvious comment is that even though *DICOPT++* is really performing for the three first examples, it failed to give any feasible result during a whole run for the remaining ones. The computational time was limited to 24 hours, afterwards it was considered that no result would be found. This is a quite deceiving trend, and from the output file it was observed that no feasible NLP sub-problem was produced through the previous MILP solution.

Table 2. Results with *GAMS* solvers

Example		1	2	3	4	5	6	7
DICOPT++	Found optimum	166079	122470	356610	-	-	-	-
	CPU time	< 1 s.	< 1 s.	< 1 s.	-	-	-	-

SBB	Found optimum	166079	120777	356610	957270	1925887	3865106	4786804
	CPU time	< 1 s.	< 1 s.	< 1 s.	4 s.	35 s.	29 min.	6.4 h.

On the other hand, *SBB* proves a great efficiency since it was able to provide an optimal result for each treated instance. Moreover, the result obtained for example 2 highlights another failure of *DICOPT*++ : the solution given by the latter is 1.4 % higher than *SBB*'s one. This remark seems to point out that *DICOPT*++ stayed trapped in a local optimum, and this assumption can be confirmed by comparing the integer variable set of both results : the two plants show different structures, particularly for batch stages 1 and 2. Figure 2 gives an illustration of both computed plants. The optimum determined by *SBB* is thus better than that found by *DICOPT*++.

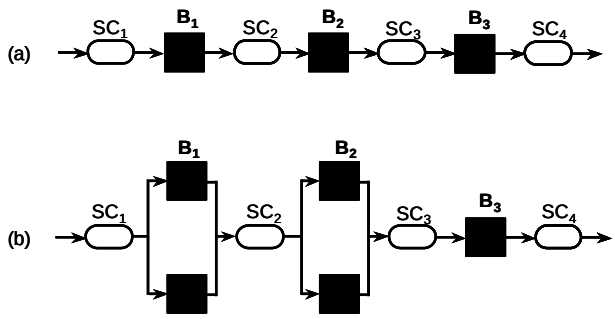

Figure 2. Structures of optima found by *DICOPT*++ (a) *SBB* (b)

With regard to *SBB* computational times, the first instances are solved very quickly, but having the plant size growing, the time increases exponentially and reaches more than 6 hours for the last example. This trend is consistent with the NP-hard nature of the problem, and is confimed by the curve shown in figure 3.

Finally, as an intermediate conclusion for the Mathematical Programming methods, the Branch & Bound algorithm proved a great efficiency. The computational time increases a lot for the last examples, but optimal solutions are provided whatever the treated example is.

Besides, the Outer Approximation method fails to give any result as soon as the problem size begins to be restrictive.

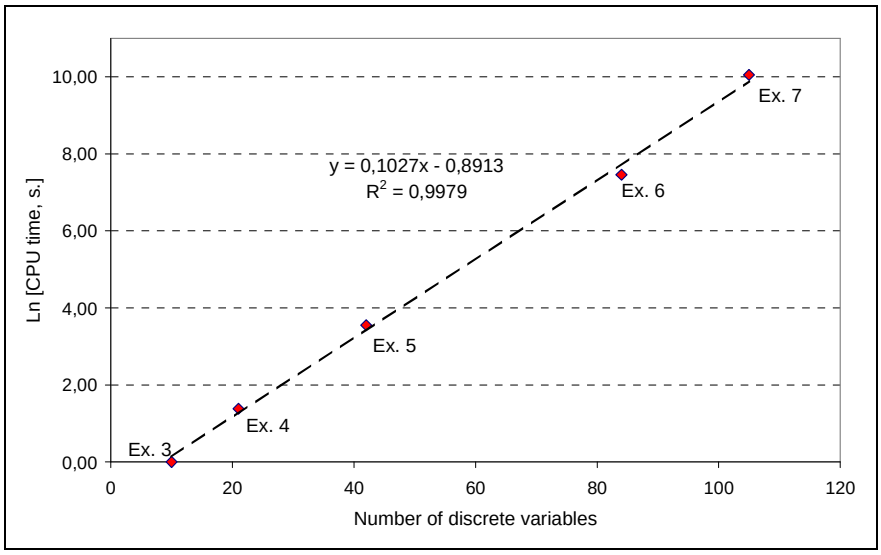

Figure 3. Computational time for *SBB*

4.2 Genetic Algorithm

Just as for Mathematical Programming, the genetic algorithm performances will be evaluated according to both result quality and computational time. The quality is estimated, of course, through the distance between the best found solution and the optimal value. But, since GAs are a stochastic method, their results have to be analysed also in terms of repeatability. So, for each test, the GA was run 100 times. The criterion for repeatability evaluation is the dispersion of the runs around GA best solution F^*_{GA}. The 2%-dispersion and 5%-dispersion are then defined as the percentage of runs providing a result lying respectively in the range $[F^*_{GA}, F^*_{GA}+2\%]$ and $[F^*_{GA}, F^*_{GA}+5\%]$.

The possibility to express the GA's computation qualities in terms of mean and standard deviation of the solution set was considered, but finally rejected : actually, standard deviations will only provide information on what confidence degree an average value should be obtained with. But it is to keep in mind that the aim is to minimize the objective function, i.e. to have a

good concentration of the solutions as close as possible to the best found value. Formulating in other words, 2% and 5%-dispersions enable to know if the solution set is well-distributed, *downward* the objective function value axis. Figure 4 gives an illusration of the advantage of the chosen repeatability criterion : each case (a) and (b) represents a set of runs, and each point is a particular solution. The mean value is (more or less) equal in both cases, and the standard deviation is better in case (b). But the 2%-dispersion highlights the fact that the run set in case (a) is better than the one in case (b), since more solutions are close to best one.

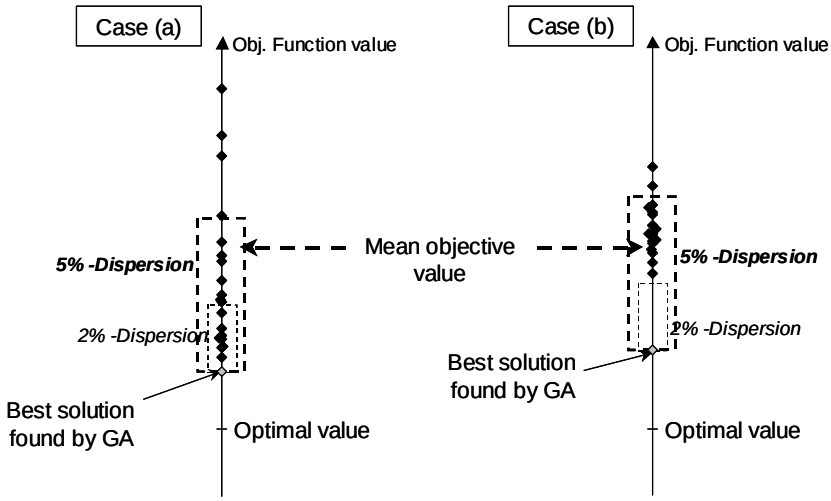

Figure 4. Repeatability indices : dispersions and mean value

The parameters chosen for the GA are the following ones : the survival (respectively mutation) rate is equal to 40 % (resp. 30 %).

The termination criterion used in the following study is the maximum number of generations. No convergence test, such as steadiness of the average/standard deviation of the objective function in consecutive populations, was performed : indeed, the variation magnitude of the associated criteria prevented from reaching a steady state from a generation to the following one. Thus, in spite of the stochastic nature of the GA, the computational time for a given example is the same for all the corresponding runs.

The maximum generation number and the population size then depend on the example complexity. Their values were chosen according to preliminary computations that enabled to determine a "globally" steady state, i.e. a generation number beyond which there was not any appreciable improvement of the (best and average) objective function. The corresponding parameters are presented in table 3.

Table 3. GA parameters for each example

Example	Generation number	Population size	Constraint handling
1	200	200	Elim.
2	200	200	Elim.
3	200	200	Elim.
4	1000	200	Single tour.
5	500	500	Single tour.
6	2000	2000	Single tour.
7	3000	6000	Single tour.

In agreement with the above-described operating mode and performance criteria, the results obtained with the GA are presented in table 4. The reported computational times correspond to only one GA run. For a more meaningful appreciation of the result quality, figure 5 also shows the gap between the best solution found by the GA and the optimum provided by *SBB*. Moreover, 2% and 5%-dispersions appear in figure 6.

Table 4. Results with the GA

Example	1	2	3	4	5	6	7
Best solution	166089	120799	356635	958778	1958190	3961817	498854
CPU time	< 1 s.	< 1 s.	1 s.	8 s.	19 s.	14 min.	1.23 h.

The first example allows validating the good behaviour of the GA, since the best found solution is almost equal to the MP methods optimum. Besides, the dispersions show that all

the runs reach this value. Furthermore, an important decrease in the objective function *mean value* in the population is observed. The latter falls from 409868 to 170991 between the first (randomly generated) generation and the last one, i.e. 58%. The computational time is very low.

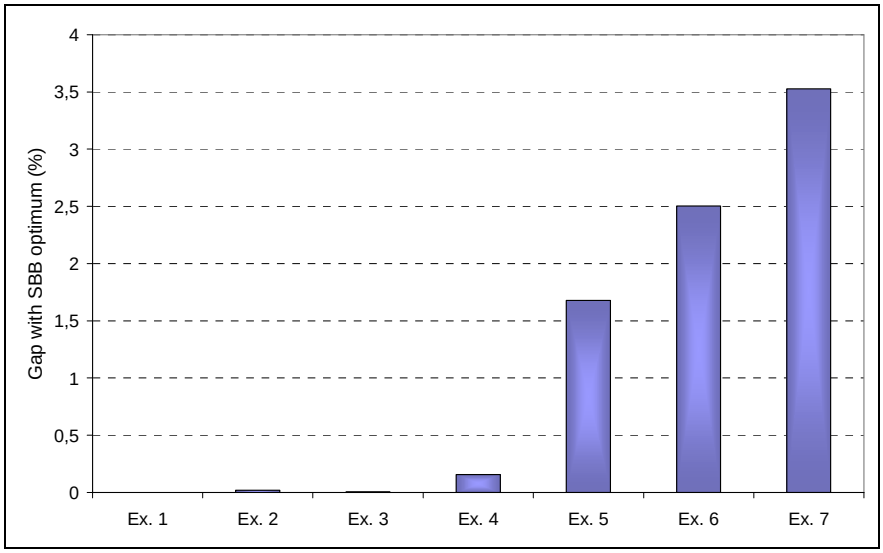

Figure 5. Distance between *SBB* optima and GA results

With regard to example 2, similar trends are noticed. However, this good performance is now underlined by the fact that for this instance, *DICOPT++* had been trapped in a local optimum, corresponding to a plant that showed a different structure from the optimal one located by *SBB* and the GA. That means that the stochastic method did enable to get around the obstacle constituted by sub-optimal solutions. Both 2% and 5%-dispersions, as well as computational time, are excellent too.

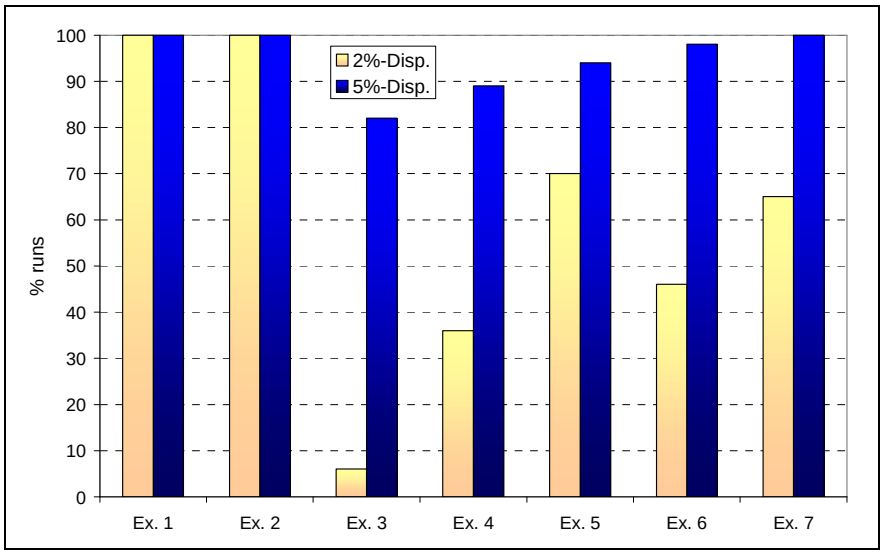

Figure 6. 2% and 5%-dispersions of GA runs

For example 3, results seem similar another time, since the optimum determined by the deterministic methods is approached with a very good precision by the GA (the distance to the optimum is lower than 0.01%). But the dispersions graph belies this assertion. Indeed, the 2%-dispersion is equal to 6%, meaning that a great proportion of the GA runs could not reach the optimum. This behaviour is actually due to a local optimum, and the analysis of the variable set at GA solution proves that some part of the runs stayed trapped on it. This local optimum shows, just as for example 2, a structure different from that of *SBB* optimum, as that can be observed in figure 7 : case (a) is the best one, which has been identified by the *GAMS* solvers. However, in spite of this failing, it is to recall that the optimum was finally determined. Moreover, 82% of the runs provide a result lying up to 5% from the optimal solution, which is still an acceptable value from an economic point of view (let us recall that we are considering a plant investment cost).

Then, from example 4 to example 7, the GA efficiency gradually deteriorates. The best solution found within the 100 runs is more distant from the deterministic optimum when the instance complexity increases. This gap to the optimum is still acceptable but reaches 3.5%

for example 7. Actually, above example 4, the integer variable set of the optimal solution found by *SBB* cannot be identified by the GA.

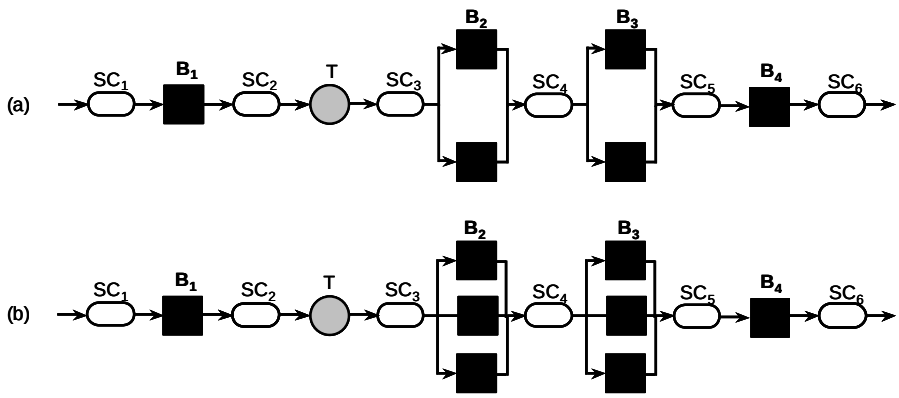

Figure 7. Structures of optima found for example 3

Concerning result repeatability, 2%-dispersion does not show a steady evolution. Nevertheless, for all instances, 5%-dispersion is quite close to 100%, meaning that the best result found by the GA is quite often approached by all runs. This remark induces that the GA proves a quite good robustness in its ability to determine an acceptable result, even if the optimum is not located for the most complex plant instances.

It is quite difficult to give a reliable explanation to the fact that GA performances decrease for the last examples. It can be however related with the number of feasible solutions visited during a run. Figure 8 shows, for the four last examples, the ratio of feasible individuals in the population at last generation : this ratio clearly decreases when the example complexity increases. Thus, it is obvious that GA faces more difficulties to find results very close to the optimum when the feasible individuals only represents about 40 % of the global population, for problem 6 and 7. It is to note that, despite this difficulty to find feasible solutions, the GA is able to locate the feasible space for all instances : there is no failed run that would mean that no one feasible solution was found during the whole search.

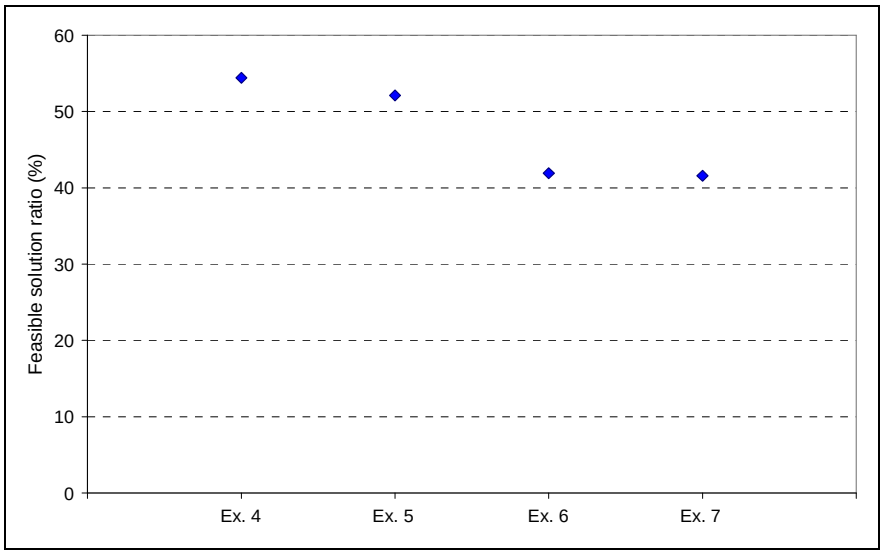

Figure 8. Evolution of feasible solution ratio in last generation

Finally, in respect of computational times, these ones always stand comparison with those of the *SBB* solver. One run takes indeed less time than the MP technique for the latest examples, and even five times quicker for example 7. This competitive feature must be moderated by the fact that *SBB* ensure an optimal solution.

So, to conclude on genetic algorithm performance, it is obvious that it cannot equal *SBB* efficiency, since it hardly identifies the optimal plant structure when the example size is very high. Examples 6 and 7 were studied in order to drive the optimisation method performances to their limit : indeed, the combinatory effect prevents the GA from approaching the optimum. However, the repeatability of GA solutions, measured through 2% and 5%-dispersions, highlights the stochastic method ability to locate feasible, good quality solutions, within a reasonable computational time.

5. Conclusions

This work tackled three optimisation techniques commonly used in Process Engineering, in order to evaluate and compare their performances on a typical batch plant design problem.

This issue is formulated as a complex, NP-hard, MINLP problem. A set of seven increasing size examples was created in order to consider complexity as a parameter of the study.

First, two Mathematical Programming methods from the *GAMS* modelling environment (Brooke et al.[22]) were investigated. The Outer Approximation technique, implemented in the *DICOPT++* solver, failed to give any result as soon as the problem exceeds a certain size. Moreover, it was proved that it might easily stay trapped in a local optimum. On the other hand, a Branch & Bound method, extended to MINLP problem treatment and available in the *SBB* module, was tested. This latter could solve all the examples to optimality, in increasing but reasonable computational times.

Besides, a stochastic technique, namely a genetic algorithm, was used to solve the seven instances. This one was able to provide a feasible, good quality result, for all instances. However, the distance between the optimum and the GA result increases with the problem complexity. CPU time is comparable and sometimes even better than *SBB*'s one. But finally, *SBB* seems to be the best fitted technique for this "equation-oriented" problem formulation, since the result optimality is a great advantage with respect to the GA.

However, two kinds of restriction must be considered, apart from the direct solving efficiency of the Branch & Bound method. Actually, the study did not, at any moment, account for the necessary effort to implement the studied optimisation methods. On the one hand, GAs are simple to implement, show an easy-understanding operating mode, and are *a priori* a generic method, adaptable to a large range of problems. On the other hand, the MP methods call for an important formulation effort that is based on a harsh study of the mathematical properties of the involved functions, and that requires time and experience. These are the prices for optimal results.

Besides, it is to underline that the considered model relies on assumptions that constitute a weakness from the realism point of view. Going into the details of the plant description,

taking operating conditions into account through process variables, adding plant scheduling features, or only considering the unit volumes as discrete variables : all of these strategies might strongly penalize the deterministic methods. On the contrary, GAs should be able to tackle efficiently such kinds of formulation, that would not be necessarily built in an "equation-oriented" way. But, despite these remarks, mathematical formulations and deterministic solution techniques should be used whenever possible. Apart from the guarantee to obtain optimal solutions, the methods force to apply a reasonning on the problem, then producing an "understanding" of the system and of its variables. No such "understanding" is proposed by stochastic techniques results.

Acknowledgements

We fully appreciated the technical support from the gridification project Grid'5000, directed by Michel Daydé from ENSEEIHT (Toulouse). We are particularly greatful to Iréa Touche, who transposed our code on the cluster and made all the necessary modifications in order to have the computations presented in the study completed.

References

(1) Padberg M.; Rinaldi, G. A branch-and-cut algorithm for the resolution of large scale symmetric travelling salesman problems. *SIAM Rev*. **1991**, *33*, 60.

(2) Hao, J. K.; Dorne R. Study of genetic search for the frequency assignment problem. *Lecture Notes in Computer Science*; Springer Verlag: **1996**, 1063, 333.

(3) Lee, S.; Grossmann, I. E. Global optimization of nonlinear generalized disjunctive programming with bilinear equality constraints : applications to process networks. *Comput. Chem. Eng*. **2003**, *27*, 1557.

(4) Teh, Y. S.; Rangaiah, G. P. Tabu search for global optimization of continuous functions with application to phase equilibrium calculations. *Comput. Chem. Eng*. **2003**, *27*, 1665.

(5) Jain, A.; Srinivasalu, S. Determination of an optimal unit pulse response function using real-coded genetic algorithms. *J. Hydrol*. **2005**, *303*, 199.

(6) Zamora, M. Z.; Grossmann, I. E. A global MINLP optimization algorithm for for the synthesis of heat exchanger networks with no stream splits. *Comput. Chem. Eng*. **1998**, *22*, 367.

(7) Guillén, G.; Badell, M.; Espuña, A.; Puigjaner, L. Simultaneous optimization of process operations and financial decisions to enhance the integrated planning/scheduling of chemical supply chains. *Comput. Chem. Eng*. **2006**, *30*, 421.

(8) Ravemark, D. E.; Rippin, D. W. T. Optimal design of a multiproduct batch plant. *Comput. Chem. Eng*. **1998**, *22*, 177.

(9) Dedieu, S.; Pibouleau, L.; Azzaro-Pantel, C.; Domenech, S. Design and retrofit of multiobjective batch plants via a multicriteria genetic algorithm. *Comput. Chem. Eng*. **2003**, *27*, 1723.

(10) Montagna, J. M.; Vecchietti, A. R. Retrofit of multiproduct batch plants through generalized disjunctive programming. *Math.Comp. Model*. **2003**, *38*, 465.

(11) Wolpert, D. H.; Macready, W. G. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput*. **1997**, *1*, 67.

(12) Grossmann, I. E. Review of nonlinear mixed-integer and disjunctive programming techniques. *Opt. Eng*. **2002**, *3*, 227.

(13) Biegler, L. T.; Grossmann, I. E. Retrospective on optimization. *Comput. Chem. Eng*. **2004**, *28*, 1169.

(14) Hao, J. K.; Galinier, P.; Habib, M. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contrainte. *Rev. Intell. Artif*. **1999**, *13*, 121.

(15) Gupta, O. K.; Ravindran, V. Branch and bound experiments in convex nonlinear integer programming. *Manag. Sc*. **1985**, *31* ,1533.

(16) Ryoo, H. S.; Sahinidis, N. V. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Comput. Chem. Eng*. **1995**, *19*, 551.

(17) Smith, E. M. B.; Pantelides C. C. A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. *Comput. Chem. Eng*. **1999**, *23*, 457.

(18) Geoffrion, A. M. Generalized benders decomposition. *J. Opt. Th. Appl*. **1972**, *10*, 237.

(19) Duran, M. A.; Grossmann, I. E. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Math. Prog*. **1986**, *36*, 307.

(20) Westerlünd, T.; Petterson, F. A cutting plane method for solving convex MINLP problems. *Comput. Chem. Eng*. **1995**, *19*, S131.

(21) Raman, R.; Grossmann, I. E. Modelling and computational techniques for logic based integer programming. *Comput. Chem. Eng*. **1994**, *18*, 563.

(22) Brooke, A.; Kendrick, D.; Meeraus, A.; Raman, R. *GAMS User's Guide*; GAMS Development Corporation, 1998.

(24) Leyffer, S. *User manual for MINLP_BB*; University of Dundee Numerical Ananlysis Report NA/XXX: Dundee, USA, **1999**.

(25) Westerlünd, T.; Lundqvist, K. *Alpha-ECP, Version 5.04. An interactive MINLP-solver based on the extended cutting plane method*; Report 01-178-A, Process Design Laboratory, Abo Akademi University: Abo, Finlande, **2003**.

(26) Kirkpatrick, S.; Gelatt Jr., C. D.; Vecchi, M. P. *Optimization by simulated annealing*; IBM Research Report RC9355, **1982**.

(27) Holland, J. H. *Adaptation in natural and artificial systems*; University of Michigan Press, Ann Arbor: MI, **1975**.

(28) Beyer, H. G.; Schwefel, H. P. Evolution Strategies, a comprehensive introduction. *Natur. Comp. Int. J*. **2002**, *1*, 3.

(29) Yang, Y. W.; Xu, J. F.; Soh, C. K. An evolutionary programming algorithm for continuous global optimization. *Eur. J. Oper. Res*. **2006**. *168*, 354.

(30) Grossmann, I. E.; Sargent, R. W. H. Optimum design of multipurpose chemical plants. *Ind. Eng. Chem. Proc. Des. Dev*. **1979**, *18*, 343.

(31) Kocis, G. R.; Grossmann, I.E. Global optimisation of nonconvex mixed-integer non linear programming (MINLP) problems in process synthesis. *Ind. Eng. Chem. Res*. **1988**, *27*, 1407.

(32) Modi, A. K.; Karimi, I. A. Design of multiproduct batch processes with finite intermediate storage. *Comput. Chem. Eng*. **1989**, *13*, 127.

(33) Patel, A. N.; Mah, R. S. H.; Karimi, I. A. Preliminary design of multiproduct non-continuous plants using simulated annealing. *Comput. Chem. Eng*. **1991**, *15*, 451.

(34) Wang, C.; Quan, H.; Xu, X. Optimal design of multiproduct batch chemical process using genetic algorithms. *Ind. Eng. Chem. Res*. **1996**, *35*, 3560.

(35) Wang, C.; Quan, H.; Xu, X. Optimal design of multiproduct batch chemical process using tabu search. *Comput. Chem. Eng*, **1999**, *23*, 427.

(36) Wang, C.; Xin, Z. Ants foraging mechanism in the design of batch chemical process. *Ind. Eng. Chem. Res*. **2002**, *41*, 6678.

(37) Ponsich, A.; Azzaro-Pantel, C.; Pibouleau, D.; Domenech, S. Constraint handling strategies in genetic algorithms – Application to optimal batch plant design. *Chemical Engineering and Processing*. Under Press.

(38) Viswanathan, J.; Grossmann, I. E. A combined penalty function and outer-approximation method for MINLP optimisation. *Comput. Chem. Eng*. **1990**, *14*, 769.

(39) Floudas, C.A. *Non-linear and mixed-integer optimization, Fundamentals and applications*; Oxford University Press, **1995**.

(40) Deb, K. An efficient constraint handling method for genetic algorithms. *Comp. Meth. Appl. Mech. Eng*. **2000**, *186*, 311.

(41) Michalewicz, Z.; Schoenauer, M. Evolutionary algorithms for constrained parameters optimization problems. *Evol. Comp*. **1996**, *4*, 1.

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

(43) Deb, K.; Agrawal, R. B. Simulated binary crossover for continuous search space. *Complex Sys*. **1995**, *9*, 115.

(44) Goldberg, D.E. *Genetic algorithms in search, optimization and machine learning*; Addison-Wesley Publishing Company Inc.: MA, **1989**.

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

Appendix A. Nomenclature

a_j : cost factor for batch stage j.

b_k : cost factor for semi-continuous stage k.

c_s : cost factor for intermediate storage tanks.

H : time horizon [h].

H_i : production time of product i [h].

i : index for products.

I : total number of products.

j : index for batch stages.

J : total number of batch stages.

k : index for semi-continuous stages.

K : total number of semi-continuous stages.

m_j : number of parallel out-of-phase items in batch stage j.

n_k : number of parallel out-of-phase items in semi-continuous stage k.

$prod_i$: global productivity for product i [kg.h^{-1}].

Q_i : demand in product i [kg.h^{-1}].

S : total number of sub-processes.

V_j : size of batch stage j [L].

V_s* : size of intermediate storage tank [L].

α_j : power cost coefficient for batch stage j.

β_k : power cost coefficient for semi-continuous stage k.

γ_s : power cost coeficient for intermediate storage.

Appendix B. Data for the tackled examples

Some data are common to all examples, and are gathered together in tables 5 (cost factors) and 6 (time horizon, size factors for semi-continuous units and storage tanks). Data specific to each example are presented from tables 7 to 10.

Table 5. Cost factors

Batch stages	a_j	250
	α_j	0.60
Semi-continuous stages	b_k	370
	β_k	0.22
Intermediate storage tanks	c_s	278
	γ_s	0.49

Table 6. General data

Time horizon H	6000 [h]
Duty factors for semi-continuous stages D_{ik}	1 [L.kg^{-1}.h^{-1}]
Size factors for storage tanks S_{is}	1 [L.kg^{-1}]

Table 7. Data for example 1

General data		***Recipe for each sub-process***	
I	2	$s = 1$	{B1,B2,B3}
J	3		
K	0		
S	1		
Q [kg]	{150.10^3, 200.10^3}		

		B1	***B2***	***B3***
S_{ij}	$i = 1$	2	3	4
	$i = 2$	4	6	3
p_{ij}^0	$i = 1$	6	16	3
	$i = 2$	7.5	8	2.25
g_{ij}	$i = 1$	0.4	0.35	0.15
	$i = 2$	0.6	0.5	0.2
d_{ij}	i = 1	0.4	0.3	0.2
	i = 2	0.4	0.3	0.2

Table 8. Data for example 2

General data		Recipe for each sub-process	
I	2	$s = 1$	{SC1, B1, SC2, B2, SC3, B3, SC4}
J	3		
K	3		
S	1		
Q {kg]	$\{150.10^3, 200.10^3\}$		

		B1	B2	B3
S_{ij}	$i = 1$	2	3	4
	$i = 2$	4	6	3
p_{ij}^0	$i = 1$	6	16	3
	$i = 2$	7.5	8	2.25
g_{ij}	$i = 1$	0.4	0.35	0.15
	$i = 2$	0.6	0.5	0.2
d_{ij}	i = 1	0.4	0.3	0.2
	i = 2	0.4	0.3	0.2

Table 9. Data for example 3

General data		Recipe for each sub-process	
I	3	$s = 1$	{SC1, B1, SC2}
J	4	$s = 2$	{SC3, B2, SC4, B3, SC5, B4, SC6}
K	6		
S	2		
Q [kg]	$\{437.10^3, 324.10^3, 258.10^3\}$		

		B1	B2	B3	B4
S_{ij}	$i = 1$	8.28	9.70	2.95	6.57
	$i = 2$	5.58	8.09	3.27	6.17
	$i = 3$	2.34	10.30	5.70	5.98
p_{ij}^0	$i = 1$	1.15	9.86	5.28	1.20
	$i = 2$	5.95	7.01	7.00	1.08
	$i = 3$	3.96	6.01	5.13	0.66
g_{ij}	$i = 1$	0.20	0.24	0.40	0.50
	$i = 2$	0.15	0.35	0.70	0.42
	$i = 3$	0.34	0.50	0.85	0.30
d_{ij}	$i = 1$	0.40	0.33	0.30	0.20
	$i = 2$	0.40	0.33	0.30	0.20
	$i = 3$	0.40	0.33	0.30	0.20

Examples 5 to 7 are built just by reproducing several times the structure of example 4. Example 5 represents two added up instances of example 4. Example 6 represents four added up instances of example 4. Example 7 represents five added up instances of example 4. The number of batch stages, semi-continuous stages and intermediate storage tanks can be deducted from the structure of example 4, so as corresponding recipes. The size factors and coefficients for the processing times computations keep being the same as in example 4. Thus, no data is given for those last examples.

Table 10. Data for example 4

General data		***Recipe for each sub-process***	
I	3	$s = 1$	{SC1, B1, SC2, B2, SC3}
J	9		
K	12	$s = 2$	{SC4, B3, SC5, B4, SC6, B5, SC7, B6, SC8}
S	3		
Q [kg]	$\{437.10^3, 324.10^3, 258.10^3\}$	$s = 3$	{SC9, B7, SC10, B8, SC11, B9, SC12}

		B1	***B2***	***B3***	***B4***	***B5***	***B6***	***B7***	***B8***	***B9***
S_{ij}	$i = 1$	8.28	6.92	9.70	2.95	6.57	10.60	7.59	6.74	8.90
	$i = 2$	5.58	8.03	8.09	3.27	6.17	6.57	5.23	4.21	5.35
	$i = 3$	2.34	9.19	10.30	5.70	5.98	3.14	7.41	3.92	6.63
p_{ij}^0	$i = 1$	1.15	3.98	9.86	5.28	1.20	3.57	6.25	2.22	10.00
	$i = 2$	5.95	7.52	7.01	7.00	1.08	5.78	4.38	4.57	4.02
	$i = 3$	3.96	5.07	6.01	5.13	0.66	4.37	3.86	1.39	5.89
g_{ij}	$i = 1$	0.20	0.36	0.24	0.40	0.50	0.40	0.47	0.16	0.68
	$i = 2$	0.15	0.50	0.35	0.70	0.42	0.38	0.29	0.25	0.51
	$i = 3$	0.34	0.64	0.50	0.85	0.30	0.22	0.32	0.27	0.45
d_{ij}	$i = 1$	0.40	0.29	0.33	0.30	0.20	0.35	0.39	0.18	0.26
	$i = 2$	0.40	0.29	0.33	0.30	0.20	0.35	0.39	0.18	0.26
	$i = 3$	0.40	0.29	0.33	0.30	0.20	0.35	0.39	0.18	0.26

Appendix C. SBX crossover

Deb and Agrawal[43] make the assumption that the probability distribution described in figure 1 (see section 4.2.2), that reproduces the single-point crossover for binary coding, can be approached by the following polynomial distribution, of order n.

$$\begin{cases} \mathscr{C}(\beta) = 0{,}5(n+1)\beta^{n} & \text{on the left-hand side of the parent,} \\ \mathscr{C}(\beta) = 0{,}5(n+1)\dfrac{1}{\beta^{n+2}} & \text{on the right hand side.} \end{cases} \tag{7}$$

According to the previous expression, for higher n values, the crossover procedure will have a more "contracting" effect, i.e. a greater probability to create children close to the parents. Then, drawing a random number u between 0 and 1 and according to the probability distribution $\mathscr{C}(\beta)$, the corresponding abscissa β^* is computed by the expression :

$$\begin{cases} \beta^* = (2u)^{\frac{1}{n+1}} & \text{if } u \leq 0.5, \\ \beta^* = \left(\dfrac{1}{2(1-u)}\right)^{\frac{1}{n+1}} & \text{elsewhere.} \end{cases} \tag{8}$$

In case of bounded variables, equation (8) becomes :

$$\begin{cases} \beta^* = (\alpha_1 u)^{\frac{1}{n+1}} & \text{if } u \leq 1/\alpha, \\ \beta^* = \left(\dfrac{1}{2-\alpha_1 u}\right)^{\frac{1}{n+1}} & \text{elsewhere.} \end{cases} \tag{9}$$

With :

$$\alpha_1 = 2-(\alpha_2)^{-(n+1)} \quad (10) \qquad \alpha_2 = 1+\frac{2}{y^{(2)}-y^{(1)}}\min\left[x^{(1)}-x^L, x^U-x^{(2)}\right] \quad (11)$$

It is to notice that in equation (11), $x^{(1)}$ is implicitly lower than $x^{(2)}$.

Finally, the children offspring are created :

$$\begin{cases} y^{(1)} = 0,5.[(1+\beta^*).x^{(1)} + (1-\beta^*).x^{(2)}] \\ y^{(2)} = 0,5.[(1-\beta^*).x^{(1)} + (1+\beta^*).x^{(2)}] \end{cases} \quad (12)$$

This procedure is carried out for all loci with a probability p_x. If the locus codes an integer variable, the decimal part is truncated to get a consistent child offspring. The p_x probability is used to control the disturbance of the chromosomes, to ensure that only a part of them is modified by the crossover (like in a single-point crossover, for binary coding). Thus, two parameters must be set for the SBX crossover implementation. This was realized thanks to preliminary sensitivity analyses :

- the order of the polynomial probability distribution ; here, n was chosen equal to 1 (which does not characterize a “contracting” but an “expanding” procedure).
- the p_x probability is set to 0.2.

List of captions

Figure 1. Probability distribution for the location of an offspring, SBX crossover

Figure 2. Structures of optima found by DICOPT++ (a) SBB (b)

Figure 3. Computational time for SBB

Figure 4. 2% and 5%-dispersions

Figure 5. Distance between SBB optima and GA results

Figure 6. 2% and 5%-dispersions of GA runs

Figure 7. Structures of optima found for example 3

Figure 8. Evolution of feasible solution ratio in last generation