PERFORMANCE ANALYSIS OF OPTIMIZATION METHODS IN PSE APPLICATIONS
Mathematical Programming Versus Grid-based Multi-parametric Genetic Algorithms

A. Ponsich[1], I. Touche[1], C. Azzaro-Pantel[1,*], M. Daydé[2], S. Domenech[1] and L. Pibouleau[1]

[1]Laboratoire de Génie Chimique, UMR 5503 CNRS/INP/UPS, Toulouse, France.
[2]Institut de Recherche en Informatique de Toulouse, UMR 5505 CNRS/INP/UPS ENSEEIHT, Toulouse, France.

Abstract: Due to their large variety of applications in the PSE area, complex optimisation problems are of high interest for the scientific community. As a consequence, a great effort is made for developing efficient solution techniques. The choice of the relevant technique for the treatment of a given problem has already been studied for batch plant design issues. However, most works reported in the dedicated literature classically considered item sizes as continuous variables. In a view of realism, a similar approach is proposed in this paper, with discrete variables representing equipment capacities. The numerical results enable to evaluate the performances of two mathematical programming (MP) solvers embedded within the GAMS package and a genetic algorithm (GA), on a set of seven increasing complexity examples. The necessarily huge number of runs for the GA could be performed within a computational framework based on a grid infrastructure; however, since the MP methods were tackled through single-computer computations, the CPU time comparison are reported for this one-PC working mode. On the one hand, the high combinatorial effect induced by the new discrete variables heavily penalizes the GAMS modules, DICOPT++ and SBB. On the other hand, the Genetic Algorithm proves its superiority, providing quality solutions within acceptable computational times, whatever the considered example.

*Correspondence to: Professor C. Azzaro-Pantel, Laboratoire de Génie Chimique, UMR 5503 CNRS/INP/UPS, 5 rue Paulin Talabot-BP 1301, 31106 Toulouse Cedex 1, France. E-mail: catherine.AzzaroPantel@ensiacet.fr

Keywords: grid computing; batch plant design; genetic algorithms; mathematical programming.

/

INTRODUCTION

A great variety of applications drawn from the Process System Engineering can be formulated as complex optimization problems. This large number of optimization 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 and complexity, basically deriving from three issues: presence of integer variables (breeding high computational times), of non-linearities (that sometimes prevent from getting a global optimum) and of severe constraints (making the search for feasible solutions a harsh process).

In order to face these problems, a significant investigation effort has been carried out to develop efficient and robust optimization methods. But if they prove to be well-fitted to the particular case they consider, the performance of these techniques is not constant over all the problems. Actually, method efficiency for a particular example is hardly predictable. 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*. A previous work, comparing the performance of three optimization methods for a few instances of some specific batch plant design problems, provided some guidelines on the most appropriate methods: it proved the superiority of a Branch & Bound technique on a genetic algorithm and an outer approximation algorithm (Ponsich *et al.*, 2006b).

However, in the formulation adopted in the abovementioned study, the item sizes (volumes for batch stages and treatment capacity for semi-continuous stages) are continuous variables. Yet, it seems obvious that in the industrial practice, the design of items equipments does not require such a level of accuracy, which seems not realistic. Besides, equipment manufacturers propose the items following defined size ranges (volumes or treatment capacity). This means that an item can only adopt a discrete number of predefined values. Actually, this approach is not a new one, since it has been already used in previous works relying on discrete event simulation (DES) for batch plant design problem modelling (Dietz, 2004).

With regard to the formulation given in Ponsich *et al.* (2006b), the new representing mode of the variables just involves a simple modification of the initial model: the item sizes can adopt a value among a discrete set, defined in the problem data. In the new problem, all the optimization variables are now integer variables. However, the remaining 'internal variables' (used only within the model module: for instance, batch sizes, or operating times ...) are still real, even though the new operating mode reduces their definition set to discrete portions of the initial one. So, the problem nature is still MINLP and a methodology similar to that developed in Ponsich *et al.* (2006b) can be used. Thus, the same optimization methods can be run on the same test problems, in order to determine if the conclusions drawn from previously experiments with continuous item sizes are still valid.

Nevertheless, although the global problem nature does not change, some modifications must be carried out. Basically, a great number of executions of the stochastic algorithm will be necessary, implying a huge computational time that a single computer would not be able to support. The experiments were thus performed using a computational grid. This latter has already proved its efficiency for parallel computing and especially for multi-parametric studies (Laforenza, 2002), which is the case here.

This paper is divided into five sections. The following section presents the framework of the new discretized grid-based approach. Then, an overview is given in the third section on the use of the Grid'5000 grid experimental platform. Some typical results are analysed in the fourth section. Finally, the fifth section is devoted to conclusions and perspectives.

A NEW APPROACH FOR BATCH PLANT DESIGN PROBLEMS

In most of the studies dealing with batch plant design problems, the classical model traditionally accounts for the item sizes involved in the process as continuous bounded variables. The corresponding formulation is briefly recalled in a first subsection. A new approach is then proposed, in order to get a more realistic representation of the plant equipment items: the new operating conditions are defined, and the methodology of the study is given. Finally, the grid-based approach is presented.

Classical Batch Plant Design Framework

Due to the growing interest for the batch operating mode, a lot of studies address the batch plant design issue (Grossmann and Sargent, 1979; Kocis and Grossmann, 1988; Modi and Karimi, 1989; Patel *et al.*, 1991; Wang *et al.*, 1996, 1999; Wang and Xin, 2002). 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, 1989). It accounts for the synthesis of I products treated in J batch stages and K semi-continuous units (pumps, heat exchangers and so on). Each (batch or semi-continuous) stage is made up of several out-of phase parallel units of same type and size. Typically, the model involves the following optimization variables:

- the size (V_j for the batch stages and R_k for the semi-continuous ones) of the items of each stage are continuous variables;
- the number (m_j and n_k) of items for each stage are discrete variables.

Moreover, $S-1$ intermediate storage tanks, with size V_s^*, divide the whole process into S sub-processes. The main feature is the minimization of the investment cost for all the items of the plant:

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

where α_j and a_j, β_k and b_k, γ_s and c_s are classical cost coefficients (a complete nomenclature is provided at the end of the paper). 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:

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

The values for Q_i are given in the problem data, while $Prod_i$ is inferred from equations involving the computation of batch sizes B_i for every product i and of the processing times of batches in both batch and semi-continuous stages (T_{ij} and θ_{ik}). Then, the maximum value of the processing times in the stages belonging to each sub-process provides the corresponding limiting cycle time T_{is}^L. Finally, the minimum ratio B_i/T_{is}^L represents the productivity of the plant $Prod_i$ for each product i. The complete model and equations will not be presented here, but the reader should report to Ponsich *et al.* (2006a) to get the detailed formulation.

Then, the aim of batch plant design problems is to find the plant configuration that respects the production requirements within the time horizon, while minimizing the economical criterion. The resulting MINLP problem proves to be non-convex and NP-hard (Wang *et al.*, 1996).

New Operating Conditions

As it has been mentioned, the adopted model has one drawback: the item sizes are represented as real variables. This point leads to solutions involving a useless accuracy on the equipment characteristics, which cannot be implemented in real word. Thus, for industrial managers, it should be really more appropriate to obtain solutions, i.e., plant configurations, that match the industrial requirements and the predefined sizes proposed by the item manufacturers. This leads to a discretization of the corresponding variables in the model, and raises the question of how this discretization task should be performed.

In reported studies (Dietz, 2004), the item size of each stage could have three possible values: a small one, a large one and a medium one. Here, since the aim is to represent the whole initial definition range of the continuous variables, three different cases of discretization range are considered, involving different numbers of possible values. The bounds of the continuous variables were initially set as:

- for batch stages: $V_{\min} = 250 \leq V_j \leq V_{\max} = 10\,000$ L, $\forall\, j$;
- for semi-continuous stages: $R_{\min} = 300$ L h$^{-1} \leq R_k \leq R_{\max} = 10\,000$ L h^{-1}, $\forall\, k$.

Then, various discretization modes are tested:

- *Large-grain discretization (LD)*: the interval between two possible values is equal to 2500 L (or L h^{-1}), globally meaning that the five possible values for batch stage size are 250; 2500; 5000; 7500; 10 000 L. And for the semi-continuous stages: 300; 2500; 5000; 7500; 10 000 L h^{-1}.
- *Medium-grain discretization (MD)*: the interval between two possible values is equal to 500 L (or L h^{-1}).
- *Small-grain discretization (SD)*: the interval between two possible values is equal to 10 L (or L h^{-1}).

Methodology

In a previous work (Ponsich *et al.*, 2006b), the study was aiming at providing some guidelines to help choose the most relevant optimization technique, in the framework of batch plant design problems. In a similar way, the objective here is to evaluate and compare the performance of three optimization techniques for the revised problem based on an integer formulation. These methods are tested over a set of seven increasing complexity examples, for which all data are detailed in Ponsich *et al.* (2006b). The only data differing from the original MINLP model are those corresponding to the discretization, which were defined in the previous subsection. Some details on each example (plant structure and stage number) are provided in Figure 1.

The methods are two mathematical programming modules from the GAMS modelling environment (SBB and DICOPT++; Brooke *et al.*, 1998) and a stochastic technique: a genetic algorithm (GA). The detailed algorithm of these methods are exhaustively described in Ponsich *et al.* (2006b), it is just useful to recall the GA internal procedures for this study:

- the variable coding is carried out with the so-called weighting box. This coding procedure is classically used for continuous variables such as presented in previous studies (Ponsich *et al.*, 2006a). The variable variation range is discretized according to some desired accuracy and a technique similar to binary coding is used to represent the resulting reduced variables. In the current case of discrete variables, the accuracy is equivalent to the distance between two possible values of the variables: the greater the number of feasible values is, the more the variable behaviour gets closer to a continuous one;
- this encoding technique allows the use of classical crossover (one cut-point) and mutation techniques (inversion of the bit value);
- an elimination method is used for small size examples for constraint handling (from example 1 to 3), while a domination-based tournament technique (from example 4 to 7) is adopted for higher-sized, more severely constrained problems (Deb, 2000; Coello Coello and Mezura Montes, 2002).

Computations are carried out for the seven examples, and for each discretization mode, the efficiency of each method is evaluated. Besides, from a mathematical point of view, it should be useful to check out the influence of the discretization function on the numerical solution. In other words, it seems interesting to wonder if, for different possible ranges of the item sizes (or discretization modes), different plant configurations can be obtained and if these plant configurations approach the optimum value previously determined with continuous sizes (MINLP problem).

Adaptation to the Chosen Optimization Tools

The changes in the optimisation variable nature do not have any spreading effect: when modelling within both GAMS environment and GA, these changes only affect the modules related to variable definition and handling. The initial lower bounds are kept unchanged, while the upper ones might be slightly modified due to discretization range.

Within the GAMS modelling environment, the discrete values are artificially created and designed by introducing binary variables for each possible discrete value of the variables definition set. For instance, let consider that for stage j, an item volume can adopt P values: v_{jp}, $p = \{1, \ldots, P\}$. Then, the associated binary variable y_{jp} is equal to 1 if the item size is v_{jp}, and 0 else. Besides, an additional restriction has to be imposed, forcing the item size to adopt only one value, i.e., for each j, there is one and only one p^* such as $y_{jp^*} = 1$. The analytical expressions of the above-stated concepts are formulated according the following equations:

$$V_j = \sum_{p=1}^{P} y_{jp} \cdot v_{jp} \quad \text{with} \quad y_{jp} \in \{0, 1\}, j = 1, Nstages \tag{3}$$

$$\sum_{p=1}^{P} y_{ip} = 1 \quad j = 1, \ldots, NStages \tag{4}$$

In the case of the GA, the abovementioned classical encoding method is used, based on the classical discretization of

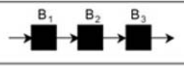
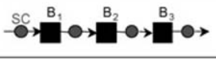
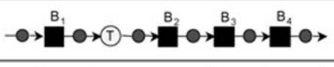
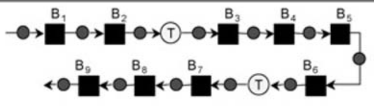
Example	Plant structure	Batch/semicont. stages
1		3 / 0
2		3 / 4
3		4 / 6
4	 B1 B2 T B3 B4 B5 B9 B8 B7 T B6	9 / 12
5	2 x Example 4	18 / 24
6	4 x Example 4	36 / 48
7	5 x Example 4	45 / 60

Figure 1. Set of increasing complexity examples.

the continuous variables and binary-like coding. Thus, a loss in accuracy results in an enlargement of the interval between two consecutive size ranges: the set of discrete possible values is created for each variable.

Some comments can be already drawn from the new operating mode. The first one is that, obviously, this operating mode will favour the GA. Actually, concerning the problem modelled with GAMS, the discretization involves the introduction of additional binary variables: it has been shown yet that this combinatorial explosion is a penalizing aspect for the reported mathematical programming optimization methods (Ponsich *et al.*, 2006b). So, in the *large-grain discretization* case (five possible values), the complexity increase is about $5^{NStages}$. This means that the GAMS solvers will have more difficulties to treat higher size examples, and also smaller discretization modes.

On the other hand, the discretization mode favours the work of the GA. Indeed, increasing the interval between two possible values means an important decrease in the chromosome size, and thus, in the search space. So, the new operating mode has opposite effects depending on the optimization methods, giving a strong advantage to the GA.

Grid Computing Approach

A particular drawback of the above-stated methodology should be underlined: the GA being a stochastic technique, it is always useful to carry out many runs for the same problem, in order to get the best possible result and to evaluate the technique repeatability. In this study, each example was treated 100 times by the metaheuristics. Moreover, the treated instances involve problems of size up to 210 variables and the problem is identified as an NP-hard problem. As a consequence, these runs involve a huge computational effort, and a classical computing infrastructure may not manage to carry out the amount of necessary computations. This is why we have considered the use of a computational grid as a suitable alternative.

Grid computing is a paradigm well-adapted to two types of applications:

- Parallel applications: an execution of the code uses several nodes to perform the calculation. Usually, the execution time decreases when increasing the number of processors until a threshold volume is reached (this maximum processor number that can be efficiently exploited, on a given problem, gives some information on the scalability of the code).
- Multi-parametric applications: the code is executed several times, but with different values of some input parameters. Using a single PC, the operating way is to run the code with a parameter value, and when finished, to execute it again with another value for the same parameter, and so on. Since the grid often provides a large number of computing nodes, it makes the user able to run the code for all parameter values simultaneously (using one node per parameter value).

The application we consider in this paper is multi-parametric: GA is a serial Fortran code that does not use any external library (which makes the installation process in every site of grid infrastructure greatly simplified). Practically, we solve various problem instances with a stochastic algorithm and repeat 100 times the run for each example. This application is multi-parametric by nature. A grid approach is perfectly relevant to this case study.

It is to note that the possibility of using a grid-based environment for the mathematical programming methods might be naturally contemplated. However, the issue of the study is not to implement a grid-based infrastructure that would support any kind of optimization technique. Here, the meaning of the grid use is that a lot of GA runs must be performed to analyse GA repeatability and to provide finally some ‘average trends’ for the stochastic technique.

GRID′5000

An Overview of Grid′5000

Grid is an active research area that provides technologies now mature enough to be used for real-life applications. Projects like e-Science, TeraGrid, Grid3, DEISA and NAREGI, to quote a few of them, demonstrate that large scale infrastructures can be deployed to provide scientists fairly easy access to resources that are geographically distributed and that belong to different administration areas. Despite its establishment as a viable computing infrastructure, there are still many issues to be solved (in terms of performance, fault tolerance, QoS, security and fairness).

Grid’5000, is a nation-wide infrastructure for research in grid computing (see http://www.grid5000.org, Bolze *et al.*, 2006). It is designed to provide a scientific tool for computer scientists similar to the large-scale instruments used by physicists, astronomers, and biologists. The project is still under construction but already in use in France. Grid’5000 is a large-scale experimental tool, with a deep reconfiguration capability, a controlled level of heterogeneity and a strong control and monitoring infrastructure.

The reconfiguration capability allows scientists to deploy, install, boot and run their specific software images (possibly including all the layers of the software stack). In a typical experiment sequence, a researcher reserves a partition of Grid’5000, deploys its software image, reboots all the machines of the partition, runs the experiment, collects results and relieves the machines. This reconfiguration feature allows all researchers to run their experiments in the software environment exactly corresponding to their needs.

The Grid’5000 platform is distributed over nine sites in France: Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia and Toulouse. Every site hosts a cluster and all sites are connected by the RENATER high-speed network (all links will provide 10 Gbps). Currently, the platform has around 2700 processors. The Toulouse site, so-called Grid’Mip, is composed of 57 nodes, dual CPU racks consisting of two AMD Opteron running at 2.2 Ghz, with 2 Gb of memory.

Application to the GA Working Mode

The gridification of the GA application is straightforward. The first step consists in modifying the code in order to be able to schedule the 100 executions over 100 different nodes. This code is composed of three steps:

- Initialization of the parameters
- Loop: do *i* from 1 to 100
- Random generation of the population
- Simulation

- $i = i + 1$
- Statistic calculations

To use the grid, the following phases have to be performed on each node:

- Initialization of the parameters.
- Random generation of the population.
- Simulation.

At the end of the executions on the 100 nodes, statistic calculations can be made. The modified code has been deployed over five clusters of Grid'5000: Lyon, Orsay, Sophia, Bordeaux and Toulouse. A Grid'5000 cluster is composed of a frontal machine, which is used for the connexion (where the OAR scheduler is accessible), and of the worker nodes, where the jobs are executed. To carry out the computations, the node number required on each cluster has to be chosen. Then, each frontal must be connected and the jobs must be submitted using OAR. This means that, for each example, 100 jobs have to be submitted with OAR. To avoid that, a web interface was developed to automatically perform the job submission.

COMPUTATIONAL RESULTS

The performed experiments are presented in three different subsections. Firstly, a comparison is made between the two extreme cases: the results obtained with a *large-grain discretization* (LD) approach and those (optima) determined in Ponsich *et al.* (2006b) for the original MINLP formulation, with continuous item sizes. Subsequently, the results corresponding to the other discretization modes (*medium-grain discretization*—MD; and *small-grain discretization*—SD), solved with the GA and with the GAMS solvers, are discussed. Each method performance will be evaluated in both terms of quality and computational times. Finally, the influence of the discretization mode on the results is analysed: in other words, passing from a discretization mode to another one, do the elements of the solution vector move towards the closest acceptable value of the new size range?

The resolution of all the considered problem instances by the GA, which is computationally intensive, is performed over the computational grid described previously. Getting 100 nodes over five clusters of Grid'5000 was not a problem. Each execution required, for the largest instances, about 4 h on a single PC, giving a total execution time of 400 h for the 100 GA runs. This global time was decreased to 4 h using Grid'5000, corresponding to a reduction factor equal to 100. This is a great benefit, but a major one compared to traditional large computer infrastructures is that time between execution launching (usually through a batch system that may limit the maximal number of processors to a lower number and impose a long wait on a specific queue) and result recovering is drastically reduced, allowing to carry out more simulations.

For illustration purpose, Table 1 displays some execution times achieved, with GA on Grid'5000 for the two instances involving the greatest computational times, i.e., batch plants 6 and 7 for the SD mode. These two examples are presented with the number of runs on each site and the time of the fastest and slowest runs. The execution time corresponds to the maximal time considering all the runs.

Table 1. Execution times on Grid'5000.

	Example 6-SD			
Site/nodes	Toulouse/20	Sophia/20	Lyon/20	Orsay/40
Min time	02:32:57	03:23:15	03:18:16	03:23:18
Max time	02:41:34	03:57:00	03:43:25	03:57:57
Total time		03:57:57		
	Example 7-SD			
Site/nodes	Toulouse/50	Sophia/10	Lyon/20	Orsay/20
Min time	02:29:37	03:23:05	03:18:20	03:23:17
Max time	02:40:16	03:57:05	03:43:24	03:57:26
Total time		03:57:26		

However, all the computational times available in the following section refer to runs carried out by a unique computer, in order to be able to compare the GA times with those required by the GAMS solvers.

Extreme Cases

The results achieved by the three methods, as well as the corresponding computational times, are reported in Table 2. The first lines give the optimal values of the original MINLP problem, and the corresponding CPU time. The solutions of this mixed-integer problem were determined with the SBB solver: actually, this MP method provided the best results for all instances (Ponsich *et al.*, 2006b). It must be highlighted that however, the MINLP optimal values are not an objective to be reached, since the discrete problem and the mixed-integerone are obviously different. But, they just provide a reference point helping to evaluate the following results.

Note that the italics style for some of the SBB values points out that the optimality conditions (intersection of the upper and lower bounds in the Branch & Bound procedure) were not obtained in a 24 h run, although a feasible solution can be provided. The GA computational times refer to only one run.

From Table 2, it first appears that DICOPT++ is not able to deal with the large combinatorial effect generated by the discretization of the item sizes. Despite the good results observed on example 1, the solver provides a solution for example 2 that is much higher than those produced by SBB and the GA. From example 3 to example 7, DICOPT + does not converge towards any feasible solution within 24 h. Actually, beyond example 2, SBB and the GA are the only ones that provide a solution to the LD problem. Figure 2 is a good illustration of the gap between these results and that of the MINLP problem (first line of Table 2). Although presenting the same results as table 2, this representation mode enables to visualize the three algorithm performances.

Concerning the quality of the obtained results, Figure 2 highlights that up to example 4, the solutions given by the *SBB* solver and the GA are the same: they all show a gap with the MINLP optima that never turns out to be lower than 5.5%. This gap is explained by the LD mode, involving that some MINLP optimal solutions are prohibited. Thus, the structure of the optimal plant for the LD problem may be different from that of the MINLP one, and induces a higher cost.

However, even though the result quality is equal for both methods, the computational times are clearly higher for SBB: the Branch & Bound method is strongly penalized by

Table 2. Results for the original MINLP and LD problems.

Example		1	2	3	4	5	6	7
MINLP	Result	166 079	120 777	356 610	957 271	1 925 887	3 865 106	4 786 804
	CPU time	<1 s	<1 s	<1 s	4 s	35 s	29 min	6.4 h
DICOPT++	Result	180 637	208 431	—	—	—	—	—
	CPU time	10 s	<1 s	—	—	—	—	—
SBB	Result	180 637	142 436	376 701	1 016 777	*2 432 203*	*5 008 147*	*6 456 102*
	CPU time	1 s	25 s	33 s	13.3 h	*24 h*	*24 h*	*24 h*
GA	Result	180637	142436	376701	1016777	2065769	4144808	5290474
	CPU time	<1 s	<1 s	3 s	9 s	14 s	34.3 min	1.6 h

the problem complexity. Indeed, up to example 3, the computational time of SBB is higher than those of the GA, although lying in a same order of magnitude. But for example 4, the difference becomes huge: 13.3 h for SBB versus 9 s for each GA run (or versus 4 s when solving the MINLP problem with SBB, thus proving that the discretization penalizes heavily the MP methods efficiency). Actually, it might be either the computational time or an insufficient computing power that causes SBB failure for example 5: the solver does not converge. The maximum time accepted for a run is set to 24 h. So, from example 5, the GA proves its superiority on the deterministic method in terms of computational time and solution quality.

However, SBB is able to provide feasible non-optimal solutions for the highest size examples, discovered during the search tree enumeration. But these solutions lie quite far away from the GA ones: for instance, for example 5, the difference between the GA result and the feasible solution located by SBB is equal to 17%. Finally, for examples 5 to 8, the GA results keep being satisfying: the gap to the MINLP optimal solution is steady.

To provide further information on the stochastic method behaviour, Figure 3 reports the times related to the MINLP and LD problems by the GA. They prove that, except for example 3, the GA is a little more efficient for solving the LD problem than solving the MINLP one. This general trend is accounted by the fact that the coding procedure of continuous variable requires more genes to get the required accuracy (see remark on variable encoding, in second section). The exception of example 3 is basically due to the constraint handling, which imposes the generation of feasible initial individuals. This is logically a more difficult task when the search space is discrete but differences between the two working modes only appear when the instance complexity becomes higher (there is no difference for examples 1 and 2). For the largest examples (from example 4 to 7), the tournament-based constraint handling mode does not impose any more the respect of constraints on the first population. Then, the influence of the chromosome size is the predominant factor, making the GA job easier for the LD problem.

Thus, the LD mode revealed the GA superiority on the mathematical programming methods. The strong combinatorial effect caused prohibitive computational times, preventing the solvers from converging beyond example 3 for DICOPT++ and example 5 for SBB. On the opposite, the GA still provides good-quality solutions, close to the MINLP optima, and within a reasonable computational effort.

Small and Medium Discretizations (SD and MD)

GA results

By considering smaller discretization modes, the number of feasible values for the variables becomes greater; according to the explanations given previously on the coding procedure, this is equivalent to gradually bringing the GA back to a standard working mode for continuous variables (with an increasing accuracy). The GA should thus show an intermediate behaviour, between the two above-considered extreme cases (MINLP and LD). Table 3 gives the best solution found by the GA for the example set; the results for LD are also recorded.

The gap between the MINLP optimal solutions and the results of Table 3 are presented in Figure 4. As predicted,

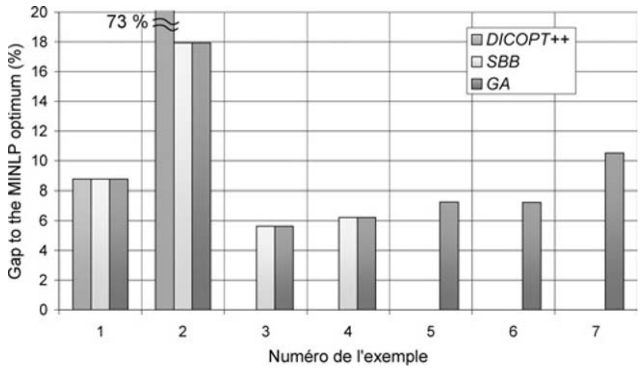


Figure 2. Distance from the MINLP optimum to the LD solutions.

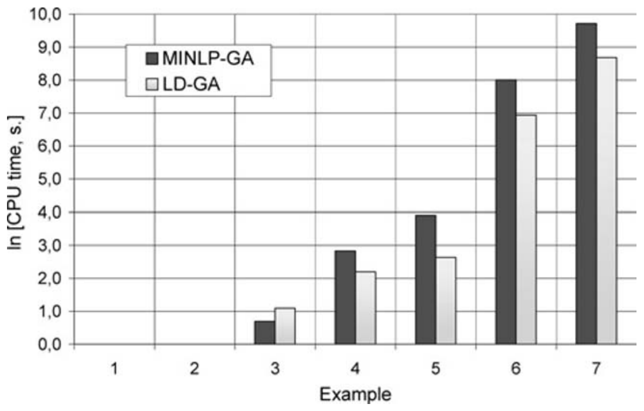


Figure 3. Computational times for MINLP and LD problems solved with SBB and the GA.

Table 3. Solutions of the GA for SD, MD, LD problems.

	Ex. 1	Ex. 2	Ex. 3	Ex. 4	Ex. 5	Ex. 6	Ex. 7
SD	166 172	122 364	370 083	972 042	1 994 946	4 010 943	5 050 716
MD	167 941	123 352	373 082	974 873	2 005 608	4 015 673	5 103 457
LD	180 637	142 436	376 701	1 016 777	2 065 769	4 144 808	5 290 474

this figure shows that the GA keeps finding good-quality solutions, and the gap to the MINLP optimum is, logically, all the more satisfying when the discretization interval is small: in these cases, the problem is clearly more similar to the MINLP one.

Concerning the computational effort, the emerging trend is that the required time is higher for MD mode, and even more for the SD one: actually, for smaller discretization intervals, the GA behaviour becomes similar to that observed for the solution of the MINLP problem (see Figure 3). So, the SD points are always above the MD ones, and above the LD ones in Figure 5, except for example 3 (for this one, the trend can be attributed to the constraint handling method, as already discussed in the previous subsection).

MP methods

For the GAMS solvers, the MD and LD modes represent a much more difficult challenge. Actually, considering that, in a given discretization mode, the item sizes can adopt P possible values, the related combinatorial effect is equal to $P^{NStages}$. So, if the discretization interval gets smaller, number P increases and the problem becomes much more complex. Then, considering the quite poor results obtained for the LD with SBB and especially DICOPT++, it is to expect that the solvers will not be more efficient for SD and MD. Actually, they might reach their solving ability limits.

Indeed, DICOPT++ manages to find optimal solutions just for the first example, and only for a MD. For the SD problem and the larger instances, no feasible solution was found within a 24 h run. Concerning SBB, the Branch & Bound method shows a better efficiency for the simplest examples. However, as soon as the complexity increases, the solution is strongly penalized by the prohibitive computational time.

Table 4 shows the distance between SBB results and the GA best solution as well as the computational time required by SBB is presented. The results prove that a threshold exists, beyond which SBB cannot conclude on the result optimality anymore. The algorithm cannot reach the termination criterion during the time allowed for a run: at the end of the search, the gap between the upper and lower bounds is still huge. SBB nevertheless gives feasible solutions (upper bound), very far away from the GA result. The computations with SBB were thus carried out up to example 3. Beyond this limit, only sub-optimal solutions would be hypothetically determined. The grey cases point out the instances for which no optimality condition was reached in the 24 h run.

Influence of the Discretization Mode

In order to provide a more detailed analysis of the computational results previously presented, it is useful to quantify the influence of the discretization mode on the final best solution. Since the mathematical programming optimization techniques were not able to solve instances beyond example 4, the GA results constitute the basis of the following analysis. Nevertheless, all the examples will not be reported here. Actually, the discretization generates cuts in the feasible space so that different plant structures (i.e., item number per stage) are computed according to the considered discretization mode. Thus, among the seven treated examples, two cases emerge: in the former (examples 1, 2, 4 and 5, for instance), the plant structure is (more or less) unchanged whatever the considered discretization mode; in the (example 3), the plant structure varies from a discretization mode to another. It is thus possible to make an analysis for each case.

For the first case, example 4 provides a representative basis when the effect of the discretization mode has fewer influence and some trends can be inferred from the following study. Actually, for example 4, no change is observed in

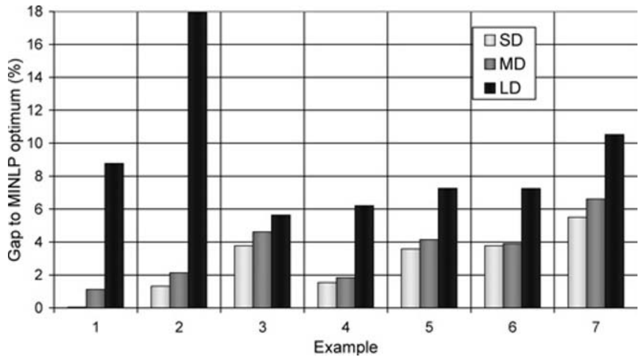


Figure 4. Gap to the MINLP optimum.

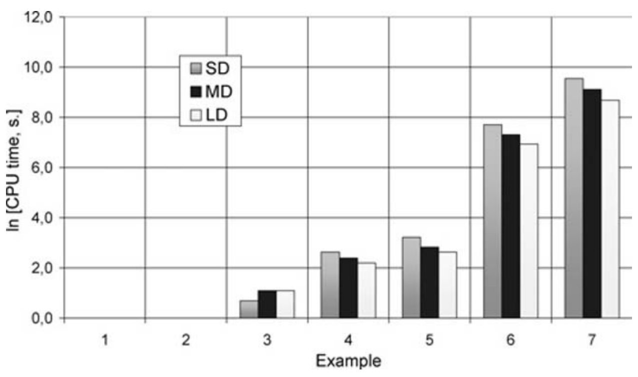


Figure 5. CPU time of GA for SD, MD, LD problems.

Table 4. Computations carried out with SBB.

	Ex. 1	Ex. 2	Ex. 3	Ex. 4	Ex. 5
SD	0% 7.3 (min)	18.3% 24 (h)	—	—	—
MD	0% 3 (s)	5.7% 24 (h)	25.2% 24 (h)	—	—

Table 5. Batch stage volumes for the solutions of MINLP, SD, MD, LD problems for example 4.

	MINLP	SD	MD	LD	Stand. dev.
B1	5827	6390	6250	5000	625
B2	8511	9050	9250	7500	783
B3	7440	8050	7750	10 000	1154
B4	4052	4040	4250	5000	453
B5	5675	6120	6250	7500	782
B6	6583	6510	6750	10 000	1695
B7	8485	8340	8750	10 000	757
B8	7534	7340	7750	7500	169
B9	9949	9690	9750	10 000	150
	Semi-continuous stages				2446

the discrete variables of the solution for the various discretization modes, except for the LD one. For the sake of illustration, Figure 6 shows the item number per batch stage, for each discretization mode (the structure corresponding to semi-continuous stages is identical in all cases). These values are different only for the LD mode: stages B2 and B3 respectively have 2 and 3 items for all cases, but 3 and 2 items for LD.

It is thus interesting to observe the volumes of the corresponding stages (see Table 5). The results corresponding to the MINLP problem are also reported, and the standard deviation of the item volumes for each stage is computed. The mean value of these nine standard deviations is equal to 730 L. For semi-continuous stages, the mean value of the standard deviation of the item volumes is presented too. It can firstly be underlined that those two values are very different: the variation of the sizes according to the discretization mode is much more intense for semi-continuous stages. This can be explained by the cost coefficients in the objective function (b_k and β_k), which are really lower for semi-continuous stages, cf. equation (1). Thus, the pressure towards minimization exerted by the stochastic optimization method on the associated variables is weaker than that exerted on the batch stages.

Besides, it should also be noted that in the 'mean value of the standard deviations', the weight induced by the LD mode is very important. Removing this mode from the mean value calculation, the resulting new 'mean value' is equal to 230 L. This trend is also illustrated in Figure 7. The volumes determined for the LD mode are quite different from those found by the other modes, although they follow the global trends. More particularly for some batch stages, the variation is more important. This is not only due to the changes in the item numbers per stage (B2 and B3), but also to the width of the discretization interval, which prohibits some zones of the MINLP problem feasible space.

These conclusions would be the same for examples 1, 2 or 5. For instance, for example 5, the volumes and item numbers are represented in Figure 8, for different discretization modes. It is clear that the number of item is always similar, except for batch stages B7, B9, B16 and 18, only for the LD mode. Consequently, the associated volumes follow the same trends, except for the LD mode again for which the issues of plant structure difference (for the mentioned batch stages) and wide range between two feasible values appear.

Finally, as mentioned previously, a second case is observed when the plant structure shows more definite differences according to the considered discretization mode. Example 3 is a quite representative instance of this behaviour. In Figure 9, the number of items per stage keeps being almost unchanged for all the semi-continuous stages. But concerning batch stages, the item number is a bit different for all discretization modes. Thus, the corresponding volumes, although following similar trends, are always different switching from a discretization mode to another one.

In conclusion, it seems that on the one hand the discretization mode has a great influence on the final solution, since the plant structure may be strongly different from a mode to another. On the other hand, when the item number per stage does not vary that much, similar trends are observed for all modes, when the discretization interval allows it. Besides, these assertions are true only if the variables

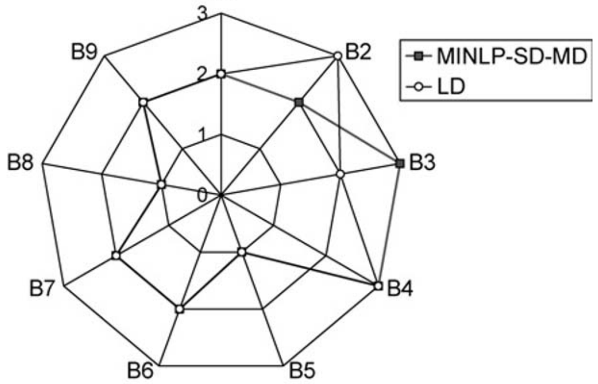


Figure 6. Discrete variables of example 4 solution (batch stages), according to the discretization mode.

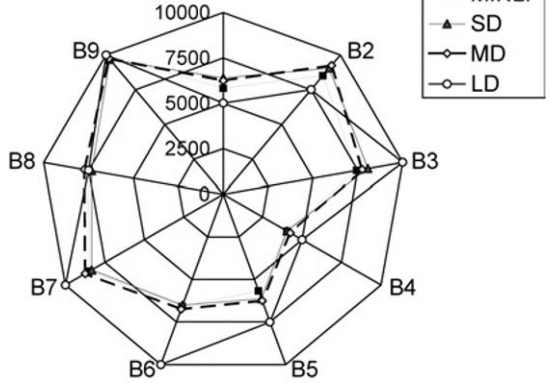


Figure 7. Batch stages volumes of example 4 solution, according to the discretization mode.

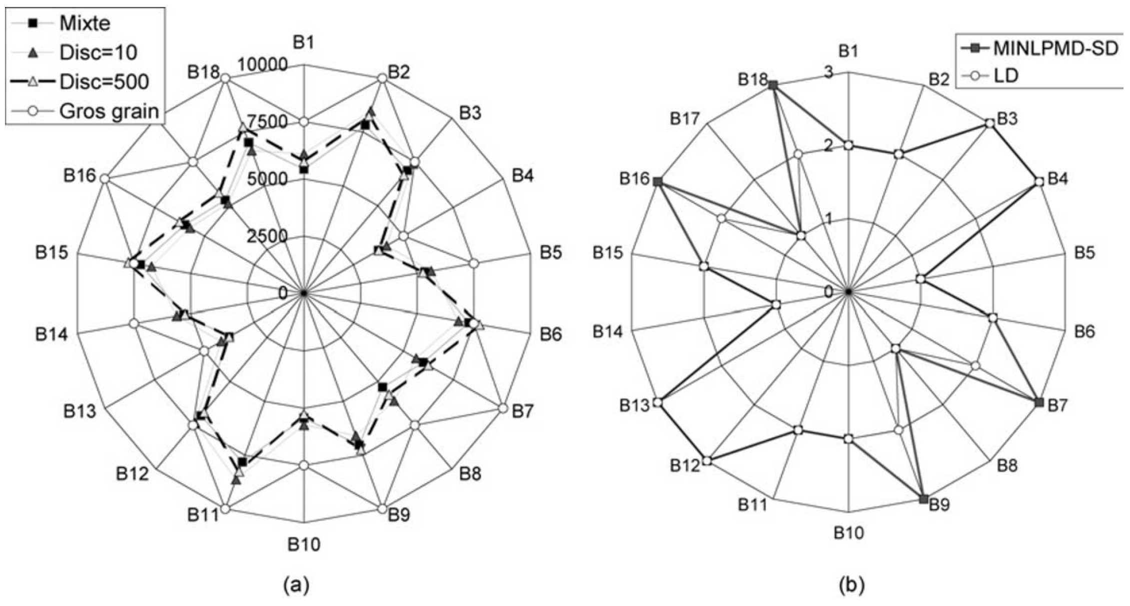


Figure 8. Batch stages volumes (a) and item number per stage (b) of example 5 solution, according to the discretization mode.

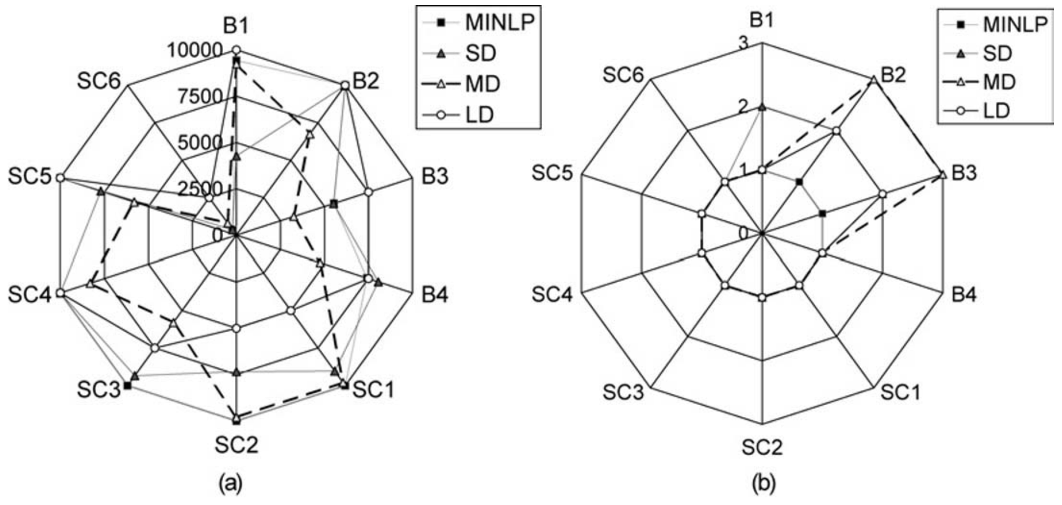


Figure 9. Batch and semi-continuous stages volumes (a) and item number per stage (b) of example 3 solution, according to the discretization mode.

have an important weight in the optimization criterion: here for instance, the item sizes of the semi-continuous stages might vary slightly without causing much change in the objective function.

CONCLUSION

In this work, a variant to the classical batch plant design problem was proposed: with a view of realism, the item sizes are not anymore considered as real variables in the model, but are now represented by discrete variables. The aim is to evaluate and compare three optimisation methods performances for this modified problem: two mathematical programming approaches and a GA. The three techniques are evaluated on seven test examples. The discretization mode, i.e., the width of the interval between two authorized values, is a parameter of the presented study.

The analysis of the behaviour of a stochastic optimization method, on various instances, with several runs for each instance, represents a huge computational cost that would be prohibitive with a classical computational tool. A grid-based approach was thus developed, in order to reduce the computing time. Indeed, for the largest instance, the elapsed time for simulation is reduced from 400 h down to 4 h. This is an impressive benefit, which allowed to have the computations completed and to draw the conclusions of this study.

The results do agree with the trends previously inferred from the working mode and variable representation of each method. On the one hand, mathematical programming techniques are heavily penalized by this operating mode, since the combinatorial effect induced by the discretization of the item sizes leads to prohibitive computational times, and sometimes prevents the solvers from converging with fulfilled optimality conditions. DICOPT++ proved its inefficiency, including small size examples. However, SBB was able to solve various instances, and when not able to converge, it provided feasible solutions (quite far away from the optimal ones).

On the other hand, at the same time, the GA work is made easier by this discretization, which does not involve any change in the variable encoding, except a great reduction of the chromosome size. So, the GA could solve all the tackled problems, for various discretization modes, showing a good performance, quite similar to that related to the original MINLP problem.

It should be underlined here that the MP methods too may have been adapted to a grid-based environment, in order not to be limited by the CPU time that prevents convergence. However, this was not the case of this paper, since the grid-based strategy was only used considering the huge number of GA runs (for repeatability tests). Actually, all the numerical results were provided and all the conclusions were drawn for a single-PC computation basis.

Finally, this approach, using discrete variables to represent item sizes, seems appropriate to treat the batch plant design problem, in both industrial and numerical points of view. In that context, the stochastic methods have proven to be the most efficient one to provide good-quality and realistic solutions.

NOMENCLATURE

a_j	cost factor for batch stage *j*
b_k	cost factor for semi-continuous stage *k*
B_i	batch size of product *i*
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	index for sub-processes
S	total number of sub-processes
T_{ij}	processing time for a batch of product *i* in batch stage *j*
T_{is}^L	limiting cycle time of product *i* in sub-process *i*
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 coefficient for intermediate storage
θ_{ik}	processing time for a batch of product *i* in semi-continuous stage *k*

REFERENCES

Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.L. and Touche, I., 2006, Grid'5000: a large scale and highly reconfigurable experimental grid testbed, *International Journal of High Performance Computing Applications*, 20: 481–494.

Brooke, A., Kendrick, D., Meeraus, A. and Raman, R., 1998, *GAMS User's Guide* (GAMS Development Corporation, Washington DC, USA).

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

Deb, K., 2000, An efficient constraint handling method for genetic algorithms, *Computers Methods Applied in Mechanical Engineering*, 186: 311–338.

Dietz, A., 2004, Optimisation multicritère pour la conception d'ateliers discontinus multiproduits: aspects économique et environnemental, PhD thesis, INP ENSIACET, Toulouse, France.

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

Kocis, G.R. and Grossmann, I.E., 1988, Global optimisation of non-convex mixed-integer non linear programming (MINLP) problems in process synthesis, *Ind Eng Chem Res*, 27: 1407–1421.

Laforenza, D., 2002, Grid programming: some indications where we are headed, *Parallel Computing*, 28: 1733–1752.

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

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

Ponsich, A., Azzaro-Pantel, C., Domenech, S. and Pibouleau, L., 2006a, Constraint handing strategies in genetic algorihms. Application to optimal batch plant design, *Chemical Engineering and Processing*, in press.

Ponsich, A., Azzaro-Pantel, C., Domenech, S. and Pibouleau, L., 2006b, MINLP optimisaion strategies for batch plant design problems, *Ind Eng Chem Res.*, 46: 854–863.

Wang, C., Quan, H. and Xu, X., 1996, Optimal design of multiproduct batch chemical process using genetic algorithms, *Ind Eng Chem Res*, 35: 3560–3566.

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

Wang, C. and Xin Z., 2002, Ants foraging mechanisms in the design of batch chemical process. *Computers and Chemical Engineering*, 41: 6678–6686.