J Glob Optim (2007) 37:481–503
DOI 10.1007/s10898-006-9075-3

ORIGINAL PAPER

Scatter search for chemical and bio-process optimization

Jose A. Egea · María Rodríguez-Fernández · Julio R. Banga · Rafael Martí

Received: 9 March 2006/Acceted: 27 July 2006 /
Published online: 7 October 2006

Abstract Scatter search is a population-based method that has recently been shown to yield promising outcomes for solving combinatorial and nonlinear optimization problems. Based on formulations originally proposed in 1960s for combining decision rules and problem constraints such as the surrogate constraint method, scatter search uses strategies for combining solution vectors that have proved effective in a variety of problem settings. In this paper, we develop a general purpose heuristic for a class of nonlinear optimization problems. The procedure is based on the scatter search methodology and treats the objective function evaluation as a black box, making the search algorithm context-independent. Most optimization problems in the chemical and bio-chemical industries are highly nonlinear in either the objective function or the constraints. Moreover, they usually present differential-algebraic systems of constraints. In this type of problem, the evaluation of a solution or even the feasibility test of a set of values for the decision variables is a time-consuming operation. In this context, the solution method is limited to a reduced number of solution examinations. We have implemented a scatter search procedure in Matlab (Mathworks, 2004) for this special class of difficult optimization problems. Our development goes beyond a simple exercise of applying scatter search to this class of problems, but presents innovative mechanisms to obtain a good balance between intensification and

J. A. Egea (✉) · M. Rodríguez-Fernández · J. R. Banga
Process Engineering Group, Instituto de Investigaciones Marinas (C.S.I.C.), Eduardo Cabello 6
36208, Vigo, Spain
e-mail: jegea@iim.csic.es

M. Rodríguez-Fernández
e-mail: mrodriguez@iim.csic.es

J. R. Banga
e-mail: julio@iim.csic.es

R. Martí
Departamento de Estadística e Investigación Operativa, Universitat de València, Dr. Moliner 50
46100, Burjassot (Valencia), Spain
e-mail: rafael.marti@uv.es

diversification in a short-term search horizon. Computational comparisons with other recent methods over a set of benchmark problems favor the proposed procedure.

Keywords Metaheuristics · Scatter search · Chemical engineering · Global optimization · Nonlinear dynamic systems

1 Introduction

Mathematical modeling, optimization, and control have become fundamental tools for optimally designing and operating production facilities in most industrial sectors such as the chemical and biotechnological process industries (e.g., see Shimizu 1996; Bailey 1998; Banga et al. 2003a, b; Biegler and Grossmann 2004; Floudas et al. 2005). Since many of these processes are operated in batch or semi-continuous modes, especially in the case of the bio-industries, they have an inherently dynamic nature. In this context, there are at least three relevant types of optimization problems: optimal operation (dynamic optimization), integrated process design and control, and parameter estimation. These problems can be stated as, or transformed into, nonlinear programming problems subject to dynamic (usually differential algebraic) constraints. Their highly constrained, non-linear and sometimes nonsmooth nature often causes nonconvexity, and therefore global optimization methods are needed in order to find suitable solutions (Floudas et al. 2005).

In this paper we address the optimization of this important class of problems i.e. the nonlinear programing problem with both differential and algebraic constraints, as given in the following formulation:

$$\underset{x}{\text{Min}}\, \phi(y,x). \tag{1}$$

Subject to:

$$f(\dot{y},y,x) = 0, \tag{2}$$

$$y(t_0) = y_0, \tag{3}$$

$$h(y,x) = 0, \tag{4}$$

$$g(y,x) \leq 0, \tag{5}$$

$$x^{\mathrm{L}} \leq x \leq x^{\mathrm{U}}. \tag{6}$$

In this model $\phi(y,x)$ is the objective function; x the vector of the *nvar* decision variables; y the set of state system variables ($\dot{y}$ is the time derivative of y), f the set of differential equations describing the system dynamics, and h and g are, respectively, the equality and inequality constraints. Finally, x^{L} and x^{U} are the lower and upper bounds, respectively, for the decision variables.

Many real world optimization problems in chemical engineering (and also in business or economics) are too complex to be given tractable mathematical formulations. Although we have used mathematical notation in the formulation above, we are considering the general case in which there is no explicit expression of the objective

function $\phi(y,x)$ since it contains multiple nonlinearities, combinatorial relationships and uncertainties inaccessible to modeling except by resorting to more comprehensive tools like computer simulation. In the context of optimizing simulations, a "complex evaluation" refers to the execution of a simulation model (which can be extremely time-consuming).

Theoretically, the issue of identifying best values for a set of decision variables x falls within the realm of optimization. Until quite recently, however, the methods available for finding optimal decisions have been unable to cope with the complexities and uncertainties posed by many real world problems of the form treated by simulation. The area of stochastic optimization has attempted to deal with some of these practical problems, but the modeling framework limits the range of problems that can be tackled with such technology. The complexities and uncertainties in these systems are the primary reason that simulation is often chosen as a basis for handling the decision problems associated with them. Advances in the field of metaheuristics have led to the creation of optimization engines that successfully guide a series of complex evaluations with the goal of finding optimal values for the decision variables.

The resolution of the differential-algebraic constraints $f(\dot{y},y,x) = 0$ is usually a hard problem. Thus, an approximate method (typically a Runge–Kutta, BDF method, or a similar numerical process) is applied to obtain the y-values corresponding to a set of decision values x. Therefore, this kind of complex problem is solved with a black-box sequential method in which the optimization takes place in the set of the decision variables x. In particular, given a set of values for the x-variables, the approximate solver of the differential-algebraic constraints computes the associated y-values. We then test the feasibility of the (x, y) solution with the h and g functions. To sum it up, a remarkable computational effort is associated with the evaluation and the feasibility test of one solution.

The disadvantage of "black-box" approaches is that the optimization procedure is generic and has no knowledge of the process employed to perform evaluations inside the box and therefore does not use any problem-specific information. The main advantage, on the other hand, is that the same optimizer can be applied to complex systems in many different settings. Therefore, although we have designed and tested our method in the process systems engineering environment, it can be directly applied to solve any kind of black-box optimization problems in other settings.

Advances in the field of metaheuristics—the domain of optimization that incorporates artificial intelligence and analogs to physical, biological or evolutionary processes—have led to the creation of new approaches that successfully integrate simulation and optimization. We have identified the following five methods as the best available to handle black-box problems. We will use them in our computational comparison in Sect. 4.

Csendes (1988) proposed *Global,* a hybrid global optimization algorithm which is essentially an improvement of the algorithm by Boender et al. (1982). It uses a random search followed by a local search routine. Initially, it carries out a clustering phase. Next, two different local search procedures can be selected for a second step. The first one is a quasi-Newton type algorithm. The second, more appropriate for problems with discontinuous objective functions or derivatives, is a robust random search method.

Differential Evolution (DE) is a heuristic algorithm for the global optimization of nonlinear and (possibly) nondifferentiable continuous functions presented by Storn and Price (1997). This population-based method handles decision variables by means

of a direct search method which outperforms other popular global optimization algorithms such as simulated annealing or genetic algorithms, and it is widely used by the *evolutionary computation* community.

Stochastic Ranking Evolution Strategy (SRES), developed by Runarsson and Yao (2000, 2005), consists of an (μ, λ) evolution strategy combined with an approach to balancing objective and penalty functions stochastically. In the (μ, λ)-ES algorithm, the evaluated objective and the penalty functions for each individual are used to rank the individuals in a population, and the best (highest ranked) μ individuals out of λ are selected for the next generation. This feature makes it especially appealing for the case of constrained problems, like those considered here.

Jones (2001) proposed *Direct* (*DIviding RECTangles*), a deterministic global optimization algorithm based on a modification of the Lipschitzian optimization scheme to solve difficult global optimization problems. The search is performed by dividing the space into hyper-rectangles and is specifically designed for those cases in which the objective function is nonsmooth, no derivative information is available, or its evaluation requires several different simulations to be performed. The algorithm operates by systematically dividing the optimization domain into hyper-rectangles, and evaluating the objective function in their centers. There are two phases to an iteration of *Direct*: first, hyper-rectangles are identified as potentially optimal (i.e. it is expected that they contain a global solution); the second phase consists of dividing potentially optimal hyper-rectangles into smaller ones. The objective function is evaluated in the centers of new hyper-rectangles and the search is directed towards unexplored regions of the domain. We use the Matlab implementation of Finkel and Kelley (2004) in our computational testing.

OptQuest is the optimization engine released by OptTek Systems Inc. As described in Laguna and Martí (2002), this is a generic optimizer that overcomes the deficiency of black box systems and successfully embodies the principle of separating the method from the model. In such a context, the optimization problem is defined outside the complex system. Therefore, the evaluator can change and evolve to incorporate additional elements of the complex system, while the optimization routines remain the same. Hence, there is a complete separation between the model used to represent the system and the procedure that solves optimization problems formulated around the model. The optimization technology embedded in OptQuest is the metaheuristic known as scatter search. The method is organized to (1) capture information not contained separately in the original points, (2) take advantage of auxiliary heuristic solution methods (to evaluate the combinations produced and to actively generate new points), and (3) make dedicated use of strategy instead of randomization to carry out component steps. In our testing we use the implementation known as OQNLP (Ugray et al. 2005) which uses OptQuest to provide starting points for any gradient-based local NLP solver. This procedure combines the superior accuracy and feasibility-seeking behavior of gradient-based local NLP solvers with the global optimization abilities of scatter search.

In this paper we propose a scatter search (SS) implementation to solve the optimization–simulation problem that arises in chemical processes. This study goes beyond a simple exercise of implementing a known method to solve a problem, and we propose innovative mechanisms and new strategies to overcome the limitations of the previous methods described above. The remainder of the paper is organized as follows. Section 2 is devoted to the generic SS methodology as it is usually applied. In Sect. 3, we show our adaptation of this methodology to solve the optimization problem

described above, in which we propose new mechanisms for search intensification and diversification. The computational comparison in Sect. 4 reports the solution of the five methods above and our proposal when solving a set of well-known problems in the context of chemical process optimization. The paper finishes with the associated conclusions.

2 Scatter search methodology

The SS was first introduced in Glover (1977) as a heuristic for integer programing. The SS orients its explorations systematically, relative to a set of reference points that typically consist of good solutions obtained by prior problem solving efforts. The *scatter search template* (Glover 1998) has served as the main reference for most of the SS implementations to date. The SS methodology is very flexible, since each of its elements can be implemented in a variety of ways and degrees of sophistication. In this section, we give a basic design to implement SS based on the well-known "five-method template" (Laguna and Martí 2003). The advanced features of SS are related to the way these five methods are implemented. That is, the sophistication comes from the implementation of the SS methods instead of the decision to include or exclude certain elements (as in the case of tabu search or other metaheuristics).

The fact that the mechanisms within SS are not restricted to a single uniform design allows the exploration of strategic possibilities that may prove effective in a particular implementation. These observations and principles lead to the following "five-method template" for implementing SS as follows:

1. A *diversification generation method* to generate a collection of diverse trial solutions, using an arbitrary trial solution (or seed solution) as an input.
2. An *improvement method* to transform a trial solution into one or more enhanced trial solutions. Neither the input nor the output solutions are required to be feasible, though the output solutions will more usually be expected to be so. If no improvement of the input trial solution results, the "enhanced" solution is considered to be the same as the input solution.
3. A *reference set update method* to build and maintain a *reference set* consisting of the b "best" solutions found (where the value of b is typically small e.g. no more than 20), organized to provide efficient accessing by other parts of the method. Solutions gain membership to the reference set according to their quality or their diversity.
4. A *subset generation method* to operate on the reference set, to produce several subsets of its solutions as a basis for creating combined solutions.
5. A *solution combination method* to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solution vectors.

Figure 1 shows the interaction among these five methods and highlights the central role of the reference set. This basic design starts with the creation of an initial set of solutions P, and then extracts from it the reference set (*RefSet*) of solutions. The darker circles represent improved solutions resulting from the application of the improvement method.

The diversification generation method is used to build a large set P of diverse solutions. The size of P (*PSize*) is typically at least ten times the size of *RefSet*. The initial reference set is built according to the Reference Set Update Method, which

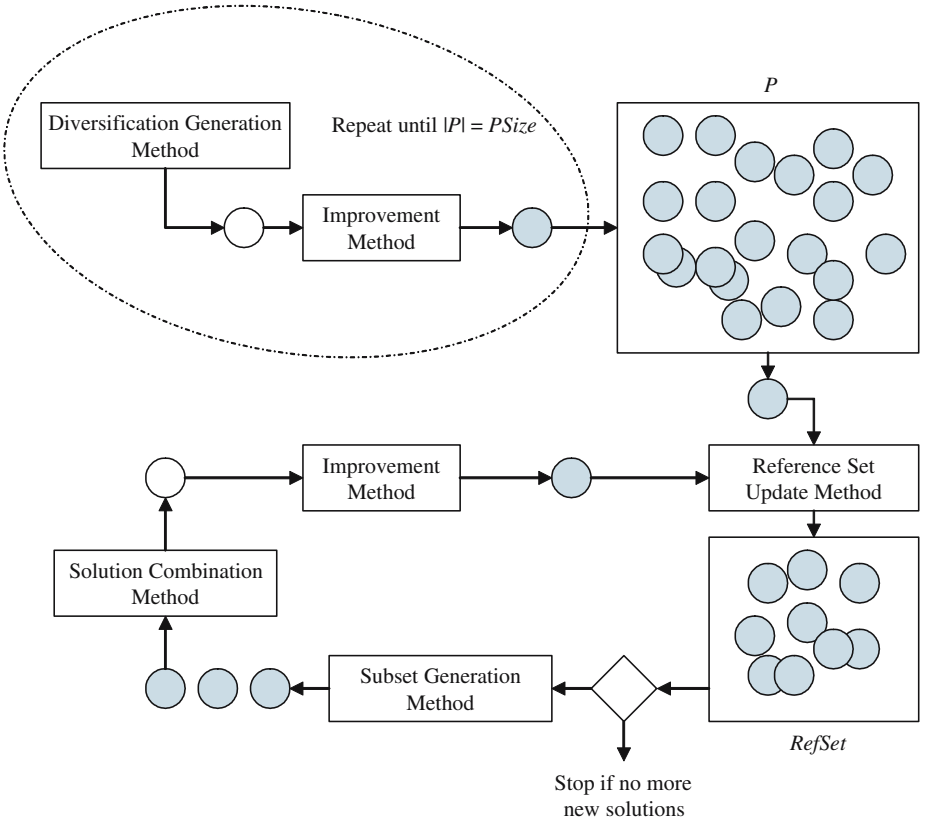

Fig. 1 Schematic representation of a basic SS design

can take the b better solutions (as regards their quality in the problem solving) from P to compose the *RefSet*. However, diversity can be considered instead of or in addition to quality for the updating. For example, the reference set update method could consist of selecting b distinct and maximally diverse solutions from P. Regardless of the rules used to select the reference solutions, the solutions in *RefSet* are ordered according to quality, where the best solution is the first one in the list. The search is then initiated by applying the subset generation method which, in its simplest form, involves generating all pairs of reference solutions. The pairs of solutions in *RefSet* are selected one at a time and the Solution Combination Method is applied to generate one or more trial solutions. These trial solutions are subjected to the Improvement Method. The Reference Set Update Method is applied once again to build the new *RefSet* with the best solutions, according to the objective function value, from the current *RefSet* and the set of trial solutions. The basic procedure terminates after all the generated subsets are subjected to the Combination Method and none of the improved trial solutions are admitted to *RefSet* under the rules of the Reference Set Update Method. However, in advanced SS designs, the *RefSet* rebuilding is applied at this point and the best $b/2$ solutions are kept in the *RefSet* and the other $b/2$ are selected from P, replacing the worst $b/2$ solutions.

The reference set, *RefSet*, is a collection of both high-quality solutions and diverse solutions that are used to generate new solutions by way of applying the Combination Method. We can use a simple mechanism to construct an initial reference set and then update it during the search. The size of the reference set is denoted by $b = b_1 + b_2 = |RefSet|$. The construction of the initial reference set starts with the selection of the best b_1 solutions from P. These solutions are added to *RefSet* and deleted from P. For each solution in *P-RefSet*, the minimum of the distances to the solutions in *RefSet* is computed. Then, the solution with the maximum of these minimum distances is selected. This solution is added to *RefSet* and deleted from P, and the minimum distances are updated. The process is repeated b_2 times, where $b_2 = b - b_1$. The resulting reference set has b_1 high quality solutions and b_2 diverse solutions.

Of the five methods in SS methodology, only four are strictly required. The Improvement Method is usually needed if high quality outcomes are desired, but a SS procedure can be implemented without it. On the other hand, a more complete SS designs could incorporate a tabu search procedure (since the TS and SS approaches are specially compatible) or another metaheuristic as the improvement procedure, although this will tipically entail a longer running time.

It is interesting to observe similarities and contrasts between SS and the original GA proposals. Both are instances of what are sometimes called "population based" or "evolutionary" approaches. Both incorporate the idea that a key aspect of producing new elements is to generate some form of combination of existing elements. However, GA approaches are predicated on the idea of choosing parents randomly to produce offspring, and further on introducing randomization to determine which components of the parents should be combined. By contrast, the SS approach does not emphasize randomization, particularly in the sense of being indifferent to choices among alternatives. Instead, the approach is designed to incorporate strategic responses, both deterministic and probabilistic, that take account of evaluations and history. The SS focuses on generating relevant outcomes without losing the ability to produce diverse solutions, due to the way the generation process is implemented.

3 Scatter search for the optimization–simulation problem

In this section, we describe the adaptation of the SS methodology to solve the optimization problem introduced in Sect. 1, in which the objective function is given by a simulation process and some constraints are differential equations describing the system dynamics.

In unconstrained problems, the evaluation of a solution is directly given by the objective function, or the output of the simulation process that defines it. In constrained problems, to make the search flexible, we allow the method to generate and combine unfeasible solutions. In particular, we add a penalty term to the objective function value defined by a weight multiplied by the maximum percentage of the violation of the constraints. We consider relative violations instead of absolute ones to take into account the different orders of magnitude among constraints. This weight, *wpen*, can be modified by the user to vary the degree of unfeasibility permitted in the search.

3.1 Diversification generation method

Our SS method implemented in Matlab begins by generating an initial set P of diverse points. This is usually accomplished by dividing the range of each variable into n

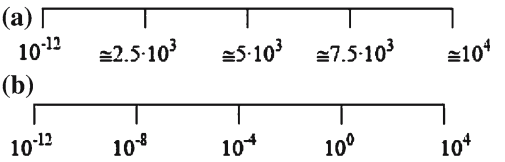

Fig. 2 Intervals within a variable range **(a)** Values uniformly distributed within the bounds **(b)** Values distributed within the different orders of magnitude

sub-ranges of equal size. Then, a solution is constructed in two steps. First, a sub-range is randomly selected. The probability of selecting a sub-range is inversely proportional to its frequency count (which keeps track of the number of times the sub-range has been selected). Second, a value is randomly chosen from the selected sub-range. The starting set of points also includes the following three solutions: the first one in which all variables are set to the lower bound, the second one in which all variables are set to the upper bound, and the third one in which all variables are set to the mid-point between both bounds. This is one of the standard SS implementations of the Diversification Generation Method for non-linear problems. It is implemented in the commercial software OQNLP described in the introduction. However, we have found that in some instances in which variables may have values in a huge range that does not contain zero, a logarithmic distribution usually provides better results.

In the context of chemical and bio-process optimization, the selection of the lower bounds for the decision variables is usually quite straightforward because of their physical meaning (e.g. a temperature can never have a value lower than zero Kelvin). However, the selection of an upper bound is not so easy and such bounds are often chosen as an arbitrarily large value to contain all the potential values for each variable. Therefore, it is expected that good solutions may lie much closer to the lower bounds than to the upper bounds. In this context, a uniform distribution for selecting diverse solutions within the bounds will not generate many trial points with good values. In contrast, a logarithmic distribution will generate more trial vectors very close to the lower bound, thus allowing the algorithm to be initialized with high quality members in the initial population, ensuring a faster convergence. Moreover, a logarithmic distribution is also helpful in the case of variables that can intrinsically have values in "different orders of magnitude" (say, for instance, around 10^{-3}, 10^{-2} or 10^{2} as is the case of pre-exponential factors in kinetic equations) or with variables without physical meaning, for which selecting bounds is a difficult task. In order to obtain good initial values for these cases, an option for selecting variables in different orders of magnitude has been added in our implementation under the name *log_var*.

Figure 2 shows this situation in one of the instances presented in Sect. 4. Consider a variable that takes values between 10^{-12} and 10^{4}. If we generate a starting set of points (say 100) between those bounds using a uniform distribution, we will approximately obtain the same number of values in every interval shown in Fig. 2a. Alternatively, if we select the *log_var* option for this variable, its values will be randomly selected with equal probability across the sub-ranges depicted in Fig. 2b. In this option, the number of subintervals is automatically adjusted so that there are a maximum of two orders of magnitude between the limits of each interval (e.g. for a variable between 10^{-12} and 10^{4}, the number of subintervals would be 8), thus generating more solutions close to zero.

3.2 Reference set update method

As described in Sect. 2, the *RefSet Update Method* is applied in two different steps of the algorithm: when building the initial *RefSet* from the set P of diverse solutions and when updating it with the combined solutions.

3.2.1 Building the RefSet

After generating the set P of diverse solutions, two strategies were investigated to select the first members of the *RefSet.* In the first strategy (used by default), a sub-set of good and diverse points is selected as the reference set. The initial *RefSet* is built selecting the best $b/2$ solutions from P as given by the evaluation–simulation process and then selecting $b/2$ additional solutions by the criterion of maximising the minimum distance between the candidate solution and the solutions currently in *RefSet*. That is, for each candidate solution x in *P-RefSet* and each solution z in *RefSet*, we calculate the Euclidean distance $d(x, z)$ and then select the candidate solution that maximizes $d_{\min}(x)$, where

$$d_{\min}(x) = \min_{z \in RefSet} \{d(x, z)\}.$$

This strategy requires $|P|$ simulations to identify the best $b/2$ solutions in terms of the objective function value. Unless we choose a low value for $|P|$, this can cause a waste of computational effort, especially in the case of the time-consuming problems we are facing in this study. We therefore propose an alternative strategy in which the initial *RefSet* is formed with three solutions: one solution with all the variables in their lower bound, another one having all the variables in their upper bounds and the third one given by the midpoint between the first two. The *RefSet* is completed using the same procedure of maximizing the minimum Euclidean distance between the candidate solutions in P and the current members of the *RefSet*. This second strategy does not involve any simulation prior to the optimization stage. We therefore have no information about the quality of these solutions, so we expect the algorithm to converge more slowly. We may say that the first strategy combines quality and diversity in the initial *RefSet,* whereas the second focuses only on diversity (and saves computational effort).

3.2.2 Updating the RefSet

In its original design, the Reference Set Update Method indicates that the *RefSet* is updated by selecting the best b solutions from the union of the reference set and the new combined solutions. However, we have empirically found that this standard mechanism would result in intermediate reference sets with very similar solutions which are unlikely to produce new good solutions by combination. We have then added a distance filter to prevent similar solutions from becoming part of the *RefSet*. Specifically, we compare each newly created solution (y,x) with the worst solution in the current *RefSet* (y^b, x^b); if it improves its value ($\phi(y, x) < \phi\ (y^b, x^b)$) instead of directly replacing it, as in the original SS implementation, we compute the distance between the new solution and the *RefSet* without the worst solution $d(x, RefSet - x^b)$ as:

$$d(x, RefSet - x^b) = \min_{z \in RefSet - \{x^b\}} \{d(x, z)\}$$

If this distance is larger than a threshold value *dth*, the new solution x replaces the old one x^b in the *RefSet*. On the other hand, if $\phi(y,x) < \phi(y^b,x^b)$ and $d(x, RefSet - x^b) < dth$ we scan the *RefSet* from x^{b-1} to x^1 in search of a solution x^j satisfying $\phi(y,x) < \phi(y^j,x^j)$ and $d(x, RefSet - x^j) > dth$ in order to replace x^j with x. If no x^j verifies it, the solution x does not enter the *RefSet*.

Figure 3 shows a schematic representation of four solutions in the *RefSet* (white circles) and a candidate solution to be included (black circle). Numbers inside each circle represent the objective function value for each solution and the distance between circles represents the Euclidean distance. In a minimization problem, in the original design of the update method, the solution with value 3 would replace the solution with value 7. In our design, with a threshold value $dth = 2$, it replaces the solution with value 4.

We have implemented an aspiration criterion (according to tabu search terminology) for this distance filter. If a solution has the best value found so far but it does not verify the distance filter, we ignore this filter and add it to the *RefSet* replacing its closest solution. The parameter *dth* is initially set to the minimum distance between the solutions in the initial *RefSet*. The distance filter avoids the inclusion in the *RefSet* of solutions that would lead to a reduction of this minimum distance. If in k consecutive iterations (where one iteration is a complete combination and update of the *RefSet*), the best solution has been improved, *dth* is increased by 10%; otherwise, it is reduced by 10% of its value. Note that in the last iterations the *dth*-value is always reduced, permitting the final refinement of the solutions.

A different criterion has been implemented when the distance strategy described above is inefficient. Indeed, Euclidean distances are normalized with respect to the bounds of the decision variables. If these bounds are not correctly chosen (e.g. the variables have no physical meaning or we simply have no information of their range) the elements in the *Refset* can be too similar. To avoid this, a second strategy based on differences in the decision variables is chosen. According to this criterion, two solutions will be different if their relative difference in all variables is equal or greater than a value specified by the user.

We have included a second filter to prevent the method from being trapped in a region for a large number of iterations. In particular, if two solutions are relatively far apart but present very similar objective values, as can happen in functions with flat landscapes, we do not replace the worst one with the slightly better one. The value filter indicates that a solution (y^j,x^j) in the *RefSet* can be replaced with a new

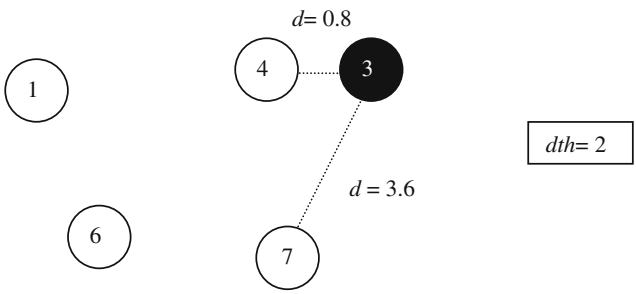

Fig. 3 Four solutions in *RefSet* and a new solution to enter it

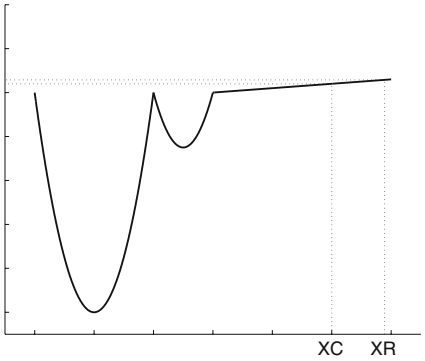

Fig. 4 Two solutions in a flat zone of the objective function

solution (y,x) if $\phi(y,x) < vth\ \phi(y^j,x^j)$, where the *threshold value, vth,* is set in the range [0.75, 1].

Figure 4 shows this situation in a minimization problem of a real function. Consider a solution XR in the *RefSet* and a candidate solution XC to enter it. Suppose that they verify the distance filter according to *dth* and XC has a slightly better value (say around 0.1% lower) than XR. Then, instead of directly replacing XR with XC, the quality filter considers that they may lie in the same flat area of the function, as shown in Fig. 4, and forbids the replacement in order to "wait" for a better solution (thus performing a more aggressive search).

Note that both filters act in a coordinated way since the assumption of flat landscapes is related with value and distance. Consider, for example, a new solution in Fig. 4 close to the origin and with the same value of XC. The situation would be completely different, since it does not belong to the same flat area of XR, as indicated by its distance. On the other hand, SS design specifies that when no new solution is added to the *RefSet*, it is rebuilt with new and diverse solutions. Therefore, if we restrict the incorporation of solutions that contribute only slight quality and diversity to the current *RefSet*, the SS design by itself will make the search more efficient over a long term horizon.

In accordance with the problem's characteristics the user adjusts this filter value, *vth*, for an optimal algorithm performance. The default value for this filter is relatively conservative, but it should be changed in problems in which we want to enhance diversity (for example when there are multiple local minima and the global optimum has a small basin of attraction). When relying on local search, the search may be more aggressive, whereas if no *Improvement Method* is present, it is recommended that the default conservative value is used. In future versions, this filter could be dynamic, being more relaxed at the beginning of the search in order to quickly locate the basin of attraction of the global minimum, and tighter at the end of the search to allow a specified tolerance in the solution. The evolution of this filter is not obvious and it is part of our current research. As is shown in the next section, the algorithm performs very well with a constant value.

3.3 Subset generation and solution combination methods

The *Subset Generation Method* consists of selecting each pair of solutions in *RefSet* and then applying the *Solution Combination Method* to them. If the *RefSet* changes

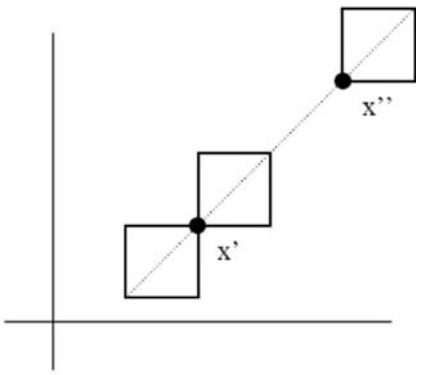

Fig. 5 Generalized combination method

after the application of the reference set update method described above, indicating that at least one new solution has been inserted in the reference set, we again apply the combination method to all the pairs in *RefSet* containing at least one new element. Otherwise, as in advanced SS designs, we resort to the Rebuilding mechanism as described in Sect. 2.

The combination method is a key element in SS implementations. This method is typically adapted to the problem context. Linear combinations of two solutions were suggested by Glover (1977) (see also Glover 1994) in the context of nonlinear optimization and are a generalization of the linear or arithmetical crossover also used in continuous and convex spaces (Michalewicz and Logan 1994). We consider the following three types of linear combinations, where we assume that the reference solutions are x' and x'' (being x' superior in quality than x''), and r is a random number in the range (0, 1):

$$1\colon x = x' - d, \qquad 2\colon x = x' + d, \qquad 3\colon x = x'' + d, \quad \text{where } d = r\frac{x'' - x'}{2}$$

Given two feasible solutions x' and x'', combination 2 always produces a solution in the segment joining these solutions. Combinations 1 and 3 produce nonconvex solutions (i.e. solutions beyond the segment joining x' and x'' but in the same line defined by them).

We also consider a similar implementation to that introduced in *OQNLP* (Ugray et al. 2005) of a generalized linear combination in which instead of producing solutions in the same segment of the reference solutions, it produces solutions in the hyper-rectangles defined over this segment. The user can select between both type of combinations according to the characteristics of the problem at hand. Specifically, we consider the three hyper-rectangles defined by the points 1: $(x' - d, x')$; 2: $(x', x' + d)$ and 3: $(x'', x'' + d)$ as shown in Fig. 5, where

$$d = \frac{x'' - x'}{2}$$

and we assume that the reference solutions are x' and x''(x' being again superior in quality to x''). We then randomly generate solutions within the rectangles, thus obtaining the combined solutions from x' and x''.

As described in Laguna and Martí (2005), depending on the relative positions of x' and x'' in the *RefSet*, different types of solutions will be generated.

- If both x' and x'' are in the first $b/2$ elements of the (sorted) *RefSet*, then we generate solutions of types 1 and 3 once and two solutions of type 2, for a total number of four solutions.
- If only x' is in the first half of the *RefSet*, then one solution of each type is generated, for a total number of three solutions.
- If neither x' nor x'' are in the first half of the *RefSet*, then one solution of type 2 is generated and another one of types 1 or 3 (randomly chosen), for a total number of two solutions.

3.4 Improvement method

The improvement method consists of a local search with the appropriate algorithm, using a carefully selected solution as the starting point. One of the advantages of implementing our optimization method in the Matlab environment is that we can easily apply any improvement method available in one of the many existing libraries. We have considered the following ten methods:

fmincon: A local gradient-based method, implemented as part of the Matlab Optimization Toolbox®, this solver finds a local minimum of a constrained multivariable function by means of a SQP (Sequential Quadratic Programing) algorithm. The method uses numerical or, if available, analytical gradients.

solnp: The SQP method by Ye (1987).

npsol: Developed by the Stanford Systems Optimization Laboratory (see Gill et al. 1998), this is usually considered a state of the art solver for dense nonlinear programming problems.

snopt: Developed by the Stanford Systems Optimization Laboratory (see Gill et al. 2002), this is a state of the art solver for sparse active-set nonlinear programming problems.

Nomadm: Nonlinear Optimization for Mixed variables And Derivatives-Matlab, abbreviated as NOMADm (see Abramson 2002), is a Matlab code that runs various Generalized Pattern Search (GPS) algorithms to solve nonlinear and mixed variable optimization problems. This solver is suitable when local gradient-based solvers do not perform well.

n2fb: This algorithm was specially designed for non-linear least squares problems by Dennis et al. (1981). The method is based on a combined approximation of a Gauss-Newton and quasi-Newton algorithm.

clsSolve: As a part of the Tomlab optimization environment (Holmström and Edvall 2004), this algorithm solves sparse or dense nonlinear least squares optimization problems with explicit handling of linear inequality and equality constraints and simple bounds on the variables.

dhc: The *Dynamic Hill Climbing* algorithm by Yuret (1994) is a direct search algorithm which explores every dimension of the search space using dynamic steps. Only the local phase of the algorithm has been implemented.

fsqp: This algorithm is a SQP method for minimizing smooth objective functions subject to general smooth constraints. The successive iterates generated by the algorithm all satisfy the constraints (Panier and Tits 1993).

ipopt: Interior Point OPTimizer is a software package for large-scale nonlinear optimization. It is designed to find (local) solutions of nonlinear programs (Wächter and Biegler, 2006).

In a classical implementation of SS, the improvement method is applied to a large number of solutions (all the initial solutions in *P* and all the combined solutions from the *RefSet*). However, in applications related to chemical and bio-process engineering, we often face time-consuming evaluation problems (i.e. every function evaluation can consume several minutes). This implies that the application of the improvement method should be restricted to a small number of promising solutions. It is expected that in the first iterations of the search process the solutions generated will be of a relatively poor quality. Therefore, we have implemented a threshold value, *Init_imp* that determines the iteration number in which the improvement method is applied for the first time (i.e. defining a number of previous function evaluations before calling the improvement method). Then, once this threshold is satisfied, a quality and a diversity filter are applied. These filters were successfully applied in Ugray et al. (2005) and they do not allow the Improvement Method to be applied from a solution of a low quality (quality filter), or from a solution close to a solution from which the Improvement Method was applied in previous iterations (diversity filter). As documented by these authors, they significantly reduce the computational time with good results.

Since the Improvement Method is selectively applied, when the SS algorithm is over (i.e. the method reaches the specified number of function evaluations or computational time), before abandoning the search, we apply the Improvement Method to the best solution found so far, just to be sure that it is not skipped, or simply to refine the best solution.

3.5 *RefSet* rebuilding

Rebuilding is a key operation associated with the reference set. It implements a mechanism to partially rebuild the *RefSet* when none of the new trial solutions generated with the Combination Method qualifies for addition to the reference set. In advanced SS designs, the method is usually the same as that used to create the initial *RefSet,* in the sense that it uses the max–min distance criterion for selecting diverse solutions. Typically, it keeps the best $b/2$ solutions in the *RefSet* and selects the other $b/2$ from the same or a new set *P* with the distance criterion.

We have modified the standard implementation of the rebuilding mechanism to incorporate the notion of ortogonality. Over a long-term horizon, the purpose of adding diverse solutions to the *RefSet* is to generate new search directions. It is therefore interesting not only to get scattered solutions in the search space, but also solutions that are able to create new search directions. Then, instead of selecting the solutions in *P* with the max–min distance, we select those with min–max cosine with the solutions already in the *RefSet*. Specifically, we choose the best element in *RefSet* as the *center of gravity* and in the first iteration apply the standard criterion to add the first diverse solution to the *RefSet*. Consider now the vector linking this new solution with the *center of gravity*. In subsequent iterations, instead of considering the solutions in *P,* we consider the vectors that they define with the *center of gravity* and select the solution associated with the vector that minimizes the maximum value of the cosine among the vectors of the solutions already in the *RefSet*.

In the standard SS design, the *RefSet* is rebuilt only when no combined solution qualifies to enter it and there is therefore no new solution to be combined. At this point the method could stop, as in basic SS implementations, or the *RefSet* can be rebuilt and the search continues. However, in our SS design we also apply the rebuilding method in two other cases. If the solutions in the *RefSet* are very similar it is

unlikely that new solutions will result from the combination of these similar solutions and we can save time if we stop the combination method at that point and resort to the *RefSet* rebuilding. Therefore, after each combination step, we compute for each variable the standard deviation of its values in the *RefSet* solutions. If the deviation of all the variables is below the threshold *v_rebuild*, we consider that the *RefSet* is too homogeneous, then stop the combination method and directly apply the rebuilding mechanism. A complementary test is to measure whether the solutions in the *RefSet* are in the same flat region of the objective function. In that case we also stop the combination method and directly apply the rebuilding method as before. Specifically, if the objective values of all the solutions in the *RefSet* are too close (the differences between the values of all the pairs of solutions being lower than *f_rebuild*) we consider that they belong to the same flat region and apply the rebuilding mechanism.

3.6 Intensification

The inclusion of the distance filter in the *RefSet Update Method* (Sect. 3.2.2) could be too restrictive if the parameter *dth* takes relatively large values, (or if the tolerance chosen by the user for when using the other strategy is too high), rejecting too many solutions to become part of the *RefSet*. Instead of keeping this parameter under low values to prevent this effect, we have experimentally found that if we store the rejected solutions with good values in a secondary reference set, *RefSet*2, we can treat them differently from the other solutions in the *RefSet*. *RefSet*2 stores the solutions that do not qualify to enter into the *RefSet* and present a value close to the value of the best solution found (specifically, better than x^2, the second solution in the *RefSet*) with a maximum of 25 solutions. During the *Solution Combination Method*, we combine the best solution in the *RefSet* with all the solutions in *RefSet*2 and check if any of the resulting solutions improve the best solution in the *RefSet*. In that case, the new solution replaces the best one in the *RefSet*; otherwise it is discarded. The solutions in *RefSet*2 are then deleted. This intensification mechanism is performed every *Inten_freq* steps where one step is a complete application of the standard combination method in the *RefSet*.

Figure 6 shows those combinations in an example with six solutions in the *RefSet* (black circles) and three solutions in *RefSet*2 (stars). The square represents the global optimum. In this example this intensification strategy makes the process converge faster since the combination of the starred solutions with the best in *RefSet* generates solutions which are very close to the global optimum of the function.

4 Computational experiments

In order to evaluate the performance of our algorithm, we have applied it to solve three well-known and hard optimization problems from the chemical and bio-process engineering area. The chemical isomerization of alpha-pinene (Box et al. 1973; Dolan et al. 2004), the design of a waste water treatment plant (Moles et al. 2003a) and a parameter estimation problem in biochemical pathways (Moles et al. 2003b). These three problems involve dynamic systems (i.e. simulations by the integration of systems of differential-algebraic equations). We have compared the results of our SS method implemented in Matlab, *SSm*, with those provided by the five global optimization algorithms identified as the best and summarized in the introduction: *DE*, *Global, SRES*,

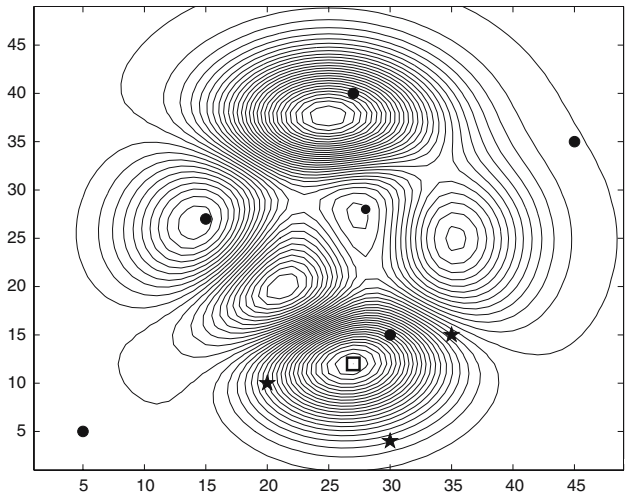

Fig. 6 Intensification strategy

Direct, and OQNLP. The last three can handle general nonlinear problems, including black-box simulations for the objective function and/or equality constraints. On the other hand, *DE* and *Global* were designed to handle bound-constrained nonlinear problems, but we have extended them for general constraints by means of penalty function approaches in order to report a complete comparison.

All the experiments were carried out on a PC Pentium-IV 3.06 GHz using Matlab 6.5, Release 13, under Windows XP Pro. We implemented *SSm* as described in Sect. 3 with the following standard parameter values: $PSize = 100$, $b = 10$, $b_1 = b_2 = 5$, $vth = 0.9999$, $Init\_imp = 100nvar$, $f\_rebuild = 10^{-5}$, $v\_rebuild = 10^{-5}$, $Inten\_freq = 20$. The diverse criterium was based in Euclidean distances, *dth*. Finally, the weight to penalize unfeasible solutions, *wpen*, was set to 1,000.

In our first experiment we compare the six methods under consideration when solving the isomerization of alpha-pinene. This is a parameter estimation problem that arises from the modeling of the chemical phenomena of an isomerization reaction. The problem was described in Box et al. (1973) and it is part of the COPS benchmarking collection maintained by Dolan et al. (2004). The dynamic model is defined by the following system of five ordinary differential equations in which y' represents the time derivative of state variable y.

$$\begin{aligned}
y_1' &= -(\theta_1 + \theta_2)y_1,\\
y_2' &= \theta_1 y_1,\\
y_3' &= \theta_2 y_1 - (\theta_3 + \theta_4)y_3 + \theta_5 y_5,\\
y_4' &= \theta_3 y_3,\\
y_5' &= \theta_4 y_3 - \theta_5 y_5.
\end{aligned}$$

The problem consists of finding the reaction coefficients, θ_i in [0,1], to minimize the differences between the values of the states obtained from experimental data and the values, y, predicted by the model. The best known solution is $\theta^* = (5.9256 \times 10^{-5}, 2.9632 \times 10^{-5}, 2.0450 \times 10^{-5}, 2.7473 \times 10^{-4}, 4.0073 \times 10^{-5})$ with value 1.9872×10^{1}.

Table 1 Comparison on the alpha-pinene problem

Solver	Best value	Worst value	Average value	CPU time (seconds)	Average No. evaluations
SS*m*	1.9872×10^1	6.8617×10^1	2.4747×10^1	41	1,163
DE	3.1951×10^4	3.2945×10^4	3.2051×10^4	46	1,250
SRES	3.3858×10^4	4.2707×10^4	3.8398×10^4	47	1,300
Global	3.1638×10^4	4.2755×10^4	3.5225×10^4	45	1,277
OQNLP	3.1252×10^4	3.1252×10^4	3.1252×10^4	51	1,565
Direct	3.6421×10^4	3.6421×10^4	3.6421×10^4	45	1,053

The improvement method applied in *SSm* for this problem is *fmincon* and all decision variables are declared as *log_var* (it is frequent in parameter estimation problems for the solution to lie very close to the bounds, and in these cases the *log_var* strategy performs remarkably well).

In order to obtain statistically significant results, we run each method ten times on this problem and report the average (Average Value), best and worst solution values of the ten runs (each run is limited to 50 s). Table 1 reports these three values as well as the average running time in seconds and the average number of function evaluations of each method. *Direct* and OQNLP only allow one run since *Direct* is a deterministic method and we are using the OQNLP implementation for Matlab, which does not permit the modification of the random generator seed. Therefore, the value of these methods must be compared with the average value of the other methods.

Table 1 shows that the best solution quality is obtained with the SS*m* method in the lowest computational time. Moreover, considering the average values over the ten runs, it also shows that SS*m* is robust, since it is able to obtain the best solutions on average. Our proposal also presents a moderate number of function evaluations when compared with the other solvers. None of the other methods obtains solutions of good quality in this problem for the computation time horizon considered.

Complementary information to compare the methods is given in Fig. 7. It depicts the convergence curves of the best three methods for this problem: DE, *Global*, and SS*m*. This experiment has the goal of showing how the value of the best solution found improves over time. The three procedures were run for 50 s and the best solution found was reported every second (approx.) As shown in Table 1, the best solution for SS*m* has, in fact, a value of 19.872; however, it is not depicted in Fig. 7 for scaling purposes.

As shown in Table 1 and Fig. 7, SSm performs significantly better than the rest of the solvers in the short time horizon considered (50 s). Actually, the best-known solution was obtained by SS*m* in nine out of the ten runs carried out. The other solvers failed to reach points close to this solution, although several of them could ultimately succeed it would be at a very large computational cost.

In our second experiment we consider another typical application of global optimization in process engineering. Specifically, we target an extremely hard problem that arises from the integrated design and control of complex processes. The objective is to find the static variables of the process design, the operating conditions and the controllers' parameters which optimize a combined measure of the plant economics and its controllability, subject to a set of constraints which ensure appropriate dynamic behaviour and process specifications. The particular case study considered here was presented by Moles et al. (2003a), and it has eight decision variables, 185

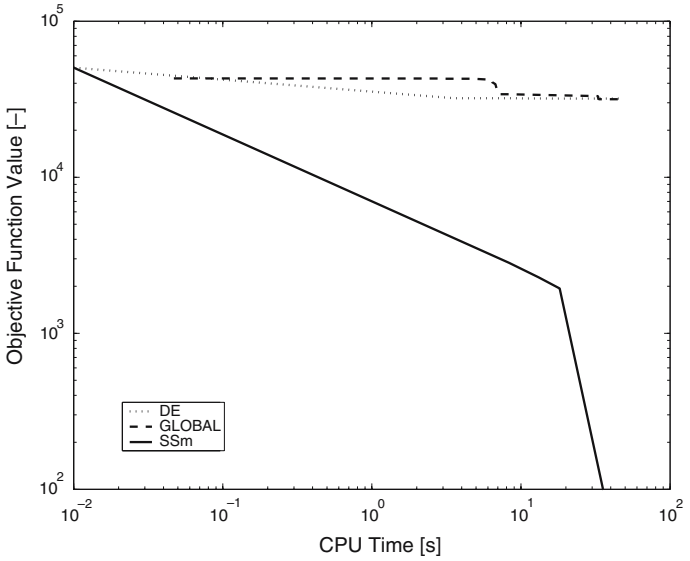

Fig. 7 Convergence curves for the alpha-pinene isomerization problem

constraints (33 differential-algebraic equality constraints and 152 nonlinear inequality constraints) and the value of the best known solution is 1.5379×10^3.

The improvement method was deactivated because it consumes excessive running time without significant solution improvement. However, a final refinement phase was activated using the solver *Nomadm*. The reason for these two special settings is the presence of discontinuities in the problem, which makes gradient-based algorithms fail or converge prematurely. The intensification phase was applied every ten iterations.

As in the first experiment, we run each method ten times on this problem and report the average (Average Value), best and worst solution values of the ten runs (each run was given an allowed time horizon of 360 s approx.) as well as the average running time in seconds and the average number of function evaluations of each method. Since our preliminary experimentation indicates that improved solutions are obtained when the local search is not applied, we also report the results of OQNLP with no local search as OQNLP* (this solver implements this option).

Table 2 shows that the best solutions are obtained with the DE and SS*m* methods. Both are able to match the best-known solution. Although SS*m* presents a slightly longer average computational time than DE(359 vs. 301 s), the former presents a lower number of function evaluations (13,219) than the latter (16,782). Moreover, comparing the best and worst values across the ten runs, SS*m* presents a very low dispersion. The other solvers under consideration are also able to obtain good solutions which are close to the best-known.

Figure 8 depicts the convergence curves of the best three methods for this problem. Specifically, it shows the best curve (over the ten runs) of the SS*m*, DE, and SRES methods. The curves in this figure show that the three algorithms present similar convergence rates. In the first fractions of one second, the SRES obtains the best

Table 2 Comparison on a design plant problem

Solver	Best value	Worst value	Avgerage value	CPU time (seconds)	Avgerage No. evaluations
SS*m*	1.5378×10^3	1.5431×10^3	1.5391×10^3	359	13,219
DE	1.5379×10^3	1.6418×10^3	1.5483×10^3	301	16,782
SRES	1.5381×10^3	1.5392×10^3	1.5385×10^3	352	13,170
Global	1.5697×10^3	1.6731×10^3	1.6049×10^3	334	14,998
OQNLP	2.4839×10^3	2.4839×10^3	2.4839×10^3	332	10,927
OQNLP*	1.7424×10^3	1.7424×10^3	1.7424×10^3	480	12,306
Direct	2.3387×10^3	2.3387×10^3	2.3387×10^3	308	18,015

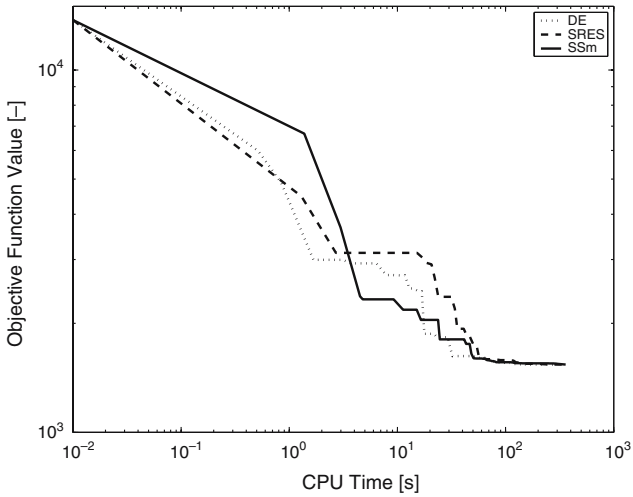

Fig. 8 Convergence curves for the integrated design problem

solutions; however, after 5 s SS*m* slightly improves on the other two methods and after 1 min the three of them obtain the same solutions in terms of quality.

In our final experiment we target the Parameter Estimation in Biochemical Pathways problem. This is a challenging parameter estimation problem introduced as a benchmark by Moles et al. (2003b). This problem has a large number of local minima, and a very small basin of attraction for the global solution, which makes it very difficult to solve in a reduced computational time. As a reference, Moles et al. (2003b) reported computation times of about 40 hours using Evolution Strategies on a PC Pentium-III/866 MHz. The problem consists of a system of eight ordinary differential equations and 36 parameters to be estimated from a set of pseudo-experimental data. Since this is a synthetic problem, its global solution is known and has a value of 0.0.

After preliminary experimentation with the SS*m* method, the following options and parameters were set to enhance diversification over intensification in the global search for an aggressive search of the solution space.

- Filters for solutions to join the *RefSet* are tightened. Only solutions that significantly improve the existing ones in the *RefSet* are allowed to join it. The function tolerance for joining the *RefSet* was 10^{-2} (i.e. *vth* = 0.99).

Table 3 Comparison on a parameter estimation problem

Solver	Best value	Worst value	Average value	CPU time (seconds)	Average No. evaluations
SS*m*	1.5821×10^{-7}	1.3889×10^{0}	1.5910×10^{-1}	325	20,586
DE	2.7429×10^{2}	4.3230×10^{2}	3.7853×10^{2}	354	18,756
SRES	2.1099×10^{2}	3.9095×10^{2}	2.5875×10^{2}	350	20,580
Global	7.4123×10^{2}	8.4587×10^{2}	7.8100×10^{2}	143	9,012
OQNLP	7.8145×10^{1}	7.8145×10^{1}	7.8145×10^{1}	416	20,718
Direct	1.1368×10^{3}	1.1368×10^{3}	1.1368×10^{3}	351	30,547

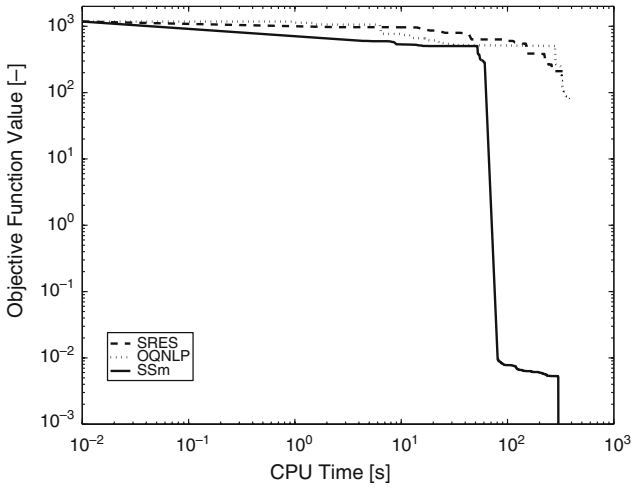

Fig. 9 Convergence curves for the parameter estimation problem

- The distance filter of local search was deactivated to perform local searches from close points, due to the small basin of attraction of the global solution.
- The local search algorithm used was *n2fb* (in double precision in the final refinement) as it is more efficient than SQP methods for challenging parameter estimation problems like this one.
- The Euclidean distance, *dth*, turned out to be inefficient for this problem since bounds for the parameters where chosen in a huge range due to the unknown nature of them. The diversity strategy based on relative tolerances was chosen. This tolerance was set to 10^{-2}.
- Unlike in the other examples, the linear combinations proved to be much more efficient for this problem than the generalized combinations based on hyper-rectangles. The reason for this is that in the global solution all the decision variables lie very close to the bounds.
- Intensification phase was applied every ten iterations.

The results obtained with the different global solvers, for a computation time horizon of 350 s, are presented in Table 3, and the best convergence curves are shown in Fig. 9.

Table 3 clearly shows the superiority in terms of solution quality of our proposal SS*m*. It is able to obtain significantly better solutions (the best solution value is

1.5821×10^{-7}) than the other six approaches (with best values from 7.8145×10^{1} to 1.1368×10^{3}) over similar running times (325 s on average). Moreover, the SS*m* is quite robust with an average value of 0.1591, closely followed by OQNLP with an average value of 7.8145.

Figure 9 shows that SS*m* clearly improves on the other two best methods for this problem. When the optimization process starts, the SS*m* procedure quickly moves to the range of high-quality solutions and maintains its lead during the rest of the solution time (this is especially true after the first minute of computation time). Within the running time horizon of 350 s, SS*m* found the global solution in eight out of ten runs, whereas the rest of the solvers failed in all the runs. It is particularly important to highlight that SS*m* decreased the computational time reported by Moles et al. (2003b) by more than two orders of magnitude, which is a very relevant result, and the best-known heuristic solution for this problem to date.

5 Conclusions

The objective of our study has been to expand and advance knowledge associated with the implementation of SS procedures. Unlike other population based methods such as the well-known genetic algorithms, scatter search has not yet been extensively studied. Specifically, we have introduced new strategies in the five key methods of SS procedure: a diversification generation method based on magnitudes, a selective reference set update method based on filters, a combination method that discriminates among reference solutions and a restrictive application of the improvement method.

In particular, we have considered a SS implementation for nonlinear continuous optimization of black-box models. We can find extremely difficult instances within this class of problems and we have compared our proposal with five well-known methods when solving three benchmark instances. The comparison favors our SS implementation.

Acknowledgements The team at CSIC acknowledges financial support from the Spanish Government (MEC AGL2004-05206-C02-01/ALI) and Xunta de Galicia (PGIDIT05PXIC40201PN). Research by Rafael Martí and Jose A. Egea is partially supported by the *Ministerio de Educación y Ciencia* (TIN2006-02696 and FPU fellowship).

References

1. Abramson, M.A.: Pattern search algorithms for mixed variable general constrained optimization problems. PhD Thesis, Rice University (2002)
2. Bailey, J.E.: Mathematical modeling and analysis in biochemical engineering: past accomplishments and future opportunities. Biotechnol. Progr. **14**, 8–20 (1998)
3. Banga, J.R., Balsa-Canto, E., Moles, C.G., Alonso, A.A.: Improving food processing using modern optimization methods. Trends Food Sci Technol. **14**, 131–144 (2003a)
4. Banga, J.R., Moles, C.G., Alonso, A.A.: Global optimization of bioprocesses using stochastic and hybrid methods. In: Floudas, C.A., Pardalos, P.M. (eds.) Frontiers In Global Optimization. Nonconvex Optimization and Its Applications, vol. 74, pp. 45–70. Kluwer Academic Publishers, Hingham, MA, USA (2003b)
5. Biegler, L.T., Grossmann, I.E.: Retrospective on optimization. Comput. Chem. Eng. **28**(8), 1169–1192 (2004)

6. Boender, C.G.E., Rinooy Kan, A.H.G., Timmer, G.T., Stougie, L.: A stochastic method for global optimization. Math. Program. **22**, 125–140 (1982)
7. Box, G.E.P., Hunter, W.G., MacGregor, J.F., Erjavec, J.: Some problems associated with the analysis of multiresponse data. Technometrics. **15**, 33–51 (1973)
8. Csendes, T.: Nonlinear parameter estimation by global optimization—efficiency and reliability. Acta Cybernet. **8**(4), 361–370 (1988)
9. Dennis, J.E., Gay, D.M., Welsch, R.E.: Algorithm 573: NL2SOL—An adaptive nonlinear least-squares algorithm. Acm. Trans. Math. Soft. **7**, 369–383 (1981)
10. Dolan, E.D., Moré, J.J., Munson, T.S.: Benchmarking optimization problems with COPS 3.0. Technical Report ANL/MCS-TM-273, Argonne National Laboratory (2004)
11. Finkel, D.E., Kelley, C.T.: An Adaptive Restart Implementation of DIRECT. Technical Report CRSC-TR04–30, NC State University (2004)
12. Floudas, C.A., Akrotirianakis, I.G., Caratzoulas, S., Meyer, C.A., Kallrath, J.: Global optimization in the 21st century: advances and challenges. Comput. Chem. Eng. **29**(6), 1185–1202 (2005)
13. Gill, P.E., Murray, W., Saunders, M.A., Wight, M.H.: User's guide for npsol 5.0: A FORTRAN package for nonlinear programming. Technical Report SOL 86–1, Systems Optimization Laboratory, Stanford University (1998)
14. Gill, P.E., Murray, W., Saunders, M.A.: SNOPT: An SQP algorithm for large-scale constrained optimization. SIAM J. Optim **12**(4), 979–1006 (2002)
15. Glover, F.: Heuristics for integer programming using surrogate constraints. Decis. Sci. **8**, 156–166 (1977)
16. Glover, F.: Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). Discrete Appl. Math. **49**, 231–255 (1994)
17. Glover, F.: A template for scatter search and path relinking. In: Hao, J.K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (eds.) Artificial Evolution, Lecture Notes in Computer Science, vol. 1363, pp. 13–54. Springer Verlag, Berlin (1998)
18. Holmström, K., Edvall, M.M.: The Tomlab optimization environment. In: Kallrath, J., Basf, A.B. (eds.) Modeling Languages in Mathematical Optimization, pp. 369–378. Kluwer Academic Publishers, Dordrecht (2004)
19. Jones, D.R.: DIRECT global optimization algorithm. In: Floudas, C.A., Pardalos, P.M. (eds.) Encyclopedia of Optimization, pp. 431–440. Kluwer Academic Publishers, Dordrecht (2001)
20. Laguna, M., Martí, R.: The OptQuest callable library. In: Voss, S., Woodruff, D. (eds.) Optimization Software Class Libraries, pp. 193–218. Kluwer Academic Publishers, Boston (2002)
21. Laguna, M., Martí, R.: Scatter Search: Methodology and Implementations in C. Kluwer Academic Publishers, Boston (2003)
22. Laguna, M., Martí, R.: Experimental testing of advanced scatter search designs for global optimization of multimodal functions. J. Global Optim. **33**, 235–255 (2005)
23. Mathworks, Matlab 6.5R13, Mathworks, Natick, MA (2004)
24. Michalewicz, Z., Logan, T.D.: Evolutionary operators for continuous convex parameter spaces. In: Sebald, A.V., Fogel, L.J. (eds.) Proceedings of the 3rd Annual Conference on Evolutionary Programming, pp. 84–97. World Scientific Publishing, River Edge (1994)
25. Moles, C.G., Gutierrez, G., Alonso, A.A., Banga, J.R.: Integrated process design and control via global optimization: a wastewater treatment plant case study. Chem. Eng. Res. Des. **81**(5), 507–517 (2003a)
26. Moles, C.G., Mendes, P., Banga, J.R.: Parameter estimation in biochemical pathways: a comparison of global optimization methods. Genome Res. **13**(11), 2467–2474 (2003b)
27. Panier, E., Tits, A.L.: On combining feasibility, descent and superlinear convergence in inequality constrained optimization. Math. Program. **59**, 261–276 (1993)
28. Runarsson, T.P., Yao, X.: Stochastic ranking for constrained evolutionary optimization. IEEE Trans. Evol. Comput. **4**, 284–294 (2000)
29. Runarsson, T. P., Yao, X.: Search biases in constrained evolutionary optimization. IEEE Trans. Syst. Man Cybern. **35**, 233–243 (2005)
30. Shimizu, K.: A tutorial review on bioprocess systems engineering. Comput. Chem. Eng. **20**, 915–941 (1996)
31. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**, 341–359 (1997)
32. Ugray, Z., Lasdon, L., Plummer, J., Glover, F., Kelly, J., Martí, R.: A multistart scatter search heuristic for smooth NLP and MINLP problems. In: Rego, C., Alidaee, B. (eds.) Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search. pp 25–58. Kluwer Academic Publishers, Dordrecht (2005)

33. Wächter, A., Biegler, L.T.: On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006)
34. Ye, Y.: Interior Algorithms for Linear, Quadratic and Linearly Constrained Non-linear programing. Phd Thesis, Stanford University (1987)
35. Yuret, D.: From Genetic Algorithms to Efficient Optimization. A.I. Technical Report No. 1569. Massachusetts Insitute of Technology (1994)