ars combination with an evolutionary algorithm for solving minlp optimization problems

gheorghe maria

department of chemical engineering, university politehnica bucharest, romania,
p.o. 15-253 bucharest, email: gmaria99m@hotmail.com, gmaria99m@netscape.net

abstract

adaptive random searches (ARS) are simple and effective optimization methods used for handling complicated non-convex / multimodal nonlinear programming (NLP) and mixed-integer nonlinear programming (MINLP) problems. ARS iteratively adapt search characteristics according to the past successful / failure steps. Periodic search domain expansions and contractions improve significantly the reliability in locating the global optimum. However, most of the ARS parameters are apriori set, and then the algorithm cannot be used at their maximum effectiveness. The present paper proposes a combination of ARS with parallel search in competitive members (‘families’) and an evolutive algorithm (EA) to automatically adapt ARS parameters and search characteristics. The analysis is applied to the MMA-ARS of Maria [1,2] adapted in the form of MMAMI for handling MINLP problems, and then coupled as MMAMI-EA rule. The effectiveness, expressed as computational effort and reliability in locating the global solution, is checked for comparison with genetic algorithms (GA), simulated annealing (SA), ARS, and EA reported results in solving six MINLP test problems. While MMAMI reports a significant decrease of computational effort comparatively with ES, GA, and SA, the combination MMAMI-EA considerably improves the search reliability.

key words

adaptive random search, evolutionary algorithm, minlp

1. introduction

solving a multimodal / nonconvex MINLP problem is a difficult task. However, an important number of identifications, optimizations and process analyses in basic sciences and engineering are formulated in such mathematical terms. This is the case, for instance, of identifying redundant terms of a model [3], of setting discrete and continuous decision variables in process synthesis and design, or of scheduling batch processes [4,5]. MINLP problems can include complex technological or physical nonlinear constraints, leading to a non-convex search domain with multiple local optima. The general objective of a MINLP is to minimize a nonlinear function:

$$\text{Min} \; f(x,y),$$

subjected to constraints:

$$g_j(x,y) \geq 0, \; j = 1,2,...,m; \; \alpha_i \leq x_i \leq \beta_i, \; i = 1,2,...,p;$$

$$\gamma_l \leq y_i \leq \xi_l, \; i = p+1,...,n,$$

where $x$ is a vector of continuous variables, $y$ is a vector of integer variables, $(\alpha_i, \beta_i)$ are bounds on parameter $x_i$, and $(\gamma_l, \xi_l)$ are bounds on parameter $y_i$, e.g. $[0,1]$ for binary variables. Attempt to solve MINLP problems with a NLP method, with considering only continuous variables and then rounding the estimate to the nearest integer, must be avoided because of non-feasible or local solutions [6]. To solve the MINLP problem, gradient or gradientless optimization algorithms were developed. Gradient methods (using function’s derivatives) solve a series of sub-problems obtained from an appropriate decomposition of the original MINLP problem. The algorithms, based on the generalized Bender’s decomposition [7] or the outer approximation [8], try to identify the sources of non-convexity. The original problem is decomposed in a ‘primal’ (the original NLP problem solved for a fixed set of integer variables), and the ‘master’ (a mixed-integer-linear problem solved for continuous variables to provide new integer variable values). Partitioning is made such that the optimum $y$ can be determined independently of $x$. Since a number of constraints must be evaluated prior to the solution, the master problem is solved as a series of relaxed sub-problems. To find the global solution, Kocis & Grossmann [9] localize first the non-convexities by local/global tests and then a relaxation is imposed to the invalid solutions. Floudas et al. [4] suggest partitioning of the original problem to ensure that the global solutions to the primal and master sub-problems are attained for all iterations. By partitioning the variables as responsible or not for non-convexities, the master and the primal problems are iterated until no improvement can be obtained. Grossmann & Sargent [10] use a branch-and-bound procedure by solving consecutive NLP sub-problems with appropriate branching criteria. In any variant, the gradient methods need to identify and
eliminate the sources of non-convexities by analyzing the objective function / constraints, by performing transformations, or by solving a perturbed NLP problem around the local solution. However, gradient methods encounter difficulties in poor-conditioned MINLP problems due to the necessity to accurately evaluate the function (constraint) first and second order derivatives. Today, with the growing availability of powerful computational means, the optimization routine efficiency is expressed more and more on their robustness to reach the global solution rather than the computational cost (usually expressed as \( n_f \) = number of objective function evaluations). Some other routine characteristics as simplicity, easy-to-use, amount of complementary calculations, independence on the initial variable guess can be decisive in choosing the right optimization algorithm.

ARS are a special class of gradientless optimization methods, reported as presenting exceptional capabilities in handling multimodal problems comparatively with gradient methods [11]. However, because some of the ARS parameters are kept constant during optimization, the algorithm still presents improvement potential. EA apply a competitive / evolutive search strategy with promising results in solving NLP and MINLP problems. Combinations of random searches usually reported considerably search improvements.

This paper is dealing with presenting two simple adaptations of an effective ARS, i.e. the MMA of Maria [1,2], for efficiently handling non-convex MINLP problems: i) the MMAMI algorithm, which preserves the MMA search policy for continuous variables while the integer variables span their feasible range; ii) the MMAMI-EA, which combines a MMAMI parallel search in competitive members (‘families’) with an EA for modifying the MMAMI search characteristics according to the competition results among members. The proposed algorithm’s effectiveness is checked both as computational effort and solution reliability in solving six MINLP test problems selected from literature. The results are discussed and compared with reported performances of some effective SA, GA, ARS, and EA.

2. RANDOM SEARCH ALGORITHMS

The gradientless random searches (RS) can by-pass most of gradient method difficulties, being simple and reliable in solving complicated non-convex / multimodal NLP and MINLP optimization problems even if their convergence is relatively slow. RS can be grouped in several classes: ARS, SA, GA, EA, and clustering algorithms (CA). ARS iteratively adapt the search (step-length, direction, checked domain, random point generator distribution) based on information on past search failure or success. ARS include several sub-classes [2]: ARS with centroid generation, Luus-Jaakola’s class, adaptive step size ARS, or combinations of these. Search strategy is usually completed with a periodic expansion and contraction of the search domain in order to refine the solution and to overcome local optima (for instance MMA [1], ARDS [14], ICRS [15], SGA [16], Luus [17]). Combinations of ARS with other RS improved efficiency in solving MINLP (for instance centroid ARS coupled with SA [12,13]).

SA methods are based on the Markov chain theory, accounting only the last step information in directing the search. In spite of their very low convergence rate, the reliability in solving complex global optimisation problems is very high. Apart from other methods, SA can accept a detrimental search step with a Boltzmann distribution probability, thus surpassing local optima [18-20]. However, the required computational effort is much higher \( [n_f \sim 10^4-10^5 n] \) comparatively with an effective ARS \( [n_f \sim 10^2-10^3 n] \). SA have been applied for solving a large variety of optimization problems: traveling salesman problem, process design, construction of evolutionary trees, molecular conformation analysis, process identification, batch process scheduling.

GA’s random point generation presents similarities with biogenetic mutation and natural selection [21]. Starting from a current point, one GA iteration implies the following steps: i) uniformly random generate a certain number (‘population size’) of feasible points (‘individuals’); ii) ‘individuals’ are randomly divided in several subsets (‘parents’); iii) from the ‘parental’ subsets, two ‘individuals’ are selected by means of a random or adaptive (proportional, ranking, etc.) rule; iv) ‘parental’ vectors are rewritten in a certain code (for instance binary) becoming a ‘chromosome’ of a certain length; v) a new ‘individual’ is created by crossover two ‘parental chromosomes’; vi) the ‘offspring’ suffers several random ‘mutations’ of parts of them, with a certain frequency, improving the global search; vii) finally, the mutant ‘individual’ is decoded and used to check the search progress (via objective function \( f \) evaluation). Steps iii-vii are repeated a certain number of times (‘number of generations’) and the best iterative point is retained. Recent GA’s improvements, by using orthogonal crossover, effective ‘crowding’ operators, and combinations with SA, avoid common GA defect of early convergence and increase the global search reliability. Hanagandi & Nikolaou [22] developed effective GA combinations with CA for solving non-convex MINLP problems. Among GA applications it is to mention: process identification, molecular design, multi-product batch process design, large-size scheduling problems, pump’s optimal configuration design.

EA mimic the evolution of the species in natural systems [23]. Like GA, EA random generate the ‘population’ of tried points based on a ‘mutation’ operator, and use only objective function and constraints in ranking and eliminating tried points. The selection procedure subsequent to a generation step can be applied in two
ways. \((\mu + \lambda)\) EA generates from \(\mu\)-parents, by mutation, \(\lambda\)-offsprings and then, from the \(\mu + \lambda\) sorted members the best \(\mu\) become the parents of the next generation. \((\mu, \lambda)\) EA generates from \(\mu\) parents, by mutation, \(\lambda > \mu\) offsprings and then, from the \(\lambda\) sorted members the \(\mu\) best of them become the future parents. EA can include parallel search for several members (‘families’) and following a certain competition policy [24]. Costa & Oliveira [25] reported effective EA for solving MINLP problems. Pham [26] proposed a general competitive EA, in which search results from several ‘populations’ (using different search strategies) allow to set the procedure operators & parameters to be used in the next step. Both ‘best’ and ‘stalled’ populations are allowed to evolve but in a different manner. The analysis is exemplified for a combined EA and GA. Wong & Wong [27] studied an evolutive hybrid of GA and SA.

There are several similarities among ARS, GA and EA to be mentioned. For instance, GA’s mutation frequency corresponds to EA’s offspring mutation frequency and to ARS’s control of local / global convergence; ARS periodic domain expansion and contraction is similar to the increasing diversity of the GA population by using crowding scheme or dissolving ‘niche/clusters’ [22]; competition-cooperation among ‘families’ in EA is equivalent with avoiding the ‘elitism’ induced by the ‘fitness’ function in GA [28], with the continuous switching between local and global search in some ARS, or with the multi-start local searches in clustering algorithms (CA).

3. MMAMI: A MMA MODIFICATION FOR SOLVING MINLP PROBLEMS

The MMA-ARS of Maria [1,2] increases the convergence rate and effectiveness in handling multimodal NLP problems by using two search strategies: a local search (‘pseudo-one-dimensional branch’) and a global search (‘pseudo-multidimensional branch’). Schematically, a MMA step consists in following one of the two branches (Table 1), each involving a certain number of iterations which contract the search domain around the current optimum. The recursive relationships in the two branches are similar and used to calculate the tried points and to store the search history in the vectors \(b\) and \(t\). However, the branches act differently: one performs successive pseudo-unidirectional searches, while the other considers all directions simultaneously. After ca. 100 unsuccessful trials to generate a feasible point, another direction is considered. The switch between branches is automatically done by means of a branch effectiveness test (Table 1).

MMA presents only four adjustable parameters: \([N_{uni},\ N_{mult},\ P_{mult},\ \alpha_f]\) (the others parameters being set to their optimal values). These parameters control the local and global search weights, and domain contraction rate respectively. Maria [2] suggested suitable MMA parameter’s values according to the problem complexity. For solving MINLP problems a modification, suggested by Salcedo et al. [6], is made to the basic MMA, thus resulting the MMAMI algorithm. The idea is to add new algorithm relationships to generate and test integer points \(y^{j+1}\) in defined ranges. During optimization, search domain for continuous variables contracts or expands following the MMA rule, while search ranges for integer variables remain invariant. In a \(j\)-th MMAMI iteration, the integer test points are generated with the formula:

\[
y^{j+1} = y + \text{INT}\left(\left(\frac{1}{2}\right)^{\text{iter}} + \frac{1}{2}n-p\right)M^j + \nu^T ; \quad M^j = \xi - y ; \quad v^T = \left[\frac{1}{2}, \ldots, \frac{1}{2}n-p\right],
\]

where \(U\) is a square-diagonal \((n-p)\) matrix consisting of random numbers in the interval \([0,1]\), \(v^T\) is a constant \((n-p)\) vector, and \(\text{INT}\) denotes the integer operator, viz., returning the largest integer less or equal to the operand. Such a strategy provides an opportunity for the integer variables to span their range of possible values, irrespectively to the evolution of search domain for continuous variables.

4. MMAMI-EA ADAPTIVE-EVOLUTIVE ALGORITHM

The previously adapted MMAMI to solve MINLP problems can be even improved in terms of search reliability in locating a global solution if the procedure parameters are varied during the search. The following MMAMI-EA is obtained by coupling MMAMI with an \((\mu+\lambda)\) EA. The basic modifications are the following.

i) Parallel search is conducted in \(N_{fam}\) independent members (or ‘families’).

ii) Each ‘family’ follows a complete MMAMI search cycle until a stop criterion is fulfilled (Table 1).

iii) In a MMAMI branch iteration, from one parent \(x^j\) several \(\lambda\)-offsprings are generated \((\lambda \leq n\) or \(\lambda \leq P_{mult}\)). This corresponds to a \((I+\lambda)\) EA with the ‘mutation rule’ inside a family controlled by the MMAMI.

iv) A search cycle is finish when all ‘families’ fulfill MMAMI stop criteria.

v) A competitive EA allows adapting the MMA parameters for each ‘family’ according to obtained results after a search cycle.

There are two competition levels in the MMAMI-EA. One is the survival competition inside a ‘family’, controlled by MMAMI-iterating rule, which choose the best individual from \((I+\lambda)\) candidates. The second is competition among families after a MMAMI complete cycle, when results obtained by every family with different MMAMI search strategies are compared and used to rank the members. After a MMAMI-EA cycle, families are line-up ranked as
following: the ‘best family’ (presenting the best search objective function $f$) is placed on the top position while the ‘worst family’ is placed on the bottom position. The philosophy of this hierarchy is to confer to the families different mutation characteristics in the next search cycle: i) as a family is better ranked, as fewer mutations will be performed, and the search will be focused on refining the local optimum; ii) as a family is below ranked, as larger mutations will be done, allowing a global search on the whole space. In fact, this competition-cooperation strategy ensures for the bottom placed families a higher chance to surpass local optima. As long as a competition driving force exists (i.e. differences in realized 'families'), the rule continues over the entire search cycle, would be possible to be moved to the rear and then another search policy will be set ad-hoc chosen as being equal with the family index. After one complete search cycle, family's obtained results are analyzed, compared and classified. For the next search cycle, each 'family' will receive a MMAMI parameter set from Table 2 according to their relative position in the 'family' hierarchy: the bottom ranked will receive the set #1, the top ranked the set #6, and the others will receive sets according to their relative position. As a stop criterion for one cycle, one recommends $n_{f,max} = 100n$ (per family). If $n_{f,max}$ will be set too small, the local search refinement will not be possible.

5. TEST PROBLEMS AND RESULTS

To check the proposed MMAMI and MMAMI-EA, six MINLP test problems have been selected from literature (Table 3). For each problem ten random starting points have been used to check the algorithm reliability in locating the global solution. Reliability was expressed as $C\%$ percentage of successful convergences to global optimum over all trials. Obtained results are compared in Table 3 with those reported when using SA (M-SIMPSA and M-SIMPS-penalised [12,13]), GA (GA-R and GA-D [25]), ARS (MSGA [6]; CRS [29]), and EA (variant [25]). The CRS-MI algorithm is the CRS code modified for solving MINLP problems in the same way as for MMAMI. The computational effort is expressed as required $n_f$ to realize a $f_{sol} / f_{initial} \geq 0.1\%$ improvement.

It is to remark that GA and SA reported for these tests 1-4 orders of magnitude higher required $n_f$ than MMAMI, for a comparable reliability. Concerning the adapted ARS (i.e. MMA and CRS), their effectiveness in handling MINLP problems seems to also depend on the original algorithm effectiveness in solving a NLP problem. Indeed, if one checks reliability in solving the Banga & Seider [30] NLP test, MMA reports $C\% = 100\%$, while CRS a lower one. Thus, a lower CRS-MI reliability for solving MINLP problems is expected (Table 3, Test #5).

MMAMI-EA reported the maximum reliability (100%) in reaching the global solution for all six tests. The required $n_f$ is still 1-2 orders of magnitude smaller than of GA and SA for the same reliability (Tests #1,3,5). In general, MMAMI-EA requires less than $n_{f,max} \times N_{fam} \times N_{cycle}$ function evaluations.

As a general conclusion, simple modifications of an effective ARS (i.e. MMA [1,2]) and a combination with ($\mu + \lambda$)EA by using parallel searches for separate members and a ranking competition schema, allow adapting the ARS parameters for each member. As a result, considerably improving in the algorithm reliability for handling non-convex MINLP problems is obtained with a reduced computational effort comparatively with the classical SA, GA, and EA.

REFERENCES

Table 1. MMA algorithm basic relationships [1].

**Pseudo-one-dimensional branch of** \( j = 1, \ldots, N_{\text{uni}} \) **iterations.** Performs \( i = 1, \ldots, p \) successive steps per iteration with:

\[
x_{i+1}^j = x_i^j + M_i (b_i^j + t_i^j Z_i^j); \text{for success } f(x_{i+1}^j) < f(x_i^j)
\]

\[
t_i^j = \alpha x_i^j = r_i^j + c_i (b_i^j + t_i^j Z_i^j); \text{for success } f(x_{i+1}^j) < f(x_i^j)
\]

\[
x_i^j = x_i^j; \text{for failure } f(x_{i+1}^j) > f(x_i^j)
\]

\[
t_i^j = \alpha x_i^j = r_i^j + c_i (b_i^j + t_i^j Z_i^j); \text{for failure } f(x_{i+1}^j) > f(x_i^j)
\]

\[
is = 1; b_i^j = 0; M_i = \lambda M (b_i - a_i); i = 1, \ldots, p; \lambda M = 1 - 2; \alpha_s = 1; r_s = 0.75; c_s = 0.5; Z_i^j \in [-1,1], \text{ random & uniform}
\]

Table 2. MMA parameter sets used in the evolutionary search MMAM-EA.

<table>
<thead>
<tr>
<th>Parameter set #</th>
<th>#1 (global search)</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6 (local search)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_{\text{mult}} )</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>( \alpha_f )</td>
<td>0.98</td>
<td>0.95</td>
<td>0.90</td>
<td>0.85</td>
<td>0.80</td>
<td>0.75</td>
</tr>
</tbody>
</table>

\( N_{\text{uni}} = N_{\text{mult}} \times 60 \times \log(0.9) / \log(\alpha_f) \)
Table 3. MINLP optimization test problems solved by several random searches (\(x=\) continuous variables; \(y=\) integer variables).

<table>
<thead>
<tr>
<th>Test</th>
<th>Test #1</th>
<th>Test #2</th>
<th>Test #3</th>
<th>Test #4</th>
<th>Test #5</th>
<th>Test #6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of ((x,y))</td>
<td>(1,1)</td>
<td>(1,1)</td>
<td>(1,1)</td>
<td>(2,1)</td>
<td>(3,4)</td>
<td>(3,2)</td>
</tr>
<tr>
<td>Ineq. constraints</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>Integer variable</td>
<td>(y_1 \in {0,1})</td>
<td>(y_1 \in {0,1})</td>
<td>(y_1 \in {0,1})</td>
<td>(y_1 \in {0,1})</td>
<td>(y_1, y_2, y_3, y_4 \in {0,1})</td>
<td>(y_1 \in {78, ..., 102})</td>
</tr>
<tr>
<td>(y_2 \in {33, ..., 45})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Procedure

<table>
<thead>
<tr>
<th>Procedure</th>
<th>(n_f)</th>
<th>C%</th>
<th>(n_f)</th>
<th>C%</th>
<th>(n_f)</th>
<th>C%</th>
<th>(n_f)</th>
<th>C%</th>
<th>(n_f)</th>
<th>C%</th>
<th>(n_f)</th>
<th>C%</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA-R*</td>
<td>6787</td>
<td>100</td>
<td>13939</td>
<td>100</td>
<td>107046</td>
<td>90</td>
<td>22489</td>
<td>100</td>
<td>102778</td>
<td>60</td>
<td>37167</td>
<td>100</td>
</tr>
<tr>
<td>GA-D*</td>
<td>6191</td>
<td>100</td>
<td>15298</td>
<td>100</td>
<td>110233</td>
<td>90</td>
<td>23730</td>
<td>80</td>
<td>34410</td>
<td>90</td>
<td>35255</td>
<td>100</td>
</tr>
<tr>
<td>((\mu + \lambda))-ES*</td>
<td>1518</td>
<td>100</td>
<td>2255</td>
<td>100</td>
<td>1749</td>
<td>80</td>
<td>stopped</td>
<td>-</td>
<td>6710</td>
<td>90</td>
<td>2536</td>
<td>100</td>
</tr>
<tr>
<td>M-SIMPSA*</td>
<td>607</td>
<td>99</td>
<td>10582</td>
<td>83</td>
<td>stopped</td>
<td>-</td>
<td>14738</td>
<td>100</td>
<td>22309</td>
<td>100</td>
<td>27410</td>
<td>100</td>
</tr>
<tr>
<td>M-SIMPSA* (penalised)</td>
<td>16282</td>
<td>100</td>
<td>14440</td>
<td>100</td>
<td>38042</td>
<td>100</td>
<td>42295</td>
<td>100</td>
<td>63751</td>
<td>87</td>
<td>33956</td>
<td>95</td>
</tr>
<tr>
<td>MSGA**</td>
<td>NR</td>
<td>100</td>
<td>CPU=25s</td>
<td>100</td>
<td>NR</td>
<td>CPU=68s</td>
<td>100</td>
<td>NR</td>
<td>CPU=25s</td>
<td>100</td>
<td>NR</td>
<td>11767</td>
</tr>
<tr>
<td>Comparision</td>
<td>(a)=383; (b)=91</td>
<td>(a)=240; (b)=120</td>
<td>(a)=192; (b)=82</td>
<td>(a)=621</td>
<td>(a)=31000; (b)=20186;</td>
<td>(a)=28851; (b)=18379;</td>
<td>(a)=55126; (b)=20848</td>
<td>(a)=60; (b)=40;</td>
<td>(a)=2710; (b)=30</td>
<td>(a)=100; (b)=90;</td>
<td>(a)=100; (b)=100</td>
<td>11767</td>
</tr>
<tr>
<td></td>
<td>(a)=20; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=fail; (b)=100</td>
<td>(a)=2880; (b)=1610;</td>
<td>(a)=1036; (d)=1293;</td>
<td>(a)=14440; (f)=2138</td>
<td>(a)=100; (b)=90; (c)=90;</td>
<td>(a)=100; (b)=100</td>
<td>5501</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=2880; (b)=1610;</td>
<td>(a)=1036; (d)=1293;</td>
<td>(a)=14440; (f)=2138</td>
<td>(a)=100; (b)=90; (c)=90;</td>
<td>(a)=100; (b)=100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>MMAMI***</td>
<td>(a)=424; (b)=435</td>
<td>(a)=137; (b)=127</td>
<td>NR</td>
<td>(b)=14272; (c)=8217;</td>
<td>(b)=7789</td>
<td>NR</td>
<td>(b)=14272; (c)=8217;</td>
<td>(b)=7789</td>
<td>NR</td>
<td>(b)=14272; (c)=8217;</td>
<td>(b)=7789</td>
<td>NR</td>
</tr>
<tr>
<td>MMAMI-EA</td>
<td>(a)=104; (b)=23</td>
<td>(a)=481; (b)=33</td>
<td>(a)=21; (b)=17</td>
<td>(a)=920; (b)=431</td>
<td>(a)=31000; (b)=20186;</td>
<td>(a)=28851; (b)=18379;</td>
<td>(a)=55126; (b)=20848</td>
<td>(a)=60; (b)=40;</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>11767</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>(a)=100; (b)=100</td>
<td>5501</td>
<td></td>
</tr>
</tbody>
</table>

**Notations:** \(n_f= \text{average number of objective function evaluations; } C\% = \text{percentage of successful convergences to global optimum over 10 trials; search failure for MMAMI is considered when } n_f > 50000; \) (*) [25]; (**) [6]; (***) MMAMI parameter set of \([\text{uniN}=60, \text{multN}=60, \alpha_f=0.8, P_{\text{multi}}=3]\); NR= not reported results.

Initial guesses for \((x,y)\): Test #1, \((a)=[1.5,0], (b)=[0.6,1]\); Test #2, \((a)=[1.0], (b)=[1.39,1]\); Test #3, \((a)=[0.99,0], (b)=[0.99,1]\); Test #4, \((a)=[0.1,0], (b)=[9,0,1]\); Test #5, \((a)=[1,1,1,0,0,0], (b)=[0.1,1,1,1,0,0.1]\); Test #6, \([35,35,35,85,40]\).