Telecommun Syst (2011) 46: 217–243
DOI 10.1007/s11235-010-9288-0

A line search approach for high dimensional function optimization

Crina Grosan · Ajith Abraham · Aboul Ella Hassainen

Published online: 5 March 2010

Abstract This paper proposes a modified line search method which makes use of partial derivatives and re-starts the search process after a given number of iterations by modifying the boundaries based on the best solution obtained at the previous iteration (or set of iterations). Using several high dimensional benchmark functions, we illustrate that the proposed Line Search Re-Start (LSRS) approach is very suitable for high dimensional global optimization problems. Performance of the proposed algorithm is compared with two popular global optimization approaches, namely, genetic algorithm and particle swarm optimization method. Empirical results for up to 10,000 dimensions clearly illustrate that the proposed approach performs very well for the tested high dimensional functions.

Keywords Global optimization · Line search · Re-start · High dimensions

C. Grosan (✉)
Department of Computer Science, Babes-Bolyai University, Cluj-Napoca, Romania
e-mail: cgrosan@cs.ubbcluj.ro

A. Abraham
Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, Washington, USA
e-mail: ajith.abraham@ieee.org

A.E. Hassainen
Quantitative Methods and IS Department, College of Business Administration, Kuwait University, P.O. Box 5486, Safat 13055, Kuwait
e-mail: abo@cba.edu.kw

1 Introduction

The objective of global optimization is to find the globally best solution of (possibly nonlinear) models, in the (possible or known) presence of multiple local optima. Formally, global optimization seeks global solution(s) of a constrained optimization model. For an unconstrained minimization problem, global optimization may be formulated as follows:

Given a function $f : \Omega \subset \Re^n \to \Re$, find the lowest value of the function. We refer the *global minimum* where there can be also several local minima.

Definition 1 (global minima) $x^* \in \Omega$ is called *global minimum* of f if: $f(x^*) \leq f(y), \forall y \in \Omega$.

Definition 2 (local minima) $x^* \in \Omega$ is called *local minimum* of f if $\exists V(x^*)$ so that $f(x^*) \leq f(y), \forall y \in V(x^*)$, where $V(x^*)$ is a neighbourhood of x^*.

The term "global" contrasts with a local minima of f, which is the lowest value of $f(x)$ in some open neighborhood in Ω. Thus a function may have multiple local minima but it can usually have only one global minimum value [6].

In this paper, we assume the following:

- the optimization problem is unconstrained;
- f is nonlinear, continuous and differentiable;
- the feasible region Ω is given by a set of lower and upper bounds on each variable, i.e.
  $\Omega = \{\min_i \leq x_i \leq \max_i,\ i = l, \ldots, n\}$ and that the global minima lies within the interior of Ω.

As mentioned by Gergel [13] and as evident from other well established works [5, 7, 10–12, 21, 22, 38, 39, 41, 44], these class of problems are of substantial interest. Even

though there is a huge amount of work dealing with global optimization, there are still not many powerful techniques to be used for dense high dimensional functions. One of the main reasons is the high computational cost involved. Usually, the approaches are computationally expensive to solve the global optimization problem reliably. Very often, it requires many function evaluations and iterations and arithmetic operations within the optimization code itself. For practical optimization applications, the evaluation of $f(x)$ is often very expensive to compute and large number of function evaluations might not be very feasible [6].

Due to the practical demands, there were some attempts in trying to use different methods to solve high dimensional global optimization problems. One solution is to use parallel global optimization methods [37]. The parallel algorithm for global optimization proposed by Hofinger et al. [19] can approach functions having up to 512 dimensions (according to [19] it took 3 days for the first 10 iterations). The parallel particle swarm algorithm proposed by Schutte et al. [42] works well for some standard functions (example: Griewank and Corona test functions) and was tested for 128 dimensions.

Among the existing meta-heuristics for global optimization, we used two popular approaches for comparisons with the LSRS approach. The selected meta-heuristics, namely, Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) are well established and found highly successful and suitable for several classes of optimization problems. As evident from the scientific literature, GA and PSO were improved and adapted in several ways so as to obtain some reasonable results. In spite of all the success stories for several applications and revisions proposed during the last several years, these techniques are still not very much suitable for large scale global optimization problems involving high dimensions.

There are impressive number of papers reporting results in applying these techniques—either in their original form or in several improved and hybrid versions. Parsopoulos and Vrahitis [40] proposed a modified and improved PSO approach, which is proven to be efficient for different types of problems: global optimization, optimization in dynamic environments as well as multi objective optimization. Authors considered only very few dimensions (maximum 9 dimensions).

Liang et al. [33] provided an exhaustive comparison between several PSO variants and a comprehensive learning PSO. Authors tested only 30 dimensions, but PSO was improved in several ways to obtain some good results for the functions they considered.

Liu et al. [34] introduced turbulence in the Particle Swarm Optimization (TPSO) algorithm to overcome the problem of stagnation. The algorithm used a minimum velocity threshold to control the velocity of particles. TPSO mechanism is similar to a turbulence pump, which supplies some power to the swarm system to explore new neighbourhoods for better solutions. The parameters and the minimum velocity threshold of the particles were tuned adaptively by a fuzzy logic controller embedded in the TPSO algorithm, which is further called as Fuzzy Adaptive TPSO (FATPSO). Authors approach gave satisfactory results for multi-modal functions up to 100 dimensions.

A hybrid GA, scatter search and tabu search was proposed by Trafalis and Kasap [45] but only for test functions having a maximum of 5 dimensions. It is obviously difficult to apply this technique for more dimensions due to its complexity.

The evolutionary techniques proposed by Emmerich et al. [9] used Gaussian random field meta-models to deal with computationally expensive function evaluation optimization problems. The test functions used in there experiments consisted of only 20 dimensions.

The genetic algorithm proposed Koumousis and Katsaras [31] used a variable population size and periodic partial reinitialization of the population in the form of a saw-tooth function. Even though the aim of these approaches is to enhance the general performance of the algorithm, the test functions consisted of only a maximum of 20 dimensions. Authors compared the performance with standard GA and micro-GA [32].

The GA using adaptive representation proposed by Grosan and Oltean [15] is achieved by using several possible encodings (not only real or binary as usual, but also using different alphabets). These encodings can switch between them when no improvements are possible for one of the representations. Authors considered only a maximum of 30 dimensions.

There are also some recent researches reported in this are as follows.

Hirsch et al. [18] used a continuous greedy randomized adaptive search procedure (C-GRASP) for continuous global optimization. Even though the algorithm does not use any additional information about the function (such as differentiable, etc.) and can be adapted for use on parallel machines, it is only applied for a maximum of 10 dimensions.

Ismael et al. [24] combined particle swarm optimization with a direct search method and applied for functions having up to 294 dimensions, but results obtained for high dimensions (for minimization problems) are in the range of 10^7 (instead of 0).

Maaranen et al. [35] tested the performance of GA by considering several ways to initialize the population (which can affect the quality of the final solution) for functions involving 50 dimensions.

A tabu search combined with gradient method (steepest descend and quasi Newton) was formulated by Stepanenco and Engels [43] and applied for functions having up to 50 dimensions.

Addis and Leyffer [1] used a trust-region method for functions involving 50 dimensions. Some of the benchmark test functions are also used in this paper for comparisons.

The paper is structured as follows: in Sect. 2 the modified Line Search- Re-Start technique is presented. In Sect. 3 the two computational techniques used for comparisons, namely, GA and PSO—are briefly introduced. Section 4 is dedicated to the numerical experiments. First we illustrate how the proposed approach could be used for unidimensional functions. We then illustrate the performance in terms of fast convergence for the two dimensional functions. Finally, experiments are reported for higher number of dimensions (between 50 and 10,000). Section 5 includes conclusions and further work ideas.

2 Modified line search technique

Line search is a well established optimization technique. The modification proposed in this paper for the standard line search technique refers to step setting and also the incorporation of a re-start approach. To fine tune the performance, the first partial derivative of the function to optimize is also made use of. The proposed three modifications are summarized below and will be described in details in the subsequent sections:

1. The first modification refers to the inclusion of multi start principle within the search process.
2. The second modification is related to the setting of the direction and step.
3. The third modification refers to the re-starting of the line search method.

After a given number of iterations, the process is restarted by reconsidering other arbitrary stating point (or other multiple arbitrary starting points) which are generated by taking into account the results obtained at the end of previous set of iterations.

Shortly, the main body of the method used can be expressed as follows:

```
For i=1 to No of re-start applications
      For t=1 No of iterations
            Line_search()
     endfor
     Re-initialization();
     Re-start();
Endfor
```

In the following subsection, the algorithm is presented in a structured form.

2.1 Generation of the starting points

It is known that line search techniques uses a starting point. There are also versions which allow the usage of multiple points and the search will start separately from each of these points.

In the proposed approach, multiple arbitrary starting points are used. Each point is randomly generated over the definition domain.

For a function of n variables and the domain of definition given by:

$$[\min_1, \max_1] \times [\min_2, \max_2] \times \cdots [\min_n, \max_n]$$

where $[\min_i, \max_i]$ is the domain of ith variable, the procedure for generating the starting points x between the considered limits is given by:

Generate starting points()

```
for i=1 to No of arbitrary starting
points
     for j=1 to No of variables
          x_ij = min_j + random*(max_j-minx_j);
     endfor
endfor
```

The *random* function generates an arbitrary number between [0, 1].

2.2 Direction and step settings

Initially, we performed several experiments in order to set an adequate value for the direction. We used the standard value $+1$ or -1 and, for some functions the value -1 was favourable for very good results. We also performed some experiments by setting the direction value as being a random number between 0 and 1. Using the random number helped to obtain overall very good performance for the entire considered test functions. But usage of the value -1 for direction, obtains almost the same performance similar to that obtained with a random value. So, either of these values (the random one and the value -1) may be used for better performance.

The step is set as follows:

$$\alpha_k = 2 + \frac{3}{2^{k^2+1}},$$

where k refers to the iteration number.

The *Line_search()* technique may be written as follows:

Line_search()

```
Set k=1 (Number of iterations)
Repeat
   for i=1 to No of starting points
       for j=1 to No of variables
              p_k=-1; //or p=random;
              α_k = 2 + 3/2^(k^2+1)
```

$$x_{ij}^{k+1} = x_{ij}^{k} + p_k \cdot \alpha_k$$

```
      endfor
    if f(x_i^{k+1}) > f(x_i^k) then x_i^{k+1} = x_i^k.
  Endfor
  k = k + 1
Until k = Number of iterations (apriori known).
```

Remarks

(i) The condition:

```
if f(x_i^{k+1}) > f(x_i^k) then x_i^{k+1} = x_i^k
```

allows us to move to the new generated point only if there is an improvement in the quality of the function.

(ii) Number of iterations for which line search is applied is apriori known and is usually a small number. In our experiments, we set the number of these iterations to 10 even for high dimensional problems (for example: 2,000 or 10,000).

(iii) When restarting the line search method (after the insertion of the re-start technique) the value of iterations number will again start from 1 (this should not be related to the value of α after the first set of iterations (and after each of the following ones)).

We have tried using several experiments to set a value for the step, starting with random values (until we will reach a point for which the objective function is getting a better value); using a starting value for the step and generating random numbers with Gaussian distribution around this number, etc. As a result of the initial experiments performed, we decided that the formula for the step provided above was performing well. But, of course, there are also several other ways to set this.

2.3 Re-start insertion

In order to restart the algorithm the best result obtained in the previous set of iterations is taken into account and by following the steps given below:

1. Among all the considered points, the solution for which the objective function is obtaining the best value is selected. If there are several such solutions, one of them is randomly selected. This solution will be a multi-dimension point in the search space and denoted by x for an easier reference.
2. For each dimension i of the point x, the first partial derivative with respect to this dimension is calculated. This means the gradient of the objective function is calculated which is denoted by g. Taking this into account, the bounds of the definition domain for each dimension is re-calculated as follows:

if $g_i = \dfrac{\partial f}{\partial x_i} > 0$ then $\max_i = x_i$;

if $g_i = \dfrac{\partial f}{\partial x_i} < 0$ then $\min_i = x_i$

The search process is re-started by re-initializing a new set of arbitrary points (using `Generate starting points ()` procedure) but between the newly obtained boundaries (between the new $\max_i$ or new $\min_i$).

The pseudo code of the Re-start technique is given below (g denotes the gradient of f):

`Re_start()`

```
Calculate the solution (out of the
 entire set of points) for which the
 value of the function is minimum.
Let x# be minimum obtained at the
 current moment of the search process
For i = 1 to No of dimensions
        if g_i(x#) > 0 then max_i = x_i#
        if g_i(x#) < 0 then min_i = x_i#
endfor
```

2.4 General Line Search with Re-Start (LSRS) procedure

The flowchart of LSRS is depicted in Fig. 1. The Line Search method presented in the previous subsections combined with the re-start technique as described above is expressed using the following pseudo code:

`General Line Search with Re-Start (LSRS)`

```
Set t = 1;
 Repeat
  Generate starting points (max, min);
    Line_search (k);
    Re_start (new values for max_i and/or
     min_i for each dimension will be
     obtained);
    t = t + 1;
  Until t = Number of applications of the
   re-start technique (a priori known).
Select the solution x* for which the
 value of the objective function is
 minimum.
Print x*.
```

3 Genetic Algorithms and Particle Swarm Optimization Algorithms

Due to the fact that these two techniques are now well established and very popular among the artificial intelligence/problem solving community, some of the fundamental ideas are presented in this section with a focus on the

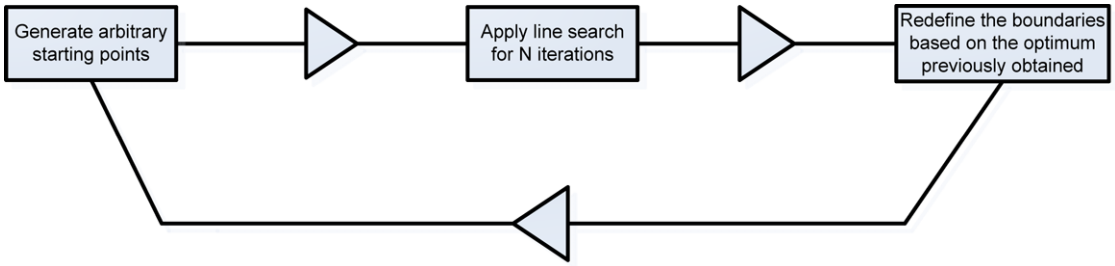

Fig. 1 The general scheme of the LSRS

algorithms structure which would help the reader to follow the experiments.

3.1 Genetic Algorithms

Genetic Algorithms are a population based search technique using principles from biological evolution which are transposed in a computational scheme. Proposed in 1975 by J. Holland [20], were developed further and successfully applied in various domains. The basic components are solution representation, population initialization, fitness function and genetic operators (such as selection, crossover and mutation). The main idea is as follows: the genetic pool of a given population potentially contains an approximate solution, to a given problem. During reproduction and crossover, new genetic combination occurs and there is chance to obtain a better solution (either than the ones which were combined or mutated or better than all the existing ones in that pool). By repeating several times these recombinations between the potential solutions for the problem, usually a very good approximation of the solution is obtained [2, 3, 14].

Genetic Algorithm Scheme

```
Set t=1;
Randomly initialize population P(t);
Repeat
      Evaluate individuals from P(t);
      Selection on P(t). Let P'(t) be
       the selected individuals;
      Crossover on P'(t). Survival
       between parents and offspring.
      Mutation on P'(t). Survival
       between parent and offspring.
      t=t+1;
      P(t)=P'(t-1);
Until t=Number of generations.
```

Remarks:

(i) The selection procedure used is binary tournament.
(ii) Crossover and mutation are performed with a given probability.
(iii) Survival between parents and offspring after crossover will return as results the best two individuals among the four (two parents and two offspring) considered.
(iv) Survival between parents and offspring after mutation is made by direct comparison (in terms of fitness function). The best one between parent and offspring will be accepted.

3.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is also a population based method, which can be successfully applied for optimization. The concept of Particle Swarms, although initially introduced for simulating human social behavior, has become very popular these days as an efficient search and optimization technique. PSO, as it is called now, does not require any gradient information of the function to be optimized, uses only primitive mathematical operators and is conceptually very simple.

In PSO, a population of conceptual 'particles' is initialized with random positions x_i and velocities v_i, and a function, f, is evaluated, using the particle's positional coordinates as input values. In an n-dimensional search space, $x = (x_1, x_2, x_3, \ldots, x_n)$ and $v = (v_1, v_2, v_3, \ldots, v_n)$. Positions and velocities are adjusted, and the function is evaluated with the new coordinates at each time-step. The best position of the particle found during the search process—pbest—as well as the position of the best particle from the entire swarm—gbest (global best)—is stored. The basic update equations for the kth dimension of a current particle $p\_current$ in PSO at iteration t may be given as:

$$v_k(t+1) = \omega \cdot v_k(t) + c_1 \cdot rand_1 \cdot (pbest_k - p\_current_k(t)) + c_2 \cdot rand_2 \cdot (gbest_k - p\_curent_k(t)))$$
$$p\_current_k(t+1) = p\_current_k(t) + v_k(t+1) \quad (1)$$

The variables $rand_1$ and $rand_2$ are random positive numbers, drawn from a uniform distribution and defined by an upper limit rand_max, which is a parameter of the system. c_1 and c_2 are called acceleration constants whereas ω is called inertia weight. $pbest_k$ is the local best solution found

so far by the particle, $gbest_k$ while represents the positional coordinates of the fittest particle found so far in the entire community (for dimension k). Once the iterations are terminated, most of the particles are expected to converge to a small radius surrounding the global optima of the search space [8, 23, 25–30]. The core of the PSO algorithm used in our experiments is presented below:

Particle Swarm Optimization

```
Initialize the population of particles.
For each particle set its pbest.
Set the gbest.
Set t=1;
Repeat
  For each particle
            update its current position
             using equations (1);
            If the value of the function
             to optimize for the new
             obtained particle is better
             than the initial one
            Then keep the new obtained
             particle
            Else keep the initial
             particle
            Update pbest
   Endfor
   Update gbest;
   t=t+1;
Until t=Number of iterations.
```

4 Numerical examples

In order to demonstrate the performance of the proposed LSRS method we present the results obtained for a set of benchmark problems which are described in Sect. 4.2. We consider a high number of dimensions varying between 50 and 10,000.

4.1 Performance assessment

According to Baritompa [4], there are two criteria which must be taken into account: effectiveness and efficiency. The first one reflects whether we reached what we wished (solution with a given approximation or so) and the second one refers to the (computational) cost required to do this. One measure of effectiveness can be given by the number of times the global optimum has been reached by a certain algorithm. For this purpose, usually several repetitions of the algorithm application are required. For measuring the efficiency, usually the number of function evaluations used is considered. The convergence speed, which is a measure of whether the solution in the next iteration is improved and how can be also considered as an efficiency criterion.

In order to asses the performances of the new proposed technique, performance graphs are used and the empirical results are also compared with genetic algorithms and particle swarm optimization algorithms.

All the algorithms including LSRS, GA and PSO have been implemented using C++ Builder 6.0 and were run on a 2.4 GHz Intel Duo Core CPU, with 2 GB RAM.

4.2 Test functions

The proposed algorithm is tested using a set of standard continuous test functions [10, 16] which are widely used in the literature and whose characteristics are diverse enough to cover many of the problems which can arise in global optimization problems as mentioned in [17].

Ackley function

$$f(x) = 20 + e - 20 \cdot e^{-0.2 \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$$

Domain of definition: $[-10, 10]^n$
Optimum point: $x^* = (0, 0, \ldots 0)$, $f(x^*) = 0$.

Levy function

$$f(x) = \sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2(1 + 10\sin^2(\pi \cdot y_i + 1)) + (y_n - 1)^2(1 + \sin^2(2 \cdot \pi \cdot x_n))$$

$$y_i = 1 + \frac{x_i - 1}{4}, \text{ for } i = 1, 2, \ldots, n$$

Domain of definition: $[-10, 10]^n$
Optimum point: $x^* = (1, 1, \ldots 1)$, $f(x^*) = 0$.

Quadric function

$$f(x) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2$$

Domain of definition: $[-10, 10]^n$
Optimum point: $x^* = (1, 1, \ldots 1)$, $f(x^*) = 0$.

Rastrigin function

$$f(x) = 10 \cdot n + \sum_{i=1}^{n}\frac{x_i^2}{4000} - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

Domain of definition: $[-5.12, 5.12]^n$
Optimum point: $x^* = (0, 0, \ldots 0)$, $f(x^*) = 0$.

Rosenbrock function

$$f(x) = \sum_{i=1}^{n-1}[100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2]$$

Domain of definition: $[-5, 10]^n$
Optimum point: $x^* = (1, 1, \ldots 1),\ f(x^*) = 0.$

Schwefel function

$$f(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|$$

Domain of definition: $[-10, 10]^n$
Optimum point: $x^* = (0, 0, \ldots 0),\ f(x^*) = 0.$

Sphere function

$$f(x) = \sum_{i=1}^{n} x_i^2$$

Domain of definition: $[-10, 10]^n$
Optimum point: $x^* = (0, 0, \ldots 0),\ f(x^*) = 0.$

Sum Squares function

$$f(x) = \sum_{i=1}^{n} i \cdot x_i^2$$

Domain of definition: $[-10, 10]^n$
Optimum point: $x^* = (0, 0, \ldots 0),\ f(x^*) = 0.$

4.3 Example of the LSRS convergence for the one-dimensional Sphere function

We consider a simple example for the Sphere function of one dimension to illustrate the convergence of LSRS for 4 arbitrary starting points. The boundaries are modified after just one step (one iteration of the line search application). Figure 2 depicts the LSRS convergence in three phases of one iteration each. Initial points plotted in Fig. 2 are randomly generated between $[-5.12, 5.12]$. These points and

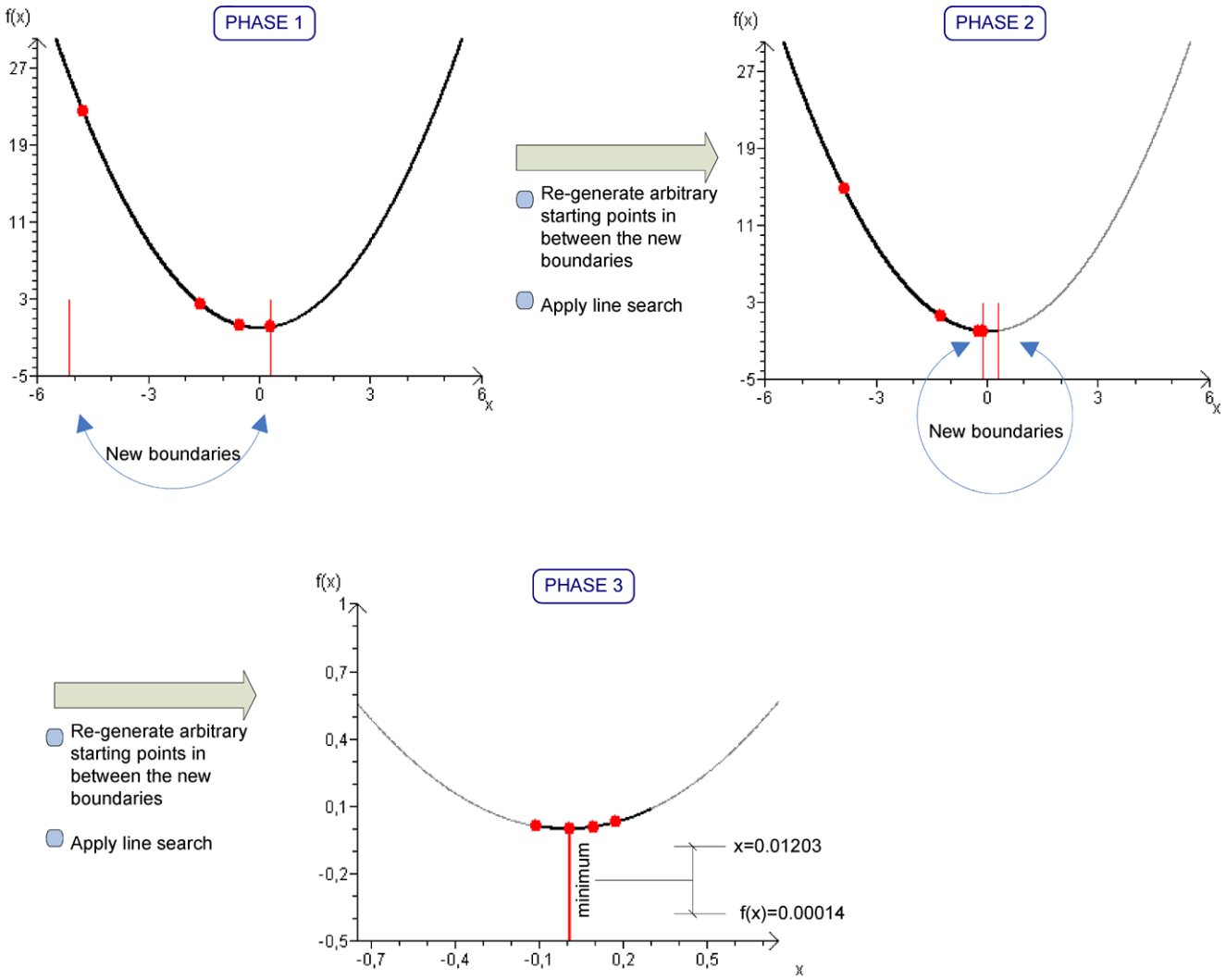

Fig. 2 Convergence of LSRS for the one-dimension Sphere function

	The four points after the Phase 2			
	Point 1	Point 2	Point 3	Point 4
	-1.589916	-4.751609	-5.218501	3.005734
f	2.527835	22.57779	27.23275	9.034441

Fig. 3 The points and the corresponding function values obtained after Phase 1

Boundaries after Phase 1	
Initial lower bound -5.12	Initial upper bound -5.12
Lower bound after the Phase 1 -5.12	Upper bound after the Phase 1 0.300573

(a)

Boundaries after Phase 2	
Initial lower bound -5.12	Initial upper bound -0.300573
Lower bound after the Phase 2 0.124962	Upper bound after the Phase 2 0.300573

(b)

Boundaries after Phase 3	
Initial lower bound 0.124962	Initial upper bound -0.300573
Lower bound after the Phase 3 0.124962	Upper bound after the Phase 3 0.012035

(c)

Fig. 4 The modification of boundaries after each Phase for the one-dimensional Sphere function

Fig. 5 The points and the corresponding function values obtained after Phase 2

	The four points after the Phase 2			
	Point 1	Point 2	Point 3	Point 4
	-0.233898	-3.842776	-0.124962	-1.239832
f	0.054708	14.766934	0.015615	1.537184

Fig. 6 The points and the corresponding function values obtained after Phase 3

	The four points after the Phase 3			
	Point 1	Point 2	Point 3	Point 4
	0.174625	0.012035	0.096014	-0.109076
f	0.0304941	0.0001448	0.009218	0.011897

the corresponding function values are given in Fig. 3. We applied line search for one iteration with direction -1 and step $2 + 3/4$. The new obtained boundaries after this first Phase are given in Fig. 4(a). Four other points are now re-generated in between the new obtained upper and lower bounds. After applying line search for one iteration, the points obtained and the corresponding function values are given in Fig. 5. The points obtained in the Phase 3 (in the same manner like in Phase 2) are presented in Fig. 6. The optimum point (and its corresponding function value) and its convergence at each Phase is depicted in Fig. 7.

4.4 The two-dimensional case: results and discussions

The PSO technique is well known as a fast convergent optimization algorithm. This is indeed the case for functions having a low number of dimensions [34]. Just few particles and iterations are good enough to provide an acceptable result for 2 objectives. This is evident from the results presented in Table 2 (for all the test functions but having only 2 dimensions) that PSO is obtaining sometimes better results than LSRS under the same conditions. But for higher dimensions PSO is getting stuck only after little iterations (same

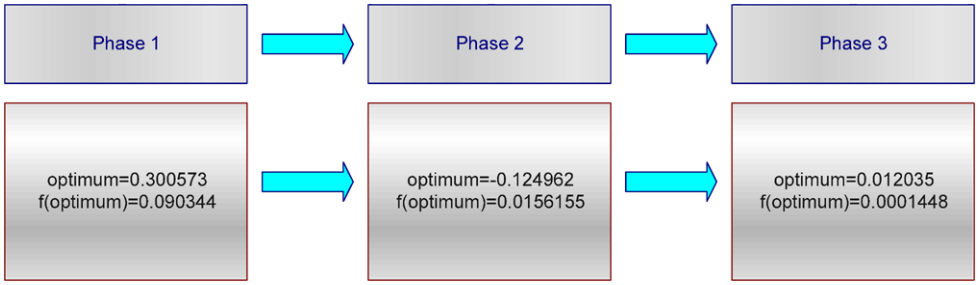

Fig. 7 The optimum point and the corresponding function value obtained at the end of each phase for the one-dimensional Sphere function

Table 1 Comparison of PSO and LSRS for two dimensional case in terms of fast convergence and best results

Test function	Swarm size (number of initial points for LSRS)	Number of iterations (N)	Actual optimum	Minimum obtained by PSO	Minimum obtained by LSRS		
					With no re-start	With 2 re-starts	With N re-starts
Ackley	5	2	0	1.71828	3.576	1.7816	1.7816
Levy	2	7	0	0.0629	5.2501	0.035	0.00002
Quadric	2	4	0	0.0064	119.45	4.068	0.0034
Rastrigin	100	2	0	5.449	20.17	16.45	16.45
Rosenbrock	2	4	0	0.8504	2559.96	17.62	1.069
Schwefel	10	2	0	2.0	2.7305	0.4823	0.4823
Sphere	10	2	0	0.7888	0.0185	0.0042	0.0042
Sum squares	3	5	0	0.0004	0.31	0.05	0.0001

conclusions can be also found in [34]) and the results obtained are much worse while compared to LSRS empirical performance.

Table 1 displays the best results which can be obtained with minimal resources (less number of iterations and less individual or particles in the population or starting points). Since PSO is the fastest technique (as the results presented also show this) we are comparing PSO and LSRS and we are providing details about the GA results obtained in the same or similar conditions in the discussions which follows due to the fact that GA convergence is a little bit slower.

4.4.1 *Remarks related to results for the two-dimensional case*

The results presented in Table 1 are obtained by considering the best minimum obtained in 100 independent runs.

- We considered these parameters values based on the results obtained for PSO. For example, for the Rosenbrock test function, the value 0.8504 is obtained in the forth iteration with only 2 particles. But the same value is obtained even after 100 iterations with a swarm of size 10. Using a higher number of particles 10, the value 0.7773 is obtained after 50 iterations. We therefore considered the initial parameters (2 particles and 4 iterations) as reference for comparison purposes.
- Three variants of LSRS are presented in Table 1:
  - LSRS with no re-start is in fact, the standard line search technique applied for N iterations;
  - LSRS with 2 re-starts is line search applied two times for $N/2$ iterations each (or $N/2$ and $N/2+1$ in case of odd N) but second time it is applied after the domain of definition reduction using the gradient.
  - LSRS with N re-starts refers to N applications of the line search for one iteration each after which the domain of definition is reduced (the boundary conditions are updated) using the gradient.
- For the Ackley test function, PSO performance was not improving after the second iteration even with a swarm size of 100. LSRS is getting a close result to PSO after only 2 iterations and the result obtained are 0.0621 for 2 initial starting points, 4 iterations and 4 re-starts (at one iteration each) and 0.5900 for 2 initial starting points, 4 iterations and 2 re-starts (at 2 iterations each). Result obtained by GA after 5 generations with a population of size 2 is 8.525. After 10 generations, with a population of size 10, the minimum obtained is 2.47.
- The minimum obtained by GA for Levy function after 7 generations with a population of 2 individuals is 1.461 which is worse than both PSO and LSRS.
- For the Quadric test function, PSO converged very fast, but results obtained by LSRS are also good in the last

case. GA obtained the value of 0.072 relatively fast (with 5 individuals and after 5 generations) but is getting trapped there and the performance could not be improved even after 1000 iterations and considering a population of size of 100.

- For the Rastrigin test function, it is obvious that only two iterations were not enough for LSRS to converge. Results obtained for 2 re-starts and for N re-starts are same because N is 2 in this case. PSO got trapped after two iterations and the performance could be improved only by increasing the swarm size. By increasing N to 10, LSRS obtained a value of 2.17E−8 (using 100 initial starting points). If only 10 starting points are considered, 10 iterations and 10 re-starts, result obtained is 0.00035 while result obtained for 10 starting points, 10 iterations and only 2 re-starts is 1.79. This is the only case where GA is obtaining the best result with the parameters given in Table 2. With a population of 50 individuals the result obtained is 2.096 while for 100 individuals the results obtained is 0.21 after only 2 generations.
- With a population of 2 individuals and using 4 generations GA is obtained the worst results for Rosenbrock test function among all the algorithms (obtained value: 8.343).

In the case of Schwefel test function, PSO performance could not be improved even with higher number of iterations (100) and higher swarm size (100). LSRS obtained the result of 0.011, if the number of re-starts is increased from 2 to 4 (one iteration for each re-start and 2 initial starting points). GA obtained 1.8786 which is better than PSO with 10 individuals and after 2 generations the performance could be improved to 0.855, if the number of generations is increased to 10.

The result obtained by GA for the Sphere test function is 1.659 (using 10 individuals and after 2 generations). For a population of 50 individuals the minimum obtained by GA is 0.824. If we increase the swarm size to 50, the optimum obtained by PSO is 0.0288 while for 100 particles in the swarm the optimum obtained is 0.0128 (after 2 iterations). While testing LSRS for 4 iterations and 4 re-starts (of one iteration each) using 2 starting points we get the result 3.31E−5.

For the Sum squares test function, result obtained by GA is 3.518 but this could be improved to 0.068, if the number of generations is increased from 5 to 10.

4.5 The high-dimensional experiments

From the previous illustrations, it is evident that PSO and GA are good candidates for a low number of dimensions. All the algorithms were also tested for higher number of dimensions. We considered 50, 100, 250, 500, 750 and 1,000 dimensions for all the test functions. Also, we test the performance of LSRS for 2,000 and 10,000 dimensions but comparisons with PSO and GA are not reported due to the poor results which is evident from the results obtained for 1,000 dimensions.

4.5.1 Parameter settings

Table 2 presents the values of the main parameters used by the three techniques involved in experiments.

Table 2 Parameters used by LSRS, PSO and GA for 50, 100, 250, 500, 750, 1,000 and 2,000 dimensions

Parameter	Parameter values						
	No of dimensions						
	50	100	250	500	750	1000	2000
LSRS							
No of starting arbitrary points	500	500	500	500	500	500	500
No of restarting (reinitialization)	50	50	100	100	100	100	100
No of iterations per each restarting phase	10	10	10	10	10	10	10
Genetic Algorithm							
Population size	500	500	500	500	500	500	
Number of generations	20,000	20,000	20,000	20,000	20,000	20,000	
Mutation probability	0.9	0.9	0.9	0.9	0.9	0.9	
Crossover probability	0.5	0.5	0.5	0.5	0.5	0.5	
Particle Swarm Optimization							
Number of particles	500	500	500	500	500	500	
Number of iterations	20,000	20,000	20,000	20,000	20,000	20,000	
c1, c2	2	2	2	2	2	2	

Table 3 Empirical performance for 50 dimensions

Approach	Function							
	Ackley	Levy	Quadric	Rastrigin	Rosenbrock	Schwefel	Sphere	Sum Squares
GA	Best							
	2.882	0.327	6.105E+3	104.86	55.34	10.99	6.544	425.57
	Average							
	3.38	0.327	6.113E+3	122.4	55.34	13.46	15.384	431.05
	Standard Deviation							
	0.477	0	545.08	14.99	0	1.896	8.111	39.454
PSO	Best							
	5.045	11.93	3.02E+4	198.86	1.58E+4	29.491	59.91	1002.2
	Average							
	6.055	18.88	7.81E+4	247.45	5.9E+4	46.80	100.16	2116.6
	Standard Deviation							
	1.017	8.38	3.43E+4	52.01	5.23E+4	10.163	36.73	786.29
LSRS	Best							
	−6.5E−19	2.9E−39	6.27E−19	0.0	2.47E−28	1.86E−11	1.34E−22	9.86E−21
	Average							
	−6.5E−19	2.9E−39	2.33E−18	0.0	1.38E−18	1.91E−11	1.38E−18	1.42E−18
	Standard Deviation							
	0	0	8.11E−19	0	1.29E−18	4.15E−12	1.29E−18	1.25E−18
Actual optimum	0	0	0	0	0	0	0	0

Table 4 Empirical performance for 100 dimensions

Approach	Function							
	Ackley	Levy	Quadric	Rastrigin	Rosenbrock	Schwefel	Sphere	Sum Squares
GA	Best							
	4.333	0.6509	2.2E+5	333.5	111.82	37.91	56.50	3.12E+3
	Average							
	4.483	0.6509	2.2E+5	348.37	2.74E+4	40.07	64.83	3.13E+3
	Standard Deviation							
	0.353	0	1.96E+5	32.76	8.61	3.78	9.79	279.63
PSO	Best							
	6.01	45.72	4.43E+5	642.57	1.05E+5	92.07	221.21	6.02E+3
	Average							
	6.78	55.72	7.81E+5	707.86	2.19E+5	104.86	249.20	9.99E+3
	Standard Deviation							
	0.83	10.39	3.04E+5	98.20	9.59E+4	15.81	32.86	3.52E+3
LSRS	Best							
	−6.5E−19	2.9E−39	9.2E−16	0	5.83E−28	7.81E−19	5.34E−19	4.68E−18
	Average							
	−6.5E−19	2.9E−39	1.15E−15	0	6.94E−16	3.98E−10	6.94E−16	6.98E−16
	Standard Deviation							
	0	0	4.38E−16	0	6.63E−16	4.97E−10	6.63E−16	6.58E−16
Actual optimum	0	0	0	0	0	0	0	0

Parameters used by LSRS for 10,000 dimensions are same for 2,000 dimensions but the number of initial arbitrary starting points is set to only 10.

4.5.2 Results and comparisons

Results for 50, 100, 250, 500, 750, 1,000 variables obtained by all the three techniques at the end of search process are presented in Tables 3, 4, 5, 6, 7 and 8 respectively. The best and average solution and standard deviation are displayed. Results obtained by LSRS for 2,000 variables are presented in Table 9 and results obtained for 10,000 variables are presented in Table 10.

The convergence of LSRS is illustrated in Figs. 8–15 for 50, 100, 250, 500, 750, 1,000, 2,000 and 10,000 dimensions respectively. The best objective function value obtained at the end of each re-start application is depicted. In Fig. 16, a graphical comparison of the convergence for all the three techniques during the first 1,000 iterations for functions having 1,000 dimensions is provided. The comparison is not 100% fair in terms of number of iterations because LSRS is re-starting after each 10 iterations, but the authors wish to illustrate the performance of LSRS when compared to GA and PSO (also, the big difference regarding the optimum obtained can be deduced from Tables 3–8). In Fig. 17 the solutions obtained at the end of search process for functions having 10,000 dimensions are depicted. Since only 10 starting points were used for this case, the structure of the final points is easier to demonstrate.

4.5.3 Discussions on the performance and results

As evident from the results presented in tables above and below, LSRS converges very fast and obtains very accurate results. Even for 10,000 dimensions, a set of 10 initial starting points are good enough to obtain good performance. It can be observed that there is not many differences between the parameters value settings for different number of dimensions. Even for 2,000 and 10,000 dimensions, the same number of re-startings and the same number of iterations for each re-start were used.

Empirical and graphical comparisons with GA and PSO also clearly indicate the big difference between these approaches in terms of quality of solutions and speed of convergence.

For the Rastrigin test function, for example, LSRS obtained the clear 0 for minimum and all the starting points converged to this value, even for 10,000 dimensions.

It is obvious than the greater the number of re-starts the faster the convergence and the more accurate the results. A greater number of re-starts will increase the computational cost (derivatives to be involved and boundaries are to be modified accordingly). But still the computational cost is very less compared to other two techniques (GA and PSO).

Table 5 Empirical performance for 250 dimensions

Approach	Function							
	Ackley	Levy	Quadric	Rastrigin	Rosenbrock	Schwefel	Sphere	Sum Squares
GA	Best							
	5.64	1.621	1.17E+7	1.53E+3	2.96E+5	195.28	286.73	5.74E+4
	Average							
	5.66	1.621	1.17E+7	1.54E+3	3.07E+5	196.31	292.73	5.74E+4
	Standard Deviation							
	0.41	0	1.06E+6	138.95	2.94E+4	17.64	27.28	5.12E+3
PSO	Best							
	7.50	202.28	9.91E+6	2.51E+3	6.14E+5	340.95	787.48	7.91E+4
	Average							
	7.60	207.38	1.34E+7	2.6E+3	6.96E+5	2.31E+10	898.47	9.71E+5
	Standard Deviation							
	0.59	18.33	4.38E+6	225.96	9.93E+4	5.15E+11	54.09	1.5E+4
LSRS	Best							
	−4.3E−19	2.9E−39	5.7E−13	0	1.57E−27	5.63E−20	2.02E−17	1.1E−15
	Average							
	−4.3E−19	2.9E−39	6.09E−13	0	3.72E−13	8.71E−20	3.95E−17	2.8E−15
	Standard Deviation							
	0	0	1.86E−13	0	3.34E−13	8.95E−21	7.38E−18	7.1E−16
Actual optimum	0	0	0	0	0	0	0	0

Table 6 Empirical performance for 500 dimensions

Approach	Function							
	Ackley	Levy	Quadric	Rastrigin	Rosenbrock	Schwefel	Sphere	Sum Squares
GA	Best							
	7.18	3.24	1.25E+8	4.58E+3	563.638	598.12	1.19E+3	4.07E+5
	Average							
	7.21	3.33	1.25E+8	4.59E+3	566.62	600.51	1.20E+3	4.07E+5
	Standard Deviation							
	0.01	0.244	1.12E+7	409.93	63.22	53.21	107.67	3.63E+4
PSO	Best							
	8.12	544.57	1.41E+8	6.11E+3	2.43E+6	884.67	211.75	4.86E+5
	Average							
	8.14	546.02	1.47E+8	6.13E+3	2.56E+6	891.19	2180.35	5.22E+5
	Standard Deviation							
	0.37	42.36	5.61E+7	11.20	2.17E+5	6.55	31.18	3.55E+4
LSRS	Best							
	−4.3E−19	2.9E−39	2.12E−11	0	3.4E−27	2.91E−19	4.54E−16	4.05E−35
	Average							
	−4.3E−19	2.9E−39	4.31E−11	0	2.61E−11	4.08E−19	9.0E−16	7.96E−35
	Standard Deviation							
	0	0	1.14E−11	0	2.32E−11	3.62E−20	1.52E−16	1.95E−35
Actual optimum	0	0	0	0	0	0	0	0

Table 7 Empirical performance for 750 dimensions

Approach	Function							
	Ackley	Levy	Quadric	Rastrigin	Rosenbrock	Schwefel	Sphere	Sum Squares
GA	Best							
	7.41	4.85	5.64E+8	7.53E+3	872.75	1058.35	2.37E+3	1.07E+6
	Average							
	3116.99	5.05	5.64E+8	7.54E+3	874.64	1060.90	2.38E+3	1.07E+6
	Standard Deviation							
	4.0E+4	0.34	19.96	672.71	67.64	94.26	212.11	9.5E+4
PSO	Best							
	8.56	1.09E+3	5.75E+8	9.85E+3	4.91E+6	1.40E+3	3.78E+3	1.26E+6
	Average							
	8.58	1.10E+3	5.75E+8	9.87E+3	5.10E+6	1.46E+12	3.81E+3	1.39E+6
	Standard Deviation							
	0.06	5.06	210.78	16.49	2.8E+5	1.20E+9	20.3	7.84E+4
LSRS	Best							
	1.0E−18	2.9E−39	5.86E−31	0	5.08E−27	1.95E−18	7.40E−36	1.1E−33
	Average							
	1.0E−18	2.9E−39	1.57E−30	0	5.55E−27	2.45E−18	1.41E−35	2.1E−33
	Standard Deviation							
	1.3E−19	0	3.59E−31	0	1.51E−28	1.94E−19	3.22E−36	4.6E−34
Actual optimum	0	0	0	0	0	0	0	0

Table 8 Empirical performance for 1,000 dimensions

Approach	Function							
	Ackley	Levy	Quadric	Rastrigin	Rosenbrock	Schwefel	Sphere	Sum Squares
GA	Best							
	7.86	6.47	6.28E+8	1.073E+4	1.12E+3	1.49E+3	3.44E+3	2.16E+6
	Average							
	7.87	7.17	6.28E+8	1.075E+4	1.12E+3	1.492E+3	3.45E+3	2.16E+6
	Standard Deviation							
	0.0056	1.07	66.81	482.31	10.8	67.0	155.21	9.72E+4
PSO	Best							
	8.91	1.61E+3	1.5E+9	1.40E+4	6.46E+6	3.9E+19	5.24E+3	2.65E+6
	Average							
	9.02	1.62E+3	1.61E+6	1.40E+4	6.58E+6	4.1E+47	5.50E+3	2.66E+6
	Standard Deviation							
	0.10	10.97	9.46E+7	20.2	1.86E+5	9.2E+48	2.7E+2	8.94E+3
LSRS	Best							
	1.3E−18	2.9E−39	5.34E−30	0	6.84E−27	9.30E−18	7.97E−19	3.78E−33
	Average							
	1.3E−18	2.9E−39	1.38E−29	0	7.41E−27	1.12E−17	1.25E−18	7.35E−33
	Standard Deviation							
	4.8E−33	0	3.68E−30	0	1.66E−28	7.33E−19	2.05E−19	1.49E−33
Actual optimum	0	0	0	0	0	0	0	0

Table 9 Empirical performance for 2,000 dimensions

Approach	Function							
	Ackley	Levy	Quadric	Rastrigin	Rosenbrock	Schwefel	Sphere	Sum Squares
LSRS	Best							
	−4.3E−19	2.9E−39	9.37E−8	0	1.40E−26	2.41E−17	9.97E−34	7.58E−31
	Average							
	−4.3E−19	2.9E−39	1.69E−7	0	1.48E−26	3.08E−17	2.35E−33	1.27E−30
	Standard Deviation							
	9.6E−35	0	3.42E−8	0	2.44E−28	2.31E−18	6.91E−34	2.02E−31
Actual optimum	0	0	0	0	0	0	0	0

Table 10 Empirical performance for 10,000 dimensions

Approach	Function							
	Ackley	Levy	Quadric	Rastrigin	Rosenbrock	Schwefel	Sphere	Sum Squares
LSRS	Best							
	6.5E−17	2.9E−39	5.10E−23	0	7.45E−26	6.77E−15	3.08E−30	5.58E−27
	Average							
	7.9E−17	5.2E−31	1.06E−22	0	7.53E−26	7.07E−15	3.88E−30	1.02E−26
	Standard Deviation							
	1.0E−17	3.4E−31	3.44E−23	0	5.54E−28	2.35E−16	5.60E−31	3.51E−27
Actual optimum	0	0	0	0	0	0	0	0

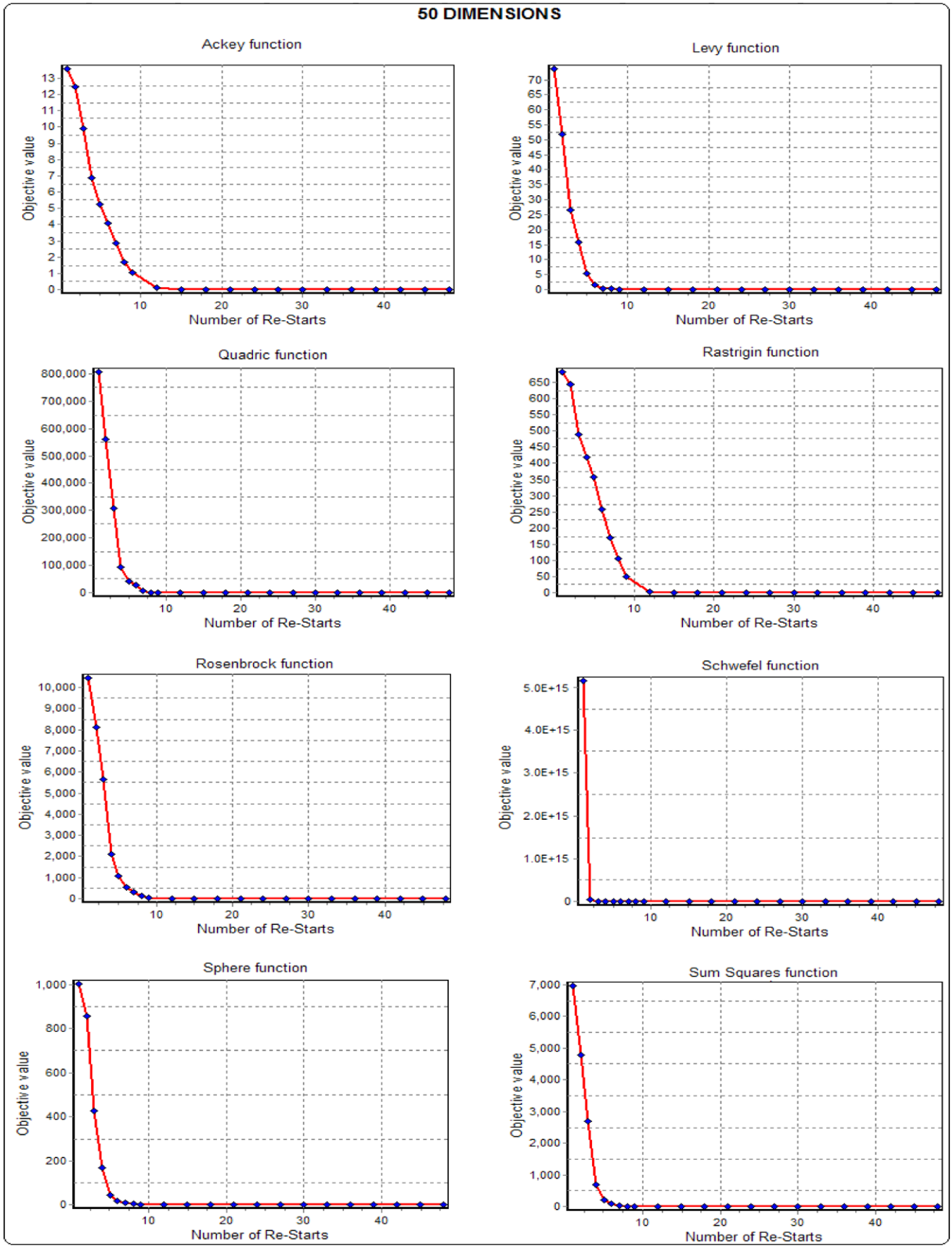

Fig. 8 LSRS convergence for 50 dimensions

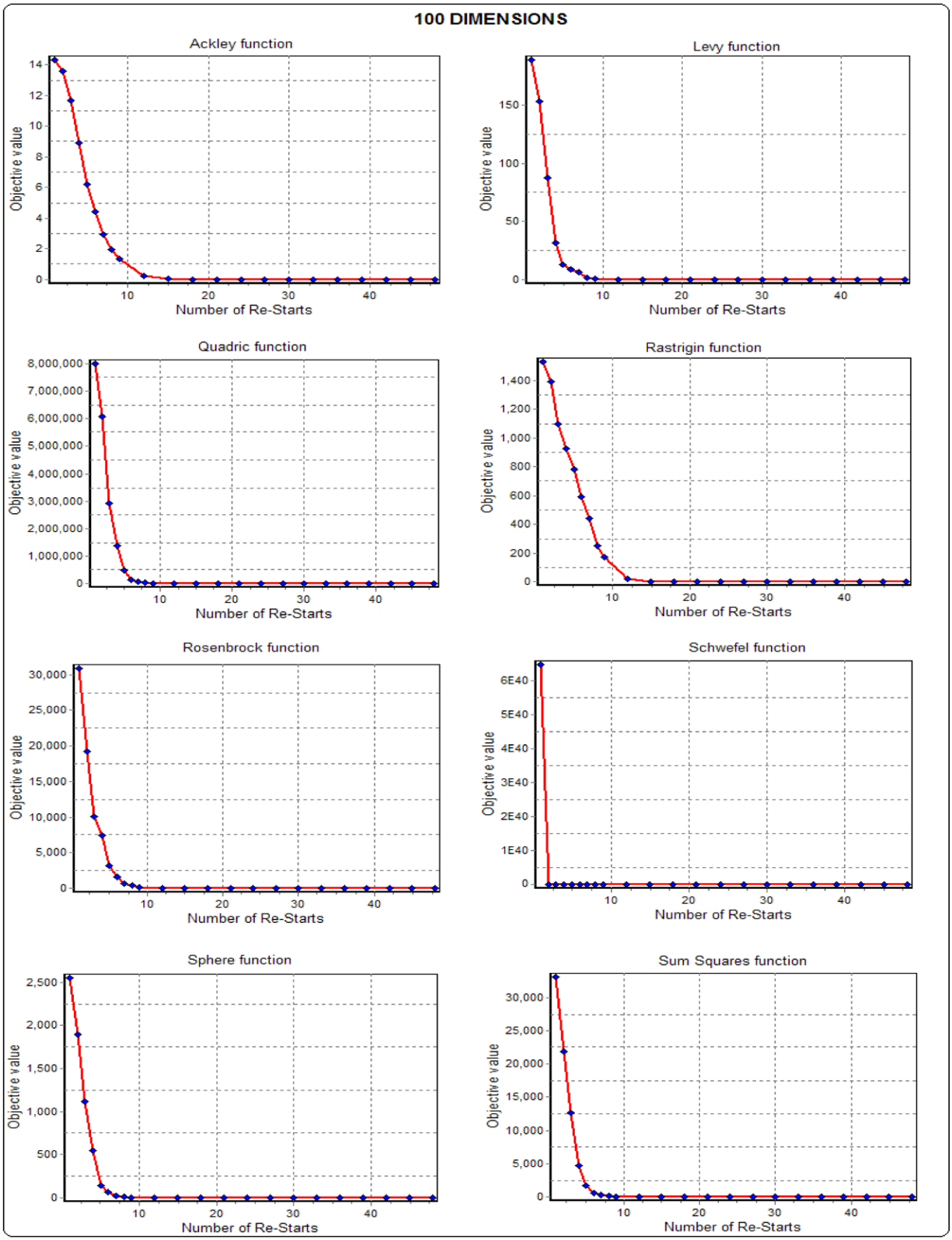

Fig. 9 LSRS convergence for 100 dimensions

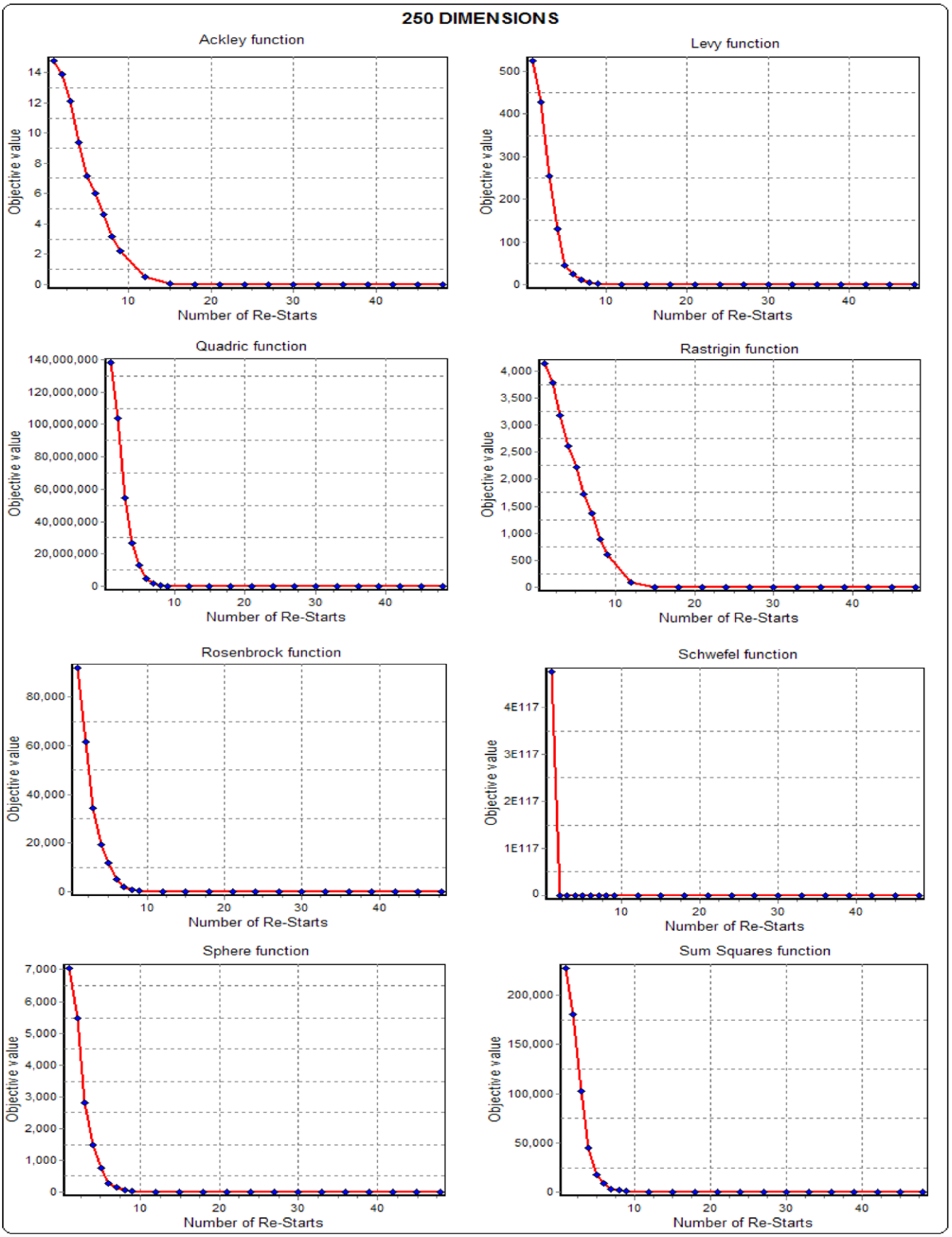

Fig. 10 LSRS convergence for 250 dimensions

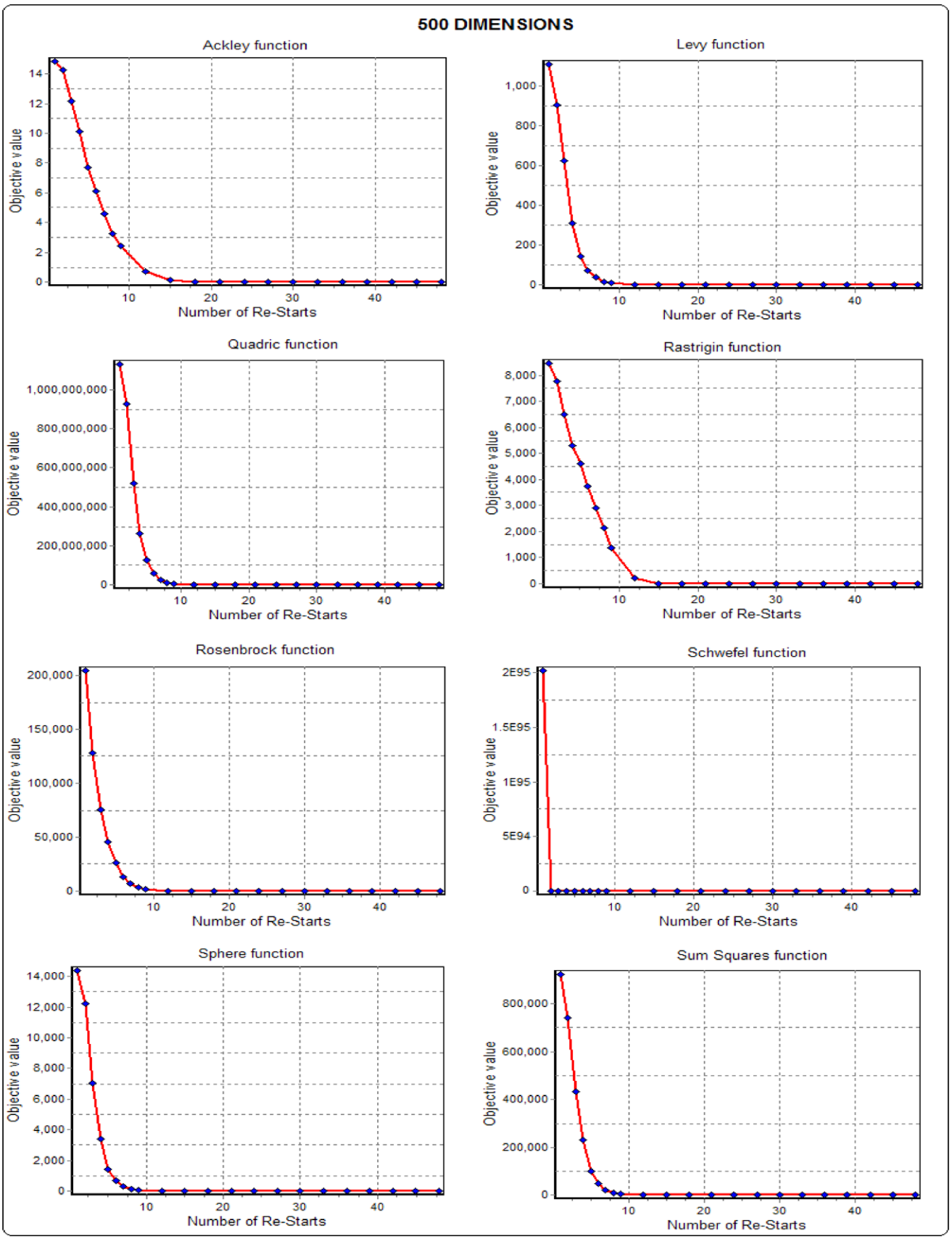

Fig. 11 LSRS convergence for 500 dimensions

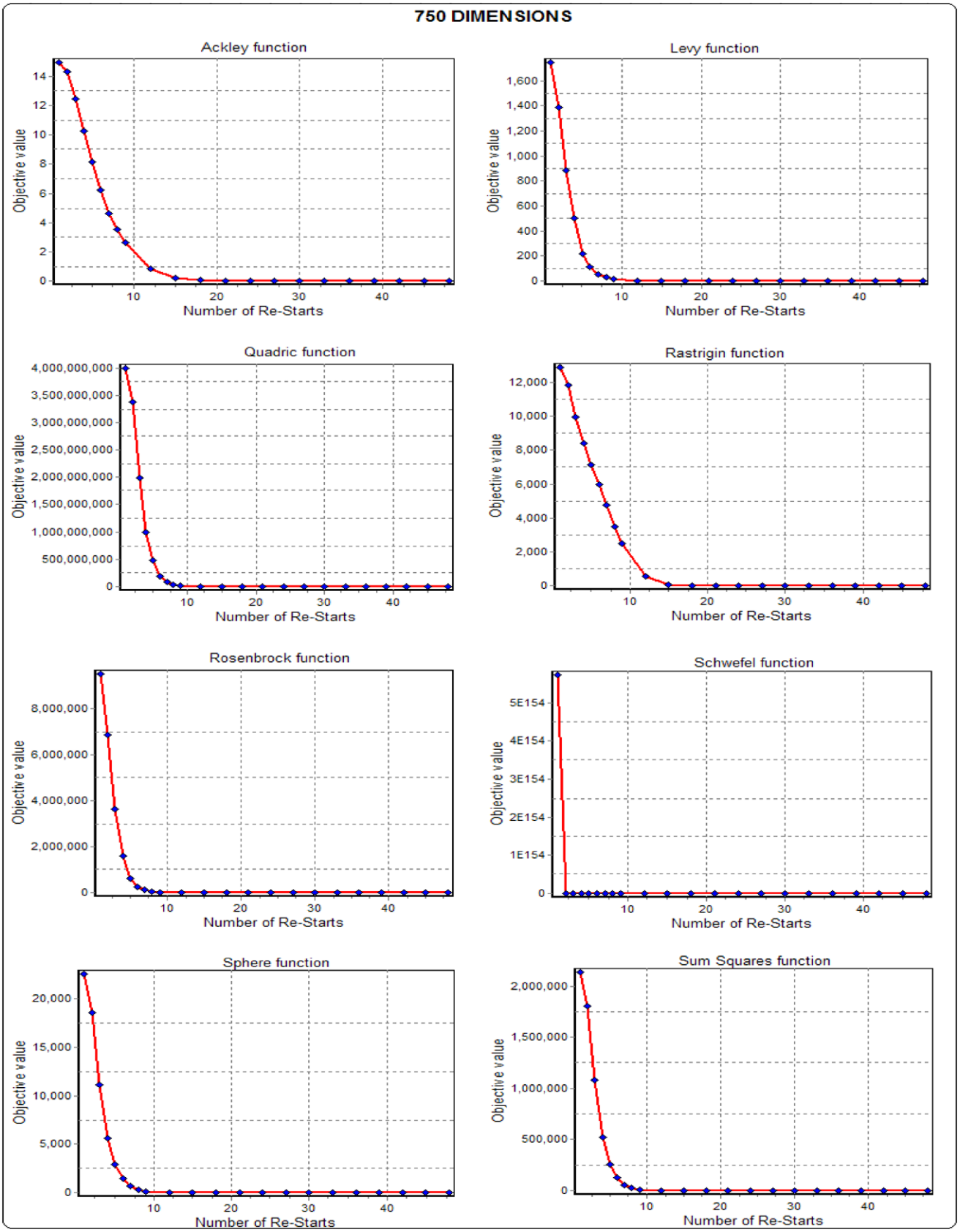

Fig. 12 LSRS convergence for 750 dimensions

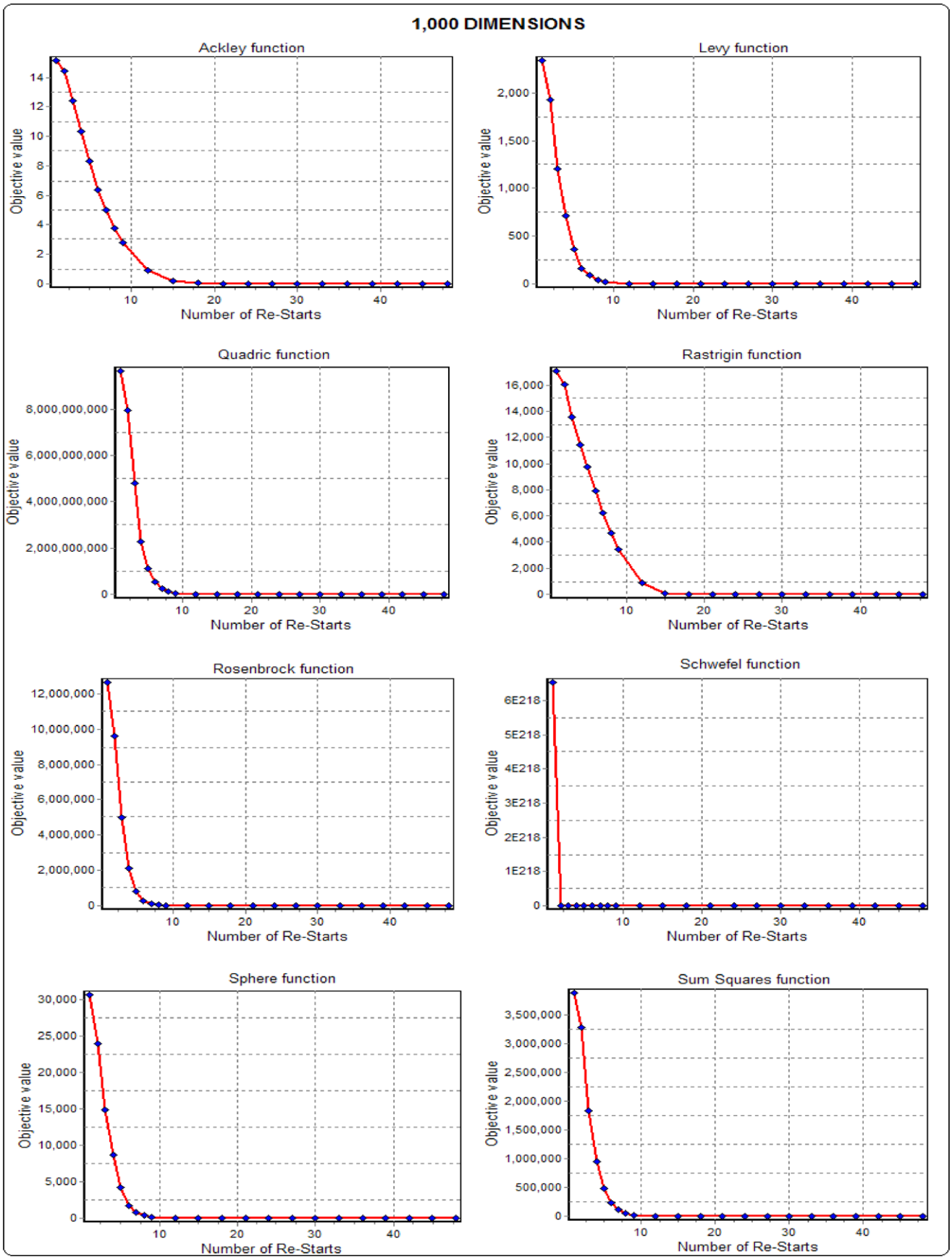

Fig. 13 LSRS convergence for 1,000 dimensions

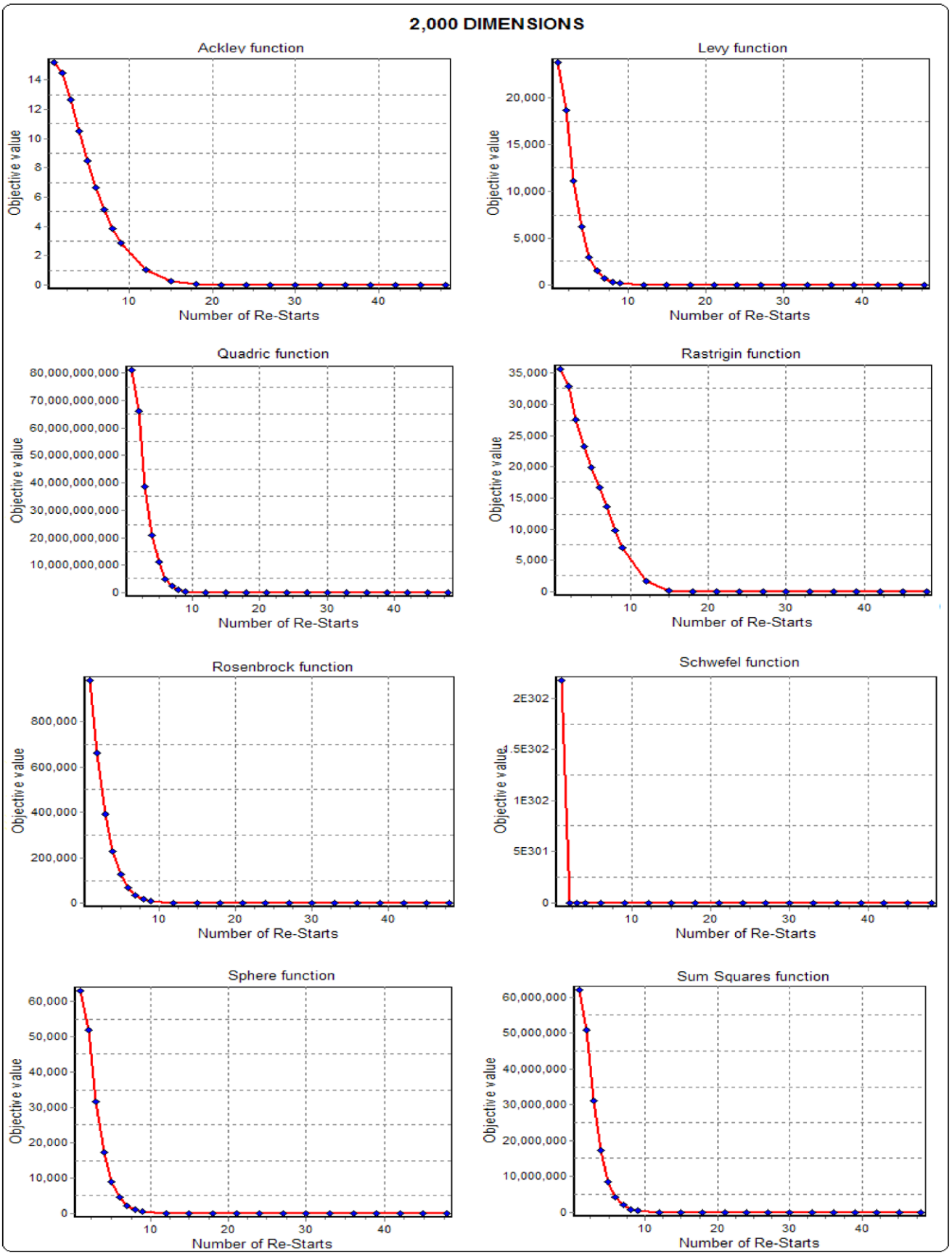

Fig. 14 LSRS convergence for 2,000 dimensions

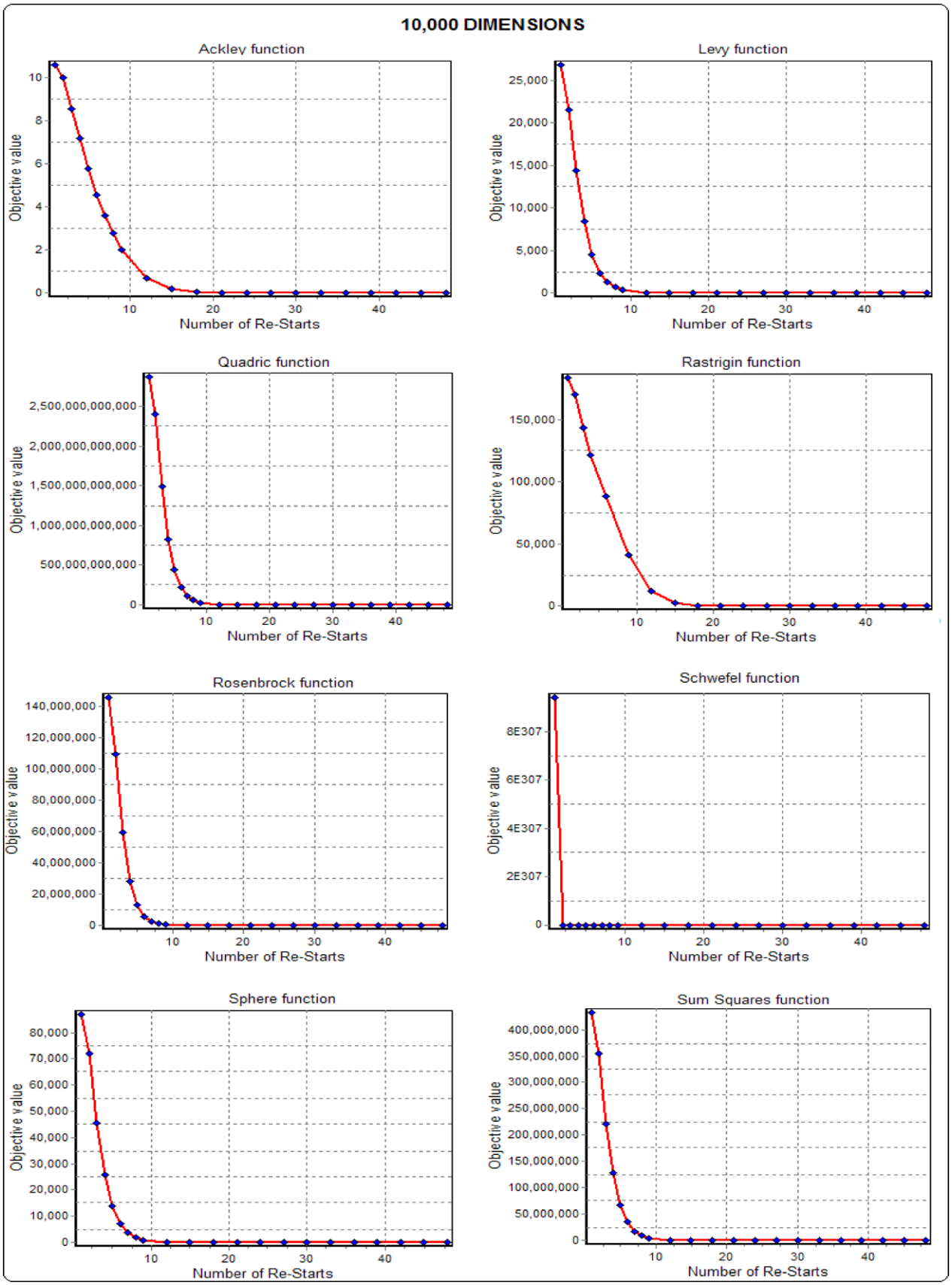

Fig. 15 LSRS convergence for 10,000 dimensions

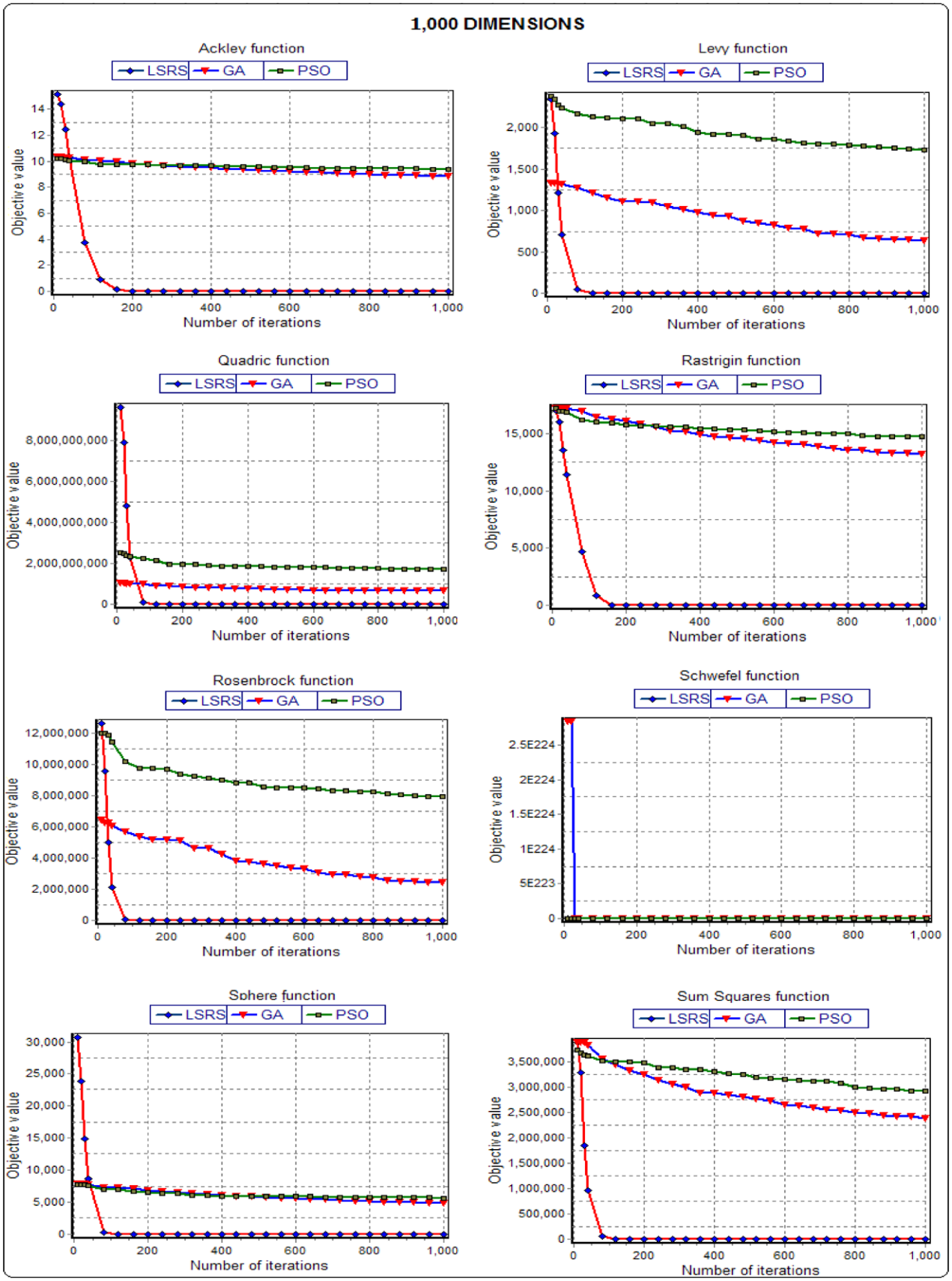

Fig. 16 Comparison of LSRS, GA and PSO for 1,000 dimensions for the first 1,000 iterations

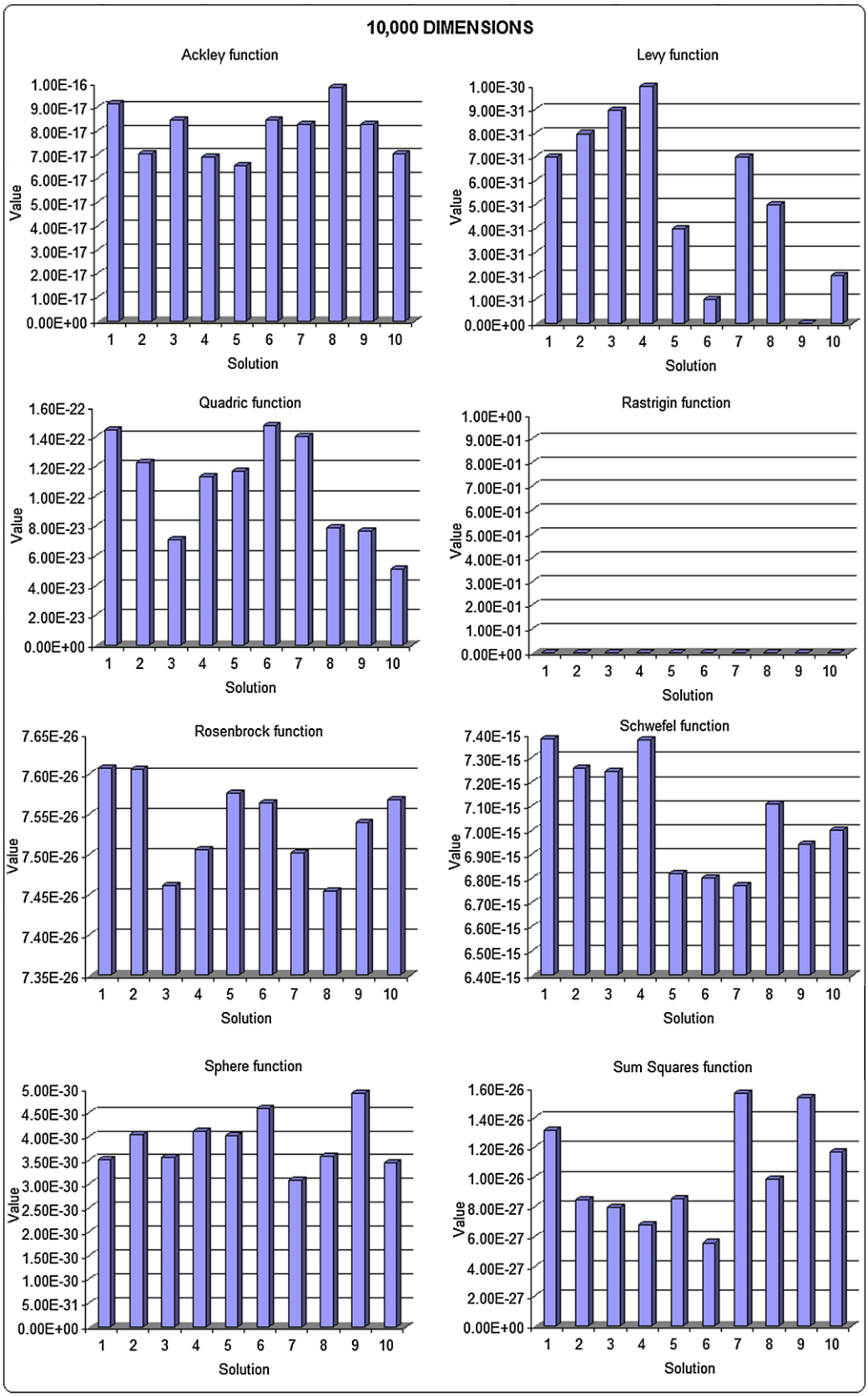

Fig. 17 The variation of the solutions obtained by LSRS at the end of search process for 10,000 dimensions (10 starting points were considered)

It takes about 5 minutes for LSRS to converge for functions having 10,000 dimensions while for GA and PSO it takes few hours even for 1,000 dimensions (due to the great number of iterations which these techniques should use in order to improve the performance). In terms of number of function evaluations LSRS is more computationally economical then PSO and GA (we refer here to objective function as well as derivatives).

4.6 Strengths and weakness of LSRS algorithm

It is well known that any approach for any kind of problems does have some weak and strong merits. There is no single algorithm which can perform perfect (or be the best) for all possible classes of problems. This is in accordance with the no free lunch theorem, which explains that for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class [36]. In sum, one should be sceptical of claims in the literature on optimization algorithms that one being proposed is substantially better than most others. Such claims are often defended through some simulations based on applications in which the proposed algorithm performed better than some familiar alternative.

We summarize some of the strengths and weakness of the proposed approach and also discuss about its suitability and inadequacy respectively for some classes of problems.

Strong points:

- It can be applied for greater number of dimensions (greater than 10,000) due to lower computational time compared to other approaches. For 10,000 variables, LSRS converges in about 5 minutes on average.
- It is simple to implement and use.
- It uses similar parameters for dimensions varying between 250 and 10,000, which means the technique is not much depending on the number of variables.
- If the computational tools are powerful enough, can be extended and applied for any number of dimensions.
- All the starting points converge to the optimum point (the technique is built like that).
- It can be effectively used for functions having a high number of local optima. This is due to the fact that the search starts with a set of initial points and there will be at least a subset which will avoid getting trapped in some of the local optima. Almost all considered test functions are highly multimodal.

Weak points:

- Require the gradient information of the function which might require more time to compute in case if the function is having different partial derivatives for each dimension. This implies it is highly suitable for functions for which the partial derivatives are having the same equation or there are at least some partial derivatives having the same equation. But if we are taking into consideration the existing mathematical software, which can easily calculate the partial derivatives this is not such a big inconvenience.
- It is restricted to a special class of functions which are continuous differentiable.
- It is more suitable for smaller domains of definition (ranges); for bigger domains LSRS might require a greater number of re-starting procedures (the interval could be reduced slowly).

5 Conclusions

A new multi-dimensional method for solving unconstraint global optimization problems is proposed in this paper. We introduced a modified line search technique called Line Search Re-Start (LSRS). The modified classical mathematical technique which incorporates the multi start method and using the re-starting techniques is found to be computationally efficient for large dimensional functions optimization problems. The computational comparisons with two well known global optimization techniques—Genetic Algorithms and Particle Swarm Optimization—clearly illustrates the superiority of the proposed approach and its independence for the number of dimensions involved.

The LSRS technique makes use of gradient information, which makes it to be restricted to a special class of functions (continuous and differentiable). But it appears to be very efficient for multi-dimensional optimization problems. We presented the empirical results and graphical illustrations for functions having up to 10,000 variables. The proposed technique can be used without any modifications for a higher number of variables.

We also intent to develop and adapt the LSRS technique so that it can be further applied for constraint optimization problems which are also of great interest in some practical optimization problems.

Acknowledgements C. Grosan acknowledges the support from the research grant CNCSIS IDEI 2412/2009.

References

1. Addis, A., & Leyffer, S. (2006). A trust-region algorithm for global optimization. *Computational Optimization and Applications*, *35*, 287–304.
2. Bäck, T., Fogel, D., & Michalewicz, Z. (1997). *Handbook of evolutionary computation*. New York: IOP Publishing and Oxford University Press.
3. Bäck, T., Fogel, D., & Michalewicz, Z. (2000). *Evolutionary computation 1: basic algorithms and operators*. Bristol: IOP Publishing.

4. Baritompa, B., & Hendrix, E. M. T. (2005). On the investigation of stochastic global optimization algorithms. *Journal of Global Optimization*, *31*(4), 567–578.
5. Bomze, I. M., Csendes, T., Horst, R., & Pardalos, P. M. (Eds.) (1996). *Developments in global optimization*. Dordrecht/Boston/London: Kluwer Academic.
6. Byrd, R. H., Dert, C. L., Rinnooy Kan, A. H. G., & Schnabel, R. B. (1990). Concurrent stochastic methods for global optimization. *Mathematical Programming*, *45*(1–3), 1–29.
7. Dixon, L. C. W., & Szegö, G. P. (Eds.) (1978). *Towards global optimization 2*. Amsterdam: North-Holland.
8. Eberhart, R. C., & Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micromachine and human science*, Nagoya, Japan, 1995 (pp. 39–43).
9. Emmerich, M. T. M., Giannakoglou, K. C., & Naujoks, B. (2006). Single and multiobjective evolutionary optimization assisted by Gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation*, *10*(4), 421–439.
10. Floudas, C. A., & Pardalos, P. M. (1990). A collection of test problems for constrained global optimization algorithms. In G. Goods & J. Hartmanis (Eds.) *Lecture notes in computer science: Vol. 455*. Berlin: Springer.
11. Floudas, C. A., & Pardalos, P. M. (Eds.) (2001). *Encyclopaedia of optimization*. Dordrecht: Kluwer Academic.
12. Floudas, C. A., & Pardalos, P. M. (Eds.) (2003). *Frontiers in global optimization*. Dordrecht: Kluwer Academic.
13. Gergel, V. (1997). A global optimization algorithm for multivariate functions with Lipschizian first derivatives. *Journal of Global Optimization*, *10*, 257–281.
14. Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading: Addison-Wesley.
15. Grosan, C., & Oltean, M. (2005). Adaptive representation for single objective optimization. *Soft Computing*, *9*(8), 594–605.
16. Hedar, A., & Fukushima, M. (2004). Heuristic pattern search and its hybridization with simulated annealing for nonlinear global optimization. *Optimization Methods and Software*, *19*, 291–308.
17. Hedar, A. R., & Fukushima, M. (2006). Tabu search directed by direct search methods for nonlinear global optimization. *European Journal of Operations Research*, *170*, 329–349.
18. Hirsch, M. J., Meneses, C. N., Pardalos, P. M., & Resende, M. G. C. (2007). Global optimization by continuous grasp. *Optimization Letters*, *1*, 201–212.
19. Hofinger, S., Schindler, T., & Aszodi, A. (2002). Parallel global optimization of high-dimensional problems. In D. Kranzlmuller et al. (Eds.) *Lecture notes in computer science: Vol. 2474*. *Euro PVM/MPI* (pp. 148–155). Berlin: Springer.
20. Holland, J. H. (1975). *Adaptation in natural and artificial system. Ann Arbor*. Michigan: The Michigan University Press.
21. Horst, R., & Tuy, H. (1996). *Global optimization—deterministic approaches*. Berlin/Heidelberg/New York: Springer.
22. Horst, R., & Pardalos, P. M. (Eds.) (1995). *Handbook of global optimization*. Dordrecht/Boston/London: Kluwer Academic.
23. Hu, X., Shi, Y., & Eberhart, R. C. (2004). Recent advances in particle swarm. In *Proceedings of congress on evolutionary computation (CEC)*, Portland, Oregon, 2004 (pp. 90–97).
24. Ismael, A., Vaz, F., & Vicente, L. N. (2007). A particle swarm pattern search method for bound constrained global optimization. *Journal of Global Optimization*.
25. Kennedy, J. (1997). The particle swarm: social adaptation of knowledge. In *Proceedings of IEEE international conference on evolutionary computation*, Indianapolis, Indiana (pp. 303–308). Berlin: Springer.
26. Kennedy, J. (1997). Minds and cultures: particle swarm implications. In *Socially intelligent agents* (Technical Report FS-97-02) (pp. 67–72). Menlo Park: AAAI Press. Papers from the 1997 AAAI Fall Symposium.
27. Kennedy, J. (1998). The behavior of particles. In *Proceedings of 7th annual conference on evolutionary programming*, San Diego, USA, 1998.
28. Kennedy, J. (1998). Thinking is social: experiments with the adaptive culture model. *Journal of Conflict Resolution*, *42*, 56–76.
29. Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization, In *Proceedings of IEEE international conference on neural networks*, 1995 (pp. 1942–1948).
30. Kennedy, J., Eberhart, R., & Shi, Y. (2001). *Swarm intelligence*. San Mateo, San Diego: Morgan Kaufmann, Academic Press.
31. Koumousis, V. K., & Katsaras, C. P. (2006). A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Transactions on Evolutionary Computation*, *10*(1), 19–28.
32. Krishnakumar, K. (1989). Micro-genetic algorithms for stationary and nonstationary function optimization. In *Proceedings of SPIE intelligent control adaptive systems* (pp. 289–296). Bellingham: SPIE Press.
33. Liang, J. J., Qin, A. K., Suganthan, P. N., & Baskar, S. (2006). Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation*, *10*(3), 281–295.
34. Liu, H., Abraham, A., & Zhang, W. (2007). A fuzzy adaptive turbulent particle swarm optimization, *International Journal of Innovative Computing and Applications*, *1*(1).
35. Maaranen, H., Miettinen, K., & Penttinen, A. (2007). On initial populations of a genetic algorithm for continuous optimization problems. *Journal of Global Optimization*, *37*, 405–436.
36. Macready, W. G., & Wolpert, D. H. (1997). The no free lunch theorems. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82.
37. Migdalas, A., Pardalos, P. M., & Storoy, S. (Eds.) (1997). *Parallel computing in optimization*. Dordrecht: Kluwer Academic.
38. Moré, J. J., Garbow, B. S., & Hillstrom, K. E. (1981). Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, *7*(1), 17–41.
39. Pardalos, P. M., & Rosen, J. B. (Eds.) (1990). Computational methods in global optimization. *Annals of Operations Research*, *25*.
40. Parsopoulos, K. E., & Vrahitis, M. N. (2002). Recent approaches to global optimization problems through Particle Swarm Optimization. *Natural Computing*, *1*, 235–306.
41. Pintér, J. D. (1996). *Global optimization in action*. Dordrecht/Boston/London: Kluwer Academic.
42. Schutte, J. F., Reinbolt, J. A., Fregly, B. J., Haftka, R. T., & George, A. D. (2004). Parallel global optimization with the particle swarm algorithm. *International Journal for Numerical Methods in Engineering*, *61*, 2296–2315.
43. Stepanenco, S., & Engels, B. (2007). Gradient tabu search. *Journal of Computational Chemistry*, *28*(2), 601–611.
44. Törn, A. A., & Zilinskas, A. (1989). In *Lecture notes in computer science: Vol. 350*. *Global optimization*. Berlin: Springer.
45. Trafalis, T. B., & Kasap, S. (2002). A novel metaheuristic approach for continuous global optimization. *Journal of Global Optimization*, *23*, 171–190.

Crina Grosan received B.S. and M.S. degrees in mathematics and a Ph.D. in computer science from Babes-Bolyai University, Cluj-Napoca, Romania in 2005. She is currently a lecturer of artificial intelligence in the Department of Computer Science, Babes-Bolyai University. Her recent research interests include optimization, mathematical programming, numerical analysis, computational intelligence, and computational biology. Dr. Grosan has over 100 scientific publications including over 25 journal articles/book chapters and 6 books written or edited. She serves on the editorial board of a number of journals and on the program committee of several international conferences.

Ajith Abraham (M'96–SM'07) received the Ph.D. degree from Monash University, Melbourne, Australia, in 2001. He currently directs the activities of the Machine Intelligence Research Labs (MIR Labs), Scientific Network for Innovation and Research Excellence, which has representation in nearly 60 countries. He has worldwide academic experience of nearly ten years with formal appointments in Monash University, Australia; Oklahoma State University; Chung-Ang University, Seoul; Jinan University, China; Rovira i Virgili University, Spain; Dalian Maritime University, China; Yonsei University, Seoul; Open University of Catalonia, Spain; and NTNU, Norway. For about two and a half years, he was working under the Institute of Information Technology Advancement Professorship Program funded by the South Korean Government. He was working with three multinational companies: Keppel Engineering, Singapore; Hyundai Engineering, Korea; and Ashok Leyland Ltd, India where he was involved in different industrial research and development projects for nearly eight years. He has authored or coauthored more than 500 research publications in peer-reviewed reputed journals, book chapters, and conference proceedings. His primary research interests are in advanced computational intelligence, with a focus on using global optimization techniques for designing intelligent systems. Their application areas include Web services, information security, Web intelligence, social networks, financial modeling, multicriteria decision making, data mining, etc. He has given more than 25 plenary lectures and conference tutorials in these areas. Dr. Abraham Co-chairs the IEEE Systems Man and Cybernetics Society Technical Committee on Soft Computing. He is a regular reviewer of IEEE Intelligent Systems, IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Neural Networks, IEEE Transactions on Fuzzy Systems, IEEE Transactions on Evolutionary Computation, IEEE Transactions on Systems, Man, and Cybernetics, and IEEE Transactions on Power Systems. He serves on the Editorial Boards of more than 30 international journals and has also guest-edited 30 special issues on various topics for international journals. Since 2001, he has been actively involved in the hybrid intelligent systems and the Intelligent Systems Design and Applications series of annual international conferences. He was the recipient of five Best Paper Awards.

Aboul Ella Hassainen is an associated professor at Cairo University, Faculty of Computer and Information Technology Department. Currently, he is a visiting professor at Kuwait University, College of Business Administration, Quantitative and Information System Department. He has involved in the organized/Scientific committees member of many international conferences. In addition, he has served as reviewer to many international conference/journals. He has received the excellence younger researcher award from Kuwait University for the academic year 2003/2004. He works in a multi-disciplinary environment involving computational intelligence, information security, bioinformatics, medical image processing and data mining and applied to various real world problems. He has directed many funded research projects. He serves the editorial board of several reputed International journals and has also guested edited many special issues on various topics.