Soft Comput
DOI 10.1007/s00500-010-0651-6

FOCUS

EM323: a line search based algorithm for solving high-dimensional continuous non-linear optimization problems

Vincent Gardeux · Rachid Chelouah · Patrick Siarry · Fred Glover

Abstract This paper presents a performance study of a one-dimensional search algorithm for solving general high-dimensional optimization problems. The proposed approach is a hybrid between a line search algorithm of Glover (The 3-2-3, stratified split and nested interval line search algorithms. Research report, OptTek Systems, Boulder, CO, 2010) and an improved variant of a global method of Gardeux et al. (Unidimensional search for solving continuous high-dimensional optimization problems. In: ISDA '09: Proceedings of the 2009 ninth international conference on intelligent systems design and applications, IEEE Computer Society, Washington, DC, USA, pp 1096–1101, 2009) that uses line search algorithms as subroutines. The resulting algorithm, called EM323, was tested on 19 scalable benchmark functions, with a view to observing how optimization techniques for continuous optimization problems respond with increasing dimension. To this end, we report the algorithm's performance on the 50, 100, 200, 500 and 1,000-dimension versions of each function. Computational results are given comparing our method with three leading evolutionary algorithms. Statistical analysis discloses that our method outperforms the other methods by a significant margin.

V. Gardeux (✉) · R. Chelouah
EISTI, L@ris, Cergy-Pontoise, France
e-mail: vincent.gardeux@eisti.fr

P. Siarry
Université de Paris 12, LiSSi, Créteil, France

F. Glover
OptTek Systems, Inc., 2241 17th Street, Boulder, CO 80302, USA

Keywords Metaheuristic · Line search · Optimization · Continuous · High-dimension

1 Introduction

We address the continuous non-linear function optimization problem in the form:

$$\text{Minimize } f(x) : L \leq x \leq U \tag{1}$$

where $x = (x_1, \ldots, x_n)$ is a vector of real-valued variables, and the vectors L and U are assumed finite and to satisfy $L < U$. The function $f(x)$ is characteristically assumed to be multimodal, so that local optima do not in general correspond to global optima. We denote the index set for the components of x, L and U by $N = \{1, \ldots, n\}$. There has recently been considerable interest in solving instances of (1) of large dimensions (i.e., large values of n), motivated by the emergence of applications in bio-computing, web and data mining (Lee 2007; Nasiri et al. 2009).

Many current optimization techniques can efficiently solve instances of (1) of small to moderate dimension, involving up to 50 variables. However, most of them are not well designed to efficiently solve high-dimensional problems, particularly those containing more than 100 variables. Moreover, the performance of these algorithms decreases with increasing dimension. We address the challenge of solving higher dimensional problems by proposing an algorithm that can be applied to any black box function without requiring Hessian or gradient information. Our goal is to produce a method that is both dimensionally robust and easy to implement. As a foundation for doing this, we have adopted a decomposition theme, which we carry to the farthest extreme by decomposing the n-dimensional problem into n one-dimensional problems,

Published online: 10 September 2010

optimizing the objective function on each dimension separately.

One-dimensional optimization is a classical problem. One of the first algorithms for solving it is Newton's method (1669). The family of one-dimensional algorithms to which Newton's method belongs, called *Line Search*, is ideally suited to provide subroutines for solving multidimensional problems. This type of optimization is particularly used in the steepest descent method which computes a direction according to the derivative of the objective function, as exemplified by the approach of Hestenes and Stiefel (1952). Contrary to this type of approach, however, our proposed method does not make use of an objective function derivative and does not try to approximate one. On high-dimensional problems, the calculation of a derivative or its approximation can be highly time-consuming. Other algorithms that have made explicit use of a one-dimensional optimization algorithm as a component strategy include those of Grosan and Abraham (2007) and Tseng and Chen (2008).

In order to adapt a line search algorithm to a multidimensional problem, we employ a simple relaxation method that takes the unit vectors $(e_1; e_2; \ldots; e_n)$ to provide a set of directions. Within this setting, line search operates by moving along the first direction until reaching a minimum (maximum), then along the second direction to its minimum (maximum) point, and so on, cycling through the set of directions as many times as necessary.

Our approach incorporates a fast global optimization procedure based on the EUS (Enhanced Unidirectional Search) algorithm of Gardeux et al. (2009). EUS is parameter-free and designed to converge quickly to a local optimum. We combine EUS with the 3-2-3 line search procedure, recently developed by Glover (2010).

In order to adapt this algorithm to non-separable problems, we enhanced the global relaxation component of EUS to incorporate several features that will be described later. In order to handle multimodal objective functions, which pose the danger of entrapping the search in a local optimum, we use a restart procedure based on a dispersion algorithm inspired by the scatter search (SS) algorithm of Marti et al. (2006). The SS algorithm component we employ computes the dispersion over all local optima found, in order to avoid re-initializing the solution near local optima already obtained. Our proposed approach is tested on a suite of 19 scalable functions and compared to three evolutionary algorithms that constitute leading methods for this class of problems. Analysis using the Wilcoxon statistical test discloses that our method outperforms the other methods by a statistically significant margin. The remainder of the paper is organized as follows. Section 2 describes the two main algorithms that we have combined to create EM323, whose detailed form is then presented in Sect. 3. Section 4 describes the experimental setup, followed by computational comparisons of our method with the three evolutionary algorithms in Sect. 5. Finally, our conclusions and directions for future work are given in Sect. 6.

2 Two algorithms based on line search: EUS and 3-2-3

2.1 Line search

Line search problems may be expressed as that of minimizing $f(y)$ on a line segment denoted by $\mathrm{LS}(x', x'')$, which passes through points x' and x'', where

$$\begin{aligned}\mathrm{LS}(x', x'') = \{x = x(\theta) : x(\theta) \\ = x' + (x'' - x')\theta \text{ for } \theta_{\min} \le \theta \le \theta_{\max}\}\end{aligned} \tag{2}$$

The values $\theta_{\min}$ and $\theta_{\max}$ are computed so that $\mathrm{LS}(x', x'')$ lies within a bounded convex region defining a set of feasible solutions either for an original problem of interest or for a mathematical relaxation of such a problem. Hence the points x' and x'' are not necessarily endpoints of $\mathrm{LS}(x', x'')$.

2.2 EUS: enhanced unidirectional search

The EUS algorithm developed by Gardeux et al. (2009) is designed to be easy to implement and robust in handling high-dimensional problems. It is a global method inspired from a classical relaxation method, using a simple line search procedure on each dimension successively. The algorithm starts from a randomly generated initial solution x. A difference ("delta") vector δ is created and initialized by setting $\delta = U - L$, followed by shrinking the size of δ on subsequent iterations and continuing until a stopping criterion is reached.

At each iteration, the algorithm focuses on optimizing the current solution x on only one dimension i, by allowing only the component x_i to be changed. The method then selects the best neighbor from the two alternatives "*x up*" and "*x down*" given by

$$x^u = x + \delta_i e_i \tag{3a}$$

$$x^d = x - \delta_i e_i \tag{3b}$$

where e_i is the unit vector with a 1 in position i and 0's elsewhere. (Hence x^u and x^d are the same as x except for the ith component, where $x_i^u = x_i + \delta_i$ and $x_i^d = x_i - \delta_i$.) The value x_i^u is reset to U_i if $x_i + \delta_i > U_i$ and x_i^d is reset to L_i if $x_i - \delta_i < L_i$. The search compares x with its two neighbor solutions x^u and x^d and updates x to be the best of these, hence setting $x = arg\ \min(f(x), f(x^u), f(x^d))$. Successive dimensions i are treated in the same manner.

```
Procedure EUS
$\delta = U - L$
begin
    for $i = 1$ to $n$
        $x^u = x + \delta_i e_i$
        $x^d = x - \delta_i e_i$
        (update x to be the best of the 3 solutions)
        $x = argmin(f(x), f(x^u), f(x^d))$
    end
    if no improvement has been found
        $\delta = \delta * 0.5$ until $\delta < \delta_{min}$
    end
end
```

Fig. 1 An iteration of the EUS algorithm

Each iteration consists of examining every dimension i, and at its conclusion the algorithm finds a restricted local optimum relative to the precision given by the vector δ. (We call this local optimum restricted, since if any change is produced during the iteration it is possible that a better solution could be produced by a new pass of the dimension i.) We then test to determine whether the solution has been improved on at least one dimension, and if not, δ is multiplied by a ratio value fixed to 0.5, therefore shrinking the size of its components. The δ vector continues to shrink in this fashion until it satisfies $\delta < \delta_{\min}$, whose components are all fixed to $1e-15$ in order to obtain a suitable precision. The parameter ratio can be tuned but experiments show that the fixed value of 0.5 is suitable.

The pseudo-code in Fig. 1 details an iteration of the algorithm. The EUS algorithm obtains good results not only on separable functions but on non-separable functions too. It does not make an intensive optimization on each dimension but only a quick approximation at each pass. So on the next pass, the function to optimize relative to a given dimension can change due to changes that have occurred on other dimensions. But complex non-separable functions, which have many interrelations between the variables, remain difficult to optimize in reasonable time. For such problems the convergence of the EUS method is too slow.

2.3 The restart procedure

The algorithm is not a population-based method, so to avoid becoming trapped in a local minimum, we use a restart procedure that keeps the best solution found so far and re-initializes δ to a new starting value after reaching the termination point given by $\delta < \delta_{\min}$. Increasing the vector $\delta_{\min}$ may increase the potential number of restarts of the algorithm, but tends to decrease the error accuracy. In order to better explore the search space, when the restart procedure is activated, a new solution is generated, that lies far from a reference set. This technique uses the diversification generation method from the Scatter Search algorithm of Marti et al. (2006). Each time the termination point is achieved, the solution found (s_i) is added to a reference set S of previously visited restricted local optima. The restart procedure then generates a collection of diverse trial solutions in the search space as in the Marti et al. approach and selects the one farthest from S. The distance dist used is a classic distance between a point and a set:

$$\mathrm{dist}(x, S) = \inf\{\mathrm{euclideanDist}(x, s)/s \in S\} \tag{4}$$

2.4 A line search algorithm: 3-2-3

We decided to replace the simple examination of the two neighbors x^u and x^d used in EUS with a more effective determination of candidates to replace the current solution x by making use of a line search procedure. For this, we selected the 3-2-3 procedure which has a structure that is highly suited for exploitation by our general design. The procedure begins by adopting an approach commonly used in non-linear line search procedures, which consists in identifying a succession of points $x(\theta_0), x(\theta_1), \ldots, x(\theta_s)$ on $LS(x', x'')$, for $s \geq 2$ such that

$$\theta_0 < \theta_1 < \cdots < \theta_s, \quad \text{where } \theta_0 = \theta_{\min}, \theta_s = \theta_{\max} \tag{5}$$

The θ_h values are parameters that identify the location of x (or more particularly $x(\theta_h)$) on the line segment $LS(x', x'')$ joining x' and x''. We may suppose, for example, that $x' = x(0)$ and $x'' = x(1)$, with $\theta_{\min}$ and $\theta_{\max}$ selected to satisfy $0 \leq \theta_{\min} < \theta_{\max} \leq 1$. A straightforward way to generate the θ_h values is to subdivide the interval $[\theta_{\min}, \theta_{\max}]$ into s equal subintervals so that

$$\theta_h = \theta_{h-1} + \Delta (= \theta_0 + h\Delta) \text{ for } h = 1, \ldots, s, \quad \text{where } \Delta = \frac{\theta_{\max} - \theta_{\min}}{s} \tag{6}$$

The 3-2-3 method focuses on sequences of points on the subdivided line defined by reference to the θ values that have either the form of a pair $(x(\theta_{h-1}), x(\theta_h))$ for $1 \leq h \leq s$ or a triple $(x(\theta_{h-1}), x(\theta_h), x(\theta_{h+1}))$ for $1 \leq h \leq s-1$. (Hence the points $x(\theta_0)$ and $x(\theta_s)$ can be endpoints of intervals defined by such pairs and triples.)

The 3-2-3 algorithm starts from an initial construction similar to that used by Golden Section methods for unimodal function optimization on a line, consisting of triples $(x(\theta_{h-1}), x(\theta_h), x(\theta_{h+1}))$ but allowing more than one starting interval and more complex ways for operating on the intervals considered. The approach can be particularly appropriate when the distances between successive points $x(\theta_h)$ are relatively small, or when the goal is to refine a coarse search of the line by focusing more thoroughly on the region around the point $x(\theta_q)$.

We select a current instance of such a triple $(x(\theta_{q-1}), x(\theta_q), x(\theta_{q+1}))$ by requiring that it satisfies

$$f(x(\theta_q)) \leqslant f(x(\theta_{q-1})) \text{ and } f(x(\theta_q)) \leqslant f(x(\theta_{q+1})) \qquad (7)$$

Such a triple always exists unless the local optima from among the points $x(\theta_0), \ldots, x(\theta_s)$ occur only at one or both of the points $x(\theta_0)$ and $x(\theta_s)$. Barring this exceptional case, the 3-2-3 algorithm is applied to each triple satisfying (7) for which $f(x(\theta_q))$ does not exceed the globally minimum $f(x(\theta))$ value over the points $x(\theta_1), \ldots, x(\theta_s))$ by more than a relatively small amount. To avoid duplicated effort, each point denoted by $x(\theta_q)$ is only permitted to lie in one of the chosen sequences. If the exceptional case exists that prevents (7) from holding, however, the 3-2-3 algorithm is applied by first launching a preliminary algorithm called the 2-1-2 algorithm.

In both the preliminary 2-1-2 algorithm and the main 3-2-3 algorithm, we initialize $x^* = x(\theta_q)$, where x^* denotes a candidate for a "best solution" obtained by the search. The three starting cases are summarized as follows, where Case 1 and Case 2 are only considered in the exceptional situation where no triple exists satisfying (7):

- Case 1: $f(x(\theta_0))$ is a global minimum over the points $x(\theta_0), \ldots, x(\theta_s)$ (and $f(x(\theta_1))$ is not). Let $x^a = x(\theta_0), x^c = x(\theta_1)$ [hence $f(x^a) < f(x^c)$]. Set $x^* = x^a$ and execute the preliminary 2-1-2 algorithm.
- Case 2: $f(x(\theta_s))$ is a global minimum over the points $x(\theta_0), \ldots, x(\theta_s)$ [and $f(x(\theta_{s-1}))$ is not]. Let $x^a = x(\theta_s), x^c = x(\theta_{s-1})$ [hence $f(x^a) < f(x^c)$]. Set $x^* = x^a$ and execute the preliminary 2-1-2 algorithm.
- Case 3: (7) is satisfied for a selected q such that $1 \leq q \leq s-1$. Let $x^a = x(\theta_{q-1}), x^b = x(\theta_q), x^c = x(\theta_{q+1})$ (hence $f(x^b) \leq f(x^a), f(x^c)$). Set $x^* = x^b$ and execute the main 3-2-3 algorithm.

The preliminary algorithm is called the "2-1-2" algorithm because each iteration starts with 2 points, generates 1 new point, and then discards one of the starting points to end with 2 points (one being the new point). The pseudo-code in Fig. 2 details this preliminary algorithm, where *maxIter* is a parameter representing the stopping criterion. Similarly, the main algorithm is called the "3-2-3" algorithm because each iteration starts with 3 points, generates 2 new points, and then discards 2 points to end with 3 points (including at least one of the new points). Viewed from the perspective of intervals, the algorithm might be called the "2-4-2" algorithm, because each iteration starts with the 2 adjacent intervals $[x^a, x^b], [x^b, x^c]$, expands them to produce the 4 adjacent subintervals $[x^a, x^{a_1}], [x^{a_1}, x^b], [x^b, x^{b_1}], [x^{b_1}, x^c]$, and then finally shrinks the latter collection to again obtain 2 adjacent intervals. If the distances $d(x^a, x^b)$ and $d(x^b, x^c)$ that define the lengths of the two initial intervals $[x^a, x^b]$ and $[x^b, x^c]$ are the same, then the lengths of the two intervals at the end of an iteration will be half the lengths of the two intervals at the start of the iteration. The pseudo-code in Fig. 3 details an iteration of the 3-2-3 line search algorithm.

```
Procedure 2-1-2
$x^* = x^a$
begin
    (Start with and maintain $f(x^a) < f(x^c)$)
    while $iter < maxIter$
        $iter = iter + 1$
        $x^b = 0.5 * (x^a + x^c)$
        if $f(x^b) \leq f(x^a)$ then
            (($x^a$,$x^b$,$x^c$) has the proper form for the 3-2-3
            algorithm)
            $x^* = x^b$
            Terminate and execute the 3-2-3 procedure
        else
            ($f(x^a) < f(x^b)$)
            Designate ($x^a$,$x^b$) to be the new ($x^a$,$x^c$)
        end if
    end while
end
```

Fig. 2 Preliminary 2-1-2 algorithm

```
Procedure 3-2-3
$x^* = x^b$
begin
    (Start with and maintain $f(x^b) \leq f(x^a), f(x^c)$)
    $x^{a_1} = 0.5 * (x^a + x^b)$
    $x^{b_1} = 0.5 * (x^b + x^c)$
    if $f(x^b) \leq f(x^{a_1})$ and $f(x^b) \leq f(x^{b_1})$ then
        Designate ($x^{a_1}$,$x^b$,$x^{b_1}$) to be the new ($x^a$,$x^b$,$x^c$)
    else if $f(x^{a_1}) \leq f(x^{b_1})$ then
        $x^* = x^{a_1}$
        Designate ($x^a$,$x^{a_1}$,$x^b$) to be the new ($x^a$,$x^b$,$x^c$)
    else
        $x^* = x^{b_1}$
        Designate ($x^b$,$x^{b_1}$,$x^c$) to be the new ($x^a$,$x^b$,$x^c$)
    end if
end
```

Fig. 3 One iteration of 3-2-3 algorithm

We illustrate the 2-1-2 and the 3-2-3 procedures in Figs. 4 and 5, respectively, using a 2-dimensional representation.

Figure 4a shows the starting configuration for the 2-1-2 procedure, where $f(x^a) < f(x^c)$. A line has been drawn connecting the points $f(x^a)$ and $f(x^c)$ to clarify their relationship, but the line has no role in the procedure itself. Remaining components of Fig. 4 include reference not only to x^a and x^c, but also to the point x^b and its function value $f(x^b)$.

Figure 4b illustrates the case where $f(x^b) \leq f(x^a)$. The lines successively joining $f(x^a)$,$f(x^b)$, and $f(x^c)$ are accentuated to indicate that this configuration qualifies as a

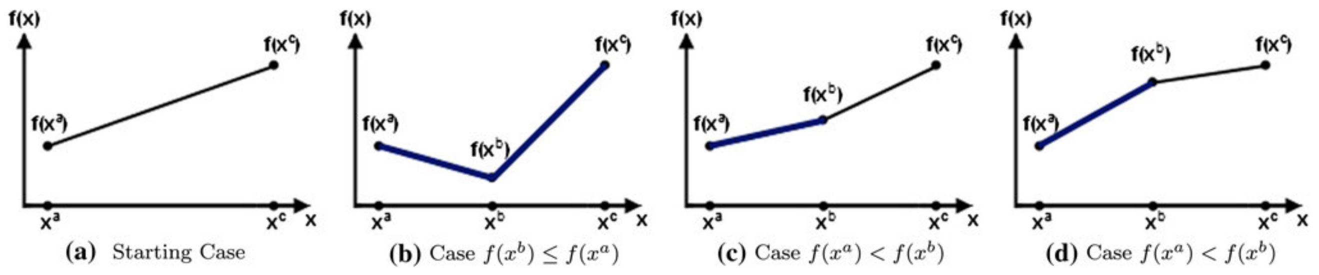
f(x)
f(x^c)
f(x^a)
x^a
x^c
x
(a) Starting Case
f(x^b)
x^b
(b) Case $f(x^b) \leq f(x^a)$
(c) Case $f(x^a) < f(x^b)$
(d) Case $f(x^a) < f(x^b)$

Fig. 4 The different cases of the 2-1-2 procedure

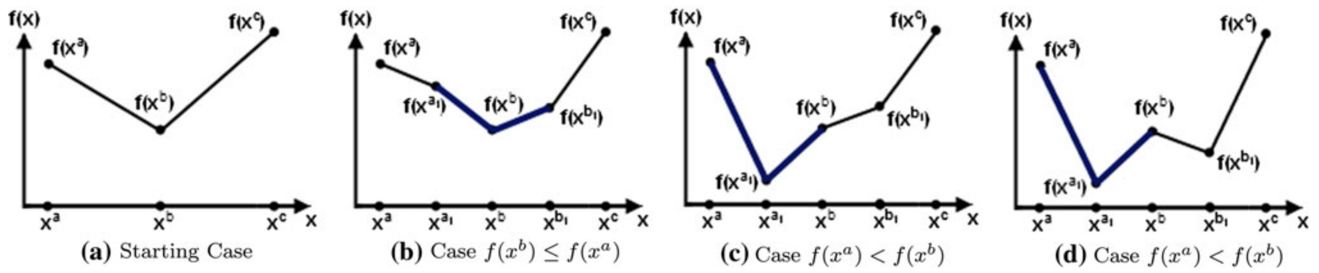
f(x)
f(x^a)
f(x^b)
f(x^c)
f(x^{a1})
f(x^{b1})
x^a
x^{a1}
x^b
x^{b1}
x^c
x
(a) Starting Case
(b) Case $f(x^b) \leq f(x^a)$
(c) Case $f(x^a) < f(x^b)$
(d) Case $f(x^a) < f(x^b)$

Fig. 5 The different cases of the 3-2-3 procedure

3-2-3 configuration, and hence the 2-1-2 procedure terminates at this point and the 3-2-3 procedure begins.

Figure 4c, d illustrate two versions of the same case, where $f(x^a) < f(x^b)$. In both instances, the line joining $f(x^a)$ and $f(x^b)$ is accentuated, to indicate that the resulting configuration qualifies as a 2-1-2 configuration. Hence the 2-1-2 procedure continues by reference to the accentuated portion of the diagram, and x^b becomes the new x^c. (A third instance of the case for $f(x^a) < f(x^b)$ is also possible, where in addition $f(x^b) > f(x^c)$. The treatment is the same as illustrated in Fig. 4c, d.)

Figure 5a shows the starting configuration for the 3-2-3 procedure, where $f(x^b) \leq f(x^a)$ and $f(x^b) \leq f(x^c)$. Again the points identifying the function values are connected by a broken line for purposes of illustration. Remaining components of Fig. 5 additionally include the points x^{a_1} and x^{b_1}, together with their function values $f(x^{a_1})$ and $f(x^{b_1})$.

Figure 5b illustrates the situation in which $f(x^b) \leq f(x^{a_1})$ and $f(x^b) \leq f(x^{b_1})$. In this case x^b retains its identity as the point having a smallest $f(x)$ value, and the sequence x^{a_1}, x^b, x^{b_1} qualifies as a 3-2-3 configuration as indicated by accentuating the lines successively joining $f(x^{a_1}), f(x^b)$, and $f(x^{b_1})$. The next iteration of the 3-2-3 procedure therefore resumes with the current x^{a_1} becoming the new x^a and the current x^{b_1} becoming the new x^c.

Figure 5c, d illustrate two versions of the case where the condition of Fig. 5b is not satisfied (hence $f(x^{a_1}) < f(x^b)$ or $f(x^{b_1}) < f(x^b)$), and in addition $f(x^{a_1}) \leq f(x^{b_1})$. Now x^{a_1} qualifies to become the new x^b, and in both of the Fig. 5c, d we have accentuated the broken line joining $f(x^a), f(x^{a_1})$ and $f(x^b)$, thus identifying the configuration that qualifies as a 3-2-3 configuration for the next iteration.

There remains the case where the conditions of both Fig. 5b, c (and 5d) are all not satisfied, and hence we have $f(x^{b_1}) < f(x^b)$ and $f(x^{b_1}) < f(x^{a_1})$. This situation is the same as the one illustrated in Fig. 5c, d with the roles of x^{b_1} and x^{a_1} interchanged, and hence we have not included an additional diagram to illustrate it.

The preceding diagrams disclose that this "basic" form of the 3-2-3 procedure neglects to examine potentially fruitful regions that might be explored more thoroughly—as in the case of Fig. 5d, where the triple (x^b, x^{b_1}, x^c) might reasonably be considered as a candidate to become the next (x^a, x^b, x^c). A more advanced version of the 3-2-3 method given in Glover (2010) is designed to handle such considerations.

3 Enhanced multidimensional 3-2-3

Our hybrid algorithm incorporates a global search procedure resembling the one used with the EUS method and replaces the EUS line search by the 3-2-3 line search algorithm to create a new algorithm called the Enhanced Multidimensional 3-2-3 (EM323), in order to solve more complex problems than EUS can handle alone. We also incorporate additional features in order to accelerate convergence of the resulting algorithm and to explore the search space more effectively.

First, we describe the global procedure that uses the Line Search algorithm, which starts by generating an initial solution x randomly. In contrast to the original EUS method, we do not proceed to a local optimum obtained by scanning all variables at each iteration. Instead we use a *strategic oscillation* approach that scans a set of dimensions N_o starting with $N_o = N(= \{1, \ldots, n\})$ and then dropping those dimensions over which the line search yields no improvement, so that the new N_o for the next iteration consists just of those dimensions that have produced a successful outcome. Let $x(i)$ and $x(i)^*$ denote the solutions x and x^* that are respectively submitted to and returned by the 3-2-3 algorithm when it is applied to dimension $i \in N_o$. Then we identify the dimensions along which the search is successful as $N_o(\text{success}) = \{i \in N_o : f(x(i)^*) < f(x(i))\}$. Consequently, at the conclusion of each iteration we update N_o by setting $N_o = N_o(\text{success})$, which gradually shrinks N_o until it becomes empty. Then we oscillate by resetting $N_o = N$ and starting over again. (This is a classical one-sided oscillation pattern. Other kinds of patterns for strategic oscillation are discussed in Glover (1995).)

The process repeats until $N_o = N$ immediately produces a set $N_o(\text{success})$ that is empty, so that all dimensions fail to yield improvement on the first iteration. At this point we refine the granularity of the 3-2-3 line search procedure, as described next, and then repeat.

We employ a vector $\delta = (\delta_1, \ldots, \delta_n)$ to determine the granularity of the 3-2-3 algorithm that is analogous to the vector δ used with the EUS algorithm. In particular, δ is used to determine the $\theta_{\max}$ and $\theta_{\min}$ values on each dimension$i$ by setting $\theta_{\max} = x_i + \delta_i$ and $\theta_{\min} = x_i - \delta_i$ so that $\theta_{\max}$ and $\theta_{\min}$ correspond to x_i^u and x_i^d. The vector δ is initialized as follows: each component δ_i is given by

$$\delta_i = (U_i - L_i) * \text{rand} \tag{8}$$

where rand is a random real value from the interval [0,1]. In order to choose the value s of the 3-2-3 algorithm, we use a parameter NCut and set $s = \text{NCut}$, hence yielding (by (6))

$$\Delta = (\theta_{\max} - \theta_{\min})/\text{NCut} \tag{9}$$

with the result of subdividing the interval from $\theta_{\min}$ to $\theta_{\max}$ into NCut equal subintervals. We then examine triples as specified in (7) and execute the 3-2-3 algorithm (preceded by the 2-1-2 algorithm if necessary).

In our present implementation we simplify the application of the 3-2-3 algorithm by applying it only to the best triple $(x(\theta_{q-1}), x(\theta_q), x(\theta_{q+1}))$, i.e., choosing $x(\theta_q)$ so that $f(x(\theta_q))$ is the local min among such triples, and executing the algorithm for only a single iteration. We restrict the algorithm in this manner because our goal is to obtain a potentially improved solution as quickly as possible. (Other algorithms for finding such solutions quickly that can be used as alternatives are also given in Glover (2010).) We also perform an oscillation in the granularity of the search. Define a block to consist of a series of passes that start from the full set $N_o = N$ and proceed until $N_o = \emptyset$. In addition, designate a block (more precisely, the execution of a block) to be successful if an improvement occurred on at least one of the passes within the block, and designate the block to be unsuccessful otherwise. Then we set $\delta_i = (U_i - L_i) * \text{rand}$ for each $i \in N_o$ to increase the granularity (from coarser grain to finer grain) at the conclusion of an unsuccessful block and set $\delta_i = (U_i - L_i)/\text{rand}$ for each $i \in N_o$ to decrease the granularity (from finer grain to coarser grain) at the conclusion of a successful block.

The pseudo-code of one iteration of the global procedure EM323 is detailed in Fig. 6. The $\delta_{\min}$ value is fixed to $1e - 15$ in order to obtain a suitable precision and the restart procedure is the same as the one used for EUS algorithm (see Sect. 2.3).

While we chose to fix the value of Ratio to 0.5 in EUS, we have set it randomly in the EM323 procedure with the anticipation that this will increase the quality of the solutions in some cases, and with the motivation that this eliminates Ratio from being considered as a parameter.

Finally, we supposed that if we managed to improve a solution after having decreased the granularity by shrinking δ, there is a chance that the improved solution is a new

```
Procedure EM323
$N_o = N$ and $N_o(success) = \emptyset$
begin
    for $i \in N_o$
        improve the solution $x$ along the dimension $i$ using
        the simplified 3-2-3 line search procedure to obtain
        a solution $x^*$
        if $f(x^*) < f(x)$ (equivalently, $x^* \neq x$) then
            $N_o(success) = N_o(success) \cup \{i\}$
        end if
        $x = x^*$
    end for
    if $N_o$ is empty then
        $N_o = N$
        if $\delta < \delta_{min}$ then
            call restart procedure
        else
            if $N_o$ went from $N_o = N$ to empty on a single
            iteration then
                (increase the granularity of the line search)
                $\delta = \delta * rand$ (rand being a random number
                from [0,1])
            else
                (decrease the granularity of the line search)
                $\delta = \delta / rand$ (rand being a random number
                from [0,1])
            end if
        end if
    end if
end
```

Fig. 6 One iteration of EM323 global algorithm

local optimum, and so it may be now possible to obtain an improvement by increasing the granularity of the search. This is the reason behind the oscillation strategy that alternately shrinks and expands the vector δ (based on setting $\delta = \delta * \text{rand}$ and $\delta = \delta/\text{rand}$).

4 Experimental setup

The proposed set of test problems includes 19 scalable functions with different characteristics. The first six are taken from the CEC'2008 test suite described in Tang et al. (2007). Other function definitions can be found in the ISDA'09 workshop report Herrera and Lozano (2009). A more detailed discussion of the extended F_{10} function can be found in Whitley et al. (1995). The last functions: F12–F19*, are hybrid composition functions built from a non-separable function and other functions of the benchmark. The procedure used to hybridize a non-separable function F_{ns} with another function F' (to create a function $F_{ns} \oplus F'$) consists of three main steps:

- Divide the solution into two components;
- Evaluate each component with a different function;
- Combine their results.

The splitting mechanism uses a parameter, m_{ns}, which specifies the ratio of variables that are evaluated by F_{ns}. Using a higher value of m_{ns}, the hybrid function becomes more difficult to optimize dimension by dimension, because there is a stronger association between the variables and the fitness. The functions we used are detailed in Table 1. For each function f, experiments are conducted in 50, 100, 200, 500 and 1,000 dimensions. Each algorithm is run 25 times and the average error e of the best solution x is computed:

$$e = f(x) - f(o) \quad \text{where } o = \text{ optimum of the function} \tag{10}$$

The maximum number of fitness evaluations is $5000n$ and determines the stopping criterion for a run.

The experiments have been implemented in Java (v.1.6), using a modular testbed called HeurisTest. The computer used had following characteristics:

- CPU: Intel Core 2 Duo, T8100 @ 2.10 GHz
- RAM: 2.00 GB
- OS: Windows XP

5 Results and discussion

5.1 EM323 average errors

EM323 method has only one tunable parameter: the NCut value used for dividing the search range in the line search procedure. The higher the value of this parameter the greater will be the granularity of line search, but the slower will be the convergence. We chose to set this parameter to 5 as a compromise between achieving a fine grained search and achieving a fast convergence. The errors obtained by the EM323 algorithm for each function are listed in Table 2. The average error values are computed for 25 runs and are reported for each of the 19 functions.

Table 1 Benchmark functions

#	Function	Separable	m_{ns}
1	Shifted sphere	Y	–
2	Shifted Schwefel problem 2.21	N	–
3	Shifted Rosenbrock	N	–
4	Shifted Rastrigin	Y	–
5	Shifted Griewank	N	–
6	Shifted Ackley	Y	–
7	Schwefel problem 2.22	Y	–
8	Schwefel problem 1.2	N	–
9	Extended F_{10}	N	–
10	Bohachevsky #1	N	–
11	Schaffer #2	N	–
12	Hybrid F9 and F1	N	0.25
13	Hybrid F9 and F3	N	0.25
14	Hybrid F9 and F4	N	0.25
15	Hybrid F10 and F7	N	0.25
16*	Hybrid F9 and F1	N	0.5
17*	Hybrid F9 and F3	N	0.75
18*	Hybrid F9 and F4	N	0.75
19*	Hybrid F10 and F7	N	0.75

It should be noted that we were unable to compute the results for the functions #7 and #15 for $D = 1{,}000$ because Java equated the objective function value with Infinity in these two cases, thus preventing a comparison between solutions. We can see that EM323 procedure is able to solve 15 out of 19 functions with suitable accuracy (e < 10E−7) for $D = 50$. The most problematic functions are Rosenbrock's function and its hybridizations (i.e functions #3, #13 and #17). The global minimum of Rosenbrock's function is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial, but to converge to the global minimum is difficult. The other function that resists solution by EM323 algorithm is Schwefel's problem 2.22 (i.e. function #8). The convergence graph of our algorithm on this function shows a very slow convergence to the optimum.

For most of the functions, the behavior of EM323 is similar for both high and low dimensions, disclosing that the method has good scalability. But functions #5, #12, #14, #16 and #18 are exceptions, providing instances where the results can deteriorate suddenly after the dimension reaches as high as $n = 100$ or $n = 200$ (e = 1e−12 before

Table 2 Error values obtained for EM323

Function	D = 50	D = 100	D = 200	D = 500	D = 1000
1	4.80E−13	9.91E−13	2.15E−12	5.80E−12	1.18E−11
2	4.08E−09	3.42E−04	1.92E−01	2.04E+01	2.31E+01
3	6.12E+01	2.10E+02	4.47E+02	1.25E+03	1.42E+03
4	5.34E−13	1.15E−12	2.23E−12	7.08E−12	1.33E−11
5	3.05E−13	5.91E−13	3.95E−04	1.77E−03	3.94E−04
6	5.66E−13	1.14E−12	2.42E−12	6.50E−12	1.29E−11
7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	–
8	2.08E+02	6.31E+02	3.37E+04	3.91E+05	1.46E+06
9	0.00E+00	3.29E−08	1.31E−08	2.21E−06	7.88E−02
10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
11	2.16E−07	9.99E−09	1.92E−06	1.10E−02	5.43E−01
12	9.88E−08	1.51E−02	4.13E−07	3.41E−01	6.16E−01
13	1.96E+01	5.97E+01	2.97E+02	1.19E+03	1.24E+03
14	1.32E−07	3.74E−07	2.10E−01	1.51E−01	1.03E+00
15	0.00E+00	0.00E+00	0.00E+00	0.00E+00	–
16*	2.52E−07	4.57E−07	9.40E−07	4.71E−03	1.29E−01
17*	8.58E+01	9.60E+01	5.73E+01	2.14E+02	3.00E+02
18*	3.12E−07	5.02E−02	2.63E−03	2.28E−01	1.66E−01
19*	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

and 1e–3 to 1e1 after). In fact, most runs encounter no problems, but when the dimension increases, some of the 25 runs become stuck in local optima and do not find the global optimum in the allotted time. This phenomenon occurs on just 1 or 2 of the 25 runs but the average error is strongly biased by these 2 runs. We speculate that this phenomenon is due to a lack of exploration in high dimensions, where the restart procedure is not called enough times to effectively explore the search space.

5.2 EM323 computational running time

In order to further investigate the behavior of the algorithm, we compute the average running time for each function of the benchmark over the 25 runs for the EM323 procedure (Table 3). We find that this running time greatly depends on the objective function as well as on the dimension. To illustrate the graph $t = f(D)$ for the function #1 is shown in Fig. 7, with t representing the running time (in milliseconds) and D the dimension. We observe that the average running time increases by what appears to be the square of the dimension. In order to confirm this relationship, we draw two new graphs from the same values, one with a logarithmic scale for t and a decimal scale for D (Fig. 8) and another with both logarithmic scales (Fig. 9). For the first case, we found a correlation coefficient $\alpha = 0.941$ and for the second we found $\alpha = 0.999$, indicating that the variation of running time according to dimension is a power function. Determining the parameters of this power function with a simple linear regression, we found: $\log(t) = 1.942 * \log(D) - 1.685$, giving $t = D^{1.94} * e^{-1.7}$. Therefore, we can conclude that the running time increases approximately as the square (1.94) of the dimension for function #1. The same test has been made for all functions and the results are quite similar, thus disclosing that the computational complexity of the algorithm is approximately $C = O(D^2)$.

Table 3 Average computational running times (in seconds) for EM323

Function	D = 50	D = 100	D = 200	D = 500	D = 1,000
1	0.394	1.368	5.233	30.605	133.380
2	0.461	1.745	6.508	35.329	162.728
3	0.413	1.539	5.868	30.443	147.325
4	1.211	4.591	16.995	90.785	379.014
5	1.333	5.160	19.476	113.553	452.131
6	1.253	4.713	18.462	110.734	452.609
7	0.400	1.286	4.587	28.468	–
8	0.351	1.140	3.977	23.515	116.735
9	10.888	38.809	153.926	1241.516	4806.078
10	1.602	6.219	25.820	147.766	645.547
11	12.104	47.855	191.999	1074.136	4784.208
12	3.152	12.434	50.221	292.506	1306.567
13	2.826	11.428	46.427	336.313	1396.375
14	3.331	13.399	52.675	377.015	1525.203
15	0.753	2.765	10.904	59.735	–
16*	6.093	23.860	95.058	620.141	2227.735
17*	8.025	30.013	119.783	863.937	3679.783
18*	8.134	30.003	123.796	882.250	3884.616
19*	1.471	5.190	21.826	122.500	489.828

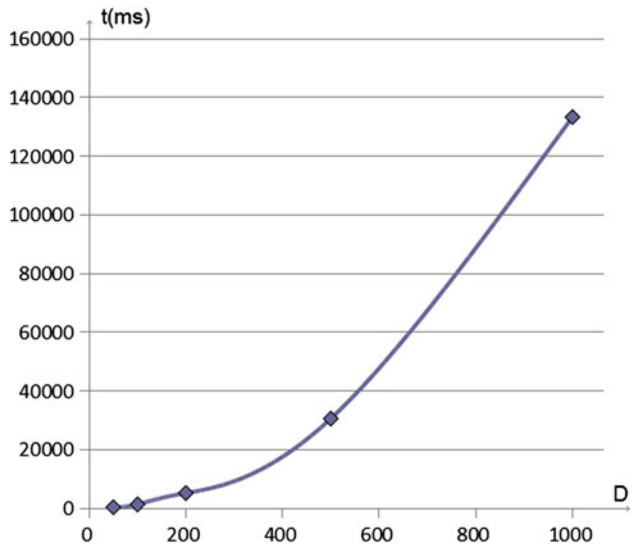

Fig. 7 Function #1: $t = f(D)$

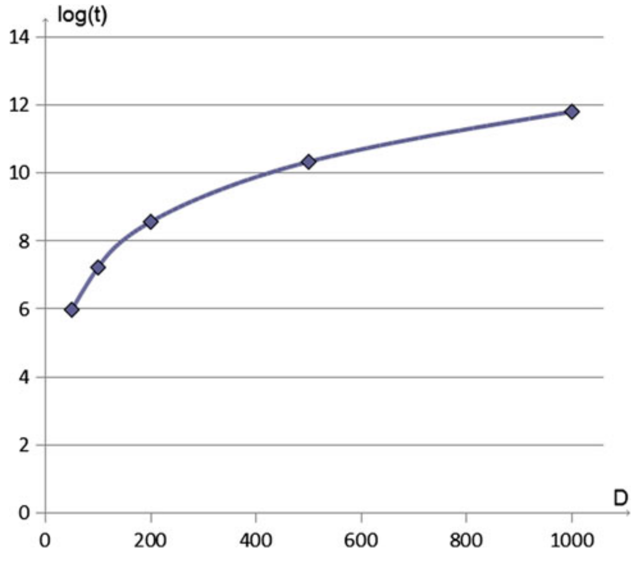

Fig. 8 Function #1: $\log(t) = f(D)$

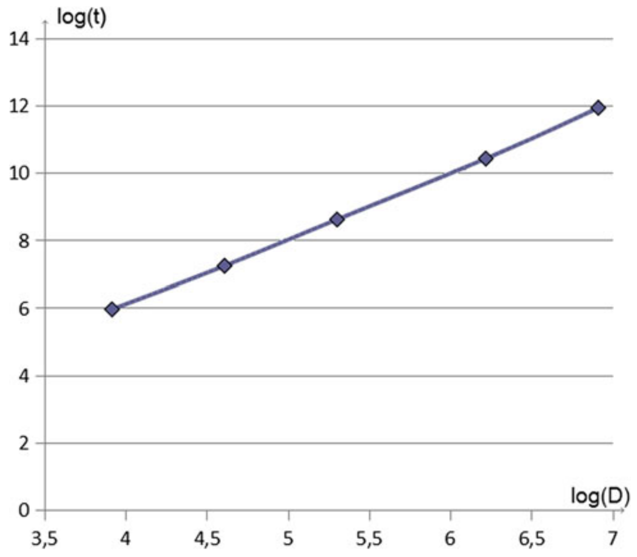

Fig. 9 Function #1: $\log(t) = f(\log(D))$

5.3 Comparison of EM323 with other algorithms

In order to have a preliminary idea of EM323's performance, we compared it with three evolutionary algorithms reputed to be effective for this type of problem, as indicated below. The parameter settings used in each case were those recommended by the authors, except in those instances where tests identified better parameter values:

- Differential evolution (DE) Storn and Price (1997): the crossover operator applied was rand/1/exp, based on the finding that this yields a better performance . The F and CR parameters were fixed to 0.5 and 0.9, respectively. Usually, the population size parameter is set in function of the problem dimension ($10D$ or $3D$), however, in high dimensionality this criterion is not adequate, and a maximum limit should be fixed. For the experiments, a population of 60 individuals was used (similar results were achieved with a population size fixed to 100).
- Real-coded CHC Eshelman and Schaffer (1993): the initial threshold is set at $L = 20 * D$. The α parameter used by the BLX-α crossover operator is set to 0.5. The population size is of 50 chromosomes.
- G-CMA-ES Auger and Hansen (2005): This method was the winner of the real-parameter optimization competition (CEC2005). The parameters used were the ones suggested by Auger and Hansen. The initial solution is uniform randomly chosen from the domain and the initial distribution size (σ) is a third of the domain size.

The errors obtained by these three algorithms for each function are listed in Tables 4, 5 and 6. The average error values are computed for 25 runs and are reported for each of the 19 functions.

We can see that our algorithm matches or outperforms the three others on all functions except functions #2 and #8, where G-CMA-ES performs very well even in high dimensions. Moreover, we can observe the same relative performance concerning the difference between the median and the average of the results on some functions for the three algorithms (DE for functions 4, 14, 18; G-CMA-ES for functions 3, 5 and CHC for functions 1, 3, 5, 10, 12, 13, 15, 17, 19). The problem is the same, i.e. in some cases, the

Table 4 Error values obtained for DE

Function	D = 50	D = 100	D = 200	D = 500	D = 1,000
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	3.60E−01	4.45E+00	1.92E+01	5.35E+01	8.46E+01
3	2.89E+01	8.01E+01	1.78E+02	4.76E+02	9.69E+02
4	3.98E−02	7.96E−02	1.27E−01	3.20E−01	1.44E+00
5	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
6	1.43E−13	3.10E−13	6.54E−13	1.65E−12	3.29E−12
7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
8	3.44E+00	3.69E+02	5.53E+03	6.09E+04	2.46E+05
9	2.73E+02	5.06E+02	1.01E+03	2.52E+03	5.13E+03
10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
11	6.23E−05	1.28E−04	2.62E−04	6.76E−04	1.35E−03
12	5.35E−13	5.99E−11	9.76E−10	7.07E−09	1.68E−08
13	2.45E+01	6.17E+01	1.36E+02	3.59E+02	7.30E+02
14	4.16E−08	4.79E−02	1.38E−01	1.35E−01	6.90E−01
15	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
16*	1.56E−09	3.58E−09	7.46E−09	2.04E−08	4.18E−08
17*	7.98E−01	1.23E+01	3.70E+01	1.11E+02	2.36E+02
18*	1.22E−04	2.98E−04	4.73E−04	1.22E−03	2.37E−03
19*	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

Table 5 Error values obtained for CHC

Function	D = 50	D = 100	D = 200	D = 500	D = 1,000
1	1.67E−11	3.56E−11	8.34E−01	2.84E−12	1.36E−11
2	6.19E+01	8.58E+01	1.03E+02	1.29E+02	1.44E+02
3	1.25E+06	4.19E+06	2.01E+07	1.14E+06	8.75E+03
4	7.43E+01	2.19E+02	5.40E+02	1.91E+03	4.76E+03
5	1.67E−03	3.83E−03	8.76E−03	6.98E−03	7.02E−03
6	6.15E−07	4.10E−07	1.23E+00	5.16E+00	1.38E+01
7	2.66E−09	1.40E−02	2.59E−01	1.27E−01	3.52E−01
8	2.24E+02	1.69E+03	9.38E+03	7.22E+04	3.11E+05
9	3.10E+02	5.86E+02	1.19E+03	3.00E+03	6.11E+03
10	7.30E+00	3.30E+01	7.13E+01	1.86E+02	3.83E+02
11	2.16E+00	7.32E+01	3.85E+02	1.81E+03	4.82E+03
12	9.57E−01	1.03E+01	7.44E+01	4.48E+02	1.05E+03
13	2.08E+06	2.70E+06	5.75E+06	3.22E+07	6.66E+07
14	6.17E+01	1.66E+02	4.29E+02	1.46E+03	3.62E+03
15	3.98E−01	8.13E+00	2.14E+01	6.01E+01	8.37E+01
16*	2.95E−09	2.23E+01	1.60E+02	9.55E+02	2.32E+03
17*	2.26E+04	1.47E+05	1.75E+05	8.40E+05	2.04E+07
18*	1.58E+01	7.00E+01	2.12E+02	7.32E+02	1.72E+03
19*	3.59E+02	5.45E+02	2.06E+03	1.76E+03	4.20E+03

Table 6 Error values obtained for G-CMA-ES

Function	D = 50	D = 100	D = 200	D = 500
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00
2	2.75E−11	1.51E−10	1.16E−09	3.48E−04
3	7.97E−01	3.88E+00	8.91E+01	3.58E+02
4	1.05E+02	2.50E+02	6.48E+02	2.10E+03
5	2.96E−04	1.58E−03	0.00E+00	2.96E−04
6	2.09E+01	2.12E+01	2.14E+01	2.15E+01
7	1.01E−10	4.22E−04	1.17E−01	–
8	0.00E+00	0.00E+00	0.00E+00	2.36E−06
9	1.66E+01	1.02E+02	3.75E+02	1.74E+03
10	6.81E+00	1.66E+01	4.43E+01	1.27E+02
11	3.01E+01	1.64E+02	8.03E+02	4.16E+03
12	1.88E+02	4.17E+02	9.06E+02	2.58E+03
13	1.97E+02	4.21E+02	9.43E+02	2.87E+03
14	1.09E+02	2.55E+02	6.09E+02	1.95E+03
15	9.79E−04	6.30E−01	1.75E+00	2.82E+262
16*	4.27E+02	8.59E+02	1.92E+03	5.45E+03
17*	6.89E+02	1.51E+03	3.36E+03	9.59E+03
18*	1.31E+02	3.07E+02	6.89E+02	2.05E+03
19*	4.76E+00	2.02E+01	7.52E+02	2.44E+06

algorithm is trapped into a local optimum and cannot find the global optimum within the given time.

For each problem instance we checked whether the differences between the solutions found by EM323 and the other algorithms are statistically significant. We used *R* statistical software to conduct a pairwise Wilcoxon's test (García et al. 2009). The method was adjusted to pairwise testing by a method of Holm. In Table 7, for each pair of algorithms (EM323 and *X*), the *p* value is computed for the following null hypothesis: $H_0 =$ "The distribution of solutions generated by EM323 and the distribution of solutions generated by X are equal".

Table 7 Wilcoxon's test results for EM323 versus 3 evolutionary algorithms

Algorithm	Dimension				
	50	100	200	500	1,000
DE					
W^+	60	57	38	32	33
W^-	60	63	82	88	87
p value	1.0000	0.8904	0.2293	0.1205	0.1354
CHC					
W^+	187	190	174	173	138
W^-	3	0	16	17	15
p value	1.907E−05	3.815E−06	0.0006	0.0008	0.0021
G-CMA-ES					
W^+	158	159	156	143	–
W^-	32	31	34	28	–
p value	0.0095	0.0082	0.0124	0.0104	–

The table shows that the *p* values are all less than 0.05 for the Real-coded CHC and the G-CMA-ES algorithms over all dimensions. Consequently, the null hypothesis is rejected with a false probability of $\alpha = 5\%$. For these algorithms, the W^+ value is very high, so we can conclude that EM323 performs better than these two algorithms for all functions over all dimensions. For the DE algorithm, since the *p* value is greater than 0.05, we have to accept the null hypothesis H_0 and cannot claim a statistically significant difference exists between the algorithms.

A careful examination of these results leads us to the further conclusion, however, that the classic Wilcoxon's test is not well designed to compare such algorithms, because it focuses on comparing average error values without reference to their relative differences. To see the effect of this, note that for function #4, EM323 (with an error of $1e-12$) performs somewhat better than DE (with an error of $1e-2$). However, the difference computed by Wilcoxon's test is only equal to $1e-2$, which discounts the actual difference between the methods on this function. This difference is further discounted when measured alongside the difference that occurs for function #3, which should presumably be considered less significant. In particular, for function #3, the results between the 2 algorithms are quite similar with errors on the order of $1e1$. DE performs a bit better than EM323, yielding a difference equal to $1e1$. But when Wilcoxon's test sorts the differences to rank them according to their importance, the $1e1$ difference is considered more important than the $1e-2$ difference, since only absolute differences rather than relative differences are taken into account, thus missing the fact that in relative terms the second difference represents a greater disparity between the two algorithms than the first.

In order to remedy this type of distortion, the calculations should be made after a logarithmic normalization, in order to appropriately compare the accuracy of each algorithm in terms of their relative errors.

5.4 EM323 versus non-separable functions

Previous computations show that the 3-2-3 line search procedure, used only with a relaxation method and without taking advantage of the combination that creates EM323, does not manage to solve non-separable functions well, and encounters serious difficulties with the last functions of the benchmark which have high correlation due to m_{ns} parameter. Such an outcome is to be expected, since a high correlation between parameters cannot be exploited by a procedure that examines only one dimension at a time. One way to compensate for this limitation would be to conduct searches in directions other than those determined by the coordinate dimensions. Embedding the 3-2-3 procedure within the global EM323 procedure adopts a different strategy, by varying both the granularity of the search on the coordinate directions and by the oscillation strategy that changes N_o and hence varies the sequence of dimensions along which the searches are carried out. As we have demonstrated, the resulting method proves effective even for non-separable functions.

6 Conclusion and future work

We present in this paper a one-dimensional search algorithm created as a hybrid of two algorithms: the EUS method and the 3-2-3 line search method. The resulting EM323 method is applied to a test suite that contains 19 high-dimensional optimization problems. We use a dimension separation process that makes our algorithms robust in handling problems of such dimensions. The experiments indicate that our method achieves high quality results on a wide range of functions in high dimension, which represents a noteworthy accomplishment for an algorithm that has only one parameter. Current work focuses on testing other line search methods within the same overall framework and exploring search directions that differ from the coordinate directions. In this latter pursuit, we are investigating the use of evolutionary algorithms to combine the search directions and to combine solutions as a basis for generating target vectors that give new search directions.

References

Auger A, Hansen N (2005) A restart CMA evolution strategy with increasing population size. In: The 2005 IEEE congress on evolutionary computation, vol 2, pp 1769–1776

Eshelman L, Schaffer J (1993) Real-coded genetic algorithms and interval-schemata. Foundation of Genetic Algorithms 2:187–202

García S, Molina D, Lozano M, Herrera F (2009) A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. J Heuristics 15(6):617–644

Gardeux V, Chelouah R, Siarry P, Glover F (2009) Unidimensional search for solving continuous high-dimensional optimization problems. In: ISDA '09: Proceedings of the 2009 ninth international conference on intelligent systems design and applications, IEEE Computer Society, Washington, DC, USA, pp 1096–1101

Glover F (1995) Tabu thresholding: improved search by nonmonotonic trajectories. ORSA J Comput 7(4):426–442

Glover F (2010) The 3-2-3, stratified split and nested interval line search algorithms. Research report, OptTek Systems, Boulder, CO

Grosan C, Abraham A (2007) Modified line search method for global optimization. In: AMS '07: Proceedings of the first Asia international conference on modelling & simulation, IEEE Computer Society, Washington, DC, USA, pp 415–420

Herrera F, Lozano M (2009) Benchmark functions 7–11. In: Tech. rep., Workshop: evolutionary algorithms and other metaheuristics for continuous optimization problems—a scalability test. http://sci2s.ugr.es/programacion/workshop/functions7-11.pdf

Hestenes MR, Stiefel E (1952) Methods of conjugate gradients for solving linear systems. J Res Natl Bureau Standards 49(6):409–436

Lee E (2007) Large-scale optimization-based classification models in medicine and biology. Ann Biomed Eng 35:1095–1109

Marti R, Laguna M, Glover F (2006) Principles of scatter search. Eur J Oper Res 169:359–372

Nasiri JA, Fard AM, Naghibzadeh M, Rouhani M (2009) High dimensional problem optimization using distributed multi-agent PSO. In: EMS '09: Proceedings of the 2009 third UKSim European symposium on computer modeling and simulation, IEEE Computer Society, Washington, DC, USA, pp 245–250

Storn R, Price K (1997) Differential evolution, a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11(4):341–359

Tang K, Yao X, Suganthan PN, MacNish C, Chen YP, Chen CM, Yang Z (2007) Benchmark functions for the CEC'2008 special session and competition on large scale global optimization. Tech. rep., Nature Inspired Computation and Applications Laboratory, USTC , China, http://nical.ustc.edu.cn/cec08ss.php

Tseng LY, Chen C (2008) Multiple trajectory search for large scale global optimization. In: IEEE congress on evolutionary computation, pp 3052–3059

Whitley D, Beveridge R, Graves C, Mathias K (1995) Test driving three 1995 genetic algorithms: new test functions and geometric matching. J Heuristics 1:77–104