Using Interval Scanning to Improve the Performance of the Enhanced Unidimensional Search Method

Mahamed G. H. Omran
Department of Computer Science
Gulf university for Science & Technology, Kuwait
Email: omran.m@gust.edu.kw

Vincent Gardeux
Ecole Internationale des Sciences du Traitement de l'Information, Cergy-Pontoise, France
Email: vincent.gardeux@eisti.fr

Rachid Chelouah
Ecole Internationale des Sciences du Traitement de l'Information, Cergy-Pontoise, France
Email: rachid.chelouah@eisti.fr

Patrick Siarry
Universite de Paris 12, LiSSi, Creteil, France
Email: siarry@univ-paris12.fr

Fred Glover
Leeds School of Business, University of Colorado, Boulder, CO 80309, USA
Email: fred.glover@colorado.edu

Abstract: This note suggests a simple modification to the Enhanced Unidimensional Search (EUS) method where interval scanning is used to find the best value for the *ratio* parameter of EUS. The proposed method, called stepped EUS (SEUS), is tested on 12 functions with encouraging results.

Keywords: Continuous nonlinear function optimization, enhanced unidimensional search, local search, interval scanning.

1. Introduction

We propose a modification of the Enhanced Unidimensional Search (EUS) algorithm (Gardeux et al. 2009) which is designed for solving the continuous nonlinear function optimization problem

(P) Minimize $f(\mathbf{x})$: $\mathbf{L} \leq \mathbf{x} \leq \mathbf{U}$

The vector $\mathbf{x} = (x_1, \ldots, x_n)$ is composed of real-valued variables, and the vectors **L** and **U** are assumed finite and to satisfy $\mathbf{L} < \mathbf{U}$. The function $f(\mathbf{x})$ is typically multi-modal, so that local optima do not in general correspond to global optima.

EUS is an enhanced variant of the Classical Unidimensional Search (CUS) for solving (P), and hence operates as a line search algorithm that runs dimension by dimension. The method is easy to implement and good in handling high dimensional problems.

In brief overview, EUS starts from a randomly-generated initial solution, **x**. A difference ("delta") vector **d** is created and initialized by setting $\mathbf{d} = \mathbf{U} - \mathbf{L}$, followed by shrinking the size of **d** on subsequent iterations and continuing until a stopping criterion is reached. The method successively focuses on a single variable x_i, $i \in N =$

{1, …,n}, and selects the best neighbor from the two alternatives "**x** up" and "**x** down" given by

$$\mathbf{x}^u = \mathbf{x} + d_i\mathbf{e}_i$$
$$\mathbf{x}^d = \mathbf{x} - d_i\mathbf{e}_i$$

where $\mathbf{e}_i$ is the unit vector with a 1 in position i and 0's elsewhere. (Hence $\mathbf{x}^u$ and $\mathbf{x}^d$ are the same as **x** except for the ith component, where $x_i^u = x_i + d_i$ and $x_i^d = x_i - d_i$.) The value x_i^u is reset to U_i if $x_i + d_i > U_i$ and x_i^d is reset to L_i if $x_i - d_i < L_i$.

The search then compares **x** with its 2 neighbor solutions $\mathbf{x}^u$ and $\mathbf{x}^d$ and updates **x** to be the best of these, hence setting $\mathbf{x} = \arg\min(f(\mathbf{x}), f(\mathbf{x}^u), f(\mathbf{x}^d))$. Then, successive dimensions i are treated in the same manner.

After one iteration, which consists of thus examining every dimension i, the algorithm finds a restricted local optimum relative to the precision given by the vector **d**. (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.) After each iteration, we test if the solution has been improved on at least one dimension, if not **d** is multiplied by a ratio value fixed to 0.5, therefore shrinking the size of its components. The **d** vector continues to shrink in this fashion until it satisfies $\mathbf{d} < \mathbf{d}_{min}$, whose components are all fixed to 1×10^{-15} in order to obtain a suitable precision. Gardeux et al. (2009) states that this *ratio* could be tuned but their experiments show that 0.5 is suitable. The pseudo-code in Figure 1 details an iteration of the algorithm, starting from the randomly generated initial **x**.

```
Procedure EUS
d = U - L
begin
    for i = 1 to n
        x^u = x + d_i e_i
        x^d = x – d_i e_i
        (update x to be the best of the 3 solutions)
        x = arg min(f(x), f(x^u), f(x^d))
    end
    if no improvement has been found
        d = 0.5d until d < d_min
    end
end
```

Fig. 1 An iteration of the EUS algorithm

To avoid becoming trapped in a local minimum, EUS uses a restart procedure that keeps the best solution found so far and re-initializes **d** to a new starting value after reaching the termination point given by $\mathbf{d} < \mathbf{d}_{min}$,. Therefore, increasing the vector $\mathbf{d}_{min}$, 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 the reference set. This technique uses the diversification generation method from the

Scatter Search algorithm. It generates a collection of diverse trial solutions and selects the one farthest from a reference set of previously visited restricted local optima.

2. The Stepped EUS (SEUS)

SEUS uses interval scanning to tune the *ratio* implicitly used in EUS. In addition, a fixed number of iterations is used to terminate EUS (rather than using $\mathbf{d}_{min}$). The proposed algorithm is shown in Figure 2.

```
ratio = 0.1
Generate an initial solution y randomly.
while ratio < 1
        x = y
        d = U - L
        for t=1 to t_max
                for i = 1 to n
                        x^u = x + d_i e_i
                        x^d = x – d_i e_i
                        (update x to be the best of the 3 solutions)
                        x = arg min(f(x), f(x^u), f(x^d))
                end
                if no improvement has been found
                        d = ratio.d
                end
        end
        ratio = ratio + 0.1
end
```

Fig. 2 Pseudo code of SEUS.

3. Experimental Results

In this section we compare the performance of SEUS with that of EUS when applied to 12 test problems with n ranging from 2 to 30. The initialization ranges for each problem are the same as in their "original sources":

http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm

http://solon.cma.univie.ac.at//glopt.html

The results reported in this section are averages and standard deviations over 30 simulations. Each simulation was allowed to run for 2000 iterations (a fixed number of iterations is used to terminate both EUS and SEUS). The same best solution (out of 1000 solutions) was used for both algorithms.

All the tests are run on an Apple MacBook computer with Intel Core Due 2 processor running at 2.0 GHz with 2GB of RAM. Mac OS X 10.5.6 is the operating system used. All programs are implemented using MATLAB version 7.6.0.324 (R2008a) environment.

Table I summarizes the results obtained by applying EUS and SEUS to the benchmark functions. The table reports the average gap, Avg. GAP, where the optimality gap for a given solution $\mathbf{x}$ and the best solution $\mathbf{x}^*$ is defined as:

$$GAP = |f(\mathbf{x}) - f(\mathbf{x}^*)|.$$

The results show that SEUS performed better than (or equal to) EUS on almost all the benchmark functions.

Notice that the comparison is relatively "unfair" because SEUS will call EUS 9 times with different values of *ratio*. However, the results show that with a simple modification that does not introduce any new parameter we could get the "best" *ratio* to use for a given problem. In addition, the results show that the *ratio* parameter has a significant effect on the performance of EUS. For example, our experiments show that the "optimal" *ratio* value for Normalized Schwefel's function was 0.9. This function was not solved with good accuracy with the 0.5 ratio, showing that this function need a slower convergence in order to be solved properly. The initial 0.5 value could result in a too fast convergence.

References

V. Gardeux, R. Chelouah, P. Siarry and F. Glover. Unidimensional search for solving continuous high-dimensional optimization problems. In the proceedings of the 2009 9th International Conference on Intelligent Systems Design and Application, pp. 1096-1101, 2009.

	EUS	*SEUS*
Function	GAP(SD)	GAP(SD)
Sphere (n = 30)	**0(0)**	**0(0)**
Rastrigin (n = 30)	6.692745e+01 (1.187047e+01)	**1.956753e+00 (1.182555e+00)**
Michalewicz (n = 10)	1.232538e+00 (4.591956e-01)	**2.819791e-01 (2.095615e-01)**
Step (n = 30)	**0(0)**	**0(0)**
Rosenbrock (n = 30)	1.107339e+00 (1.787840e+00)	**5.152956e-03 (5.678218e-03)**
Ackley (n = 30)	4.612607e-14 (1.223366e-14)	**3.558635e-14 (7.503647e-15)**
Griewank (n = 30)	5.753899e-03 (5.469051e-03)	**0(0)**
Normalized Schwefel (n = 30)	9.766790e+01 (4.995761e+00)	**5.400125e-13 (8.672276e-14)**
Salomon (n = 30)	5.066540e+00 (2.020299e+00)	**1.796540e+00 (3.221515e-01)**
Rotated Hyper-ellipsoid (n = 30)	9.244927e-20 (2.239516e-19)	**6.229013e-25 (2.423284e-24)**
Goldstein and Price (n = 2)	4.588922e-14 (2.845551e-15)	**3.782160e-14 (1.265088e-14)**
Shekel (n = 4)	5.294651e+00 (3.341180e+00)	**4.847920e+00 (3.571269e+00)**

Table I. Mean and standard deviation (SD) of the function optimization results.