2004 International Symposium on Nonlinear
Theory and its Applications (NOLTA2004)
Fukuoka, Japan, Nov. 29 - Dec. 3, 2004

Continuous Global Optimization Algorithm using Lattice-based Sampling and Topographical Multilevel Single Linkage

Hideo KANEMITSU[1)], Hideaki KONNO[2)], Masaaki MIYAKOSHI[3)] and Masaru SHIMBO[4)]

[1) 2)] Hokkaido University of Education
Hakodate 040-8567, Japan
Phone: +81-138-44-{4351[1)], 4366[2)]}
Fax: +81-138-44-{4351[1)], 4366[2)]}
{hkanemt[1)], konno[2)]}@cc.hokkyodai.ac.jp

[3)] Hokkaido University
Sapporo 060-8628, Japan
Phone: +81-11-706-6810
Fax: +81-11-706-7830
miyakosi@main.eng.hokudai.ac.jp

[4)] Hokkaido Information University
Ebetsu 069-8585, Japan
Phone: +81-11-385-4411
Fax: +81-11-384-0134
shimbo@do-johodai.ac.jp

Abstract—We propose an algorithm for solving global optimization problems with objective functions of *n*-real variables and bounded constraints. This method basically uses sample lattice points and uses topographical information on objective function and a multilevel single linkage technique for avoiding repeated convergence to the same local minima. We show that the algorithm can calculate neighboring points in a much shorter time than that in the case of using the TGO method. Moreover, in order to avoid exponentially increasing sample size with increased number of divisions of each coordinate, we propose a subspace sampling. We show by numerical experiments in simple test functions that the algorithm effectively finds the global minima in a short computational time and in a few function evaluations.

1. INTRODUCTION

In recent years, many deterministic and stochastic algorithms have been proposed for solving a global optimization (minimization) problem (P) of a multivariate function with bounded constraints:

$$
\text{(P)} \quad \left| \begin{array}{l} \text{Min} \;\; f(\boldsymbol{x}) \equiv f(x_1, x_2, \ldots, x_n)\,, \\ \text{Subject to} \;\; L_i \le x_i \le U_i\,, \quad i = 1, 2, \ldots, n\,, \\ \quad D^n = \{\, \boldsymbol{x} \in R^n \,|\, [L_1, U_1] \times \cdots \times [L_n, U_n] \,\}\,. \end{array} \right.
$$

Deterministic algorithms [2] repeatedly divide a given region into subregions, select a subregion in which a global optimum is included, and give a guarantee of successfully finding the global optimum under highly restrictive conditions on objective functions (for example, Lipschitz continuity with a known Lipschitz constant).

On the other hand, stochastic methods involve random sampling or a combination of random sampling and local search. The latter algorithms, called *multistart* methods, can find the global optimum with a high degree of accuracy.

However, these algorithms are very inefficient because of repeated convergence of many sample points to the same local minima that have already been found.

To overcome this problem, *clustering* methods [4] have been used. These methods consist of the three steps: i) taking uniformly random sample points, ii) selecting sampled points for grouping sample points around each local optimum, and iii) forming groups of mutually close points using a clustering technique and applying the local optimizer from the best point of each group. However, if multiple local optima are included in a group, only one local optimum can be found by this method. In *clustering* methods, sampled points are basically grouped by using only information on the distance between two points. Overviews of the clustering methods have been described in [4].

As more advanced methods, the *Multi-Level Single Linkage* (MLSL) algorithm [3] and the *Topographical Global Optimization* (TGO) method [5], which use information not on only the distance between two neighboring points but also on the function values of these two points, have been proposed. The MLSL method uses a local optimizer for each new sample point $\boldsymbol{x}_i$ except if there is already a sampled point $\boldsymbol{x}_j$ with $f(\boldsymbol{x}_j) < f(\boldsymbol{x}_i)$ and $\| \; \boldsymbol{x}_j - \boldsymbol{x}_i \; \| \le r_k$, where r_k is the critical distance. However, this method sometimes find local optima that have already been found. The TGO method detects sample points so that all l-neighboring points around each sample point are higher than the sample point, and a local optimizer is used for each detected point. This method, however, sometimes finds the same local optima if l is small. Moreover, a long time is needed to calculate neighboring points of the current point in both methods.

To overcome the problem of inefficient use of CPU resource for finding l-neighboring points, we propose a hybrid algorithm consisting of the following steps: i) sampling points on a lattice, ii) detecting points by the TGO method, iii) selecting a starting point by the MLSL method and iv) terminating the algorithm and estimating sample points on the next iteration by Boender's expectation [1] of the number of local minima.

The remainder of the paper is organized as follows. Notations are given in section 2. In section 3, MLSL and TGO algorithms are described as background information. The main algorithm is described in detail in section 4. Lattice-based sampling and subspace sampling are described in section 5. In section 6, results of tests on the proposed algorithm by numerical experiments in Shekel functions are shown. Finally, concluding remarks are presented.

2. PRELIMINARIES

In a problem (P), suppose the objective function $f(\boldsymbol{x})$ is continuous and smooth, and it has a finite number of isolated local minima $\boldsymbol{x}_k^* \in D^n$ $(k = 1, 2, \ldots, M)$.

A set X^* of the isolated local minima and a set F^* of their values are written by

$$\begin{cases} X^* &= \{\, \boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \ldots, \boldsymbol{x}_M^* \,\}\,, \\ F^* &= \{\, f(\boldsymbol{x}_1^*), f(\boldsymbol{x}_2^*), \ldots, f(\boldsymbol{x}_M^*) \,\}\,. \end{cases} \tag{1}$$

Notation 1 Given a set A that consists of finite elements, the size of A is denoted by $|A|$.

Notation 2 An algorithm is expressed by using the following notation:

$$(r_1, r_2, \ldots, r_q) \ \leftarrow a\_name\,(a_1, a_2, \ldots, a_p).$$

This means that if input p-arguments $(a_1, a_2, \ldots,\ a_p)$ are given by applying the algorithm *a_name*, results $(r_1, r_2, \ldots, r_q)$ are obtained.

3. BACKGROUND OF OUR ALGORITHM

3.1. MLSL Algorithm

The MLSL algorithm [3] is one of the most efficient mutistart-based methods.

In this algorithm, the local search procedure is applied to every sample point in the k-th iteration except in the case in which there is another sample point within the following *threshold-distance* r_k,

$$r_k = \frac{1}{\sqrt{\pi}} \left(\sigma \cdot \mu(D^n) \cdot \Gamma\left(1 + \frac{n}{2}\right) \frac{\log N}{N} \right)^{1/n}, \tag{2}$$

which has a lower function value. Here, σ is a constant, $\mu(D^n)$ is the Lebesgue measure of D^n, $\Gamma(\cdot)$ is the gamma function, and N is the total number of sample points.

Furthermore, the algorithm repeats sampling until the expected number of minima $|X^*|(\gamma N - 1)\,/\,(\gamma N - |X^*| - 2)$ exceed the number of different minima found $|X^*|$ by less than 0.5. Thus, the algorithm stops if

$$\frac{|X^*|\,(\gamma N - 1)}{\gamma N - |X^*| - 2} \leq |X^*| + 0.5, \tag{3}$$

where γ $(0 < \gamma < 1)$ is the reduced ratio of sampling, that is, only γN-lowest points are used as candidate starting points of a local search based on equation (2).

In the first iteration, $N^{(1)}$ (= N as the initial value) is given, and in the subsequent iteration, $N^{(k+1)}$ $(k = 1, 2, \ldots)$ and N are updated by

$$\begin{cases} N^{(k+1)} \leftarrow \dfrac{1}{\gamma}(2|X^*|^2 + 3|X^*| + 2)\,, \\ N \leftarrow N + N^{(k+1)}. \end{cases}$$

The MLSL algorithm finds the set of local minima X^* and the set of minimal function values F^* for a function f over a searching region D^n for a given number of initial sampling points $N^{(1)}$, for a given reduced ratio $\gamma(0 < \gamma < 1)$, and for a given constant $\sigma > 0$. The steps of the algorithm are as follows.

$(F^*,\ X^*) \leftarrow MLSL\,(D^n,\ f,\ N^{(1)},\ \gamma,\ \sigma)\,;$

M1. [Initialize] Set $X^* \leftarrow \emptyset\,;\ \ F^* \leftarrow \emptyset\,;\ \ k \leftarrow 1\,;\ \ N \leftarrow N^{(1)}$.

M2. [Sample points and select sample] Sample $N^{(k)}$ points and select the number of $\gamma N^{(k)}$ lowest points. Set $N_r \leftarrow \gamma N^{(k)}$. Transform the sample by selecting the fraction γ of lowest points.

M3. [Determine r_k and starting point of a local search by MLSL, and apply the local search] Determine r_k by equation (2) and apply a local search from each new point $\boldsymbol{x}_i$ $(i = 1, 2, \ldots, N_r)$ except if there is a sample point $\boldsymbol{x}_j$ with $f(\boldsymbol{x}_j) < f(\boldsymbol{x}_i)$ and $\|\,\boldsymbol{x}_j - \boldsymbol{x}_i\,\| < r_k$. Add the obtained local minimum $\boldsymbol{x}^*$ to X^*.

M4. [Test for stop] If the stop condition (3) is satisfied, then stop. Otherwise set a new sample point $N^{(k+1)} \leftarrow \frac{1}{\gamma}(2|X^*|^2 + 3|X^*| + 2)$, set $N \leftarrow N + N^{(k+1)}$, set $k \leftarrow k + 1$, and go to step **M2**.

The method is very simple, and no clustering is applied. The only difficulty is to find the nearest neighbor for each new point in an efficient way in the growing sample of uniformly distributed points.

In spite of its simplicity, the theoretical properties of the method are quite strong. If $\sigma > 4$, then even if sampling continues forever, the total number of local searches is finite with probability 1 [1].

3.2. TGO Algorithm

The TGO algorithm was proposed by Törn [4] and is also a very efficient mutistart-based method.

The TGO algorithm consists of the following steps: i) sampling the number of N_s points, ii) detecting $\boldsymbol{x}_j$ such that function values at its neighbor l -points are higher than $f(\boldsymbol{x}_j)$,

$$f(\boldsymbol{x}_j) < f(\boldsymbol{x}_i) \qquad (i = 1, 2, \ldots, l), \tag{4}$$

and iii) applying a local search from points detected.

This algorithm is also very simple, but there is a waste of time in finding neighbor l-points.

4. MAIN ALGORITHM

In this section, we show the main algorithm of our new method. This algorithm consists of i) lattice-based sampling, ii) detecting points close to each local minimum using TGO, iii) determining the starting point by MLSL criteria, and iv) applying a local search.

In the MLSL algorithm, only γN-lowest points are selected after N sampling. On the other hand, in our algorithm γ is set to 1 because of all N sample points are selected at step i) and points are detected by TGO at step ii).

The main algorithm finds the set of local minima X^* and the set of minimal function values F^* for a function f over a searching region D^n, for a given number of initial samplings $N^{(1)}$, and for a given constant $\sigma > 0$. The detailed steps of the algorithm are as follows.

$(F^*, X^*) \leftarrow GM(f, D^n, N^{(1)}, \sigma)$;

G1. [Initialize] Set $X^* \leftarrow \emptyset$, $F^* \leftarrow \emptyset$, $k \leftarrow 1$, $N \leftarrow N^{(1)}$, and $\gamma \leftarrow 1$. Set the set of these function values F at sample points, the set of detected points X_c and the set of these function values F_c as follows: $F \leftarrow \emptyset$, $X_c \leftarrow \emptyset$, and $F_c \leftarrow \emptyset$.

G2. [Sample points] Generate $N^{(k)}$ lattice-based sampling points (See the next section for details.) $\boldsymbol{x}_i^{(k)} \in D^n$ $(i = 1, \ldots, N^{(k)})$, evaluate these function values $f_i^{(k)} \equiv f(\boldsymbol{x}_i^{(k)})$ $(i = 1, \ldots, N^{(k)})$, and add these function values to the set F.

G3. [Detect (select) points by TGO] Detect a point $\boldsymbol{x}_j$ such that function values at its neighbor l-points are higher than $f(\boldsymbol{x}_j)$ like equation (4). Then add the point $\boldsymbol{x}_j$ to the set X_c and its function value $f_j \equiv f(\boldsymbol{x}_j)$ to the set F_c.

G4. [Determine r_k and starting point of a local search by MLSL, and apply the local search] Determine r_k by equation (2), and apply a local search from each new point $\boldsymbol{x}_i \in X_c$ except if there is a sample point $\boldsymbol{x}_j \in X_c$ with $f(\boldsymbol{x}_j) < f(\boldsymbol{x}_i)$ and $\| \boldsymbol{x}_j - \boldsymbol{x}_i \| < r_k$. Add the obtained local minimum $\boldsymbol{x}^*$ to X^* and its function value f^* to F^*.

G5. [Test for stop and set a new number of sample points] If the stop condition (3) is satisfied, then stop. Otherwise set a new number of sample points $N^{(k+1)} \leftarrow (2|X^*|^2 + 3|X^*| + 2)$, update the total number of sample points $N \leftarrow N + N^{(k+1)}$, set $X_c \leftarrow \emptyset$, $F_c \leftarrow \emptyset$, $k \leftarrow k + 1$, and go to step **G2**.

From the set of local optima X^* and the set of optimal values F^* obtained by the algorithm, we can obtain the global optimum $\boldsymbol{x}^{**}$ and the global optimal value f^{**} with the smallest value in the set of local optimal values F^*.

5. SAMPLING

5.1. Lattice-based Sampling

In order to reduce wasted CPU time for finding l-neighbor points, we propose lattice-based sampling. The sampling consists of the following three stages.

First stage: This stage is used in the first iteration $k = 1$. Divide each coordinate into N_d and generate points $\boldsymbol{x} \equiv (x_1, x_2, \ldots, x_n)$ on a lattice as follows:

$$h_i \leftarrow (U_i - L_i) \,/\, N_d \ (i = 1, 2, \ldots, n),$$
$$x_i \leftarrow L_i + h_i \,/\, 2 + k_i \cdot h_i \ (k_i = 0, 1, \ldots, N_d - 1).$$

Then set $N^{(1)} \leftarrow (N_d)^n$.

Therefore, the k_i-th lattice $(i = 1, 2, \ldots, n)$ point can be denoted by $\boldsymbol{x}_{k_1,k_2,\ldots,k_n}$, and its function value can also be denoted by $f_{k_1,k_2,\ldots,k_n}$. In case such a sampling performed, the condition (4) for detecting point $\boldsymbol{x}_j$:

$$f(\boldsymbol{x}_j) < f(\boldsymbol{x}_i) \qquad (i = 1, 2, \ldots, l),$$

can be transformed into

$$\left| \begin{array}{ll} f_{k_1,k_2,\ldots,k_n} < f_{k_1+1,k_2,\ldots,k_n}, & f_{k_1,k_2,\ldots,k_n} < f_{k_1-1,k_2,\ldots,k_n}, \\ f_{k_1,k_2,\ldots,k_n} < f_{k_1,k_2+1,\ldots,k_n}, & f_{k_1,k_2,\ldots,k_n} < f_{k_1,k_2-1,\ldots,k_n}, \\ \cdots\cdots\cdots\cdots & \cdots\cdots\cdots\cdots \\ f_{k_1,k_2,\ldots,k_n} < f_{k_1,k_2,\ldots,k_n+1}, & f_{k_1,k_2,\ldots,k_n} < f_{k_1,k_2,\ldots,k_n-1}. \end{array} \right. \qquad (5)$$

In this condition, the number of neighbor points becomes $l = 2n$, and we can easily calculate these neighbor $2n$-points.

Second stage: This stage is used in the second iteration $k = 2$. Divide each coordinate into N_d and generate points $\boldsymbol{x}_j \equiv (x_1, x_2, \ldots, x_n)$ on a lattice as follows:

$$h_i \leftarrow (U_i - L_i) \,/\, N_d \ (i = 1, 2, \ldots, n).$$
$$x_i \leftarrow L_i + k_i \cdot h_i \ (k_i = 1, 2, \ldots, N_d - 1).$$

Then set a set of grid points $X^G \leftarrow X^G + \{ \boldsymbol{x} \equiv (x_1, x_2, \ldots, x_n) \}$, where X^G is initialized by an empty set. Let an index set $I^G \leftarrow \{1, 2, \ldots, |X^G|\}$. Then sample points $\boldsymbol{x}_i^{(2)}$ $(i = 1, \ldots, N^{(2)})$ are selected by the following steps.

```
if $N^{(2)} \le |X^G|$ then
    select $l_i$ randomly $l_i \in I^G$ and set $I^G \leftarrow I^G - \{l_i\}$,
    $\boldsymbol{x}_i^{(2)} \leftarrow X_{l_i}^G$, $X^G \leftarrow X^G - \{\boldsymbol{x}_i^{(2)}\}$ $(i = 1, \ldots, N^{(2)})$.
else set $\boldsymbol{x}_i^{(2)} \leftarrow X_i^G$ $(i = 1, \ldots, |X^G|)$, $X^G \leftarrow \emptyset$.
    if $N^{(2)} > |X^G|$ then
        generate uniformly random points $\boldsymbol{x} \in D^n$,
        and set $\boldsymbol{x}_i^{(2)} \leftarrow \boldsymbol{x}$ $(i = |X^G| + 1, \ldots, N^{(2)})$.
```

Since the neighbor l-points of a point $\boldsymbol{x}_j$ become neighbor lattice points at the first stage, we can also easily find these neighbor points.

Third stage: This stage is used after the second iteration $k = 3, 4, \ldots$. In this stage, if $|X^G| = 0$, then sample points $\boldsymbol{x}_i^{(k)}$ $(i = 1, \ldots N^{(k)})$ are generate randomly in D^n, otherwise sample points $\boldsymbol{x}_i^{(k)}$ $(i = 1, \ldots, N^{(k)})$ are selected by similar steps in the second stage.

If an appropriate number $N^{(1)}$ is determined, sampling at step **G2** will be terminated by the second stage $(k = 2)$.

5.2. Subspace Sampling

In the lattice-based sampling, sample size $(N_d)^n$ exponentially increases with respect to dimension n. In order to avoid this problem, we propose *subspace sampling,* which samples on a reduced $n_p(< n)$-dimensional space. We illustrate an example of dividing the original even-dimensional space R^n into two half-dimensional subspaces $R^{n/2}$ and $R^{n/2}$. Moreover, in order to easily understand the method, we treat the Shekel-m problem ($\mathrm{P_S}$) defined on a 4-dimensional space as follows:

$$(\mathrm{P_S})\left|\begin{array}{l}\text{Min } f(x_1,x_2,x_3,x_4) = -\sum_{i=1}^{m} \dfrac{1}{\sum_{j=1}^{4}\{(x_j-a_{ij})^2\}+c_i},\\ \text{Subject to } \quad 0 \le x_1,x_2,x_3,x_4 \le 10,\ (i=1,2,3,4),\\ \quad D^4 = \{\,\boldsymbol{x}\in R^4 \mid [0,10]\times[0,10]\times[0,10]\times[0,10]\,\},\end{array}\right.$$

where m is the number of local minima.

We split the original problem ($\mathrm{P_S}$) defined on a 4-dimensional space into the following two subproblems ($\mathrm{P_{S1}}$) and ($\mathrm{P_{S2}}$) defined on a 2-dimensional subspace:

$$(\mathrm{P_{S1}})\left|\begin{array}{l}\text{Min } f_1(x_1,x_2) = -\sum_{i=1}^{m} \dfrac{1}{\sum_{j=1}^{2}\{(x_j-a_{ij})^2\}+c_i},\\ \text{Subject to } \quad 0 \le x_1,x_2 \le 10,\ (i=1,2),\\ \quad D^2 = \{\,\boldsymbol{x}\in R^2 \mid [0,10]\times[0,10]\,\}.\end{array}\right.$$

$$(\mathrm{P_{S2}})\left|\begin{array}{l}\text{Min } f_2(x_3,x_4) = -\sum_{i=1}^{m} \dfrac{1}{\sum_{j=3}^{4}\{(x_j-a_{ij})^2\}+c_i},\\ \text{Subject to } \quad 0 \le x_3,x_4 \le 10,\ (i=3,4),\\ \quad D^2 = \{\,\boldsymbol{x}\in R^2 \mid [0,10]\times[0,10]\,\}.\end{array}\right.$$

Let the number of divisions at each coordinate be $N_d = 10$ in the case where the sample size $10^4 = 10000$ in $\mathrm{P_S}$ is vastly reduced to the size $2\cdot 10^2 = 200$ in ($\mathrm{P_{S1}}$) and ($\mathrm{P_{S2}}$).

In two such subspaces, sampling and detecting points by TGO are performed. Let two groups of the points by detected TGO in the two subproblems ($\mathrm{P_{S1}}$) and ($\mathrm{P_{S2}}$) be $\boldsymbol{x}_j^1 \equiv (x_{1j}^1, x_{1j}^1) \in D^2$ $(j = 1,\dots,m_1)$ and $\boldsymbol{x}_k^2 \equiv (x_{3k}^1, x_{4k}^2) \in D^2 (k = 1,\dots,m_2)$, respectively. Then, the full-dimensional detected points $\boldsymbol{x}_j \in D^4$ $(j = 1,\dots,m_1)$ are constructed as follows:

$$\boldsymbol{x}_j \equiv (x_{1j},x_{2j},x_{3j},x_{4j}) \leftarrow (x_{1j}^1, x_{2j}^1, \underline{x}_{3k}^1, \underline{x}_{4k}^1) \quad j = 1,\dots,m_1\,,$$

where

$$(\underline{x}_{3k}^1, \underline{x}_{4k}^1) = \operatorname*{argmin}_{1\le k\le m_2}\left\{ f(x_{3k}^1, x_{4k}^1)\right\}.$$

6. SIMPLE NUMERICAL EXPERIMENTS

Our algorithm was implemented in programming language C, and the local search used in the experiment was the Quasi-Newton method (BFGS formula) with finite difference approximation for calculating gradient ∇f.

We tested the algorithm for Shekel test functions: S-m $(m = 5,7,10)$ given in the problem (S). A comparison among numerical results obtained by our method, the TGO method [5] and the MLSL method [3] is shown in Table 1.

Table 1: Sekel's test functions S-m of 4 variables with m local minima in an interval $0 \le x_i \le 10$ of each coordinate

Function	TGO		MLSL		Our method	
Name	N_{fe}	UT	N_{fe}	UT	N_{fe}	UT
S-5	265	10.8	404	1	174	0.37
S-7	–	–	432[a]	1	207	0.53
S-10	–	–	564	2	210	0.67

N_{fe} : Number of function evaluations.
UT : Unit time ($UT = 1$ is 1000 function evaluations of S5).
[a] : The global minimum was not found in one of four trials.

These results show that our algorithm can effectively find the global minimum in a fewer function evaluations and a shorter computational time than can other methods.

7. CONCLUDING REMARKS

We have proposed an algorithm for solving global optimization problems that consists of the following steps: i) lattice-based and subspace sampling, ii) detection of points by the TGO method, iii) determination of the starting point by the MLSL criterion, and iv) termination of the algorithm and estimation of the number of sample points on the next iteration by Boender's expectation [1] of the number of local minima.

We have shown that our algorithm can find a global minimum in a short time because calculation for finding neighbor points is completed in a much shorter time than that in the case of using the TGO method.

References

[1] Boender, C.G.E., Rinnooy Kan, A.H.G. and Timmer, G.T.: "A Stochastic Method for Global Optimization," Mathematical Programming, **22**, 125–140, 1982.

[2] C. A. Floudas: "Deterministic Global Optimization –Theory, Methods and Applications–," Kluwer A.P., 2000.

[3] Rinnooy Kan, A.H.G. and Timmer, G.T. : "Stochastic Global Optimization Methods. Part II: Multi Level Methods," Mathematical Programming, **39** 57–78, 1987.

[4] Törn, A.A. and Žilinskas, A.A.: "*Global Optimization,*" Springer-Verlag, Lecture Notes in Computer Science 350, Berlin, 1989.

[5] Törn, A.A. and Viitanen, S.: "Topograhical Global Optimization," In *Recent Advances in Global Optimization,* Floudas, C.A. and Pardaros, P.M eds., Princeton Series in Computer Science: U.S.A, 384–398, 1992.