Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2012, Article ID 969104, 13 pages
doi:10.1155/2012/969104

Research Article

SOMO-*m* Optimization Algorithm with Multiple Winners

Wei Wu and Atlas Khan

Department of Applied Mathematics, Dalian University of Technology, 116024 Dalian, China

Correspondence should be addressed to Atlas Khan, atlas900@gmail.com

Received 1 April 2012; Revised 6 June 2012; Accepted 6 June 2012

Academic Editor: M. De la Sen

Self-organizing map (SOM) neural networks have been widely applied in information sciences. In particular, Su and Zhao proposes in (2009) an SOM-based optimization (SOMO) algorithm in order to find a wining neuron, through a competitive learning process, that stands for the minimum of an objective function. In this paper, we generalize the SOM-based optimization (SOMO) algorithm to so-called SOMO-*m* algorithm with *m* winning neurons. Numerical experiments show that, for $m > 1$, SOMO-*m* algorithm converges faster than SOM-based optimization (SOMO) algorithm when used for finding the minimum of functions. More importantly, SOMO-*m* algorithm with $m \geq 2$ can be used to find two or more minimums simultaneously in a single learning iteration process, while the original SOM-based optimization (SOMO) algorithm has to fulfil the same task much less efficiently by restarting the learning iteration process twice or more times.

1. Introduction

Self-organizing map (SOM) neural networks have been widely applied in information sciences [1–7]. In particular, an SOM-based optimization (SOMO) algorithm is proposed in [8, 9] to find a winning neuron, through a competitive learning process, that stands for the minimum of an objective function. They compared the SOM-based optimization (SOMO) algorithm with genetic algorithms [10, 11] and particle swarm optimization algorithm [12–17], and they showed that the SOM-based optimization (SOMO) algorithm can locate the minimum much faster than the genetic algorithm and the particle swarm optimization algorithm.

The aim of this paper is to generalize SOMO algorithm to so called SOMO-*m* algorithm to find *m* winning neurons in a single learning process. A separation technique is introduced in the learning process to prevent the accumulation of the m-wining neurons. Numerical experiments show that, for $m \geq 2$, SOMO-m algorithm converges faster than SOMO algorithm when they are used for finding the minimum of functions. A more important merit of SOMO-m algorithm with $m \geq 2$ is that it can find two or more minimums simultaneously in a single learning iteration process, while the original SOMO algorithm has to fulfil the same task much less efficiently by restarting the learning iteration process twice or more times.

This paper is organized as follows. In Section 2, a brief introduction of SOMO algorithm is given. Our SOMO-m algorithm is proposed in Section 3. Section 4 is devoted to some supporting numerical simulations.

2. Original SOM-Based Optimization (SOMO) Algorithm

Self-organizing map (SOM) is an unsupervised learning algorithm proposed by Kohonen [18–20]. The principal goal of the SOM algorithm is to map an incoming pattern in a higher dimensional space into a lower (usually one or two) dimensional space, and perform this transforation adaptively in a topological ordered fashion. The applications of SOM range widely from simulations used for the purpose of understanding and modeling of computational maps in the brain to subsystems for engineering applications such as speech recognition, vector quantization and cluster analysis [18–26].

Different from the usual SOM algorithm, an SOM-Based optimization (SOMO) algorithm is introduced in Su and Zhao [8] for continuous optimization. In the following, let us describe SOMO algorithm [8] used for finding the minimum point x^* of a function $f(x)$, $x \in R^n$. The SOMO network contains $M \times N$ neurons arranged as a two dimensional array. For each neuron (i, j), its weight $\underline{W}_{i,j}$ is a vector in R^n, where $1 \leq i \leq M$ and $1 \leq j \leq N$ for some positive integers M and N. For a special input $\underline{x} = (x_1, \ldots, x_n) = (1, 1, \ldots, 1) \in R^n$, the winner out of all the neurons is defined as

$$\begin{aligned}
(i^*, j^*) &= \underset{1\leq i\leq M, 1\leq j\leq N}{\arg} \min f\left(\left(\underline{W}_{i,j}\right)_1 \times x_1, \ldots, \left(\underline{W}_{i,j}\right)_n \times x_n\right) \\
&= \underset{1\leq i\leq M, 1\leq j\leq N}{\arg} \min f\left(\left(\underline{W}_{i,j}\right)_1 \times 1, \ldots, \left(\underline{W}_{i,j}\right)_n \times 1\right) \\
&= \underset{1\leq i\leq M, 1\leq j\leq N}{\arg} \min f\left(\underline{W}_{i,j}\right).
\end{aligned} \tag{2.1}$$

The idea of SOM training is applied to the network such that the weight $\underline{W}_{i^*,j^*}$ of the winner will get closer and closer to the minimum point x^* during the iterative training process. The detail of the training process is as follows:

Step 1. Initialization.

Substep 1 (Initialization of the neurons on the four corners). The weight vectors of the neurons on the corners are initialized as follows:

$$\underline{W}_{1,1} = (l_1, l_2, \ldots, l_n)^T,$$

$$\underline{W}_{M,N} = (h_1, h_2, \ldots, h_n)^T,$$

$$\underline{W}_{1,N} = \left(l_1, l_2, \ldots, l_{\lfloor n/2 \rfloor}, h_{\lfloor n/2 \rfloor + 1}, \ldots, h_n\right)^T,$$
$$\underline{W}_{M,1} = \left(h_1, h_2, \ldots, h_{\lfloor n/2 \rfloor}, l_{\lfloor n/2 \rfloor + 1}, \ldots, l_n\right)^T. \tag{2.2}$$

Here, the two points $(l_1, l_2, \ldots, l_n)^T$ and $(h_1, h_2, \ldots, h_n)^T$ are randomly chosen and far enough from each other.

Substep 2 (Initialization of the neurons on the four edges). The initialization of the weights of the neurons on the four edges is as follows:

$$\begin{aligned}
\underline{W}_{1,j} &= \frac{\underline{W}_{1,N} - \underline{W}_{1,1}}{N-1}(j-1) + \underline{W}_{1,1} \\
&= \frac{j-1}{N-1}\underline{W}_{1,N} + \frac{N-j}{N-1}\underline{W}_{1,1}, \\
\underline{W}_{M,j} &= \frac{\underline{W}_{M,N} - \underline{W}_{M,1}}{N-1}(j-1) + \underline{W}_{M,1} \\
&= \frac{j-1}{N-1}\underline{W}_{M,N} + \frac{N-j}{N-1}\underline{W}_{M,1}, \\
\underline{W}_{i,1} &= \frac{\underline{W}_{M,1} - \underline{W}_{1,1}}{M-1}(i-1) + \underline{W}_{1,1} \\
&= \frac{i-1}{M-1}\underline{W}_{M,1} + \frac{M-i}{M-1}\underline{W}_{1,1}, \\
\underline{W}_{i,N} &= \frac{\underline{W}_{M,N} - \underline{W}_{1,N}}{M-1}(i-1) + \underline{W}_{1,N} \\
&= \frac{i-1}{M-1}\underline{W}_{M,N} + \frac{M-i}{M-1}\underline{W}_{1,N},
\end{aligned} \tag{2.3}$$

where $i = 2, \ldots, M-1$ and $j = 2, \ldots, N-1$.

Substep 3 (Initialization of the remaining neurons).

$$\begin{aligned}
\underline{W}_{i,j} &= \frac{\underline{W}_{i,N} - \underline{W}_{i,1}}{N-1}(j-1) + \underline{W}_{i,1} \\
&= \frac{j-1}{N-1}\underline{W}_{i,N} + \frac{N-j}{N-1}\underline{W}_{1,N},
\end{aligned} \tag{2.4}$$

where $i = 2, \ldots, M-1$ and $j = 2, \ldots, N-1$.

Substep 4 (Random noise). A small amount of random noise is added for each weight so as to keep the weight vectors from being linearly dependent as follows:

$$\underline{W}_{i,j} = \underline{W}_{i,j} + \underline{t} \tag{2.5}$$

for $1 \le i \le M$ and $1 \le j \le N$, where $\underline{t}$ denotes a small random noise.

Step 2. Winner finding.

$$\left(i^*, j^*\right) = \underset{1\le i\le M, 1\le j\le N}{\arg} \min f\left(\underline{W}_{i,j}\right) \tag{2.6}$$

Step 3. Weights updating. The weights of the winner and its neighbors are adjusted by the following formula:

$$\begin{aligned}\underline{W}_{i,j}(t+1) &= \underline{W}_{i,j}(t) + \eta\beta\left(i^*, j^*, i, j\right)\left[\underline{W}_{i^*,j^*}(t) - \underline{W}_{i,j}(t)\right] \\ &\quad + \lambda\left(1 - \beta\left(i^*, j^*, i, j\right)\right)\underline{p} \quad \text{for } 1 \le i \le M,\ 1 \le j \le N,\ t = 0, 1, \ldots,\end{aligned} \tag{2.7}$$

where the parameters η and λ are real-valued constants which can be either constants predefined by the user or time-varying parameters which decrease gradually when time t increases. $d(i^*, j^*, i, j)$ is the lateral distance between winning neuron (i^*, j^*) and neuron (i, j); the randomly chosen vector $\underline{p}$ is called the perturbation vector, and

$$\beta\left(i^*, j^*, i, j\right) = 1 - \frac{d\left(i^*, j^*, i, j\right)}{\sqrt{M^2 + N^2}}. \tag{2.8}$$

Step 4. Go to step 2 until a prespecified number of generations is achieved, or some kind of termination criterion is satisfied.

3. SOMO-*m* Algorithm

In this section, let us present our SOMO-*m* algorithm. We divide the section into two subsections, dealing with the training algorithms for finding one minimum and two minima of a function, respectively.

3.1. SOMO-m Algorithm for Finding One Minimum

The SOMO-*m* algorithm for finding one minimum of a function $f(x)$ is as follows:

Step 1. The initialization of SOMO-*m* algorithm is the same as that of SOMO algorithm.

Step 2. This step aims to find m winning neurons, denoted by $(i_1^*, j_1^*), (i_2^*, j_2^*), \ldots, (i_m^*, j_m^*)$ with the best objective function values among the neurons as follows:

$$\begin{aligned}
(i_1^*, j_1^*) &= \underset{1\le i\le M, 1\le j\le N}{\arg} \min f\left(\underline{W}_{i,j}\right), \\
(i_2^*, j_2^*) &= \underset{i\in\{1,\ldots,M\}\setminus S_1, j\in\{1,\ldots,N\}\setminus T_1}{\arg} \min f\left(\underline{W}_{i,j}\right), \\
&\vdots \\
(i_m^*, j_m^*) &= \underset{i\in\{1,\ldots,M\}\setminus S^m, j\in\{1,\ldots,N\}\setminus T_m}{\arg} \min f\left(\underline{W}_{i,j}\right),
\end{aligned} \tag{3.1}$$

where

$$S_1 = \left\{i : \left|i - i_1^*\right| \le R_1\right\}, \qquad T_1 = \left\{j : \left|j - j_1^*\right| \le R_1\right\}, \tag{3.2}$$

$$\begin{aligned}
S_2 = \left\{i : \left|i - i_2^*\right| \le R_2\right\}, &\qquad T_2 = \left\{j : \left|j - j_2^*\right| \le R_2\right\}, \\
&\vdots
\end{aligned} \tag{3.3}$$

$$S_m = \{i : |i - i_m^*| \le R_m\}, \qquad T_m = \left\{j : \left|j - j_r^*\right| \le R_m\right\}, \tag{3.4}$$

$$S^m = (S_1 \cup \cdots \cup S_m), \qquad T_m = (T_1 \cup \cdots \cup T_m), \tag{3.5}$$

and $R_1, R_2, \ldots, R_m$ are suitably chosen sizes of neighborhoods.

Step 3. The weights of the winners and its neighbors are adjusted by the following formula:

$$\begin{aligned}
\underline{W}_{i,j}(t+1) = \underline{W}_{i,j}(t) &+ \sum_{r=1}^{m} \eta_r \beta(i_r^*, j_r^*, i, j)\left[\underline{W}_{i_r^*, j_r^*}(t) - \underline{W}_{i,j}(t)\right] \\
&+ \sum_{r=1}^{m} \lambda_r (1 - \beta(i_r^*, j_r^*, i, j))\underline{p} \quad 1 \le i \le M,\ 1 \le j \le N,
\end{aligned} \tag{3.6}$$

where

$$\beta(i_r^*, j_r^*, i, j) = 1 - \frac{d(i_r^*, j_r^*, i, j)}{\sqrt{M^2 + N^2}}. \tag{3.7}$$

The η_r and λ_r are real-valued parameters which can be constants predefined by the user or time-varying parameters, which decreases gradually with increasing time t, and $\underline{p}$ is a perturbation vector.

Step 4. Go to step 2 until a prespecified number of generations is achieved or some kind of termination criteria is satisfied.

We first remark that in the above Step 2, we require the m winners to be away from each other by using the neighborhoods S_m and T_m, such that the winners can spread around rather than accumulate in one point. Secondly, we find in our numerical experiments that it is OK to choose λ_r and η_r as either small constants independent of time t or small variables decreasing with increasing time t. Thirdly, we find that it is better to choose $\eta_1 < \eta_2 \leq \cdots \leq \eta_m$. The reason for this may be the following: A smaller constant η_1 might help for the first winner to converge to the minimum, while the larger η_r for $r > 1$ might help for the other winners to reach a larger searching area.

3.2. SOMO-2 Algorithm for Two Minima

Here we discuss SOMO-2 algorithm for finding two minima of a function simultaneously. It is an easy matter to generalize the algorithm for finding three or more minima. SOMO-2 algorithm has the same training steps as those in the last subsection with $m = 2$ for finding one minimum, except the step of weights updating process, which is as follows:

For the neurons (i, j) in the neighborhood of the first winner (i_1^*, j_1^*) satisfying $p_1 \leq i \leq p_2$, $q_1 \leq j \leq q_2$, where

$$\begin{aligned} p_1 &= \max\left(i_1^* - R_1, 1\right), \\ p_2 &= \min\left(i_1^* + R_1, M\right), \\ q_1 &= \max\left(j_1^* - R_1, 1\right), \\ q_2 &= \min\left(j_1^* + R_1, N\right). \end{aligned} \tag{3.8}$$

The weights updating rule is

$$\underline{W}_{i,j}(t+1) = \underline{W}_{i,j}(t) + \eta_1 \beta_1\left(i_1^*, j_1^*, i, j\right)\left[\underline{W}_{i_1^*, j_1^*}(t) - \underline{W}_{i,j}(t)\right] + \lambda_1\left(1 - \beta_1\left(i_1^*, j_1^*, i, j\right)\right)\underline{p}, \tag{3.9}$$

where

$$\beta_1\left(i_1^*, j_1^*, i, j\right) = 1 - \frac{d\left(i_1^*, j_1^*, i, j\right)}{\sqrt{M^2 + N^2}}. \tag{3.10}$$

For the rest (i, j) neurons,

$$\underline{W}_{i,j}(t+1) = \underline{W}_{i,j}(t) + \eta_2 \beta_2\left(i_2^*, j_2^*, i, j\right)\left[\underline{W}_{i_2^*, j_2^*}(t) - \underline{W}_{i,j}(t)\right] + \lambda_2\left(1 - \beta_2\left(i_2^*, j_2^*, i, j\right)\right)\underline{p}, \tag{3.11}$$

where

$$\beta_2\left(i_2^*, j_2^*, i, j\right) = 1 - \frac{d\left(i_2^*, j_2^*, i, j\right)}{\sqrt{M^2 + N^2}}. \tag{3.12}$$

We see that, in the training, the second winner (i_2^*, j_2^*) does not affect the neighboring neurons of the first winner (i_1^*, j_1^*), but does affect all the other neurons. Therefore, hopefully the first winner will converges to the minimum point of the function $f(x)$, while the second winner converges to another minimum point.

4. Simulation Results

4.1. Objective Functions

In this subsection, we use our SOMO-m methods to minimize the following functions.

(1) Step function

$$f(\underline{x}) = \sum_{i=1}^{30} (|x_i + 0.5|)^2. \tag{4.1}$$

(2) Griewant function

$$f(\underline{x}) = \frac{1}{4000}\sum_{i=1}^{30}(x_i - 100)^2 - \prod_{i=1}^{10}\cos\left(\frac{x_i - 100}{\sqrt{i}}\right) + 1. \tag{4.2}$$

(3) Giunta function

$$\begin{aligned} f(\underline{x}) = \sum_{i=1}^{30} \sin\left(\frac{16}{15}x_i - 1\right) + \sin^2\left(\frac{16}{15}x_i - 1\right) \\ + \frac{1}{50}\sin\left(4\left(\frac{16}{15}x_i - 1\right)\right) + 0.3. \end{aligned} \tag{4.3}$$

(4) U_1 function

$$f = -\exp(-8\sin(\pi x)) - \exp(9.85\sin(2\pi y)). \tag{4.4}$$

(5) U_2 function

$$f = \exp(-8\sin(\pi x) + \sin(\pi y)) - \exp(\sin(2.5\pi x) + \sin(\pi y)). \tag{4.5}$$

4.2. Parameters of Simulation

In Table 1 we present the global minima, dimensions, and the upper bound of the number of generations for optimization algorithms. Figure 1 illustrates the graphs of these five functions in two dimensional space. The neurons are arranged as a 30×30 array, namely, $M = N = 30$. For SOMO algorithm, we set $\eta = 0.2$ and $\lambda = 0.01$; for SOMO-2, $\eta_1 = 0.2$, $\eta_2 = 0.3$, $\lambda_1 = 0.01$, and $\lambda_2 = 0.001$; for SOMO-3, $\eta_1 = 0.2$, $\eta_2 = 0.3$, $\eta_3 = 0.35$, $\lambda_1 = 0.01$, $\lambda_2 = 0.001$, and $\lambda_3 = 0.0005$.

Table 1: Parameters for five functions.

Test function	Dimensions	Initial range	Minimum	Number of generations
Step	10	$-100 \le x_i \le 100$	0	100
Griewank	10	$-600 \le x_i \le 600$	0	100
Giunta	30	$-10 \le x_i \le 10$	≈ 0.9	100
U_1 function	2	$0 \le x_i \le 1$		100
U_2 function	2	$0 \le x_i \le 1$		100

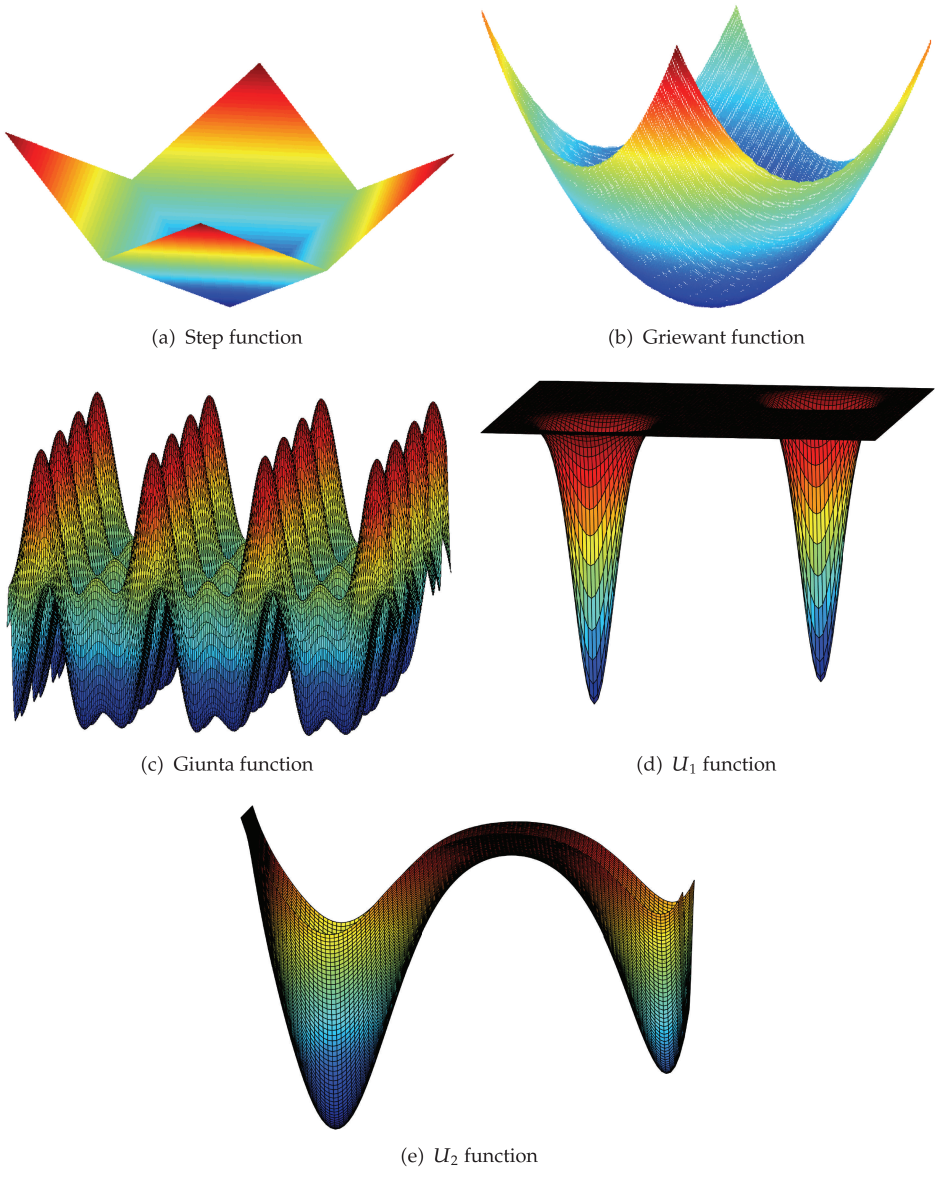

(a) Step function (b) Griewant function (c) Giunta function (d) U_1 function (e) U_2 function

Figure 1: Graphs of the five objective functions.

Table 2: Comparison of SOMO, SOMO-2, and SOMO-3 for finding one minimum.

Test function	Algorithm	Mean	Standard deviation	Time mean (s)	Time SD (s)
Step	SOMO	$1.6778e-016$	$2.4504e-016$	1.3813	1.4008
	SOMO-2	$1.1321e-019$	$8.3427e-020$	1.0973	1.1419
	SOMO-3	**$7.8070e-020$**	$5.7362e-020$	**0.8743**	0.8686
Griewant	SOMO	$5.1773e-015$	$2.8126e-016$	1.7502	1.1681
	SOMO-2	$2.8126e-016$	$3.7420e-016$	1.1681	1.1324
	SOMO-3	**$2.2204e-017$**	$6.14813e-017$	**0.9849**	
Giunta	SOMO	0.9670	$1.4901e-009$	1.0116	1.0049
	SOMO-2	0.9670	$9.2269e-011$	0.7087	0.7053
	SOMO-3	**0.9670**	$5.8655e-012$	**0.6217**	0.6451
U_1 function f	SOMO	$-1.896401167771170e+004$	$1.018269572071048e-007$	0.8009	0.7584
	SOMO-2	$-1.896401167771323e+004$	$2.722830731013220e-009$	0.6168	0.6438
	SOMO-3	**$-1.896401167771335e+004$**	$2.146099170750331e-011$	**0.4657**	0.4778
U_2 function f	SOMO	−7.36465824205229	$9.781475238601051e-014$	2.1060	2.1278
	SOMO-2	−7.36465824208548	$2.101813757029311e-015$	1.5919	1.5198
	SOMO-3	**−7.36465824208548**	$2.394240849331165e-015$	**0.9831**	0.9525

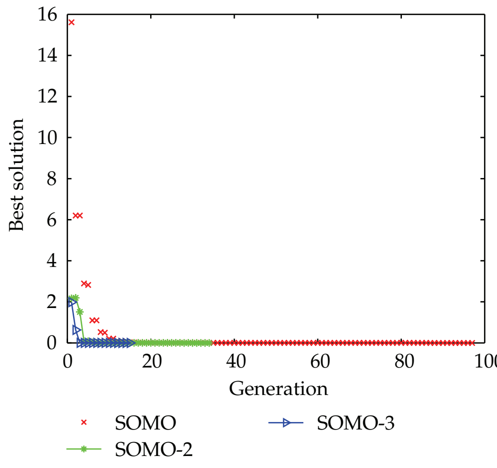
4.3. Simulations of SOMO-m Algorithm for One Minimum

In this subsection, we investigate the performance of SOMO-m algorithm for finding one minimum. For each function, each algorithm conducted 30 runs. The stop criterion for each run is that either 100 generations are iteratively generated, or before then if the difference of two successive minima in the iteration process is less than a prescribed tolerance $\epsilon > 0$. The best solutions found for each function and each run were recorded and, for instance, the mean column in Table 2 presents the average of the 30 best solutions of the 30 runs, respectively. To measure and compare the performance of SOMO-3, the mean time, that is, the average of processing time over 30 runs is recorded. Table 2 tabulates the comparison of the simulation results of SOMO, SOMO-2, and SOMO-3 algorithms. The mean column and the standard deviation (SD) column represent the mean and the standard deviation (SD) of the best solutions of 30 runs. The highlighted (bold) entries correspond to best solutions found by SOMO-3. Figure 2 shows the performances of SOMO, SOMO-2, and SOMO-3 algorithms in typical runs.

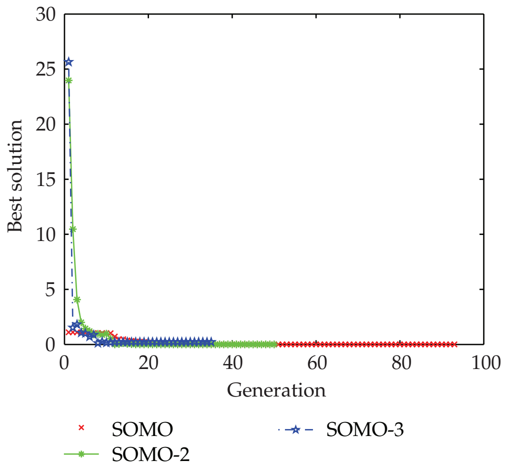
Now, we discuss the accuracy of the algorithms. We measure the errors for each algorithm over the 30 runs as follows:

$$E = \frac{1}{30}\sum_{l=1}^{30}\left|\underline{x}^{l} - \underline{x}^{*}\right|, \tag{4.6}$$

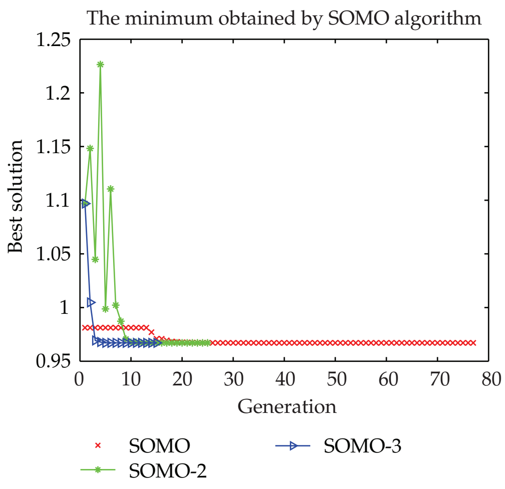
where $\underline{x}^{l}$ is the approximate solution of lth run, and $\underline{x}^{*}$ is the real minimum of the function. Table 3 tabulates the errors of SOMO, SOMO-2, and SOMO-3 algorithms, respectively.



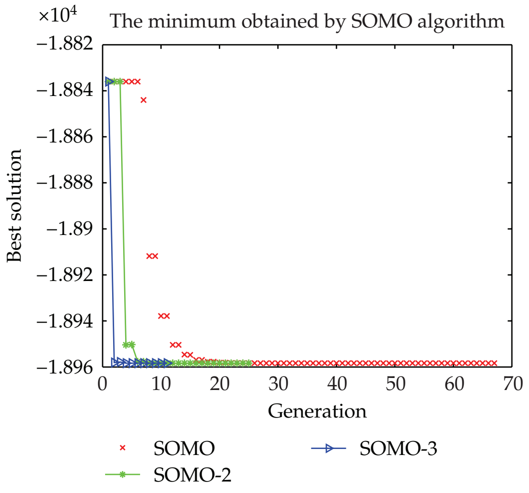
(a) Step function



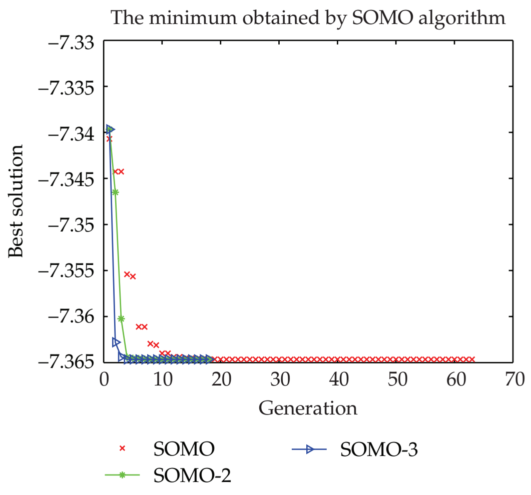
(b) Griewant function



(c) Giunta function



(d) U_1 function



(e) U_2 function

Figure 2: Best performance curves for SOMO, SOMO-2, and SOMO-3 algorithms, respectively.

Table 3: Error of SOMO, SOMO-2, and SOMO-3.

Test function	Algorithm	Error
Step	SOMO	$7.2868e - 017$
	SOMO-2	$4.0385e - 019$
	SOMO-3	**$5.3813e - 020$**
Griewank	SOMO	$6.4936e - 017$
	SOMO-2	$2.2973e - 019$
	SOMO-3	**$6.8401e - 020$**
Giunta	SOMO	0.067056308053595
	SOMO-2	0.067056308053587
	SOMO-3	**0.067056308053585**

Table 4: Comparison of SOMO-2 and the original SOMO algorithms for finding two minima.

Test function	Algorithm	Mean	Standard deviation	Time mean (s)	Time SD (s)
U_2 function	SOMO				
	First minimum	−7.36465824205229	$9.781475238601051e - 014$	2.1060	2.1278
	Second minimum	−5.99322845918991	0.00379247207700	2.9716	2.865
	SOMO-2				
	First minimum	−7.36465824124180	$4.047965312150572e - 009$	1.4967	1.4237
	Second minimum	−6.07080647154294	$6.025651408745671e - 013$		
U_1 function	SOMO				
	First minimum	$-1.895835670663015e + 004$	$3.927781801202260e - 010$	1.2414	1.1168
	Second minimum	$-1.895837311768572e + 004$	$4.152186941602799e - 009$	1.1343	1.0938
	SOMO-2				
	First minimum	$-1.895837311745155e + 004$	$2.482103550304831e - 007$	0.5536	0.5631
	Second minimum	$-1.895837311768680e + 004$	$2.906907306783492e - 009$		

We draw the following conclusions from the simulations in this subsection for finding one minimum.

(1) SOMO-3 algorithm shows the best results for all functions based on the comparisons of the means of the best objective value and the processing time.

(2) As shown in Figure 2, the SOMO-3 algorithm locates the minima faster than SOMO and SOMO-2 algorithms.

(3) In Table 3, the bold entries show that the error corresponding to SOMO-3 algorithm is smaller than those of SOMO and SOMO-2 algorithms.

4.4. Simulations of SOMO-2 Algorithm for Two Minima

In this subsection, we present the simulation results for the case of finding simultaneously two minima of a function. The best solutions found for each run after prespecified number of generations were recorded. Table 4 tabulates the comparison of the simulation results. The mean column and the standard deviation column represents the mean and the standard deviation of the best solutions of 30 runs.

Based on observation from Table 4, our conclusion for finding simultaneously two minima of a function is as follows:

The proposed SOMO-2 algorithm can find two minima of a function simultaneously in a single learning iteration process, while the original SOM-based optimization (SOMO) algorithm has to fulfil the same task much less efficiently by restarting the learning iteration process twice or more times.

Acknowledgments

The authors wish to thank the Associate Editor and the anonymous reviewers for their helpful and interesting comments. This work was supported by National Science Foundation of China (11171367) and the Fundamental Research Funds for the Central Universities of China.

References

[1] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[2] Y. Xiao, C. S. Leung, T. Y. Ho, and P. M. Lam, "A GPU implementation for LBG and SOM training," *Neural Computing and Applications*, vol. 20, no. 7, pp. 1035–1042, 2010.

[3] I. Valova, D. Beaton, A. Buer, and D. MacLean, "Fractal initialization for high-quality mapping with self-organizing maps," *Neural Computing and Applications*, vol. 19, no. 7, pp. 953–966, 2010.

[4] M. Rubio and V. Giménez, "New methods for self-organising map visual analysis," *Neural Computing and Applications*, vol. 12, no. 3-4, pp. 142–152, 2003.

[5] A. Delgado, "Control of nonlinear systems using a self-organising neural network," *Neural Computing and Applications*, vol. 9, no. 2, pp. 113–123, 2000.

[6] N. Ahmad, D. Alahakoon, and R. Chau, "Cluster identification and separation in the growing self-organizing map: application in protein sequence classification," *Neural Computing and Applications*, vol. 19, no. 4, pp. 531–542, 2010.

[7] T. Kohonen, *Self-Organizing Maps*, vol. 30 of *Springer Series in Information Sciences*, Springer, Berlin, Germany, 2nd edition, 1997.

[8] M. C. Su and Y. X. Zhao, "A variant of the SOM algorithm and its interpretation in the viewpoint of social influence and learning," *Neural Computing and Applications*, vol. 18, no. 8, pp. 1043–1055, 2009.

[9] M. C. Su, Y. X. Zhao, and J. Lee, "SOM-based Optimization," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 781–786, Budapest, Hungary, July 2004.

[10] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Mich, USA, 1975.

[11] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, UK, 1989.

[12] M. S. Arumugam and M. V. C. Rao, "On the performance of the particle swarm optimization algorithm with various inertia weight variants for computing optimal control of a class of hybrid systems," *Discrete Dynamics in Nature and Society*, vol. 2006, Article ID 79295, 17 pages, 2006.

[13] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*, Academic Press, New York, NY, USA, 2001.

[14] M. S. Arumugam and M. V. C. Rao, "On the optimal control of single-stage hybrid manufacturing systems via novel and different variants of particle swarm optimization algorithm," *Discrete Dynamics in Nature and Society*, no. 3, pp. 257–279, 2005.

[15] R. Eberhart and J. Kennedy, "New optimizer using particle swarm theory," in *Proceedings of the 6th International Symposium on Micro Machine and Human Science*, pp. 39–43, October 1995.

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, December 1995.

[17] P. Umapathy, C. Venkataseshaiah, and M. S. Arumugam, “Particle swarm optimization with various inertia weight variants for optimal power flow solution,” *Discrete Dynamics in Nature and Society*, vol. 2010, Article ID 462145, 15 pages, 2010.

[18] T. Kohonen, *Self-Organizing Maps*, vol. 30 of *Springer Series in Information Sciences*, Springer, Berlin, Germany, 3rd edition, 2001.

[19] T. Kohonen, E. Oja, O. Simula, A. Visa, and J. Kangas, “Engineering applications of the self-organizing map,” *Proceedings of the IEEE*, vol. 84, no. 10, pp. 1358–1383, 1996.

[20] T. Kohonen, *Self-Organization and Associative Memory*, vol. 8 of *Springer Series in Information Sciences*, Springer, New York, NY, USA, 3rd edition, 1989.

[21] J. Malone, K. McGarry, S. Wermter, and C. Bowerman, “Data mining using rule extraction from Kohonen self-organising maps,” *Neural Computing and Applications*, vol. 15, no. 1, pp. 9–17, 2006.

[22] M. Oja, S. Kaski, and T. Kohonen, “Bibliography of self-organizing map (SOM),” *Proceedings of the IEEE*, vol. 84, no. 10, pp. 1358–1383, 2003.

[23] S. Kaski, J. Kangas, and T. Kohonen, “Bibliography of self organizing map,” *Neural Computing Surveys*, vol. 1, 1998.

[24] P. K. Sharpe and P. Caleb, “Self organising maps for the investigation of clinical data: a case study,” *Neural Computing and Applications*, vol. 7, no. 1, pp. 65–70, 1998.

[25] H. Merdun, “Self-organizing map artificial neural network application in multidimensional soil data analysis,” *Neural Computing and Applications*, vol. 20, no. 8, pp. 1295–1303, 2010.

[26] B. Lamrini, E. K. Lakhal, M. V. Le Lann, and L. Wehenkel, “Data validation and missing data reconstruction using self-organizing map for water treatment,” *Neural Computing and Applications*, vol. 20, no. 4, pp. 575–588, 2011.