Paper

Tracking optima in dynamic problems by an optimizer based on piecewise-rotational chaos system

Yoshikazu Yamanaka\textsuperscript{1a)} and Tadashi Tsubone\textsuperscript{2b)}

\textsuperscript{1} Utsunomiya University, 7-1-2 Yoto, Utsunomiya, Tochigi 321-8585, Japan
\textsuperscript{2} Nagaoka University of Technology, 1603-1, Kamitomioka, Niigata 940-2188, Japan

\textsuperscript{a)} yyamanaka@cc.utsunomiya-u.ac.jp
\textsuperscript{b)} tsubone@vos.nagaokaut.ac.jp

Received September 25, 2017; Revised March 27, 2018; Published October 1, 2018

Abstract: Tracking optima in dynamic problems is achieved by a multi-population optimizer based on piecewise-rotational chaotic system (OPRC) using memory update within a tolerance. Tracking optima is a difficult task for multi-population-based optimizers because of two issues, called outdated memory and divergent loss. To solve the outdated memory issue, a simple procedure named memory update within a tolerance is proposed. The proposed procedure is applied to our previous proposed optimizer OPRC, and its outstanding tracking performance is observed. This result shows OPRC can solve the divergent loss issue without any modification of its searching dynamics. The tracking mechanism of OPRC is also considered, and it is uncovered that the searching behavior given by the folding dynamics of the chaotic system contributes to catch the shifting optima.

Key Words: optimization methods, dynamic optimization problems, chaos, nonlinear dynamical systems

1. Introduction

Optimization problems and its solving methods called optimizers are important research topics in engineering. Many multi-population-based optimizers have been proposed, which search the optimal solution by searching points, such as Nelder-Mead method [1] and particle swarm optimization (PSO) [2, 3]. In general, subjects of these methods are static optimization problems, whose optimal solution is stationary. However, in the real-world problems, we sometimes have to deal with dynamic optimization problems, whose optimal solution would change depending on the time, such as the maximum power point tracking in photovoltaic systems [4, 5] and adaptive PID control in mechanical systems [6]. The dynamic problems for maximizing a function $f: \mathbb{R}^d \times \mathbb{Z} \rightarrow \mathbb{R}$ is

$$\max \{ f(x, n) \mid x \in S, n \geq 0 \},$$

(1)
where $\mathbb{R}$ and $\mathbb{Z}$ are sets of real-numbers and integers, respectively; $n = 0, 1, 2, \ldots$ is time-step; $x = \{x_1, x_2, \ldots, x_d\}$ is a feasible solution; $d$ is the dimension of the problem; and $S \in \mathbb{R}^d$ is the set of the feasible solutions. $x_{opt}(n)$ where $f(x_{opt}(n), n) \geq f(x, n), \forall x \in S$ is the optimal solution at $n$. For the dynamic problems, it is required not only finding but also tracking the changing optimal solution. The multi-population-based optimizers may be a particularly suitable candidate for this type of problems [7], and a large number of their variants, which are designed to solve the dynamic problems, have been studied as described in survey papers [8, 9].

In this paper, we consider the tracking for a few dimensional, unconstrained ($S = \mathbb{R}^d$) and continuously changing problems where the optimal solution changes with small distance, therefore, the optima can be assumed to be continuously shifted depending on $n$. This is a basic and important dynamic problem, and many researchers have focused on this problem [6, 8–13]. We also focus on simple multi-population-based optimizers, whose searching points are updated by a dynamical or stochastic system. Some researchers have studied about more complex dynamic problems, such as more than 4-dimensional and discontinuously changing optimal solutions, and optimizers for them [7, 14–16], however, these problems and methods are beyond of this paper. These optimizers have been extended the basic optimization methods by using multi-swarm [7], time-variant dynamical systems [14] and hierarchical population [15, 16]. Because of that, this paper focuses on the fundamental study and will contribute to enhance such extended optimizers by improving their basic performance.

Even though the tracking task in continuous changing problems is relatively easy, the basic multi-population-based optimizers which are designed to solve the static problems probably fail to do the task. As the reasons for that, outdated memory and divergent loss issues have been reported [13]. In general, the multi-population-based methods memorize the best obtained solution and its fitness value, such as global best $gb(t)$ and their fitness $F_{gb} = f(gb(t))$ of PSO. In dynamic optimization problems, however, the memorized fitness could be outdated because the function is time-variant: $f(gb(t), n + \Delta) \neq f(gb(t), n)$. When the stored fitness becomes outdated, optimizers do not correctly update the best obtained solution, and fail to track the optimal solution (we show the detail of this in Sec. 3.2). This problem is called outdated memory. The other issue is divergent loss. The population-based optimizers search the better solutions in the specific range around the memory, such as $gb(t)$. This is because they are designed based on the proximate optimality principle [17, 18]. Therefore, if the optimal solution jumps to the outside far away from their searching range, optimizers would not catch the jumped optima. Note that even though the outdated memory issue is solved, once all searching points converge to narrow range, it takes a lot of iterations to follow the shifted optima, then, searching points would be left behind. To solve these two issues, one of the classical schemes is periodically restarting the searching procedure [12]. This scheme is simple and can solve both issues, however, it is difficult to select the resetting period and re-initializing range of the searching points. In addition, the real-world problem would be non-periodic. Therefore, many researchers have focused on the non-periodic approaches. The survey papers [8, 9] have reported that most of the recently proposed methods observe object functions, and once change of the functions is detected, they update the memory and execute procedures to prevent the divergent loss, such as re-evaluating the memory and re-initializing the searching points. However, such detection-based optimizers would worsen the performance for noisy environments because of false detections [9]. Some real-world problems, such as [4–6], should contain the noise, therefore, novel methods without detecting technique have been required [9]. Without detecting, some researchers have tried to solve the two important issues one by one. To solve the outdated memory, some novel memory update procedures have been proposed: worsening the stored fitness values by multiplying a constant [19, 20], simply re-evaluating the fitness for each time-step [6], and worsening the memorized fitness by adding a constant every time-step [21]. However, there are some drawbacks: costs of fitness calculation is increased, and applicable problems (object functions) are limited (see in Sec. 3.2). To solve the divergent loss, atomic PSO (APSO) [13] have been proposed, which modifies the dynamics of PSO by adding a vector to introduce repulsion behavior. APSO is simple and succeeds to prevent the divergent loss, however, the tracking behavior would be rough (This behavior is shown in Sec. 4).

In our previous paper, we have proposed a novel optimizer based on piecewise-rotational chaotic
system (OPRC), and reported its searching performance for static optimization problems [22]. In this paper, we propose a novel memory update procedure and apply it to OPRC. As the results, OPRC achieves the superior tracking performance and solves some of the drawbacks described above. Our novel memory update procedure is called memory update within a tolerance. The procedure is quite simple: the memory can be updated when its fitness is improved and also is worsened within a tolerance value. Even though the proposed procedure is based on almost same idea with [19, 20], this novel procedure can solve the outdated memory for various object functions than [19, 20]. In addition, the procedure does not require additional evaluation costs. These advantages are described in more detail in Sec. 3.2.

We apply the proposed memory update procedure to Nelder-Mead method, PSO, bare born PSO (BBPSO) [23], APSO and OPRC, then, compare the tracking performance for a well-known dynamic benchmark problem: moving peaks benchmark (MPB) [11, 24]. As the results, OPRC tracked the optima with the best accuracy among them. This result is interesting because the chaotic system of OPRC did not modify unlike APSO. In other words, OPRC autonomously prevented the divergent loss issue. We also consider the tracking mechanism of OPRC, and it is revealed that the behavior is caused by the folding mechanism of the chaotic system.

This paper organize as follows: Sec. 2 introduces the compared optimizers with traditional memory update procedure, the memory update within a tolerance and its applied way are described in Sec. 3, Sec. 4 shows the tracking experiments, the consideration is given in Sec. 5, and Sec. 6 concludes this paper.

2. Basic multi-population optimizers for dynamic problems

In this section, Nelder-Mead method [1], PSO [2, 3, 25], BBPSO [23], APSO [13] and our previous proposed method, OPRC [22] are introduced. These methods have originally been proposed for static problems. Here, we consider these algorithms for a maximizing dynamic problem $f$ described by Eq. (1). To search and track the optimal solution, these algorithms update $d$-dimensional vectors called searching points with time-step $t$,

$$ x_i(t) = \{x_{i1}(t), x_{i2}(t), \ldots, x_{ij}(t), \ldots, x_{id}(t)\}, \quad (2) $$

where $i = 1, 2, \ldots, N$ is an index of the searching points and $j = 1, 2, \ldots, d$ is an index of the dimension. In this paper, $x_i(0)$ is given randomly. Let $F_i(t) \in \mathbb{R}$ be the fitness value of $x_i(t)$ given by $F_i(t) = f(x_i(t), n)$.

2.1 Nelder-Mead method

Nelder-Mead method [1] searches the optimal solution by $N = (d+1)$ searching points. The algorithm is described in Algorithm 1. Nelder-Mead method updates the searching point $x_w(t)$ with reflection, expansion and contraction procedure, where $w$ is the index whose searching point obtains the worst fitness at $t$. In other words, $x_w(t)$ is always improved in these procedures. With shrinking procedure, all searching points except the best one, $x_b(t)$, are updated. Therefore, Nelder-Mead method updates the searching points with almost improving $F_i(t)$, then, the diversity loss could be taken place. In this paper, the system parameter $\alpha = 1.0, \beta = 0.5, \gamma = 2.0$ [1] are used.

2.2 Particle swarm optimization (PSO)

The original PSO have been proposed by Kennedy and Eberhart [2, 3], and it has been modified by many researches. One of the well-known PSOs is proposed in [25], which contains an inertia weight and consists of a star neighborhood topology for information sharing (the global best model). In this paper, let the modified PSO [25] be PSO. PSO searches the optimal solution by updating $x_i(t)$ and independent $d$-dimensional vectors

$$ v_i(t) = \{v_{i1}(t), v_{i2}(t), \ldots, v_{ij}(t), \ldots, v_{id}(t)\}. \quad (3) $$
Algorithm 1: Nelder-Mead method for a maximizing dynamic problem $f$

1: **procedure 1: INITIALIZATION**
2: \hspace{1em} $t = 0, n = 0$
3: \hspace{1em} $x_i(t) \in \mathbb{R}^d, \ i = \{1, 2, \ldots, (d + 1)\}$
4: **for** $i = 1$ to $(d + 1)$ **do**
5: \hspace{2em} $F_i(t) = f(x_i(t), n)$
6: \hspace{2em} $n = n + 1$
7: \hspace{2em} $w = \arg\min_i F_i(t) \quad \triangleright \text{the worst}$
8: \hspace{2em} $b = \arg\max_i F_i(t) \quad \triangleright \text{the best}$
9: \hspace{2em} $\mathbf{x}(t) = \frac{1}{d} \sum_{i=1, i \neq w}^{d+1} x_i(t)$

10: **procedure 2: REFLECTION**
11: \hspace{2em} $x^*(t) = \mathbf{x}(t) + \alpha(\mathbf{x}(t) - w(t))$
12: \hspace{2em} $F^*(t) = f(x^*(t), n)$
13: \hspace{2em} $n = n + 1$
14: **if** $F^*(t) > F_i(t)$ **then**
15: \hspace{3.5em} Go to Expansion (procedure 3)
16: **else if** $F^*(t) < F_w(t)$ **then**
17: \hspace{3.5em} Go to Contraction (procedure 4)
18: **else**
19: \hspace{3.5em} $x_w(t+1) = x^*(t)$
20: \hspace{3.5em} $F_w(t+1) = F^*(t)$
21: \hspace{3.5em} $x_i(t+1) = x_i(t), \ i \neq w$
22: \hspace{3.5em} $F_i(t+1) = F_i(t), \ i \neq w$
23: \hspace{3.5em} Go to procedure 6

24: **procedure 3: EXPANSION**
25: \hspace{2em} $x^{**}(t) = \mathbf{x}(t) + \gamma(x^*(t) - \mathbf{x}(t))$
26: \hspace{2em} $F^{**}(t) = f(x^{**}(t), n)$
27: \hspace{2em} $n = n + 1$
28: **if** $F^{**}(t) > F_b(t)$ **then**
29: \hspace{3.5em} $x_w(t+1) = x^{**}(t)$
30: \hspace{3.5em} $F_w(t+1) = F^{**}(t)$
31: **else**
32: \hspace{3.5em} $x_w(t+1) = x^*(t)$
33: \hspace{3.5em} $F_w(t+1) = F^*(t)$
34: \hspace{3.5em} $x_i(t+1) = x_i(t), \ i \neq w$
35: \hspace{3.5em} $F_i(t+1) = F_i(t), \ i \neq w$
36: \hspace{2.5em} Go to procedure 6

37: **procedure 4: CONTRACTION**
38: \hspace{2em} $x^{**}(t) = \mathbf{x}(t) + \beta(x_w(t) - \mathbf{x}(t))$
39: \hspace{2em} $F^{**}(t) = f(x^{**}(t), n)$
40: \hspace{2em} $n = n + 1$
41: **if** $F_w(t) < F^{**}(t)$ **then**
42: \hspace{3.5em} Go to Shrinking (procedure 5)
43: **else**
44: \hspace{3.5em} $x_w(t+1) = x^{**}(t)$
45: \hspace{3.5em} $F_w(t+1) = F^{**}(t)$
46: \hspace{3.5em} $x_i(t+1) = x_i(t), \ i \neq w$
47: \hspace{3.5em} $F_i(t+1) = F_i(t), \ i \neq w$
48: \hspace{3.5em} Go to procedure 6

49: **procedure 5: SHRINKING**
50: \hspace{2em} $x_i(t+1) = \frac{1}{2}(x_i(t) + x_b(t))$
51: **for** $i = 1$ to $(d + 1)$ **do**
52: \hspace{3.5em} $F_i(t+1) = f(x_i(t+1), n)$
53: \hspace{2em} $n = n + 1$
54: \hspace{2.5em} Go to procedure 6

55: **procedure 6: EVALUATION AND CHECK TERMINATION**
56: \hspace{2em} $w = \arg\min_i F_i(t+1)$
57: \hspace{2em} $b = \arg\max_i F_i(t+1)$
58: **if** $t + 1 = t_{\text{max}}$ **then**
59: \hspace{3.5em} terminate
60: \hspace{3.5em} $\triangleright \text{the obtained solution}$
61: **else**
62: \hspace{3em} $\mathbf{x}(t+1) = \frac{1}{d} \sum_{i=1, i \neq w}^{d+1} x_i(t+1)$
63: \hspace{2.5em} $t = t + 1$
64: \hspace{2.5em} return to procedure 2

In this paper, $\mathbf{v}_i(0)$ is given as $\{0, \ldots, 0\}$. Note that the number of $x_i(t)$ and $\mathbf{v}_i(t)$, $N$, can be selected independently of the problem dimensions unlike Nelder-Mead method. Let

$$ pb_i(t) = \{pb_{i1}(t), pb_{i2}(t), \ldots, pb_{ij}(t), \ldots, pb_{id}(t)\}, $$

be $x_i(t)$ that denotes the best fitness value in the $i$-th searching points history $\{F_i(0), F_i(1), \ldots, F_i(t)\}$, and let the fitness value of $pb_i(t)$ be $Fpb_i(t) \in \mathbb{R}$. Let

$$ gb(t) = \{gb_1(t), gb_2(t), \ldots, gb_j(t), \ldots, gb_d(t)\}, $$

be $pb_i(t)$ that denotes the best fitness value in $\{Fpb_1(t), Fpb_2(t), \ldots, Fpb_N(t)\}$, and let the fitness value of $gb(t)$ be $Fgb(t)$. $x_i(t)$ and $\mathbf{v}_i(t)$ are updated by the following dynamical system:
BBPSO has been proposed by Kennedy in [23]. In that paper, Kennedy studied behavior of an
2.3 Bare born PSO (BBPSO)

BBPSO has been proposed by Kennedy in [23]. In that paper, Kennedy studied behavior of an
one-dimensional PSO’s searching point, and reported that while \( pb(t) \) and \( gb(t) \) are constants, the
searching distribution can be approximated by Gaussian distribution with mean \( \mu = \frac{1}{2}(pb(t) + gb(t)) \)
and variance \( \sigma^2 = |pb(t) + gb(t)|. \) Based on the study, BBPSO updates the searching points by

\[
\begin{align*}
  y_{ij}(t+1) &= N(\mu, \sigma^2), \\
  x_{ij}(t+1) &= y_{ij}(t+1) + \frac{1}{2}(pb(t) + gb(t)),
\end{align*}
\]

where \( N(\mu, \sigma^2) \) is the Gaussian distribution. As shown in Eq. (9), the searching points are governed
by a simple stochastic system. Excepting the update rule, the searching procedures of BBPSO, such as
updating \( pb(t) \) and \( gb(t) \), are same as PSO (see Sec. 2.2 and Algorithm 2).
2.4 Atomic PSO (APSO)
As described in Sec. 2.2, the divergent loss problem can occur in PSO. APSO [13] is a modified PSO algorithm to maintain the diversity of searching points by adding the acceleration vector $a_i(t)$:

$$a_i(t) = \{a_{i1}(t), a_{i2}(t), \ldots, a_{iq}(t), \ldots, a_{id}(t)\},$$

(10)

into the velocity update rule of PSO (see Eq. (6a)). Therefore $v_{ij}(t)$ is updated by

$$v_{ij}(t + 1) = \omega v_{ij}(t) + c_1 r_{1ij}(p b_{ij}(t) - x_{ij}(t)) + c_2 r_{2ij}(g b_{ij}(t) - x_{ij}(t)) + a_{ij}(t).$$

(11)

$a_i(t)$ is calculated by

$$a_i(t) = \begin{cases} \sum_k q_i q_k r_{ik}(t)(x_{i}(t) - x_k(t)) & \text{for } p_{\text{min}} < r_{ik} < p_{\text{max}}, \\ 0 & \text{otherwise,} \end{cases}$$

(12)

where $k \in \{1, 2, \ldots, N\} \land k \neq i$; $q_i$ is a charge parameter for $i$-th searching point; $p_{\text{min}}$ and $p_{\text{max}} = (x_{\text{max}})^{1/3}$ are system parameters; and $r_{ik}(t) = ||x_i(t) - x_k(t)||$, where $|| \cdot ||$ is norm. In this paper, following [13], we use $p_{\text{min}} = 1, x_{\text{max}} = 100$, $q_i = 16$ for half the number of the searching points, and $q_i = 0$ for the others. The $i$-th searching point with $q_i \neq 0$ is called charged particle, and it repels other charged particles based on $a_i(t)$, therefore, it is expected to prevent the diversity loss. As well as BBPSO, the basic searching procedure of APSO is same as PSO except the dynamical system.

2.5 An optimizer based on piecewise-rotational chaotic system (OPRC)
OPRC has been proposed in our previous research [22]. OPRC also searches the optimal solution by updating searching points based on a dynamical system likewise PSO, however, the system is chaotic. In OPRC, $x_i(t)$ and $v_i(t)$ are updated by a piecewise-rotational chaotic system (PRC):

$$\left[ \begin{array}{c} y_{ij}(t+1) \\ v_{ij}(t+1) \end{array} \right] = \begin{cases} \left[ \begin{array}{c} 2 \text{sgn}(y_{ij}(t))T h_{ij}(t) - y_{ij}(t) \\ 0 \end{array} \right] & \text{for } (v_{ij}(t), y_{ij}(t)) \in \Pi_{ij}(t), \\ R \left[ \begin{array}{cc} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{array} \right] \left[ \begin{array}{c} y_{ij}(t) \\ v_{ij}(t) \end{array} \right] & \text{otherwise,} \end{cases}$$

(13a)

(13b)

where

$$y_{ij}(t) = x_{ij}(t) - \frac{1}{2}(p b_{ij}(t) + g b_{ij}(t)), \quad T h_{ij}(t) = \frac{1}{2}|p b_{ij}(t) - g b_{ij}(t)|, \quad \Pi_{ij}(t) = \{(v_{ij}(t), y_{ij}(t)) \mid |y_{ij}(t)| > T h_{ij}(t), \text{sgn}(v_{ij}(t)y_{ij}(t)) = -1\}, \quad \text{sgn}(a) = \begin{cases} 1 & \text{for } a > 0, \\ -1 & \text{otherwise,} \end{cases}$$

(14)

(15)

(16)

(17)

and $R$ and $\theta$ are system parameters. It is guaranteed that PRC exhibits chaos with $R > 1$ and $0 < \theta < \frac{\pi}{2}$ [22]. In this paper, $R = 1.4$ and $\theta = 50[\text{deg}]$ are used, by which OPRC obtains good performance in static optimization problems [22]. The typical behavior of $x_{ij}(t)$ and $v_{ij}(t)$ is discussed in Sec. 5. The Algorithm of OPRC equals to PSO (Algorithm 2) except the dynamical system.

3. Memory update within a tolerance
As described in Sec. 1, the outdated memory problem [13] should be solved to track the optimum. In this section, we propose memory update within a tolerance to solve the issue, and some related works are introduced.
3.1 Proposed method

Memory update within a tolerance is simple: we just consider a tolerance parameter, $\tau \in \mathbb{R}$, to update the memory. For PSO, BBPSO, APSO and OPRC, the proposed procedure is

$$\begin{bmatrix}
pb_i(t+1) \\
Fpb_i(t+1)
\end{bmatrix} = \begin{cases} 
\begin{bmatrix}
x_i(t+1) \\
F_i(t+1)
\end{bmatrix} & \text{for } F_i(t+1) \geq Fpb_i(t) - \tau, \\
pb_i(t) & \text{otherwise}
\end{cases}$$  \quad (18)

$$\begin{bmatrix}
gb(t+1) \\
Fgb(t+1)
\end{bmatrix} = \begin{cases} 
\begin{bmatrix}
pb_g(t+1) \\
Fpb_g(t+1)
\end{bmatrix} & \text{for } Fpb_g(t+1) \geq Fgb(t) - \tau, \\
gb(t) & \text{otherwise}
\end{cases}$$  \quad (19)

When $\tau = 0$, Eqs. (18 and 19) equal to the conventional procedure Eqs. (7 and 8). For Nelder-Mead method, memory update within a tolerance is considered in the expression of IF statement as shown in Algorithm 3. In this paper, we use $\tau = 5$.

Algorithm 3 Nelder-Mead method for a maximizing dynamic problem $f$ using memory update within a tolerance. The skipped lines equal the original Nelder-Mead method (Algorithm 1).

1: procedure 1: Initialization
... 
10: procedure 2: Reflection
... 
14: if $F^*(t) > F_b(t) - \tau$ then
15: Go to Expansion (procedure 3)
16: else if $F^*(t) < F_w(t) + \tau$ then
17: : 
24: procedure 3: Expansion
... 
28: if $F^{**}(t) > F_b(t) - \tau$ then

37: procedure 4: Contraction
... 
38: if $F_w(t) < F^{**}(t) + \tau$ then
39: Go to Shrinking (procedure 5)
40: else
49: procedure 5: Shrinking
... 
55: procedure 6: Evaluation and check termination
...

Let us consider the behavior of a memory with proposed update. Figure 1(a) shows typical $p_{b1}(t)$ and its fitness $Fpb_i(t)$ for a 1-dimensional maximizing dynamic problem. In the figure, the green and blue ranges show the updatable ranges of the memory with conventional update and with proposed update, respectively. With conventional update, the memory can be updated only when $Fpb_i(t+1) \geq Fpb_i(t)$, therefore, the green range shrinks, then it is going to be hard for searching points to hit the range. Note that if the memory equals to the global optimum, the length of the green range is zero (see Figs. 1(a) and 1(b)). Once the function changes, such as shifting from Figs. 1(b) to 1(c), the searching points must hit the green range to track the shifted optima, but it is difficult as described above. On the other hand, the memory update within a tolerance keeps the width of updatable range even though the memory equals to the optima (see blue line in Fig. 1(b)). It is expected for searching points to hit the updatable range when the function changes as shown in Fig. 1(c). Therefore, considering only $\tau$, the memory can be updated and optimizers will track the optimum.

3.2 Related works

In conventional studies, some memory update procedures have been proposed to solve the outdated memory issue. Cui et al. have proposed a simple procedure which updates the memory when $F_i(t) \geq kFpb_i(t)$ (for maximizing problems), where $k$ is a constant [19, 20]. Using this procedure, optimizers
Fig. 1. The updatable range of the memory. The green and blue line shows the range using the conventional update ($\tau = 0$), and using the proposed update ($\tau \neq 0$), respectively. The green range becomes narrow when $p_i(t)$ is close to the optimum as shown in (b). The blue ranges are kept wider than conventional one, therefore, even though the function shifts, $p_i(t)$ can be updated and will track the optimum as shown in (c).

Fig. 1. The updatable range of the memory. The green and blue line shows the range using the conventional update ($\tau = 0$), and using the proposed update ($\tau \neq 0$), respectively. The green range becomes narrow when $p_i(t)$ is closed to the optimum as shown in (b). The blue ranges are kept wider than conventional one, therefore, even though the function shifts, $p_i(t)$ can be updated and will track the optimum as shown in (c).

can track the solution for the function $f : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$, where $\mathbb{R}_{\geq 0} = \{ z \in \mathbb{R} | z \geq 0 \}$ or $f : \mathbb{R}^d \to \mathbb{R}_{\leq 0}$, where $\mathbb{R}_{\leq 0} = \{ z \in \mathbb{R} | z \leq 0 \}$, however, it could fail to track for $f : \mathbb{R}^d \to \mathbb{R}$. For example, the $0 < k < 1$ solves the outdated memory issue for maximizing $f : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$; because $k F p_i(t)$ is reduced than the original $F p_i(t)$ and the decreased fitness memory makes wider the updatable range (see Fig. 1). In contrast, such $k$ value makes the issue worse for maximizing $f : \mathbb{R}^d \to \mathbb{R}_{\leq 0}$ since $F p_i(t) < k F p_i(t)$ (note that $F p_i(t) < 0$) makes smaller the updatable range than original one. This is critical issue of [19, 20] because the property of $f$ is unknown in the real-world problems.

On the other hand, our proposed method, memory update within a tolerance, can be applied for $f : \mathbb{R}^d \to \mathbb{R}$.

Online PSO [6] is a high accuracy method, which re-evaluates the memories for each time step $t$, however, it requires calculations for the fitness twice, such as $f(x_i(t), n)$ and $f(p_b(t), n)$. Our proposed procedure needs to calculate only $f(x_i(t), n)$.

$\varepsilon$PSO [21] worsen the stored fitness values when the memory is not updated, for maximizing problems, such as,

$$
\left[ \begin{array}{c}
 p_b(t+1) \\
 F p_b(t+1)
\end{array} \right] = 
\left\{ 
\begin{array}{l}
 \left[ \begin{array}{c}
 x_i(t+1) \\
 F_i(t+1)
\end{array} \right] \\
 p_b(t) \\
 F p_b(t) - \varepsilon
\end{array} \right. 
\right. 
\begin{array}{l}
 \text{for } F_i(t+1) \geq F p_b(t), \\
 \text{otherwise}
\end{array}
$$

where $\varepsilon$ is a positive constant. Comparing $\varepsilon$PSO, our memory update is simple because of no updating the stored fitness for each step.

4. Experiments

In this section, Nelder-Mead method, PSO, BBPSO, APSO and OPRC are compared using conventional and proposed procedures of memory update. The experiment conditions are described in Sec. 4.1. Section 4.2 shows tracking behavior, and its quantitative comparison is shown in Sec. 4.3.

4.1 Conditions

The tracking performance are measured by moving peaks benchmark (MPB) [11, 24], which is implemented in [30]. MPB is a maximizing problem with a multimodal function, which has a global optimum and some local optima. These optimal solutions are called peaks. In general, their position, width and height change depending on number of evaluation, $n$. However, in this paper, let the height be constant, because we consider tracking performance for continuously changing optimum (see Figs. 2(c), 3(e) and 3(f)). The fitness value of $x_i(t)$ is given by
Fig. 2. Shape of the generated function with \( d = 1 \) (a, b), and its history of peak’s positions (c). The red rectangle in (a, b) and line in (c) indicate the global optimum, and those of green color indicate local optima. As shown in (c), the changes of the optimum can be assumed as continuously.

Fig. 3. Shape of the generated function with \( d = 2 \) (a-d), and its history of peak’s positions (e, f). The red rectangle in (a-d) and line in (e, f) indicate the global optimum, and those of green color indicate local optima.

\[
f(x_i(t), n) = \max_p \left( H_p - W_p(n) \sum_{j=1}^{d} (x_{ij}(t) - x^*_p(n))^2 \right), \tag{20}
\]

where \( p \) is index number of the peaks; \( H_p \) and \( W_p(n) \) are height and width of \( p \)-th peak, respectively; and \( x^*_p(n) \in \mathbb{R}^d \) is the global optimum or local optima, which equals to position of the top. \( H_p \) is given randomly following uniform distribution with [30, 70]. \( W_p(n) \) and \( x^*_p(n) \) are updated depending on \( n \) by

\[
W_p(n) = \begin{cases} W_p(n - \Delta) + \mathcal{N}(0, 1) & \text{for } n \mod \Delta = 0, \\ W_p(n - 1) & \text{otherwise}, \end{cases} \tag{21}
\]

\[
x^*_p(n) = \begin{cases} x^*_p(n - \Delta) + V_p(n) & \text{for } n \mod \Delta = 0, \\ x^*_p(n - 1) & \text{otherwise}, \end{cases} \tag{22}
\]

\[
V_p(n) = \frac{l}{||r + V_p(n - \Delta)||} \left( (1 - \lambda) r + \lambda V_p(n - \Delta) \right), \tag{23}
\]
where $\Delta$ is a period of the function change; $\mathcal{N}(0, 1)$ is a random value following normal distribution with mean 0 and variance 1; $\lambda$ is a inertia coefficient; $l$ is distance of peak's shift; and $r = \{r_1, r_2, \ldots, r_d\}$ is a random vector where the element $r_j$ is given by uniform distribution with $[-0.5, 0.5]$. In short, every $\Delta$ evaluation times, the width $W_p(n)$ is changed, and the position of top $x^*_p(n)$ is shifted. In this paper, $\Delta = 100$, $l = 1.0$ and $\lambda = 0.5$ are used. $W_p(n)$ and $x^*_p(n)$ is clumped $[1.0, 12.0]$ and $[0, 100]$, respectively. We compare the algorithms with $d = 1, 2, 3, 4$, and the maximum number of the evaluation is 10,000. For a fair comparison, functions are generated in advance. Figures 2 and 3 shows the generated function with $d = 1$ and 2, respectively.

In this experiment, the initial searching points $x_i(0)$ are given by uniform distribution with range $[x^*_M(0) - 1, x^*_M(0) + 1]$, where $M$ is the peak's index of the global optimum (see initial point of the red line in Figs. 2(c), 3(e) and 3(f)). The number of searching points is 10 for PSO, BBPSO, APSO and OPRC, and $d + 1$ for Nelder-Mead method, respectively. Even though the number of searching points is not equal, the performance is measured under fair condition, because we consider the tracking behavior with $n$.

4.2 Tracking behavior

First, let us consider the tracking behavior using conventional procedure of memory update. Figure 4 shows the behavior for 1-dimensional (1-d) MPB. As show in these figures, all of the compared methods did not track the optima. This is because of the outdated memory issue. Even though algorithms are designed to prevent the divergent loss, such as APSO, they cannot track the optima without solving the outdated memory problem as shown in Fig. 4(d).

Figures 5 and 6 show the tracking behavior for same problem using memory update within a tolerance. Comparing with conventional one, the tracking performance was improved by all optimizers except BBPSO. For BBPSO, all of the searching points converged to the global optimum before $n = 100$ as shown in Fig. 6(c), therefore, even though using proposed memory update, BBPSO missed the changes of the optima at $n = 100$. In other words, the divergent loss strongly occurred with BBPSO. Nelder-Mead method was also influenced by the divergent loss. As shown in Fig. 6(a),

Fig. 4. Tracking behavior of 5 trials for 1-d MPB using conventional procedure of memory update. Red and green lines show the global optimum and local optima, respectively (See Fig. 2(c)). In each figure, there are 5 blue lines (almost overlapped), where a blue line shows a history of the best obtained solution by the optimizer, such as $gb_1(t)$ and $x_b(t)$. Black points show the searching points of the 5 trials (almost overlapped with blue lines, except APSO). All optimizers did not track the optima.
Tracking behavior of 5 trials for 1-d MPB using memory update within a tolerance ($\tau = 5$). Like Fig. 4, red, green and blue lines and black points show the global optimum, local optima, obtained best solutions, and searching points, respectively. The tracking performance was improved for all optimizers excepting BBPSO. Especially, OPRC seems to track the optimum with the best accuracy.

Nelder-Mead method tracked the optimum before $n = 700$, however, the tracking behavior exhibited delay after $n = 700$. The searching points could track the optimum until $n = 700$, however, after that, they were left behind the shifted optimum. This is because the searching points can move only small distance when they are closed (see Algorithm 1). PSO tracked the optima for all trials until $n = 1000$, however, in some trials, PSO missed the function’s changes for $n > 1000$ as shown in Fig. 5(b). APSO followed the solution during this experiment, however, sometimes it exhibited huge errors as shown in Fig. 5(d). For APSO, $gb_1(t + 1)$ would be updated to far away from $gb_1(t)$ by the memory update within a tolerance, because the charged particles are spread in wide range. Compared with the others, OPRC seems to track the optima with the best accuracy as shown in Fig. 5(e).

Let us focus on the tracking behavior for 2-d and 3-d MPB by PSO, APSO and OPRC using the memory update within tolerance ($\tau = 5$), that obtained relatively better results for 1-d MPB, and also Nelder-Mead using the conventional memory update ($\tau = 0$). Figures 7 and 8 shows the 30 trials of tracking for 2-d and 3-d, respectively. Increasing $n$, PSO missed the optimum more and more, then, most trials failed to track at $n = 10000$. APSO obtained better results than PSO, however, the tracking behavior was also unstable like the 1-d result. In addition, it lost the optima.
Fig. 7. Tracking behavior of 30 trials for 2-d MPB, focusing on around the global optimum. Red-dashed and blue line shows the global optimum (see Fig. 3) and obtained best solutions (overlapped 30 trials), respectively. The memory update within a tolerance ($\tau = 5$) was applied to PSO, APSO, and OPRC; and the conventional memory update ($\tau = 0$) was applied to Nelder-Mead. PSO lost the optima in most trials. APSO could follow the optimum, however, the behavior was unstable. Nelder-Mead and OPRC succeeded the tracking in most trials.

for $6000 < n < 8000$ for 3-d MPB in most trials. Nelder-Mead using conventional memory update succeeded the tracking for 2-d MPB (see Figs. 7(a) and 7(b)), which is an interesting result because the method failed the tracking for 1-d MPB as shown in Fig. 4(a). One of the possible causes is the shrinking procedure which is described in Algorithm 1, where the memorized fitness values $F_i(t)$ of the searching points $x_i(t)$ are forcibly updated to the new fitness values at the current number of function evaluation, $n$. In other words, by the shrinking procedure, Nelder-Mead can solve the outdated memory issue even using the conventional memory update. However, remember that the divergent loss issue can be taken place with Nelder-Mead as described before, and note that the Nelder-Mead failed for 3-d MPB (see Figs. 8(a)–8(c)). On the other hand, OPRC tracked the optima in most trials even for 2-d and 3-d problems.

### 4.3 Quantitative comparison
Here, we quantitatively consider the tracking performance. There are many performance measurements for dynamic problems [8]. In this paper, we use two simple measurements of them, called offline errors, that are based on fitness values $F_{err}(n)$, and based on the distance from the obtained best solution to the global optimum $D_{err}(n)$:

$$F_{err}(n) = \frac{1}{30} \sum_{m=1}^{30} \{ f(x^*_M(n), n) - f(x_{best}(m, n), n) \},$$  \hspace{1cm} (24)

$$D_{err}(n) = \frac{1}{30} \sum_{m=1}^{30} \{|x^*_M(n) - x_{best}(m, n)|\},$$  \hspace{1cm} (25)

where $m$ is the index of the trials, and $x_{best}(m, n)$ is the obtained best solution at $n$ of $m$-th trial. For each $n$, $x_{best}(m, n)$ was recorded by $x_b(t)$ and $gb(t)$ with Nelder-Mead method and others, respectively.

Mean, standard deviation, median, minimum and maximum values of $F_{err}(n)$ are shown in Table I, and those of $D_{err}(n)$ are shown in Table II. We can see these statistics denote the tracking performance well. For example, the median values obtained by APSO were relatively small, however, the maximum and standard deviation of the errors were large. This result qualitatively equals to
Fig. 8. Tracking behavior of 30 trials for 3-d MPB. The colored lines and the compared algorithms are all in common with those of Fig. 7. PSO failed to track the optimum (see (d)–(f)); the tracking behavior of Nelder-Mead and APSO were unstable (see (a)–(c) and (g)–(i)); in contrast, OPRC still succeeded the tracking in most trials and obtained better accuracy than others.

the tracking behavior as described above (see Figs. 5–8). Nelder-Mead using the conventional procedure of memory update and BBPSO using the proposed procedure obtained the minimal and the second minimal errors, respectively, because their searching points quickly converge before the first change (see Figs. 4(a) and 6(c)). In other words, the minimum value of the statistics is not important to consider the tracking behavior. OPRC using the proposed memory update obtained the best values except the minimum for \( d = 1, 3, 4 \). For \( d = 2 \), Nelder-Mead using the conventional memory update denoted the best statistics, and OPRC using the proposed memory update was second best. Although the results obtained by OPRC were slightly worse than those of the Nelder-Mead, both of them obtained remarkable small errors than others (see Fig. 7).
Table I. Statistic values of $F_{err}(n)$. The gray rectangles and '*' markers show the smallest (best) and the second smallest statistics, respectively. OPRC using the proposed memory update obtained the best values except the minimum for $d = 1, 3, 4$. For $d = 2$, both of the OPRC and Nelder-Mead using conventional memory update obtained remarkable small errors than others, where the Nelder-Mead results were slightly better than the OPRC. For all dimensions, the OPRC obtained the best or the second best results except the minimum.

| dimension | Using conventional memory update ($\tau = 0$) | Using memory update within a tolerance ($\tau = 5$) |
|-----------|---------------------------------------------|-----------------------------------------------|
|           | Nelder-Mead | PSO | BBPSO | APSO | OPRC | Nelder-Mead | PSO | BBPSO | APSO | OPRC |
| mean      | 11.99       | 11.19 | 11.21 | 11.19 | 11.11 | * 5.912   | 5.851 | 5.961 | 5.904 | 5.890 |
| std       | 5.883       | 5.050 | 5.964 | 5.945 | 5.904 | 4.005    | 3.264 | 5.963 | 2977  | 0.318 |
| med       | 12.81       | 12.90 | 13.00 | 12.90 | 12.87 | 2.297    | 6.435 | 13.00 | 39.64 | 8.037 |
| max       | 20.83       | 20.97 | 21.00 | 20.97 | 20.87 | 13.71    | * 11.57 | 21.00 | 1421  | 35.31 |
| min       | 0.3448      | 0.191 | 0.193 | 0.193 | 0.192 | * 0.191  | 0.191 | 0.191 | 0.191 | 0.191 |

Table II. Statistic values of $D_{err}(n)$. The cell color and marker are all in common with those of Table I. The obtained statistics indicate a similar tendency as that of $F_{err}(n)$, and OPRC using the proposed memory update obtained the best or the second best results for all conditions.

| dimension | Using conventional memory update ($\tau = 0$) | Using memory update within a tolerance ($\tau = 5$) |
|-----------|---------------------------------------------|-----------------------------------------------|
|           | Nelder-Mead | PSO | BBPSO | APSO | OPRC | Nelder-Mead | PSO | BBPSO | APSO | OPRC |
| mean      | 11.93       | 8.191 | 8.185 | 8.203 | 8.139 | * 5.841   | 5.812 | 5.841 | 5.812 | 5.841 |
| std       | 9.158       | 3.884 | 3.888 | 3.884 | 3.884 | 4.895    | 3.771 | 3.869 | 6.090 | 1.080 |
| med       | 14.69       | 8.002 | 7.993 | 8.009 | 7.963 | 10.42    | * 7.715 | 7.982 | 8.325 | 1.409 |
| max       | 25.80       | 15.37 | 15.37 | 15.39 | 15.30 | 18.22    | * 13.73 | 15.29 | 350.7 | 4.523 |
| min       | 0.1606      | 0.0160 | 0.0160 | 0.0160 | 0.0160 | 0.0160   | 0.0160 | 0.0160 | 0.0160 | 0.0160 |

The OPRC obtained the best or the second best results except the minimum for all dimensions. Therefore, we have qualitatively confirmed that OPRC obtained the best tracking performance.
5. Consideration

In this section, we consider the tracking mechanism of OPRC. In general, the divergent loss occurs and optimizers fail the tracking, even though using the memory update within a tolerance as shown in above results. One of the solution for the divergent loss is modification of searching dynamics, such as APSO. On the other hand, OPRC obtained the better performance than APSO even though the searching dynamical system was not modified. We consider this reason focusing on PSO, BBPSO and OPRC. Because, in spite of these methods are designed based on the same concept, searching the optima using $p_{b_i}(t)$ and $g_{b_j}(t)$, the tracking performances were remarkably different.

Let us consider the mechanism by following steps. First, the distribution of a searching point is discussed in Sec. 5.1 with an assumption that the searching points are converged around the typical memories. Second, in Sec. 5.2, we compare the searching behavior immediately following the memories are updated. These two discussions are given under the simplified and pseudo situations. To confirm them, we focus on the real tracking behavior for MPB again in Sec. 5.3.

5.1 Searching distribution with static memories and converged state

Here, we consider a searching point in 1-d space with $p_{b_i}(t)$ and $g_{b_j}(t)$ are constant: $p_{b_i}$ and $g_{b_j}$. Note that, $x_{i,j}(t)$ is updated independently of other dimensions and other searching points, therefore, we can discuss the searching behavior without loss of generality. Let $p_{b_i}$ and $g_{b_j}$ be 1 and $-1$, respectively, and let $p_{b_i} < x_{i,j}(0) < g_{b_j}$. This condition assumes that these memories have not been updated for a while, and a searching point has almost converged to specific range around the memories.

Figure 9 shows time-series of $x_{i,j}(t)$, where $x_{i,j}(0)$ was given randomly within the range following uniformed distribution, and $v_{i,j}(0) = 0$ was given for PSO and OPRC. As shown in the figure, the searching points oscillated and did not converge to a specific point. For PSO and OPRC, the oscillating behavior were also observed in the phase space as shown in Fig. 10. Figure 11 shows the histogram of $x_{i,j}(t)$ given by $10^6$ iterations. BBPSO and OPRC denoted relatively similar distribution, and PSO

![Fig. 9](image1.png)

**Fig. 9.** Time-series of a searching point with static memories, $p_{b_i} = 1$ and $g_{b_j} = -1$. The initial value was given by $p_{b_i} < x_{i,j}(0) < g_{b_j}$ for all methods, and $v_{i,j}(0) = 0$ for PSO and OPRC.

![Fig. 10](image2.png)

**Fig. 10.** Typical behavior on $x_{i,j}(t)$-$v_{i,j}(t)$.

![Fig. 11](image3.png)

**Fig. 11.** Histogram of $x_{i,j}(t)$ until $t = 10^6$. 

obtained the long-tail distribution than others, however, the tracking performance of these 3 methods had remarkably differed as shown in Sec. 4. Therefore, the superior tracking performance of OPRC would not be given by the searching distribution in such converged state.

5.2 Transient searching behavior at memory changes
Here, we focus on searching behavior immediately following the memories change. Comparing with above converged state, let us call this behavior as transient behavior.

Figures 12 and 13 show $x_{ij}(t)$ updated by PSO and OPRC, respectively. In these figures, $x_{ij}(0) = 100, v_{ij}(0) = 0$ is given. The condition $|x_{ij}(0) - (gb_j + pb_{ij})/2| \gg 0$ can simulate the situation just when $pb_{ij}$ and $gb_j$ are updated and become closed. Note that the searching points of BBPSO does not exhibit such transient behavior; that is $x_{ij}(t)$ immediately follows the memory because its update rule is not based on a dynamical system. The searching points of PSO converge to a specific range, and the speed of convergence seems independent of the distance between $gb_j$ and $pb_{ij}$ as shown in Fig. 12. On the other hand, the searching point of OPRC exhibited damped vibrational motion as shown in Fig. 13. This is because the searching point is continuously folded by Eq. (13a) when $|y_{ij}(t)| > |Th_{ij}(t)|$. Also, the converging speed depended on the memory distance, because $|y_{ij}(t + 1)| - |y_{ij}(t)| \propto |Th_{ij}(t)|$ during the vibration. Note that $y_{ij}(t) = x_{ij}(t)$ when $gb_j(t) + pb_{ij}(t) = 0$ (see Eq. (14)), and $|Th_{ij}| = |gb_j| = |pb_{ij}|$ (see Eq. (15)) in Fig. 13. Therefore, the converging speed differs among PSO, BBPSO and OPRC when $gb_j$ and $pb_{ij}$ are suddenly closed. Here, we consider the behavior for $gb_j + pb_{ij} = 0$, however, this transient behavior has generality even though $gb_j + pb_{ij} \neq 0$. Also, after the transient behavior, searching points exhibits the searching motion as described in above section.

5.3 Single trial of tracking behavior for MPB
Let us focus on the real tracking behavior given in the experiment Sec. 4. Figure 14 shows typical trials of the MPB tracking already shown in Fig. 5. The searching points of BBPSO quickly converged
Fig. 14. A typical tracking trial from Fig. 5. A point shows $x_{i1}(t)$, where $t = \lfloor n/10 \rfloor$, $\lfloor r \rfloor = \max\{m \in \mathbb{Z} \mid m \leq r \in \mathbb{R}\}$, where $r, m$ is real numbers and integers, respectively. $x_{i1}(t)$ is colored depending on $i$. Black line shows the global optimum $x^{*}_{M}(n)$. BBPSO and PSO missed the optima because all searching points converged to small range (divergent loss). For OPRC, some searching points also converged to small range likewise PSO, however, some points widely vibrated (g, h).

As shown in Fig. 14(b). Based on the above consideration, this quick convergence is caused by the non-transient behavior (Remember that the searching distributions of BBPSO and OPRC are almost equal for converged state.). Unlike BBPSO, PSO and OPRC did not exhibit such quick convergence, therefore, it seems that the transient behavior given by dynamical systems helps to avoid the divergent loss. However, the searching points of PSO slowly converged to a narrow range as shown in Fig. 14(d), then, they missed the changes of the optima at $n = 2500$ as shown in Fig. 14(e). On the other hand, OPRC kept the divergence of the searching points during the experiment, because some searching points exhibited the vibrational behavior as we have considered above. In Fig. 14(g), the typical vibrating searching points are marked with lines ($i = 5, 7$), and Fig. 15 shows them with the optimum, $gb_{1}(t)$ and $pb_{i1}(t)$. Let us take a look about Fig. 15(a), where $pb_{51}(t)$ and $gb_{1}(t)$ became closed at $n = 130$, then, $x_{51}(t)$ started widely vibrating. This vibrational motion caught the shifted optima at $n = 320$, then, it stayed close to the optima with chaotic motion. Such small perturbation can help to improve the accuracy of $gb_{1}(t)$. Other vibrational behavior was observed for $x_{71}(t)$ as shown in Fig. 15(b). It also started vibrating at $n = 350$, then, hit the shifted optima at $n = 520$. Focusing on both Figs. 15(a) and 15(b), we can see coexistence with the vibrating and chaotic searching points in same time-steps. Therefore, it is expected that OPRC autonomously succeeded to catch the shifted optima and to improve the accuracy for the stopped optima without modification of its searching
6. Conclusion
We have studied the tracking of optimal solutions in dynamic optimization problems. To achieve the tracking, both the outdated memory and the divergent loss issues had to be solved. To solve the outdated memory, we have proposed a simple procedure of memory update, called memory update within a tolerance. We have applied the proposed procedure to our previous proposed optimizer OPRC and 4 well-known methods, and have compared their tracking performance.

As the results, PSO and BBPSO failed to track the optimal solution in most trials due to the divergent loss issue. APSO, which is designed to avoid the divergent loss, could follow the optimal solution, however, the tracking behavior was rough. Nelder-Mead method showed capability to track the optimal solution especially for a 2-dimensional problem, however, its performance was limited for 1, 3 and 4-dimensional problems. Comparing with them, OPRC achieved the best tracking performance, even though its dynamical system of the searching points was not modified.

We have considered the tracking mechanism of OPRC, then, we have uncovered that the searching points caught the shifted optima by the transient behavior when the memories become suddenly closed. We described the transient behavior is given by the folding mechanism of its chaotic dynamical system. In other words, OPRC could solve the divergent loss problem autonomously, therefore, OPRC can search the stationary optimal solution (in static environment) and also can track the optimal solution when it moves (in dynamic environment). This feature is strong advantage for the real-world optimization problems with uncertainty environment.

In the future work, we will analyze the effects of the tolerance parameter and will provide its parameter setting guideline. Also, the study about OPRC for the complex dynamic problems whose optimal solution discontinuously changes, and application for the real-world dynamic problems are future topics.

Acknowledgments
This work was supported by JSPS KAKENHI Grant Number JP17H06552.

References
[1] J. Nelder and R. Mead, “A simplex method for function minimization,” Computer Journal, vol. 7, pp. 308–313, 1964.
[2] J. Kennedy and R. Eberhart, “Particle swarm optimization,” Proc. IEEE international conference on neural networks, vol. 4, pp. 1942–1948, November 1995.
[3] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” Proceedings of the International Symposium on Micro Machine and Human Science, pp. 39–43, October 1995.
[4] J.H.R. Enslin, M.S. Wolf, D.B. Snyman, and W. Swiegers, “Integrated photovoltaic maximum power point tracking converter,” *IEEE Transactions on Industrial Electronics*, vol. 44, no. 6, pp. 769–773, 1997.

[5] K. Tanakajima and T. Saito, “Application of particle swarm optimization to dynamic maximum power point tracking,” *Transactions of the Japanese Society for Artificial Intelligence*, vol. 7, no. 2, pp. 24–31, 2016.

[6] T. Nishida, T. Sakamoto, and N.I. Giannoccaro, “Self-tuning PI control using adaptive PSO of a web transport system with overlapping decentralized control,” *The transactions of the Institute of Electrical Engineers of Japan. D, A publication of Industry Applications Society*, vol. 131, no. 12, pp. 1442–1450, 2011.

[7] T. Blackwell and J. Branke, “Multiswarms, exclusion, and anti-convergence in dynamic environments,” *IEEE Transactions on Evolutionary Computation*, vol. 10, pp. 459–472, August 2006.

[8] T.T. Nguyen, S. Yang, and J. Branke, “Evolutionary dynamic optimization: A survey of the state of the art,” *Swarm and Evolutionary Computation*, vol. 6, pp. 1–24, 2012.

[9] C. Li, T.T. Nguyen, M. Yang, S. Yang, and S. Zeng, “Multi-population methods in unconstrained continuous dynamic environments: The challenges,” *Information Sciences*, vol. 296, pp. 95–118, 2015.

[10] H.G. Cobb and J.J. Grefenstette, “Genetic algorithms for tracking changing environments,” *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 1–8, 1993.

[11] J. Branke, “Memory enhanced evolutionary algorithms for changing optimization problems,” *Proc. IEEE CEC.*, pp. 1875–1882, 1999.

[12] R. Eberhart and Y. Shi, “Tracking and optimizing dynamic systems with particle swarms,” *Proceedings of the 2001 Congress on Evolutionary Computation*, vol. 1, pp. 94–100, 2001.

[13] T.M. Blackwell and P.J. Bentley, “Dynamic search with charged swarms,” *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO’02*, pp. 19–26, 2002.

[14] M. Kamosi, A.B. Hashemi, and M.R. Meybodi, “A hibernating multi-swarm optimization algorithm for dynamic environments,” *2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC)*, pp. 363–369, December 2010.

[15] S. Yang and C. Li, “A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments,” *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 959–974, 2010.

[16] U. Halder, S. Das, and D. Maity, “A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments,” *IEEE Transactions on Cybernetics*, vol. 43, no. 3, pp. 881–897, 2013.

[17] F. Glover and M. Laguna, *Tabu search*. Kluwer Academic Publishers, 1997.

[18] F. Charles and G. Fred, “Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory,” *INFORMS Journal on Computing*, vol. 11, no. 2, pp. 198–204, 1999.

[19] X. Cui, C. Hardin, R. Ragade, T. Potok, and A. Elmaghraby, “Tracking non-stationary optimal solution by particle swarm optimizer,” *Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN’05)*, vol. 2005, pp. 133–138, 2005.

[20] X. Cui and T.E. Potok, “Distributed Adaptive Particle Swarm Optimizer in Dynamic Environment,” *2007 IEEE International Parallel and Distributed Processing Symposium*, pp. 1–7, 2007.

[21] T. Nishida and T. Sakamoto, “Adaptive PSO for online identification of time-varying systems,” *Electronics and Communications in Japan*, vol. 95, no. 7, pp. 10–18, 2012.

[22] Y. Yamanaka and T. Tsubone, “Analysis of an optimizer based on piecewise-rotational chaotic system,” *IEICE Trans. Nonlinear Theory and Its Applications (NOLTA)*, vol. 7, pp. 557–575, October 2016.
[23] J. Kennedy, “Bare bones particle swarms,” Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706), pp. 80–87, 2003.

[24] M.C. du Plessis and A.P. Engelbrecht, “Self-adaptive differential evolution for dynamic environments with fluctuating numbers of optima,” Metaheuristics for Dynamic Optimization, pp. 117–145, 2013.

[25] Y. Shi and R. Eberhart, “A modified particle swarm optimizer,” Proc. IEEE CEC., pp. 69–73, May 1998.

[26] R. Eberhart and Y. Shi, “Comparing inertia weights and constriction factors in particle swarm optimization,” Proceedings of the 2000 Congress on Evolutionary Computation. CEC2000, vol. 1, pp. 84–88, July 2000.

[27] M. Clerc and J. Kennedy, “The particle swarm - explosion, stability, and convergence in a multidimensional complex space,” IEEE Transactions on Evolutionary Computation, vol. 6, no. 1, pp. 58–73, 2002.

[28] K. Yasuda, N. Iwasaki, G. Ueno, and E. Aiyoshi, “Particle swarm optimization: a numerical stability analysis and parameter adjustment based on swarm activity,” IEEJ Transactions on Electrical and Electronic Engineering, vol. 3, pp. 642–659, November 2008.

[29] C.W. Cleghorn and A.P. Engelbrecht, “A generalized theoretical deterministic particle swarm model,” Swarm Intelligence, vol. 8, pp. 35–59, January 2014.

[30] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, “DEAP: Evolutionary Algorithms Made Easy,” Journal of Machine Learning Research, vol. 13, pp. 2171–2175, July 2012.