Applied Soft Computing 47 (2016) 130–140

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Species-based Particle Swarm Optimizer enhanced by memory for dynamic optimization

Wenjian Luo[a,*], Juan Sun[a], Chenyang Bu[a], Houjun Liang[b]

[a] Anhui Province Key Laboratory of Software Engineering in Computing and Communication, School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China
[b] School of Computer Science and Technology, Anhui University of Finance and Economics, Bengbu, Anhui, China

ARTICLE INFO

Article history:
Received 22 October 2015
Received in revised form 1 May 2016
Accepted 23 May 2016
Available online 28 May 2016

Keywords:
Dynamic optimization
Particle swarm optimization
Species
Memory

ABSTRACT

Both the species strategy and the memory scheme are efficient methods for addressing dynamic optimization problems. However, the combination of these two efficient techniques has scarcely been studied. Thus, this paper focuses on how to hybridize these two methods. In this paper, a new swarm updating method is proposed to enhance a representative species-based algorithm, i.e., SPSO (Species-based Particle Swarm Optimization), and the new algorithm is named MSPSO. MSPSO has two characteristics. First, the number of replaced particles in the current swarm is set adaptively according to the number of species. To not substantially destroy the exploitation capability of each species, no more than one particle in each species is replaced by the memory. Second, the retrieved memory particles are categorized according to their fitness values and their distances to the seed of the closest species. Aimed at enhancing the search in both promising areas and existing species, each category is processed by different operations. The MPB, Cyclic MPB and DRPBG are used to test the performance of MSPSO. Experimental results demonstrate that MSPSO is competitive for dynamic optimization problems.

1. Introduction

Due to the dynamism of real-world optimization problems, dynamic optimization problems (DOPs) have received extensive interest in recent years [1,2]. Due to the changes in the landscape, to solve DOPs, the aim is not only to locate the optima in a specified environment, but also to track the movement of the optima over different environments. To achieve this goal, Evolutionary Algorithms (EAs) seem to be feasible candidates. Over the years, using EAs to cope with DOPs has attracted many researchers, and some strategies have been developed, e.g., diversity increasing schemes [3,4], diversity maintaining schemes [5,6], multi-swarm schemes [7], memory schemes [8], adaptive schemes [9], and predictive schemes [10].

Because Species-based Particle Swarm Optimizers are often regarded as multi-swarm (or multi-population) algorithms [2,11–14], we briefly review the typical Species-based Particle Swarm Optimizers together with multi-swarm (and multi-population) algorithms as follows. Many experimental results have shown that multi-swarm algorithms seem to be efficient for locating and tracking multiple optima in DOPs [15–17,11]. The main idea is to divide the entire search space into multiple sub-spaces and then exploit these sub-spaces separately. Indeed, in this manner, as mentioned in Ref. [1], multi-population algorithms could maintain good diversity for further exploitation whenever an environmental change is detected and could track the movements of multiple optima. However, one issue in using multi-swarm algorithms is how to create an appropriate number of sub-swarms, where different sub-swarms attempt to exploit different sub-spaces. According to recent studies, Species-based Particle Swarm Optimization (SPSO) [15,16] demonstrates good performance. At each step, the swarm in SPSO is re-partitioned, and multiple species are generated. The partition is based on the distribution of particles, and the number of species is evolved adaptively. Furthermore, in SPSO, to prevent particles from crowding in a few known areas, an allowed maximum size of the species works efficiently.

Memorizing old solutions from past environments for later environmental changes is an effective method for DOPs as well. Many works in the literature have proved the effectiveness of the memory scheme, especially in problems where the optima reappear. By retrieving the memory, particles may locate the optima directly or evolve with a bias toward some optimal areas. However, the memory scheme may lose its superiority in a drastically chang-

* Corresponding author. Fax: +86 55163601812.
E-mail addresses: wjluo@ustc.edu.cn (W. Luo), sunjuan@mail.ustc.edu.cn (J. Sun), bucy1991@mail.ustc.edu.cn (C. Bu), lhj@126.com (H. Liang).

http://dx.doi.org/10.1016/j.asoc.2016.05.032
1568-4946/

ing environment. In such cases, as mentioned in Ref. [1], memory schemes may be inefficient unless they are combined with various diversity strategies.

Based on the aforementioned description, combining the multi-swarm strategy and the memory scheme may utilize the advantages of both. However, combinations of the memory scheme and the multi-swarm strategy have scarcely been studied. In most cases, applying the traditional memory-based swarm updating method to the multi-swarm strategy does not yield ideal results. First, whenever an environmental change is detected, the number of replaced particles is difficult to determine. In existing works, this value is often predefined artificially. For example, a fixed number of ten is used in Ref. [18]. Second, different memory particles may play different roles in the new environment, and applying the same operation to them may be unreasonable. However, as for the classification and different operations of the retrieved memory, no literature attempts a further discussion.

In this paper, a new swarm updating method is proposed to solve the two aforementioned problems. First, the number of replaced particles is set adaptively according to the number of sub-swarms. To not substantially destroy the exploitation capability of each sub-swarm, no more than one particle is replaced in each sub-swarm. Second, the retrieved memory particles are classified into different categories, and each category is processed by different operations. The proposed swarm updating method is embedded into a representative multi-swarm algorithm, i.e., SPSO, and the new algorithm is named MSPSO. Experimental results demonstrate that MSPSO is competitive for dynamic optimization problems constructed by moving peaks, cyclic moving peaks and dynamic rotation peak benchmark generators.

The rest of this paper is organized as follows. Section 2 describes the related works on multi-swarm and memory approaches and their combinations. Section 3 introduces the particle swarm optimization and the Species-based Particle Swarm Optimizer in detail. Section 4 presents the proposed method. Benchmark problems, performance measurement, experimental settings, experimental results and analysis are given in Section 5. Finally, Section 6 briefly concludes this paper.

2. Related works

Currently, population-based algorithms, including Evolutionary Algorithms (EAs) and Particle Swarm Optimizations, are considered to be the most widely used approaches to solve DOPs, and the multi-swarm scheme and memory scheme are two efficient strategies for these algorithms. For DOPs, maintaining a high level of population diversity is a necessity. Many experimental studies have shown that the multi-swarm scheme is an efficient method to enhance the diversity. Moreover, many studies have proved that making use of the memory scheme to save historical solutions from past environments could be helpful for DOPs. In the following subsections, we will introduce some representative state-of-the-art algorithms using the multi-swarm scheme, the memory scheme and their hybrids.

For convenience, in the later text, we do not strictly distinguish multi-swarm from multi-population, as well as species.

2.1. Multi-swarm and species-based algorithms

As stated in Ref. [9], the maintenance of diversity is important for dynamic optimization problems. It aids in responding to environmental changes quickly and improves the chance of discovering the solutions. However, diversity loss is a common phenomenon in DOPs. The most widely used approach to cope with diversity loss is to use multiple swarms or divide the main swarm into multiple sub-swarms, with each swarm or sub-swarm exploiting a sub-space in the landscape.

Branke et al. put forward a method named Self-Organizing Scouts (SOS) [19], in which a parent population continuously explores for new peaks, and a number of child populations exploit promising previously detected areas. In the process of searching, if the condition to create a child population is satisfied, then a child population is created and splits off from the parent population. This general idea scatters particles on different peaks, which is helpful in the context of a dynamic environment.

Blackwell introduced an algorithm based on the Charged PSO for dynamic optimization [20]. Due to inter-particle repulsion, CPSO maintains population diversity and good tracking for optima; thus, it is well suited to dynamic optimization problems. On the basis of this idea, CPSO-based multi-swarm algorithms are studied in Ref. [20].

In Ref. [17], two multi-swarm models were proposed. One of these, multi-CPSO, is a multi-swarm version of CPSO. The main idea is to, by constructing interacting multi-swarms, extend the single swarm PSO and Charged Particle Swarm Optimization (CPSO) methods. The other, multi-Quantum Swarm Optimization (multi-QSO), uses quantum swarms, which broadens the atomic analogy in CPSO to a quantum model. These two algorithms start with a predefined number of swarms, and swarms exclude each other in the evolution. If one attractor is within r_{excl} of another attractor, then the swarm with the worse swarm attractor value is reinitialized. Compared with single-swarm algorithms on the MPB, the results obtained by these two models are encouraging.

In Ref. [11], a new variant of particle swarm optimization was proposed, i.e., mQSO. The population is composed of a number of swarms, and each swarm consists of both neutral and quantum particles. Instead of evolving according to some specified regulations as neutral particles, quantum particles, randomly distributed around the best particle of the swarm within radius r_{cloud}, provide the swarm with a certain level of diversity to track the moving optima. In addition to operation exclusion between swarms, another operator is introduced, i.e., anti-convergence, which reinitializes the worst swarm in the entire search space whenever all the swarms attain a state of convergence.

For the two algorithms mentioned above, the number of swarms remains unchanged in the process of searching, and this number must be predefined. Ideally, this value should be set as the number of peaks in the search space. However, in most of the cases, the number of peaks is difficult to obtain. To solve this problem, methods with a variable or adaptive number of swarms or sub-swarms are developed. In the following, multi-swarm methods of this type are introduced.

Parrott and Li presented a new classification technology based on the conception of species, i.e., SPSO [15] with spatially close particles assigned to a particular species or sub-swarm. Similar to Ref. [21], particles are similar to each other within the same sub-swarm and discrepant between sub-swarms. At each step, multiple seeds are identified by a list of all the particles sorted in descending order according to their fitness. Once a seed is selected, all the particles within the species radius of the seed are regarded as members of the same species. Particles may be assigned to different species in successive generations. The partition is based on the distribution of particles, and thus, the number of species changes adaptively.

An improved version of SPSO was developed two years later by the same authors and called DSPSO [16]. In DSPSO, a parameter P_{max} is introduced to prevent the convergence of particles on a few known peaks. If the number of particles in a species exceeds P_{max}, only the fittest P_{max} particles can be the members of the species, and redundant individuals with low fitness are re-initialized randomly over the entire search space. In this way, the population is

prevented from concentrating on a few regions and, as a result, spreads over the search space.

In terms of dividing the swarm into multiple groups, a Clustering Particle Swarm Optimization algorithm (CPSO) [22,23] has been proposed for DOPs as well in an attempt to cope with some difficulties of multi-swarm approaches. In the process of generating sub-swarms, CPSO implements a hierarchical clustering method on the basis of the Euclidean distance between particles. At each step, two clusters that have the closest distance among all the cluster pairs are picked to merge if the total number of their particles is not more than *subSize*. If no cluster pair meets the condition, then the clustering procedure exits. Without the influence of a key parameter as in SPSO, i.e., the species radius, the CPSO may be more robust. However, with respect to execution time, CPSO is inferior to SPSO because the procedure of partitioning the population is more time-consuming in CPSO than in SPSO.

2.2. Memory strategies

The memory strategy is to store all complete or partial good solutions and re-use them in the future. Many works have addressed such a strategy.

An associative memory scheme has been developed for PBILs for DOPs by Yang with some promising preliminary results [24]. This PBIL-specific explicit memory scheme was further investigated in Ref. [25]. Within this memory scheme, both the best sample, which is created by the working probability vector, and the probability vector are stored in the memory at a certain time. When an environmental change is detected, stored samples need to be re-evaluated. Then, the probability vector associated with the best sample in the new environment is retrieved to compete with the current working probability vector for further iterations. Results show that the memory scheme is effective for PBILs in dynamic environments.

A memory-based approach for addressing DOPs was proposed by Yang in Ref. [26], named MIGA, which hybridizes random immigrants and memory for GAs. Instead of depending on the detection of environmental changes, the memory is retrieved in each generation and hybridized with the random immigrant scheme. For each generation, the memory is re-evaluated, and immigrants are generated based on the best memory point via mutation. Then, the worst individuals in the population are replaced by the immigrants. Experiments were carried out on a series of systematically constructed dynamic problems, and the results showed that the hybrid approach improves the performance of genetic algorithms in dynamic environments.

Results obtained in Ref. [26] imply that a combination of memory and diversity strategies may have good performance. In Ref. [27], Branke studied some representative memory strategies and drew some conclusions: (1) if the optima appear frequently, memory approaches could track the moves of optima quickly; (2) good solutions can be retrieved if they are saved in the memory; and (3) if the optima change, even if it is a small change, the memory strategy may lose its advantage. To cope with this problem, the memory strategy would be combined with various diversity methods. According to the previous descriptions, combining the multi-swarm approach with memory may be feasible. In the following, some combinations of multi-swarm and memory schemes will be introduced.

2.3. Combinations of multi-swarm and memory schemes

Thus far, for DOPs, how to combine the memory schemes with the multi-swarm strategies has scarcely been studied. To our best knowledge, the works in Refs. [28] and [18] focused on the combinations of memory schemes and multi-swarm strategies for dynamic optimization.

In Ref. [28], a Cluster-based Dynamic Differential Evolution with external Archive (i.e., CDDE_Ar) for global optimization in dynamic fitness landscapes was proposed. By using the *k*-means algorithm [29], the entire population is partitioned into several clusters according to the spatial locations of the trial solutions. The clusters are evolved separately using a standard differential evolution algorithm. Every certain number of iterations, the population is re-partitioned, and the number of clusters is updated adaptively. Whenever a swarm is about to converge, its best solution is saved to the memory and the swarm is then reinitialized in the search space. These preserved solutions in the memory are added to the population whenever an environmental change is detected. Experiments are carried out on both MPB and GDBG benchmarks. Compared with several peer algorithms, CDDE_Ar obtains competitive results. Moreover, experimental results show that using the memory enhances the diversity of the population.

In Ref. [18], Zhu et al. proposed the "large memory" strategy. As stated in the paper, memory schemes may be inadequate if the memory size is small because multiple optima obtained by MPAs (Multi-Population Algorithms) could not be saved entirely. A large memory scheme (LM) is given to solve the problem, with more solutions from past environments being kept. The pseudo-code of the memory updating in Ref. [18] is shown in Algorithm 1. Based on the LM, two population updating models are presented, i.e., LM1 and LM2. In LM1, the best memory particles are retrieved to update the population whenever an environmental change is detected. In LM2, memory seeds are selected as replacers. LM2 takes both the fitness values and spatial distributions of memory particles into consideration. In both LM1 and LM2, the replaced particles are picked evenly from sub-swarms. That is, the worst particles in each sub-swarm, rather than the worst particles from the entire swarm, are selected and replaced by the memory. Experiments are carried out on both the MPB (Moving Peaks Benchmark) and CMPB (Cyclic MPB), and the results demonstrate LM's feasibility.

Algorithm 1. Updating the memory in LM.

```
1   Mark sub-swarms that have meet the requirement of convergence
2   while the number of converged sub-swarms is less than 5 and there
    are still some sub-swarms that have not been marked do
3     Mark each sub-swarm that has not been marked until the number of
    marked ones equals 5
4   end while
5   for each marked sub-swarm I do
6     if the memory is not full then
7       Put s into the memory, s is the seed of I, set the age of s to 0
8     else
9       Find the closest particle p in the memory to s
10      if distance (p, s) < Update-Distance then
11      Replace p with s, set the age of s to 0
12      else if uniform (0,1) < 2/3 then
13      Replace the eldest individual in the memory with s, set the age of s
    to 0
14      end if
15    end if
16  end for
```

3. PSO and SPSO

3.1. Particle Swarm Optimization

The initial particle swarm optimization algorithm was proposed by Eberhart and Kennedy [30]. Social interaction modeled by the PSO guides the particles moving toward the most promising area in the landscape. In PSO, each particle i is regarded as a candidate solution and represented by three parameter vectors at time t, i.e., the current location $\boldsymbol{x_i}(t)$, personal best location $\boldsymbol{p_i}(t)$ and velocity $\boldsymbol{v_i}(t)$. Each particle i updates its velocity according to the $\boldsymbol{p_i}(t)$ and $\boldsymbol{g}(t)$, where $\boldsymbol{g}(t)$ is the neighborhood best. During the process of searching, particles are not allowed to cross the boundary. If not,

particles should be pulled back to the search space. To constrain the maximum velocity in each dimension, a parameter called $V_{\max}$ is necessary. In most cases, the variation range of the velocity in each dimension is set to $[-V_{\max}, V_{\max}]$.

There are many variations of PSO, one of which is called the Constriction coefficient PSO (CoPSO) [31] and was proposed by Clerc and Kennedy. In the method, the velocity and position are updated according to the following formulas:

$$v_i(t+1) = \aleph(v_i(t) + \varphi_1(p_i(t) - x_i(t)) + \varphi_2(g(t) - x_i(t))) \quad (1)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (2)$$

where

$$\varphi_1 = c_1 r_1, \quad \varphi_2 = c_2 r_2, \quad c = c_1 + c_2,$$

$$\aleph = \frac{2}{|2 - c - \left(c^2 - 4c\right)^{1/2}|}$$

In Eq. (1), $\aleph$ is a constriction factor that produces a damping effect on the amplitude of a particle's velocity in each dimension. φ_1 and φ_2 are accelerating factors corresponding to the self-cognitive and social part, respectively. r_1 and r_2 are random numbers distributed uniformly in the interval [0,1]. Both c_1 and c_2 are accelerating constants, and $\aleph$ is calculated by them.

3.2. Species-based Particle Swarm Optimization

The Species-based PSO (SPSO) was developed by Parrot and Li [15], in which the entire swarm is partitioned into several species according to the fitness and spatial locations of the trial solutions. A parameter r_s is used to define the coverage area of a species, which denotes the species radius from the center to the boundary. Particles located in the coverage area of a species are grouped into this species. A seed is defined as the particle with the best fitness in a species. Regarding what fitness it refers to between the current fitness and personal best fitness, different algorithms have different settings. At each step, the swarm is re-partitioned, and multiple seeds are identified by a list of all the particles sorted in descending order according to their fitness. To prevent the convergence of particles at a few known optimal areas, a parameter $P_{\max}$ is introduced in Ref. [16]. If the number of particles in a species exceeds $P_{\max}$, only the fittest $P_{\max}$ particles are allowed to be the members of the species. Redundant particles with low fitness will be re-initialized randomly.

The framework of SPSO is shown in Algorithm 2, and the algorithm for dividing the population into species is shown in Algorithm 3. The set *Set_Seed* is initially set to null. All the particles are sorted in descending order according to their fitness. For each particle in the population, which species it belongs to is checked. Each particle is added to its corresponding species if the size of this species is less than $P_{\max}$. If a particle does not belong to any species, this particle is selected as a seed, and a new species is created.

Algorithm 2. The framework of the SPSO [16].

```
1  Initialize the swarm with randomly generated particles
2  Evaluate all the particles in the swarm
3  Divide the swarm into species (Algorithm 3)
4  Adjust particles' locations according to Eqs. (1) and (2)
5  Go back to step 2 unless the termination condition is met
```

Algorithm 3. Dividing the swarm into species in SPSO [16].

```
1   Set_Seed ⟵ Ø
2   Sort all the particles in descending order according to the fitness
3   for each particle p do
4     found ⟵ false
5     for each seed s in the Set_Seed do
6       if distance (p, s) < r_s then
7         found ⟵ true
8         if the size of the species that s belongs to is less than P_max then
9         Add p to the species that s belongs to
10        else
11        Re-initialize p
12        end if
13        break
14      end if
15    end for
16    if found = false then
17      Add p to Set_Seed, create a new species that p belongs to
18    end if
19  end for
```

4. The proposed method

4.1. Primary ideas

The primary advantage of multi-swarm algorithms is that they maintain good diversity, which is beneficial for tracking the trajectory of the optima. Memory schemes, by injecting previous solutions to the population, may guide particles to evolve toward favorable directions. By combining the memory scheme with the multi-swarm strategy, we expect to achieve two purposes: (1) the parameter that determines the number of replaced particles each time can be removed; (2) areas that are not covered by any species in the current swarm could be located by some small size sub-swarms that are triggered by the memory if there are historical records associated with those areas in the memory, while existing sub-swarms could be accelerated by the memory.

To achieve the first purpose, more memory particles should be retrieved when there are many peaks in the landscape, while this number should be reduced when there are few peaks. Meanwhile, to not substantially destroy the exploitation capability of each sub-swarm, the number of replaced particles in each sub-swarm should not be more than one. Therefore, we think that the number of replaced particles should be adjusted adaptively according to the number of sub-swarms in the swarm. Meanwhile, as stated in Ref. [18], the replaced particles should be selected evenly from each sub-swarm. Therefore, when an environmental change is detected, we think that the total number of replacements should be no greater than the number of sub-swarms, and the worst particle in each sub-swarm is replaced.

To achieve the second purpose, we classify the memory individuals into different types according to their fitness and distances to the seed in the closest sub-swarm. Only when the replacer is better than the replaced particle is the replacement allowed to be executed. If the replacer is far from the seed of its closest sub-swarm, extra particle(s) should be generated around the replacer. Specifically, if the replacer is outside any sub-swarms and better than its closest seed, a small-sized sub-swarm will be generated around the replacer because it could be in a promising area. Contrarily, if the replacer is near the seed of its closest sub-swarm, no extra particle is generated.

Based on the above ideas, a novel algorithm, named MSPSO, is proposed for DOPs. The details of MSPSO are described in the following subsections.

4.2. The framework of MSPSO

The primary contribution of MSPSO is a new swarm updating algorithm, which is used to update the current population with the memory. Before the new swarm updating method is introduced, we first describe the framework of MSPSO, as shown in Algorithm 4. MSPSO is based on the framework of SPSO in Ref. [16] and the large memory scheme in Ref. [18], and the new swarm updating method is embedded. If an environmental change is not detected, a generation of SPSO is executed, i.e., line 9. Otherwise, lines 5–7 should be executed. First, Algorithm 1 must be implemented to update the memory. After re-evaluating the population and the memory particles in line 6, the swarm is updated with the memory, i.e., Algorithm 5. The new swarm updating algorithm will be given in Subsection 4.3.

In Algorithm 1, regarding the identification of converged species, the method used in Ref. [32] is adopted. That is, if the personal best fitness of a seed could not be improved in several successive generations, then this species can be viewed as converged, and the seed should be marked as a particle to be saved. In this paper, no less than 5 seeds are saved to the memory each time. If the number of truly converged species is less than 5, then unmarked seeds are labeled until this number reaches 5.

It should be noted that, when performing Algorithm 1, there is a small change in our process in that we use a probability of 1/2 rather than 2/3 when replacing the eldest memory particle, i.e., lines 12–13 in Algorithm 1.

Algorithm 4. The framework of MSPSO.

```
1   Initialize the swarm
2   Memory ← Ø
3   while stop criteria is not satisfied do
4     if change is detected then
5       Update memory (conduct Algorithm 1)
6       Re-evaluate the population and memory
7       Update the swarm with memory (conduct Algorithm 5)
8     else
9         Execute a generation of SPSO
10    end if
11  end while
```

4.3. The swarm updating strategy

After, and the pseudo-code is describing the framework of MSPSO in Subsection 4.2, the new swarm updating method will be introduced in the following given in Algorithm 5.

For convenience, we denote the current swarm and the memory as P and M, respectively. *Set_Retrieved,* which denotes the set of replacers from the memory, is initially set to null. *Species* (i) denotes the species (sub-swarm) whose seed is numbered i. P is divided into species by Algorithm 3, and the set of seeds is denoted as *Set_Seed*. Particles in *Set_Seed* and M are sorted in descending order according to the fitness. For each memory particle *mp*, the distance to each seed in *Set_Seed* is calculated, and then, the closest distance *cdis* and the corresponding seed *cs* are recorded. Then, the memory particles are grouped into four categories, and different operations are employed for different types. A detailed description is provided as follows.

Case 1. If *mp* is better than *cs* and *cdis* is no less than the species radius r_s, a new species is established around *mp*. This new inserted species includes three particles. First, *mp* is added. Then, two randomly generated particles are generated around *mp*, and their distances to *mp* are set to a tenth of. r_s The replaced three particles are the worst particles from the three species (the worst one of each species is replaced). It is noted that the species are sorted in descending order by seeds.

The reasons for this case are given in the following. In a dynamic environment, keeping track of the moving peaks is essential. Because *mp* is better than the closest seed *cs*, the probability of locating the global optima by *mp* is larger than that for *cs*. Because exploiting a promising area by one particle is inadequate, the creation of extra particles is necessary. To not substantially destroy the exploitation capabilities of other species, only three particles are included in the new species.

Case 2. If *mp* is better than *cs*, and *cdis* is less than r_s but not less than $0.5 \times r_s$, *mp* is added to the current population, and only one extra particle is generated. This extra particle is randomly generated around *mp*, and its distance to *mp* is set to a tenth of. r_s Therefore, two replacements are conducted in this case.

Here, we *decrease* the number of extra particles by one compared to *Case* 1. This is because other particles in the corresponding species could move to *mp* in the near future.

Case 3. If *mp* is better than *cs* and *cdis* is less than $0.5 \times r_s$, only *mp* is added to the current population. Correspondingly, the number of replacements in this case *decreases* to one.

Because other particles in the species are close to the replacer, there is no need to create any extra particles. Moreover, with the insertion of the replacer, other particles in the same species may evolve toward a more favorable direction.

Case 4. If *mp* is not better than *cs* and *cdis* is less than, r_s a fitness comparison between the replacer and the particle to be replaced, *wp*, has to be executed. If *mp* is better than *wp*, *mp* replaces *wp*.

Because the replacer is not better than *cs*, the fitness check is necessary to not introduce a poor particle to the species.

Inspired by Refs. [33] and [34], the velocities of all the replacers and newly generated particles are initialized to zero.

Fig. 1 gives an example to illustrate our algorithm. Fig. 1(a) gives the state before updating. Fig. 1(b) shows the state after updating. Assume there are five peaks in the search space, numbered 1–5 in Fig. 1(b). However, only four peaks are detected by the current population, numbered 1–4 in Fig. 1(a). In the figure, $\oplus$ denotes a replacer from the memory, Δ denotes an extra particle generated around the replacer, and $\otimes$ denotes a replaced particle in the population. We can see that no more than one particle is replaced in each species.

Because $\oplus$ in *peak* 5 is better than the closest seed, i.e., the seed in species 2, and its distance to the closest seed is larger than r_s, species 5 is created to exploit this area.

In species 3, $\oplus$ is better than the closest seed, i.e., the seed in species 3, and the distance between it to the closest seed is larger than $0.5 \times r_s$ and less than r_s; thus, only one Δ is created around $\oplus$.

In species 2, $\oplus$ is better than the closest seed, i.e., the seed in species 2, but the distance between it to the closest seed is less than $0.5 \times r_s$, so only $\oplus$ is added to this species.

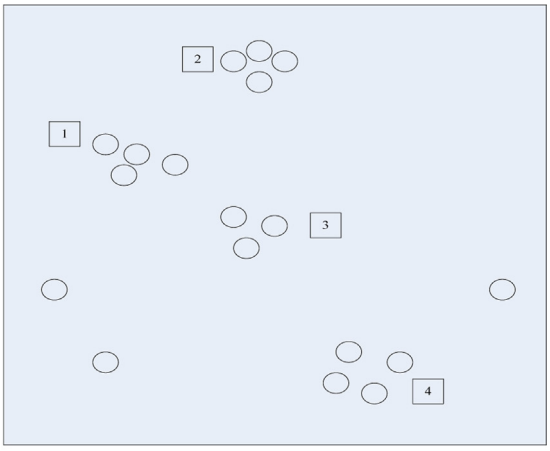
There is one case that has not been described in this example. Here, we suppose the replacer is not better than the worst particle in species 1, so no replacement is conducted in this species.

5. Experimental evaluation

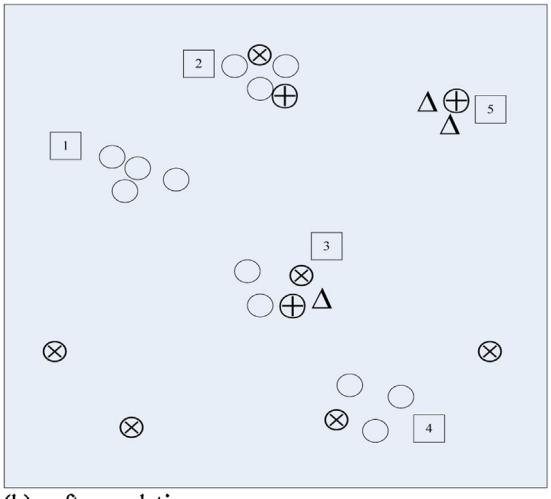
5.1. Benchmark problems

5.1.1. The Moving Peaks Benchmark

The dynamic benchmark problem called the "Moving Peaks" Benchmark (MPB) was proposed by Branke in Ref. [27] and has been one of the most widely used dynamic optimization test suites. It is a real-valued dynamic environment with N_p peaks in a D-dimensional search space, where the height, location and width



(a) before updating



(b) after updating

Fig. 1. Description of the states both before updating and after updating.

of each peak change at a certain frequency. The change of peak i is formally denoted as:

$$W_i(t) = W_i(t-1) + \text{width\_severity} \times \sigma \tag{3}$$

$$H_i(t) = H_i(t-1) + \text{height\_severity} \times \sigma \tag{4}$$

$$\boldsymbol{X}_i(t) = \boldsymbol{X}_i(t-1) + \boldsymbol{v}$$
$$\sigma \in N(0,1) \tag{5}$$

where $H_i(t)$, $W_i(t)$ and $X_i(t)$ denote the height, width and position of peak i at time t, respectively, and σ is a normal distributed random number with mean 0 and standard deviation 1. The location of each peak is moved by a vector v of fixed length s in a random direction. More details on the MPB can be found in Ref. [27].

Different landscapes can be constructed by defining the shape of peaks, and the conical peak is a typical one. The form of the conical function is given as follows:

Algorithm 5. The population-updating algorithm of MSPSO.

```
1   Execute Algorithm 3, sort seeds in Set_Seed and particles in M in
    descending order according to the fitness Set_Retrieved ⟵ Ø,
    changespecies ⟵ Set_Seed.size()-1
2   for each memory particle mp in M do
3     for each seed s in Set_Seed do
4       Calculate distance (mp, s)
5     end for
6     Find the closest distance cdis, the closest seed is named cs
7     if mp is better than cs and cdis is no less than r_s then
8       if changespecies ≥ 0 then
9         Set_Retrieved ⟵ Set_Retrieved ∪ mp, replace the worst particle in
    species(changespecies) with mp, changespecies ⟵ changespecies-1
10        for i=1 → 2 do
11          if changespecies ≥ 0 then
12            Generate a new particle newp around mp (its distance to mp
    is set to a tenth of r_s), replace the worst particle in
    species(changespecies) with newp, changespecies ⟵ changespecies-1
13          else
14            break
15          end if
16        end for
17      else
18        break
19      end if
20    else if mp is better than cs, cdis is less than r_s and not less than
    0.5 × r_s then
21      if changespecies ≥ 0 then
22        Set_Retrieved ⟵ Set_Retrieved ∪ mp, replace the worst particle in
    species(changespecies) with mp, changespecies ⟵ changespecies-1
23        if changespecies ≥ 0 then
24          Generate a new particle newp around mp (its distance to mp is
    set to a tenth of r_s), replace the worst particle in species(changespecies)
    with newp, changespecies ⟵ changespecies-1
25        else
26          break
27        end if
28      else
29        break
30      end if
31    else if mp is better than cs and cdis is less than 0.5 × r_s then
32      if changespecies ≥ 0 then
33        Set_Retrieved ⟵ Set_Retrieved ∪ mp, replace the worst particle in
    species(changespecies) with mp, changespecies ⟵ changespecies-1
34      else
35        break
36      end if
37    else if mp is not better than cs and cdis is less than r_s then
38      if changespecies ≥ 0 then
39        if mp is better than the worst particle wp in
    species(changespecies) then
40          Set_Retrieve ⟵ Set_Retrieved ∪ mp, replace wp with mp,
    changespecies ⟵ changespecies-1
41        end if
42      else
43        break
44      end if
45    end if
46  end for
47  Increase the ages of memory particles by one and set the ages of
    memory particles in Set_Retrieved to zero
```

$$\boldsymbol{F}(\mathbf{x}, t) = \max_{i=1\ldots N_p} \left(H_i(t) - W_i(t) \times \sqrt{\sum_{d=1}^{D} (x_d(t) - X_{id}(t))^2} \right) \tag{6}$$

where $X_{id}(t)$ denotes the d-th dimension of peak i at time t in a D-dimension space, and $x_d(t)$ denotes the d-th dimension of particle $\boldsymbol{x}$ at time t.

5.1.2. The Cyclic Moving Peaks Benchmark

The CMPB is a special case of the MPB, in which the location of each peak changes cyclically every l times. For each peak X_i, given the cyclic length l and the shifting severity s, a rotation matrix $\boldsymbol{R}$ and a corresponding vector $\boldsymbol{v}_0^i$ are generated. The height and width of each peak change in the same way as in the MPB, while the location is adjusted in the following manner:

$$\mathbf{X}_i(0) = \boldsymbol{v}_0^i \tag{7}$$

$$\mathbf{X}_i(t) = \boldsymbol{R} \times \mathbf{X}_i(t-1) = v_t^i, \quad t > 0 \tag{8}$$

Table 1
Basic parameter settings of the MPB and CMPB.

Parameters	Value
Change frequency	50,000
Number of peaks (N_p)	10
Number of changes	100
height_severity	7.0
width_severity	1.0
shift_severity(s)	1.0
Peak shape	Cone
Number of dimensions (D)	5
S	∈[0,100]
H	∈[30,70]
W	∈[1,12]
I	50

Suppose that $\boldsymbol{F}(x)$ is a test function in the real space, and that $\boldsymbol{O}$ is the optimum. In a cyclic changing environment, a transformed form of $\boldsymbol{F}(x)$ can be represented as follows:

$$\boldsymbol{F}^*(\boldsymbol{x}, t) = \boldsymbol{F}(\boldsymbol{x} - \boldsymbol{v}(t) + \boldsymbol{O}) \tag{9}$$

where

$$\boldsymbol{v}(t) = \boldsymbol{R}^t \times \boldsymbol{v}(0), \quad t > 0 \tag{10}$$

The generation of $\boldsymbol{R}$, $v(0)$ and some other details are explained in Refs. [18,35].

The default parameter settings used in the MPB and CMPB are listed in Table 1. The environment changes every 5000 fitness evaluations, and there are 100 changes in total. The changing severity of the height and width are set to 7.0 and 1.0, respectively. The default number of peaks and shifting severity are set to 10 and 1.0, respectively. Moreover, S, H, W, and I represent the variation range of the search space, variation range of the height, variation range of the width and initial height of each peak, respectively.

5.1.3. The GDBG system and the DRPBG

For the purpose of constructing dynamic problems for real, binary and combinational spaces, a generalized dynamic benchmark generator (GDBG) was proposed by Li et al. [36] and used for the CEC 2009 competition on dynamic optimization [37]. Later in the year 2012, it was updated to a new version and used as the benchmark generator for the IEEE WCCI-2012 competition on evolutionary computation for dynamic optimization problems [38].

There are six change types of the system control parameters in the earlier GDBG [36], i.e., small step (T_1), large step (T_2), random (T_3), chaotic (T_4), recurrent (T_5) and recurrent with noise (T_6). In the real space, a rotation method is introduced to the GDBG instead of shifting the positions of peaks as in the MPB. The dynamism of the GDBG is implemented by tuning the control parameters. It can be described as follows:

$$\emptyset(t+1) = \emptyset(t) \oplus \Delta\emptyset \tag{11}$$

where Ø is the system control parameter, and ΔØ is the deviation from the current control parameter which is determined by the specified change type. Thus, the new environment at time $t+1$ can be expressed as follows, and f is the cost function.

$$f(\boldsymbol{x}, \emptyset, t+1) = f(\boldsymbol{x}, \emptyset(t) \oplus \Delta\emptyset, t) \tag{12}$$

Two different real DBGs are constructed using different methods in the system: the dynamic rotation peak benchmark generator (DRPBG) and the dynamic composition benchmark generator (DCBG). In this experiment, the DRPBG is used.

Table 2
Basic parameter settings of the DRPBG.

Parameters	Value
Change frequency	50,000
Number of peaks (N_p)	10
Number of changes	60
height_severity	5.0
width_severity	0.5
noise_severity	0.8
Number of dimensions (D)	5
S	∈[−5,5]
H	∈[10,100]
W	∈[1,10]
I	50
P	12

The fitness function of the DRPBG (i.e., F_1 in the GDBG [38]) with D-dimension and N_P peaks is defined as:

$$\boldsymbol{F}(\mathbf{x}, t) = \max_{i=1\ldots N_p} \left(H_i(t) \Big/ \left(1 + W_i(t) \times \sqrt{\sum_{d=1}^{D} \frac{(x_d(t) - X_{id}(t))^2}{D}} \right) \right) \tag{13}$$

The basic parameter settings of the DRPBG in Ref. [38] are given in Table 2, where P denotes the change period in T_5 and T_6. Moreover, a new factor *change_ratio* is introduced in Ref. [38], which controls the changing ratio of peaks in the dynamism. We use the same settings as the original literature, and the parameters for all peaks, including the locations, the widths, and the heights, are all the same.

5.2. Performance measurement

To fairly compare the algorithms involved in this paper, as in many other papers, we use the *offline error,* which is named the *best error before change* in Ref. [1], as the performance measure. It is calculated as the average of the minimum fitness errors and is obtained by the algorithm at the end of each period just before a new environmental change is introduced. The form of the *offline error* is as follows:

$$e_{offline-error} = \frac{1}{T}\sum_{t=1}^{T}(optimum_t - fbest_t) \tag{14}$$

where T is the number of environments, $optimum_t$ is the optimum in the t-th environment in theory, and $fbest_t$ is the best solution obtained by the algorithm just before the t-th change occurs.

5.3. Experimental settings

In Eq. (1), the accelerating constants c_1 and c_2 are both set to 2.05. Parameter V_{max} is set to the size of the variation range of the search space. In Algorithm 1, *Update-Distance* is set to 0.8. In Algorithm 3, P_{max} is set to 10. The size of the swarm and the capacity of the memory are set to 100 and 500, respectively. The species radius is set to 30 for the MPB and CMPB. For the DRPBG, the species radius is set to 3.5 and 7.0 for the 5-D and 10-D space, respectively. The heights of the peaks in the MPB and CMPB change uniformly in the range [30,70] (different with formula (4)).

The experiment is carried out in four groups. To test the validity and feasibility of the viewpoints proposed in this paper, MSPSO is tested on the MPB and CMPB in the first two groups and compared with SPSO and/or its variants. In the third group, MSPSO is implemented on $T_1 - T_6$ of the DRPBG with the parameter settings in Table 2 and [37], and compared with other algorithms. The chang-

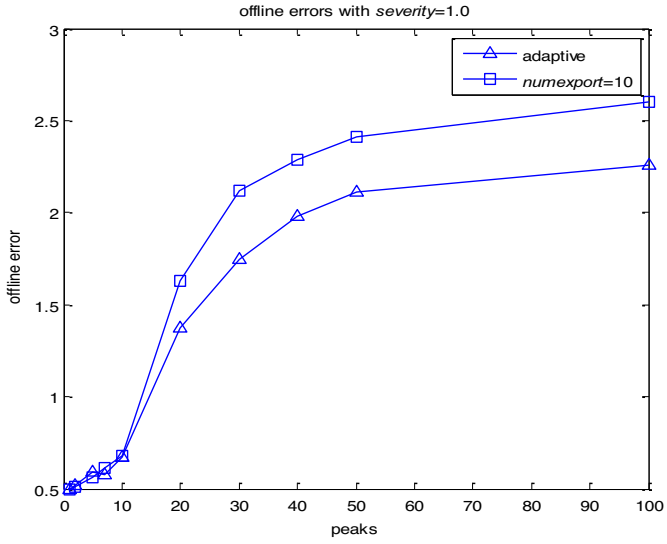


Fig. 2. Comparisons of the number of replaced particles on the MPB with different numbers of peaks.

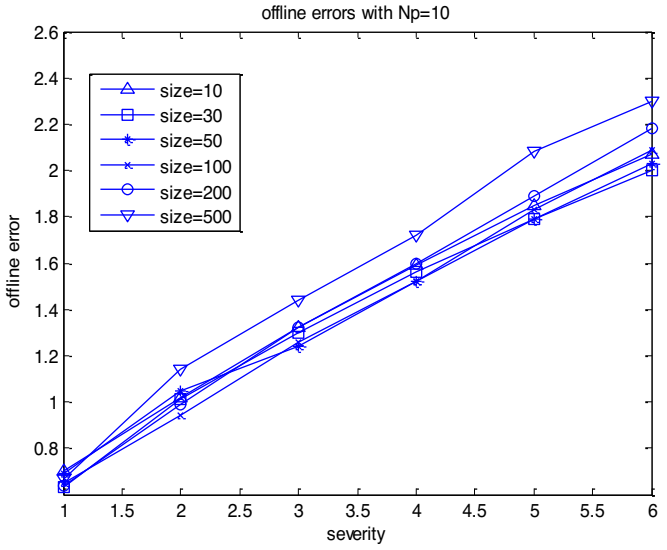


Fig. 3. Comparisons of different memory sizes on the MPB with different shifting severities.

ing ratio *change_ratio* is set to 1.0. In the last part, an influence test on the memory size is implemented.

5.4. Experimental results and analysis

5.4.1. The number of replaced particles

The LM2 algorithm in Ref. [15] is adopted for comparisons. The number of replaced particles in the LM2 algorithm is denoted by *numexport*, and this number is set to 10. In Fig. 2, the proposed algorithm in this paper is denoted as "adaptive," and the LM2 algorithm is denoted as "*numexport* = 10."

Fig. 2 shows that the proposed adaptive method performs much better when the number of peaks (N_p) is larger than 10. Furthermore, the difference between the two curves becomes larger with the increase of N_p. Thus, the proposed adaptive method not only leads to the number of replaced particles evolving adaptively, but also produces smaller offline errors.

5.4.2. Comparisons over the MPB and CMPB

In this subsection, we compare the performance of MSPSO with that of several other variants of SPSO on both the MPB and CMPB.

The results on the MPB are presented in Tables 3 and 4. The results show that MSPSO has the best performance in almost all the cases. The results also show that MSPSO performs much better than the other three algorithms with the increase of shifting severity (i.e., s) or the number of peaks (i.e., N_p). That is, MSPSO has a more stable performance than the compared algorithms with different numbers of peaks and different shifting severities.

The experimental results on the CMPB with different shifting severities are provided in Tables 5–8. In Tables 5–7, most of the best results are obtained by MSPSO. In Table 5 , MSPSO is slightly worse than LM1SPSO for the cases (s, l, n) = (1, 20, 10) and (1, 20, 20). Table 8 shows that, for the cases with a large shifting severity, MSPSO has the best performance for all the cases.

According to the results in Tables 3–8 above, to compare the performance differences between MSPSO and the other three algorithms, Wilcoxon signed ranks test is performed and the results are provided in Table 9. Table 9 shows that MSPSO has a significant improvement over SPSO, LM1SPSO and LM2SPSO on both the MPB and CMPB with a significance level $\alpha = 0.05$.

Overall, MSPSO is competitive for dynamic optimization problems on both the MPB and CMPB. Compared with SPSO and two variants of SPSO, MSPSO is significantly better than them in most of the cases. Moreover, the advantage of MSPSO is more noticeable with the increase of s and N_p.

5.4.3. Comparisons over the DRPBG

This experiment aims to study the performance of the proposed method in comparison with several other algorithms on the DRPBG. On the DRPBG with the parameter settings in Table 2, the compared algorithms include CPSOR [39] and MLSDO [40].

When the DRPBG is adopted with the parameter settings in Ref. [37], the involved algorithms include jDE [41], DASA [42], CPSO [22], dopt-aiNet [43] and EP [44], and the results of these algorithms are from the corresponding references.

Table 10 shows that MSPSO obtains better results than CPSOR and MLSDO for the cases T_1–T_3. For the cases T_4 and T_6, MSPSO is better than CPSOR and worse than MLSDO.

Table 11 provides the comparative results of our method MSPSO with five other algorithms on the DRPBG with settings in Ref. [37]. From the table, we can see that, in the 10-D search space with 10 peaks, MSPSO has better performance than other algorithms in the cases of T_1–T_3, T_5 and T_6. When the number of peaks (i.e., N_p) is set to 50, MSPSO has better performance than the others only in the cases of T_2 and T_6. Among the remainder of the cases, jDE or CPSO obtain the best results. Furthermore, the result of the MSPSO on the case of $N_p = 100$ is provided as well.

The results show that the proposed algorithm is suitable for addressing DOPs with not only shifting changes (i.e., the MPB) but also rotating changes (i.e., the DRPBG).

5.4.4. The memory size

The comparisons of different memory sizes on the MPB with different shifting severities (s) are shown in Fig. 3. The number of peaks (N_p) is set to 10. Fig. 3 shows that, when the memory size is set to 500, MSPSO obtains the largest offline errors (i.e., performs worst) in most of the cases, and differences among the other five curves are relatively small.

The comparisons of different memory sizes on the MPB with different numbers of peaks (N_p) are presented in Fig. 4. The shifting severity (s) is set to 1.0. Fig. 4 shows that, for a fixed N_p, the offline errors tend to decrease with the increase of the memory size when $N_P > 10$. That is, for those six values of the memory size, the algorithm has the worst performance when the memory size is set to 10

Table 3
Comparisons on the MPB with different numbers of peaks.

Algorithms	Number of peaks (N_p)									
	1	2	5	7	10	20	30	40	50	100
SPSO	0.50 ± 0.09	0.71 ± 1.05	0.67 ± 0.44	0.70 ± 0.34	1.07 ± 0.32	2.39 ± 0.35	2.81 ± 0.35	3.03 ± 0.38	3.26 ± 0.44	3.59 ± 0.41
LM1SPSO	0.54 ± 0.05	0.58 ± 0.05	0.58 ± 0.11	0.61 ± 0.13	0.70 ± 0.15	1.78 ± 0.14	2.23 ± 0.13	2.49 ± 0.15	2.61 ± 0.12	2.86 ± 0.17
LM2SPSO	0.57 ± 0.06	0.56 ± 0.10	0.60 ± 0.17	0.63 ± 0.17	0.66 ± 0.14	1.73 ± 0.12	2.25 ± 0.17	2.44 ± 0.17	2.55 ± 0.15	2.81 ± 0.12
MSPSO	0.49 ± 0.04	0.50 ± 0.03	0.55 ± 0.08	0.59 ± 0.11	0.70 ± 0.13	1.38 ± 0.13	1.81 ± 0.16	1.99 ± 0.14	2.07 ± 0.10	2.28 ± 0.10

Table 4
Comparisons on the MPB with different shifting severities.

Algorithms	Shifting severity (s)					
	1.0	2.0	3.0	4.0	5.0	6.0
SPSO	1.07 ± 0.32	1.45 ± 0.30	1.90 ± 0.34	2.38 ± 0.45	2.65 ± 0.38	3.09 ± 0.41
LM1SPSO	0.70 ± 0.15	1.23 ± 0.13	1.62 ± 0.18	2.05 ± 0.17	2.45 ± 0.19	2.90 ± 0.18
LM2SPSO	0.66 ± 0.14	1.19 ± 0.15	1.60 ± 0.16	1.99 ± 0.13	2.47 ± 0.18	2.86 ± 0.18
MSPSO	0.70 ± 0.13	1.07 ± 0.10	1.41 ± 0.11	1.72 ± 0.12	1.99 ± 0.13	2.28 ± 0.19

Table 5
Comparisons on the CMPB. The shifting severity (s) = 1.0. Parameter N_p is the number of peaks, and l is the change period.

Algorithms	(s, l, N_p)							
	(1,5,5)	(1,5,10)	(1,5,20)	(1,5,40)	(1,20,5)	(1,20,10)	(1,20,20)	(1,20,40)
SPSO	0.65 ± 0.30	0.91±0.36	1.85±0.33	2.64±0.33	0.73±0.37	0.90±0.26	2.11±0.33	2.88±0.46
LM1SPSO	0.08 ± 0.03	0.14±0.06	0.77±0.19	1.15±0.16	0.14±0.19	0.24 ± 0.14	0.99±0.06	1.56±0.07
LM2SPSO	0.07 ± 0.02	0.17 ± 0.10	0.81±0.18	1.37±0.19	0.14±0.13	0.31±0.09	1.07±0.14	1.63±0.17
MSPSO	0.06 ± 0.03	0.13 ± 0.05	0.54 ± 0.10	1.07 ± 0.13	0.10 ± 0.04	0.25 ± 0.09	0.99 ± 0.14	1.55 ± 0.19

Table 6
Comparisons on the CMPB. The shifting severity (s) = 2.0. Parameter N_p is the number of peaks, and l is the change period.

Algorithms	(s, l, N_p)							
	(2,5,5)	(2,5,10)	(2,5,20)	(2,5,40)	(2,20,5)	(2,20,10)	(2,20,20)	(2,20,40)
SPSO	0.94 ± 0.29	1.22±0.29	2.46 ± 0.33	3.23 ± 0.51	0.99±0.29	1.47±0.34	2.59±0.36	3.39±0.34
LM1SPSO	0.08 ± 0.01	0.19 ± 0.07	0.79±0.15	1.17 ± 0.13	0.20±0.14	0.42±0.12	1.13 ± 0.14	1.71 ± 0.17
LM2SPSO	0.09 ± 0.03	0.21±0.12	0.86±0.14	1.32±0.14	0.24±0.07	0.36 ± 0.19	1.22±0.11	1.79±0.18
MSPSO	0.07 ± 0.03	0.18 ± 0.10	0.67 ± 0.12	1.18 ± 0.13	0.15 ± 0.06	0.36 ± 0.10	1.15 ± 0.16	1.66 ± 0.13

Table 7
Comparisons on the CMPB. The shifting severity (s) = 5.0. Parameter N_p is the number of peaks, and l is the change period.

Algorithms	(s, l, N_p)							
	(5,5,5)	(5,5,10)	(5,5,20)	(5,5,40)	(5,20,5)	(5,20,10)	(5,20,20)	(5,20,40)
SPSO	2.08 ± 0.61	2.49 ± 0.36	3.75 ± 0.39	4.59±0.41	2.25±0.39	3.02±0.48	4.15 ± 0.38	4.51±0.37
LM1SPSO	0.10 ± 0.03	0.26 ± 0.08	0.65 ± 0.09	1.12±0.10	0.46 ± 0.09	0.84±0.08	1.51±0.29	2.05±0.15
LM2SPSO	0.14 ± 0.07	0.26±0.09	0.77±0.10	1.16±0.11	0.41±0.08	0.77 ± 0.09	1.49 ± 0.15	2.09±0.12
MSPSO	0.08 ± 0.04	0.20 ± 0.09	0.72 ± 0.12	1.10 ± 0.13	0.31 ± 0.19	0.64 ± 0.18	1.34 ± 0.13	1.84 ± 0.10

Table 8
Comparisons on the CMPB. The shifting severity (s) = 10.0. Parameter N_p is the number of peaks, and l is the change period.

Algorithms	(s, l, N_p)							
	(10,5,5)	(10,5,10)	(10,5,20)	(10,5,40)	(10,20,5)	(10,20,10)	(10,20,20)	(10,20,40)
SPSO	3.79 ± 0.59	4.71 ± 0.50	5.25±0.43	5.34±0.41	4.62±0.63	5.54±0.52	5.79 ± 0.40	5.37±0.41
LM1SPSO	0.15 ± 0.04	0.33±0.06	0.74±0.10	1.17 ± 0.10	0.86 ± 0.09	1.51±0.08	2.35±0.28	2.78 ± 0.24
LM2SPSO	0.15 ± 0.04	0.33±0.09	0.73±0.09	1.17±0.09	0.82±0.11	1.52±0.10	2.31±0.24	2.77±0.19
MSPSO	0.12 ± 0.08	0.28 ± 0.12	0.66 ± 0.10	1.03 ± 0.12	0.59 ± 0.26	1.20 ± 0.19	1.92 ± 0.19	2.27 ± 0.13

Table 9
The results of the Wilcoxon signed ranks test on the MPB and CMPB. MSPSO has a significant improvement over SPSO, LM1SPSO and LM2SPSO with a significance level $\alpha = 0.05$.

Comparison	MPB		CMPB	
	p-value	Detected difference	p-value	Detected difference
MSPSO-SPSO	0.000	$\alpha = 0.05$	0.000	$\alpha = 0.05$
MSPSO-LM1SPSO	0.001	$\alpha = 0.05$	0.000	$\alpha = 0.05$
MSPSO-LM2SPSO	0.001	$\alpha = 0.05$	0.000	$\alpha = 0.05$

Table 10
Comparisons on the DRPBG with the parameter settings in Table 2 and *change_ratio* = 1.0.

Algorithms	Change type					
	T_1	T_2	T_3	T_4	T_5	T_6
CPSOR	2.82 ± 4.15	1.13 ± 1.37	1.92 ± 0.60	0.09 ± 0.26	0.00 ± 0.01	0.25 ± 0.80
MLSDO	0.583 ± 1.315	1.129 ± 1.110	1.736 ± 0.623	0.003 ± 0.006	0.046 ± 0.025	0.013 ± 0.007
MSPSO	0.002 ± 0.002	0.447 ± 0.313	0.113 ± 0.026	0.033 ± 0.019	0.411 ± 0.074	0.131 ± 0.089

Table 11
Comparisons on the DRPBG with the parameter settings in Ref. [37]. Parameter N_p represents the number of peaks.

Dimension	N_p	Algorithms	Change type					
			T_1	T_2	T_3	T_4	T_5	T_6
10	10	jDE	0.03 ± 0.44	3.59 ± 7.84	3.00 ± 7.13	0.02 ± 0.29	2.18 ± 4.39	1.15 ± 5.73
		DASA	0.18 ± 1.25	4.18 ± 9.07	6.37 ± 10.70	0.48 ± 1.95	2.54 ± 4.80	2.34 ± 8.66
		CPSO	0.04 ± 0.43	2.72 ± 6.52	4.13 ± 8.99	0.09 ± 0.79	1.87 ± 4.49	1.06 ± 4.81
		dopt-aiNet	0.14 ± 1.01	5.87 ± 10.28	4.25 ± 8.18	5.36 ± 8.94	4.44 ± 5.55	9.94 ± 15.82
		EP	5.71 ± 9.68	10.66 ± 13.85	10.87 ± 13.50	1.50 ± 3.00	8.30 ± 13.23	8.23 ± 13.10
		MSPSO	0.00 ± 0.00	0.01 ± 0.05	0.03 ± 0.02	0.44 ± 0.13	0.23 ± 0.22	0.58 ± 0.31
10	50	jDE	0.17 ± 0.76	4.09 ± 6.45	4.29 ± 6.75	0.09 ± 0.25	0.95 ± 1.77	1.77 ± 5.83
		DASA	0.44 ± 1.39	4.86 ± 7.00	8.42 ± 9.56	0.51 ± 1.09	1.18 ± 2.18	2.07 ± 5.97
		CPSO	0.26 ± 0.94	3.28 ± 5.30	6.32 ± 7.44	0.13 ± 0.39	0.85 ± 1.78	1.48 ± 4.39
		dopt-aiNet	0.36 ± 0.93	4.75 ± 6.76	5.25 ± 6.68	2.66 ± 5.98	2.86 ± 4.16	6.83 ± 11.88
		EP	5.74 ± 6.84	13.29 ± 12.94	15.90 ± 13.37	1.41 ± 2.45	2.27 ± 4.24	3.16 ± 5.60
		MSPSO	2.56 ± 1.61	1.43 ± 0.32	5.83 ± 1.60	0.82 ± 0.14	2.52 ± 1.67	1.10 ± 0.44
	100	MSPSO	3.25 ± 2.08	1.59±0.60	4.65 ± 0.75	0.40 ± 0.15	3.81 ± 2.32	2.06 ± 0.63

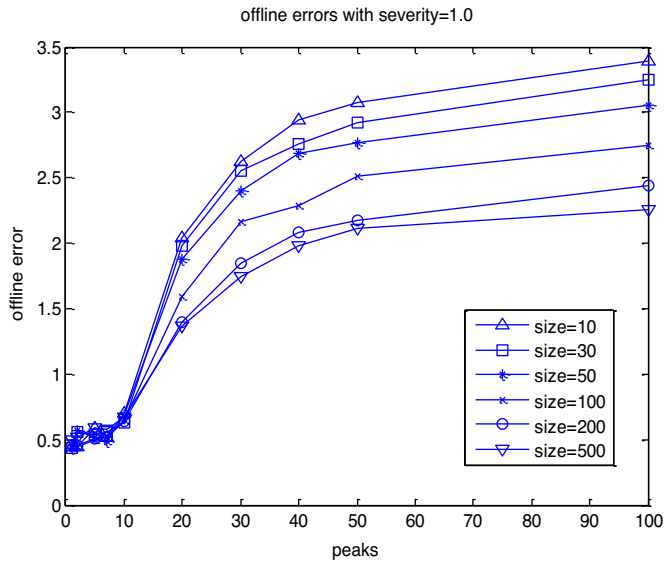


Fig. 4. Comparisons of different memory sizes on the MPB with different numbers of peaks.

and has the best performance when the memory size is set to 500. It is difficult to say which one is better when $N_P \leq 10$ because the changes of the curves are unstable. Another phenomenon is that the rising tendency of the curve is rather slow with the increase of N_p when the memory size is set to 500.

By comparing Fig. 3 and Fig. 4, a noticeable difference that can be observed is that the difference between each of the two curves in Fig. 4 is clearer than that in Fig. 3. Thus, we can obtain another conclusion that the optimal setting of the memory size is more influenced by the number of peaks than by the shifting severity.

Overall, if there are few peaks in the search space, a large memory size is not a wise choice. This is because there are many redundant or useless records in the memory. When re-evaluating the memory, many fitness evaluations are wasted on them. As a result, the benefit of the memory is damped. By contrast, a large memory size would be helpful if there are many peaks in the search space because more historical solutions can be kept.

6. Conclusion and future work

Both the memory scheme and the multi-swarm strategy are effective techniques for dynamic optimization problems. However, their hybrids have scarcely been studied. In most cases, employing the traditional population updating methods to enhance the performance of the multi-swarm algorithms by the memory may not yield ideal results. Aimed at the two aforementioned problems, a new swarm updating method is proposed in this paper and embedded into the typical species-based algorithm SPSO. The number of replaced particles, instead of being predefined, is set adaptively according to the number of species. This may alleviate the impact generated by the unsuitable assignment. To not substantially destroy the exploitation capability of each species, no more than one particle is replaced by the memory in each species. The retrieved memory particles are grouped into four categories and processed by different strategies.

To demonstrate the validity and feasibility of the new method, the proposed algorithm MSPSO is compared with SPSO and its variants on both the MPB and CMPB. Moreover, MSPSO is tested on the DRPBG and compared with other algorithms. The experimental results show that MSPSO is competitive.

The effect of the memory size on the performance of the algorithm is tested in different dynamic scenarios. The experimental results indicate that the optimal setting of the memory size is largely influenced by the number of peaks in the search space. If there are few peaks, a small memory size would be favorable. When the number of peaks becomes larger, a large memory size would be better. How to set the memory size adaptively is a topic of our future work.

In the future, the memory scheme will be combined with other multi-swarm methods and other PSO algorithms, e.g., works proposed in Refs. [45] and [46].

Acknowledgement

This work is partly supported by the National Natural Science Foundation of China (No.61573327).

References

[1] T.T. Nguyen, S. Yang, J. Branke, Evolutionary dynamic optimization: a survey of the state of the art, Swarm Evol. Comput. 6 (2012) 1–24.
[2] Y. Jin, J. Branke, Evolutionary optimization in uncertain environments—a survey, IEEE Trans. Evol. Comput. 9 (3) (2005) 303–317.
[3] T.T. Nguyen, Continuous Dynamic Optimisation Using Evolutionary Algorithms, University of Birmingham, 2011.
[4] X. Hu, R.C. Eberhart, Adaptive particle swarm optimization: detection and response to dynamic systems, Proc. IEEE Congr. Evol. Comput. (2002) 1666–1670.
[5] J.J. Grefenstette, Genetic algorithms for changing environments, Proc. PPSN (1992) 137–144.
[6] R.W. Morrison, Designing Evolutionary Algorithms for Dynamic Environments, Springer Science & Business Media, 2004.
[7] T. Blackwell, Particle swarm optimization in dynamic environments, in: Evolutionary Computation in Dynamic and Uncertain Environments, Springer, 2007, pp. 29–49.
[8] H. Richter, S. Yang, Memory based on abstraction for dynamic fitness functions, in: Applications of Evolutionary Computing, Springer, 2008, pp. 596–605.
[9] R.K. Ursem, Multinational GAs: multimodal optimization techniques in dynamic environments, Proc. GECCO (2000) 19–26.
[10] C. Rossi, M. Abderrahim, J.C. Díaz, Tracking moving optima using Kalman-based predictions, Evol. Comput. 16 (1) (2008) 1–30.
[11] T. Blackwell, J. Branke, Multiswarms, exclusion, and anti-convergence in dynamic environments, IEEE Trans. Evol. Comput. 10 (4) (2006) 459–472.
[12] C. Li, S. Yang, A general framework of multipopulation methods with clustering in undetectable dynamic environments, IEEE Trans. Evol. Comput. 16 (4) (2012) 556–577.
[13] S. Hui, P. Suganthan, Niching-based self-adaptive ensemble DE with MMTS for solving dynamic optimization problems, Proc. IEEE Congr. Evol. Comput. (2014) 1536–1541.
[14] C. Cruz, J.R. González, D.A. Pelta, Optimization in dynamic environments: a survey on problems, methods and measures, Soft Comput. 15 (2011) 1427–1448.
[15] D. Parrott, X. Li, A particle swarm model for tracking multiple peaks in a dynamic environment using speciation, Proc. IEEE Congr. Evol. Comput. (2004) 98–103.
[16] D. Parrott, X. Li, Locating and tracking multiple dynamic optima by a particle swarm model using speciation, IEEE Trans. Evol. Comput. 10 (4) (2006) 440–458.
[17] T. Blackwell, J. Branke, Multi-swarm optimization in dynamic environments EvoWorkshops, vol. 3005, Springer, 2004, 2016, pp. 489–500.
[18] T. Zhu, W. Luo, L. Yue, Combining multipopulation evolutionary algorithms with memory for dynamic optimization problems, Proc. IEEE Congr. Evol. Comput. (2014) 2047–2054.
[19] J. Branke, T. Kaußler, C. Smidt, H. Schmeck, A multi-population approach to dynamic optimization problems, in: Evolutionary Design and Manufacture, Springer, 2000, pp. 299–307.
[20] T.M. Blackwell, Swarms in dynamic environments, Proc. GECCO (2003) 1–12.
[21] J.-P. Li, M.E. Balazs, G.T. Parks, P.J. Clarkson, A species conserving genetic algorithm for multimodal function optimization, Evol. Comput. 10 (3) (2002) 207–234.
[22] C. Li, S. Yang, A clustering particle swarm optimizer for dynamic optimization, Proc. IEEE Congr. Evol. Comput. (2009) 439–446.
[23] S. Yang, C. Li, A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments, IEEE Trans. Evol. Comput. 14 (6) (2010) 959–974.
[24] S. Yang, Population-based incremental learning with memory scheme for changing environments, Proc. GECCO (2005) 711–718.
[25] S. Yang, X. Yao, Population-based incremental learning with associative memory for dynamic environments, IEEE Trans. Evol. Comput. 12 (5) (2008) 542–561.
[26] S. Yang, Memory-based immigrants for genetic algorithms in dynamic environments, Proc. GECCO (2005) 1115–1122.
[27] J. Branke, Memory enhanced evolutionary algorithms for changing optimization problems, Proc. IEEE Congr. Evol. Comput. (1999) 1875–1882.
[28] U. Halder, S. Das, D. Maity, A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments, IEEE Trans. Cybernetics 43 (3) (2013) 881–897.
[29] J. MacQueen, Some methods for classification and analysis of multivariate observations, Proc. the Fifth Berkeley Symposium on Mathematical Statistics and Probability (1967) 281–297.
[30] J. Kennedy, R. Eberhart, Particle swarm optimization, Proc. Int. Conf. on Neural Network (1995) 1942–1948.
[31] M. Clerc, J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, IEEE Trans. Evol. Comput. 6 (1) (2002) 58–73.
[32] S. Bird, X. Li, Enhancing the robustness of a speciation-based PSO, Proc. IEEE Congr. Evol. Comput. (2006) 843–850.
[33] W.-N. Chen, J. Zhang, Y. Lin, N. Chen, Z.-H. Zhan, H.S.-H. Chung, et al., Particle swarm optimization with an aging leader and challengers, IEEE Trans. Evol. Comput. 17 (2) (2013) 241–258.
[34] A. Engelbrecht, Particle swarm optimization: velocity initialization, Proc. IEEE Congr. Evol. Comput. (2012) 1–8.
[35] T. Zhu, W. Luo, L. Yue, Dynamic optimization facilitated by the memory tree, Soft Comput. 19 (2015) 547–566.
[36] C. Li, S. Yang, A generalized approach to construct benchmark problems for dynamic optimization, Proc. 7th Int. Conf. Simulated Evol. Learn. (2008) 391–400.
[37] C. Li, S. Yang, T. Nguyen, E. Yu, X. Yao, Y. Jin, et al., Benchmark generator for CEC 2009 competition on dynamic optimization, University of Leicester, University of Birmingham, Nanyang Technological University, Tech. Rep. (2008).
[38] C. Li, S. Yang, D.A. Pelta, Benchmark generator for the IEEE WCCI-2012 competition on evolutionary computation for dynamic optimization problems, China University of Geosciences, Brunel University, University of Granada, Tech. Rep. (2011).
[39] C. Li, S. Yang, M. Yang, Maintaining diversity by clustering in dynamic environments, Proc IEEE Congr. Evol. Comput. (2012) 1–8.
[40] J. Lepagnot, A. Nakib, H. Oulhadj, P. Siarry, A dynamic multi-agent algorithm applied to challenging benchmark problems, Proc. IEEE Congr. Evol. Comput. (2012) 1–8.
[41] J. Brest, A. Zamuda, B. Boskovic, M.S. Maucec, V. Zumer, Dynamic optimization using self-adaptive differential evolution, Proc. IEEE Congr. Evol. Comput. (2009) 415–422.
[42] P. Korošec, J. Silc, The differential ant-stigmergy algorithm applied to dynamic optimization problems, Proc. IEEE Congr. Evol. Comput. (2009) 407–414.
[43] F.O. De França, F.J. Von Zuben, A dynamic artificial immune algorithm applied to challenging benchmarking problems, Proc. IEEE Congr. Evol. Comput. (2009) 423–430.
[44] E. Yu, P.N. Suganthan, Evolutionary programming with ensemble of explicit memories for dynamic optimization, Proc. IEEE Congr. Evol. Comput. (2009) 431–438.
[45] I. Fister, X.-S. Yang, K. Ljubič, D. Fister, J. Brest, I. Fister, Towards the novel reasoning among particles in pso by the use of rdf and sparql, Sci. World J. 2014 (2014).
[46] I. Fister Jr., A. Tepeh, J. Brest, I. Fister, Population size reduction in particle swarm optimization using product graphs, in: Mendel 2015, Springer, 2015, pp. 77–87.