Applied Soft Computing Journal xxx (xxxx) xxx

Contents lists available at ScienceDirect

Applied Soft Computing Journal

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

Dual-archive-based particle swarm optimization for dynamic optimization

Xiao-Fang Liu [a], Yu-Ren Zhou [a,c,*], Xue Yu [a], Ying Lin [b]

[a] School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, PR China
[b] Department of Psychology, Sun Yat-sen University, Guangzhou 510006, PR China
[c] The Key Laboratory of Machine Intelligence and Advanced Computing, Sun Yat-sen University, Ministry of Education, China

A R T I C L E   I N F O

Article history:
Received 12 April 2019
Received in revised form 15 August 2019
Accepted 14 October 2019
Available online xxxx

Keywords:
Particle swarm optimization
Dynamic optimization problems
Evolutionary computation
Information reuse

A B S T R A C T

In dynamic optimization problems, although the problem environments keep changing, a new environment is usually related to its previous environments. Based on the relevance, the search experience in the previous environments can be reused to accelerate the optimization of the current new environment. Thus, in this paper, we propose a dual-archive-based particle swarm optimization to utilize the useful information accumulated in past environments as well as to explore the emerging information of each new environment. Specifically, in the proposed algorithm, the good solutions found in past environments are stored in two different archives, i.e., a fine-grained archive and a coarse-grained archive, so as to preserve both detailed information and systemic information, respectively. Once the environment is changed, the solutions in the two archives will be used for guidance to quickly find high-quality solutions in the new environment. The proposed algorithm is evaluated on the famous moving peaks benchmark in terms of two performance measures. The experimental results show that the proposed algorithm is competitive with state-of-the-art algorithms for dynamic optimization problems.

1. Introduction

During the past two decades, evolutionary algorithms (EAs) have successfully solved lots of static optimization problems, but there are still challenges to apply EAs to dynamic optimization problems (DOPs) whose optima dynamically change with time [1–5]. Usually, when the environment changes, the optimization process of an EA tends to be already convergent in the previous environments and hence lacks enough diversity to find the new optimum of the new environment [6]. To overcome this diversity loss problem, it is intuitive to treat each new environment as a new static problem and restart the algorithms in each new environment [7]. However, frequent restarting shall greatly reduce the algorithm's search efficiency since all historical search experience in past environments have been discarded [8,9]. Actually, it has always been a challenge to utilize historical information accumulated in past environments when solving DOPs. Besides, the multimodal characteristic of DOPs further increases the difficulty of solving the problem. Due to these challenges, it has become a research hotspot to design efficient algorithms for DOPs.

In the literature, researchers have proposed different strategies to overcome the diversity loss problem in dynamic optimization, such as introducing new individuals to enhance the population diversity when the environment changes [10] and maintaining diversity along the whole evolutionary process [11]. Alternatively, some researchers have proposed memory schemes that use archives for solution storage and, when necessary, bring the archived solutions into the population to increase diversity [12]. However, many of existing memory schemes do not differentiate solutions from different environments and use only one archive to store all these different solutions [13,14]. Thus, these studies always ignore the unique characteristics of each single environment and the correlation between successive environments. Even, some of these studies only store the best solution discovered for each environment [15], and hence they often encounter difficulties in multimodal DOPs. Focusing on the multimodal feature, some researchers have developed multipopulation methods to track multiple moving peaks. However, these multipopulation methods often directly inherit the information (e.g., populations) of all past environments [16] or simply the most recent environment [17]. As a result, they pay too much or too little attention to the search experience of early environments, causing inefficient information reuse, especially on DOPs with large numbers of peaks.

* Corresponding author.
E-mail addresses: nliuxf@qq.com (X.-F. Liu), zhouyuren@mail.sysu.edu.cn (Y.-R. Zhou).

https://doi.org/10.1016/j.asoc.2019.105876
1568-4946/© 2019 Elsevier B.V. All rights reserved.

Based on the above discussion, it should be necessary to develop a granular solution storage scheme to replace the traditional unitary storage scheme. Besides, since each new environment is always different from the previous environments, it is also essential to design schemes to explore new information while reusing historical information once the environment changes [18]. To meet these requirements, in this paper, we propose a dual-archive-based particle swarm optimization algorithm, namely DPSO, to solve DOPs. In the proposed DPSO, two-level archives, i.e., a fine-grained archive and a coarse-grained archive, are adopted to better identify the useful information accumulated in the previous environments. Specifically, the fine-grained archive only stores the good solutions found in the most recent previous environment so as to better exploit the most relevant historical information, while the coarse-grained archive collects high-quality solutions from all past environments so as to better summarize the whole dynamic environmental information and provide better global guidance. The performance of the proposed algorithm is validated on the widely used moving peaks benchmark [19] in comparison with several typical EAs for DOPs. In the experiments, we assume that environmental changes can be explicitly observed and following [20], we preset the change frequency in terms of the number of function evaluation times so as to decouple the change detection from change handling. Experimental results show that the proposed algorithm is competitive to the compared algorithms. Generally speaking, the contribution of this paper lies in the following three aspects.

(1) Solution storage: We use *two archives* for preserving historical information. One archive is fine-grained and it considers the detailed subarea feature of a single environment; the other archive is coarse-grained and it implies the comprehensive information of the search space in a dynamic environment.
(2) Solution reuse: Solutions in the two archives are utilized for different purposes. The solutions in the fine-grained archive are used for *exploitation*, and the solutions in the coarse-grained archive are adopted for *exploration*. The combination of these two kinds of solutions helps to achieve more efficient reuse of historical information
(3) New particle swarm optimization: Considering the complexity of DOPs, the traditional particle swarm optimization (PSO) performs unsatisfactorily due to its incapability of reusing the search experience. To improve the PSO, we investigate how to use historical solutions for guidance in dynamic environments so as to accelerate the optimization of PSO in new environments.

The rest of the paper is organized as follows. Section 2 describes the DOPs and reviews the related work. Section 3 presents the proposed DPSO in detail. Experiments are performed in Section 4 to evaluate the performance of the proposed algorithm as well as to investigate the effect of each algorithm component. Section 5 discusses the advantages and disadvantages of the DPSO. Finally, conclusions and future work are summarized in Section 6.

2. Preliminaries

2.1. Problem description

A DOP can be formulated as

$$\text{Optimize}\, F(\boldsymbol{x}, \varepsilon(t)) = F(x_1, x_2, \ldots, x_n, \varepsilon(t)) \tag{1}$$

where $\boldsymbol{x} = [x_1, x_2, \ \ldots, \ x_n]$ is an n-dimensional vector in the decision space, $\varepsilon(t)$ represents the environmental factors that change with time t, and F is the fitness function of the problem to be optimized. Assume that the environment of a DOP changes at time $t_1, t_2, \ldots, t_j, \ldots,$ and t_T, then the dynamic environment can be divided into a series of independent static environments, where the $(j+1)$th static environment starts from t_j and ends at time t_{j+1}. Notably, the fitness function also varies as the environment changes, resulting in the change of the optimal solutions. To solve a DOP, EAs need to track the moving optima.

2.2. Related work

Different techniques have been developed to solve DOPs and a survey can be found in [21]. These techniques can be loosely categorized as the following three types.

2.2.1. Memory scheme

Memory schemes store the search experience of past environments for future utilization. The memory schemes can be implicit and explicit. Implicit memory schemes store redundant representations [22], while explicit memory schemes store historical information in an external archive [19]. The memory is often updated when the environment changes [23], periodically or at a random time interval [24]. Usually, solutions for storage are selected according to factors such as ages and fitness values, and then these solutions may be simply stored [13] or further converted to population probability vectors [24] to represent the search achievement. Besides the update of memory, how to use the memory is another crucial problem. It is popular to introduce the solutions from memory into the population to increase diversity and accelerate convergence when the environment changes [25]. For example, in [26], the authors try adding all solutions in memory to the population or just select some excellent solutions to replace the inadequate individuals in the population; in [27], a variant of the best solution in memory is used to exploit its neighboring areas; Luo et al. proposed to select some solutions into the population according to their fitness values and distances to the nearest species to enhance diversity [28]. Also, some researchers have proposed to use the memory to replace inferior solutions during the whole evolutionary process [29].

2.2.2. Diversity maintenance

The population of an algorithm tends to lose diversity as its optimization proceeds. Once the population has collapsed and all individuals have become very similar, it will be hard for the algorithm to explore new environments anymore. Thus, it is essential to maintain population diversity in dynamic environments [30]. Different methods have been developed to enhance population diversity:

First, a common method is to increase the population diversity when a change is detected, which is often achieved by randomizing some parts of or the whole population. For example, in [10], random immigrate strategies replace some individuals with randomly generated solutions to diversify the population; in [20], each individual is relocated using a different radius that is calculated according to the information obtained in previous environments; in [31], new individuals far away from the existing individuals are added into the population.

Second, it is also popular to consistently maintain diversity along the whole evolutionary process. For example, to enhance swarm diversity, multiple topologies have been designed for PSO, such as a gridlike neighborhood structure [32] and a hierarchical structure [33]; repulsion mechanisms are proposed to avoid swarm convergence; charged particles orbit around neutral particles to keep diversity [34]; multiple particles are composed together for regaining local diversity [35]; quantum mechanics and Brownian motions are simulated to spread around the area that encompasses the possible changes in [36–38].

2.2.3. Multipopulation method

Since DOPs often have multiple peaks, multipopulation methods are popular to locate and track multiple moving peaks [39, 40]. In a multipopulation method, different populations have different roles. Usually, some populations are used for exploration and some populations exploit the detected peaks [41, 42]. Similarly, in [43] and [44], one population is employed to maintain diversity while another population is used to improve convergence. Multiple populations can also be created by dividing a large population to search different subareas. For population partition, niching techniques are often utilized. For example, in [45], multiple populations are adaptively formed by speciation techniques; in [46], the dot product of two vectors is used to define niches. In addition to niching techniques, clustering techniques [47], [48] are also popular, such as hierarchical clustering [49] and k-means [50]. Since the number of peaks may vary with time, the number of populations is expected to be adaptively adjusted according to some relevant real-time information, such as the appearance of stagnating populations [51], the difference of the number of populations between current and previous increasing points [52], and heuristic relationships between the number of historical populations and current survivors [16]. During the multipopulation optimization, scheduling mechanisms are often adopted to allocate more computing resources to the well-performing populations [53]. Usually, a population will be hibernated [54] or restarted [55] to avoid the waste of computing resources if it has converged. To enhance the diversity among populations, multiple techniques have also been developed, e.g., exclusion and anti-convergence operators [56] to strengthen interactions between populations.

3. Methodology

The proposed dual-archive-based particle swarm optimization method, called DPSO, includes two optimization components: one exploitation component to exploit the historical experience of past environments, and one exploration component to discover new information of new environments. The good solutions found by the two optimization components are stored in two archives for reuse. In the following, we will describe the detailed procedure of the DPSO in one environment indexed by i as an example. The situation is similar in other environments. For clarity, the archives for solution storage are introduced first since they play an important role in the optimization components. Then the two optimization components are presented. The complete algorithm will be given at the end.

3.1. Two archives for historical solution storage

The solutions found in past environments represent the search experience and they are often useful in new environments. Thus, it is vital to select useful historical solutions and store them in a reasonable way for subsequent reuse. First, for solution selection, due to the multimodal characteristic of DOPs, the solutions that can describe the feature of each subarea may be useful. Hence, in each environment, the best solution found in each subarea can be selected for storage. Second, for solution storage, since each environment is different from but interrelated to each other, the environmental property of the solutions should be considered and we adopt two archives to store the selected solutions in a fine-grained scheme and a coarse-grained scheme, respectively. Particularly, the fine-grained archive A_F focuses on a single environment and aims to store the complete area information of the environment. In contrast, the coarse-grained archive A_C tries to preserve the systemic information of the whole search space that has been explored in all past environments and A_C has a limited size of N to avoid redundancy. In this way, the solution storage takes the multimodal characteristic of the problem into account and keeps the good solutions of past environments in a more granular mode.

The two archives are updated when the environment is changed. In the process of dynamic optimization, each population is indeed responsible for searching a different subarea and hence the best solution found by the population can be regarded as the local optimum of the corresponding subarea. Assume that in the previous environment $i-1$ ($i > 1$), the best solution found in each subarea, i.e., local optimum, is collected into a temporary set L. To update the fine-grained archive A_F, the archive A_F is emptied and then all solutions in L are added to A_F. In contrast, only a fraction of solutions in L will be selected into archive A_C according to their Euclidean distances in the search space for better diversity. First, we put all solutions in L and archive A_C in a temporary set O. If the size of O exceeds N, then the closest solution pair in the search space is selected and a random one in the solution pair will be discarded. The discarding step continues until the size of O becomes N. At last, archive A_C is updated by set O. The updating procedure of archive A_C is presented in **Algorithm 1**. Note that after updating the two archives, the solution set L will be emptied.

Algorithm 1: Archive A_C updating procedure

```
Input: archive $A_C$, fixed archive size of $N$, solution set $L$
Output: $A_C$
1:  $O = L \cup A_C$
2:  Calculate the Euclidean distance between any two solutions in $O$
3:  While $|O| > N$
4:          Find the nearest solution pair $(s_1, s_2)$
5:          If rand(0,1) $\le 0.5$
6:              Remove $s_1$ from $O$
7:          Else
8:              Remove $s_2$ from $O$
9:          End
10:  End
11:  $A_C = O$
```

3.2. Exploit historical information of the last environment

Since the current environment i is changed from the previous environment $i-1$, the optima of these two successive environments may be geographically close. Hence, the good solutions of the environment $i-1$ (stored in archive A_F) are very likely to be favorable in the environment i. Thus, it should be promising to apply local search to these solutions to quickly find high-quality solutions in the new environment. Herein, we use the simple yet powerful Rosenbrock method [57] for local search. In the local search, each solution in archive A_F plays as a starting point and then a detection stage and a direction construction stage will be carried out in turn to approximate the optimum in the restricted subarea.

3.2.1. Detection stage

For local search, given one starting point X_0 and D directions $e_1, e_2, \ldots, e_D$ with the corresponding step sizes $l_1, l_2, \ldots, l_D$, respectively, the detection stage goes through multiple iterations on each direction until it fails to find a better solution on any direction. In the kth iteration, assume that the starting point is X_k and it tries a step in each direction. At first, we set $Y = X_k$. In the dth step, Y tries to move in direction e_d to obtain a new point V_d as

$$V_d = Y + l_d e_d \qquad (2)$$

Assume the DOP is a maximization optimization problem. If the new point V_d has a better fitness value than Y, the step size

l_d will be increased and Y will be updated as V_d; otherwise, the step size l_d will be decreased and reversed by

$$l_d = \begin{cases} \alpha l_d, & \text{if } \mathrm{f}(V_d) > \mathrm{f}(Y) \\ -\beta l_d, & \text{otherwise} \end{cases} \tag{3}$$

$$Y = \begin{cases} V_d, & \text{if } \mathrm{f}(V_d) > \mathrm{f}(Y) \\ Y, & \text{otherwise} \end{cases} \tag{4}$$

where $\alpha > 1$ is amplification factor and $0 < \beta < 1$ is reduction factor. We set α as 3 and set β as 0.5 following [57]. At the end of the kth iteration, the starting point X_{k+1} for the (k+1)th iteration is set as Y. If the kth iteration has successively improved the solution in one or more directions, the detection stage enters the next iteration; otherwise, it is terminated. At the end of the detection stage, X_0 is updated as Y if Y is better. Note that the local search process will be terminated if the step sizes for all directions $|l_d| \le \varepsilon$ or $\|Y - X_k\| \le \varepsilon$. Then Y is considered as the approximated optimum in the area. The ε is set as 1e−5 following [57].

3.2.2. Direction construction stage

After the detection stage, we construct D new orthogonal directions using the successful steps for faster movements towards the optimum in the next detection stage. First, we generate a group of linearly independent directions p_d ($1 \le d \le D$) according to the search information in the detection stage. Suppose that λ_d is the algebraic sum of all the successful steps l_d in direction e_d during the detection stage. Then we let

$$p_d = \begin{cases} e_d, & \text{if } \lambda_d = 0 \\ \sum_{i=d}^{D} \lambda_i e_i, & \text{otherwise} \end{cases} \tag{5}$$

where p_1 is the vector joining the initial and final points obtained by the use of directions $e_1, \ldots, e_n$ in the detection stage and p_d is the sum of all the advances made in all directions except for the first $(d-1)$ ones.

Then we use the Gram–Schmidt process [58] to obtain an orthogonal basis (a group of new directions e_d) as

$$e_d = \begin{cases} p_d, & \text{if } d = 1 \\ p_d - \sum_{i=1}^{d-1} \frac{e_i^T p_d}{e_i^T e_i} e_i, & \text{otherwise} \end{cases} \tag{6}$$

The new directions e_d are normalized later. These D linearly independent and orthogonal directions will be used for the next cycle of movements.

After finishing the above direction construction, we return to the detection stage. These two procedures are performed in turn until reaching the termination condition given in the detection stage. Then, the best-found solution will be added to the solution set L.

3.3. Explore new areas with the guidance of archive A_C

Since the exploitation component mainly finds high-quality solutions that are close to the good solutions found in the last environment, it may fail to find the global optimum if the peak where the global optimum is located has not been discovered before. Thus, we further design an exploration component based on particle swarm optimization to search for undiscovered subareas. It not only helps to get out of local optima but also helps to better adapt to the dynamic environment. Due to the multimodal characteristic of the DOP, we divide the swarm into multiple subswarms to enhance the exploration ability of the swarm. Each subswarm moves independently to search one subarea. Besides, since the subswarms are aimed to explore new subareas, the coarse-grained archive A_C may be useful since it can afford some global information of the whole dynamic search space. Hence, the solutions in archive A_C are employed to assist the optimization of subswarms. Overall, the exploration component is composed of three parts, i.e., swarm generation, swarm division, and subswarm search. Assume that the size of the initial swarm is S and the number of subswarms to create is C.

3.3.1. Swarm generation

The swarm is initialized by some randomly generated solutions plus several solutions from archive A_C. The wide variety of solutions from all past environments, i.e., archive A_C, should be quite potential to locate different subareas in the whole search space and can act as leaders to push the swarm towards multiple locally optimal solutions. Hence, the number of solutions selected from A_C, denoted n_{leader}, is defined by a random integer in the range $1 \ldots C$ to balance the subarea exploration based on historical experience and completely random search. That is, n_{leader} solutions will be randomly selected from the archive A_C. Note that each solution in the archive A_C is only allowed to be selected once and the solutions used for exploitation cannot be selected again to avoid repeatedly searching the explored subareas. The selected archived solutions are then used to initialize the historical best position $pBest_k = [pBest_{k1}, \ldots, pBest_{kD}]$ and position $X_k = [X_{k1}, \ldots, X_{kD}]$ of n_{leader} particles, where k represents particle index and $1 \le k \le n_{\text{leader}}$. For the other ($S$-$n_{\text{leader}}$) particles, the X_k of particle k is initialized by a randomly generated solution in the search space and the $pBest_k$ is set as X_k, where $n_{\text{leader}} + 1 \le k \le S$. The velocity $V_k = [V_{k1}, \ldots, V_{kD}]$ of each particle k is randomly generated in the range of $[-0.2 \times (U_d - L_d), 0.2 \times (U_d - L_d)]$ for each dimension d, where L_d and U_d are the lower and upper bound of the search space and $1 \le k \le S$.

In a word, the use of archive A_C for swarm generation works on two sides. On the one hand, the archive A_C records the good search experience accumulated in all past environments, which can help to avoid blind search under the limited computation in the current environment. On the other hand, the reuse of the diversified solutions in archive A_C can greatly accelerate the algorithm's global exploration especially when there are large numbers of peaks.

3.3.2. Swarm division

The initial swarm is then divided into multiple subswarms. To ensure that each subswarm can search a different subarea, we divide the particles in the swarm according to their positions in the search space and assign the neighboring particles around the good solution of a subarea into the same subswarm. The speciation technique [59] is adopted for swarm partition, based on Euclidean distance evaluation. The particles are first sorted by their performance from the best to worst according to their fitness values. Then the best one is selected from the available particles to be the seed. The seed and its $(S/C-1)$ nearest particles form a new subswarm. Note that each particle can be selected into one and only one subswarm. The division procedure continues until all the particles are assigned to a subswarm. Finally, C subswarms are created.

3.3.3. Search process of subswarm

In each subswarm, each particle k updates its position X_k by

$$V_{kd} = \omega V_{kd} + c_1 r_1 (pbest_{kd} - X_{kd}) + c_2 r_2 (gbest_d - X_{kd}) \tag{7}$$

$$X_{kd} = X_{kd} + V_{kd} \tag{8}$$

where ω is the inertia weight, c_1 and c_2 are acceleration coefficients, r_1 and r_2 are random numbers in the range of [0,1], and *gbest* is the best one among all the *pBest* in the subswarm. If the new position X_k is better than the $pBest_k$, then the $pBest_k$ will be updated as X_k.

To avoid computational waste, when a subswarm has converged or stagnated, the subswarm is restarted. A subswarm is supposed to be converged if the radius of the subswarm in the search space is less than the predefined threshold $\delta = 1e - 4$. Similarly, a subswarm is regarded as stagnated if the *gbest* stagnates more than 50 generations. Once a subswarm has converged or stagnated, the local search method detailed in Section 3.2 will be performed on the *gbest* solution to increase solution accuracy and the solution obtained after the local search is added to the temporary solution set L. Note that the maximum available function evaluations (FEs) for the local search is set as (maxFEs-usedFEs)/C, where maxFEs is the maximum available FEs of the current environment and usedFEs is the number of consumed FEs. To restart a subswarm, we randomly reinitialize the subswarm. One solution, with a predefined probability q, will be selected into the subswarm from archive A_C. That is, a random real number r within [0, 1] is generated and compared with q. If $r < q$, then a random solution from A_C is directly selected into the subwarm to replace a particle; otherwise, no solution from A_C is used and all particles are randomly generated. The probability q is set as 0.5 to balance the utilization of problem information learnt from past environments and the completely new exploration.

To balance the FEs spent for exploitation and exploration, the maximum available FEs for each local search in the exploitation component is set as $(\text{maxFEs- usedFEs})/(N_{LS} + C)$, where maxFEs is the maximum available FEs of the current environment and usedFEs is the number of FEs that have been consumed before the exploitation component. In this way, each local search and each subswarm are expected to be assigned an equal number of FEs for search.

All in all, in the proposed DPSO, the exploitation and exploration components work together to find good solutions in the new environment using the two-level archives. The newly-found good solutions will be further used to update the archives for the next new environment. An example of the implementation of the DPSO in an environment is illustrated in Fig. 1.

3.4. Complete algorithm

The flowchart of the proposed DPSO in dynamic environments is shown in Fig. 2. In the first environment, since there is no historical information, only the exploration component is executed. When the environment changes, the good solutions found during the optimization process are stored in set L and then used to update the two archives A_F and A_C. In the new environment, the exploitation component is applied. Later the exploration component is carried out until the next new environment comes. When an new environment arrives, the good solutions discovered are collected into the archives A_F and A_C. The optimization procedure of new environments continues until meeting the termination condition.

Specifically, in the exploitation component, the solutions in A_F are used to perform local search and the new solutions obtained by the local search are stored in set L. In the exploration component, a new swarm is randomly generated and several solutions are selected to add into the swarm from archive A_C. Then the swarm is divided into C subswarms and each subswarm searches independently. If the subswarm has converged or stagnated, the best solution in the subswarm is taken to perform a local search to increase accuracy and the obtained solution is stored in the set L. Later, the subswarm is reinitialized and, with probability $q = 0.5$, selects one solution to add from archive A_C. The search process of each subswarm continues until the environment is changed.

3.5. Computational complexity of DPSO in one environment

The computational complexity of DPSO in one environment includes that of the exploitation component, the exploration component, and the archive update. Given a DOP in D-dimensional decision space. Assume that the maximum sizes of the archives A_F and A_C are N_{LS} and N, respectively.

First, in the exploitation component, each solution in the archive A_F performs a local search. Assume that the number of cycles in a local search is T, and in each cycle, the number of iterations for the detection stage is K. Then one cycle of the local search procedure has a complexity of $\text{O}((D+D)\times K + D\times K + D\times D\times D) = O(\times K + D^3)$, where $\text{O}((D + D)\times K)$ is for the detection stage and $\text{O}(D\times K + D\times D\times D)$ is for the direction construction stage. Hence the local search procedure with T cycles on N_{LS} solutions requires $\text{O}((D\times K + D^3)\times T\times N_{LS})$ computations, that is $\text{O}(D\times K\times T\times N_{LS}+D^3\times T\times N_{LS})$.

Second, in the exploration component, there are three parts: swarm generation, swarm division, and subswarm search. Assume that the swarm size is S, the number of subswarms is C, the size of each subswarm is $P = S/C$, and the number of generations of PSO is G. The swarm generation with the selection of archived solutions requires $\text{O}(S\times D + N\times N_{LS}\times D)$ computations and the swarm division requires $\text{O}(S^2\times D + C\times S + C\times S\times \log P)$ computations. For each created subswarm, the optimization procedure requires $\text{O}(D\times P\times G)$ computations, and the local search on the global best solution requires $\text{O}(D\times K\times T + D^3\times T)$ computations, where K and T are local search parameters as defined above. Hence the computational complexity of the search process of a subswarm is $\text{O}(D\times P\times G + D\times K\times T + D^3\times T)$. Thus, the computational complexity of the exploration component is $\text{O}(S\times D + N\times N_{LS}\times D + S^2\times D + C\times S + C\times S\times \log P+(D\times P\times G + D\times K\times T + D^3\times T)\times C)$, that is $\text{O}(N\times N_{LS}\times D + S^2\times D + C\times S\times \log P + D\times S\times G + D\times K\times T\times C + D^3\times T\times C)$.

Third, the update of the archives A_F and A_C requires $((C+N_{LS}+N)^2\times D)$ computations.

Therefore, the overall worst-case complexity of DPSO in one environment is $\text{O}(D\times K\times T\times N_{LS}+D^3\times T\times N_{LS}+N\times N_{LS}\times D + S^2\times D + C\times S\times \log P + D\times S\times G + D\times K\times T\times C + D^3\times T\times C+(C + N_{LS} + N)^2\times D)$. In our simulations, archive A_F is formed by collecting the global best solution of each subswarm and we set $1<C<S$, $N\le 2\times S$, and hence we have $N_{LS}\approx C, P<S$, and $\log P<P$. Thus, the worst-case complexity of DPSO in one environment can be simplified to $\text{O}(D\times S\times G + S^2\times D + D\times K\times T\times C + D^3\times T\times C)$, that is, the largest one among $\text{O}(D\times S\times G)$, $\text{O}(S^2\times D)$, $\text{O}(D\times K\times T\times C)$, and $\text{O}(D^3\times T\times C)$. This shows that the complexity of the DPSO is dominated by the complexity of particle swarm optimization $\text{O}(D\times S\times G)$, the complexity of distance evaluation $\text{O}(S^2\times D)$, or the complexity of local search $\text{O}(D\times K\times T\times C + D^3\times T\times C)$.

4. Experiment

In this section, the performance of the proposed algorithm DPSO is validated. Our experiments are divided into two parts. The first part is to investigate the effect of the components in DPSO, which helps analyze and understand the search behavior of the proposed DPSO. The second part is to compare the proposed algorithm with several typical algorithms for DOPs, which helps demonstrate the performance of DPSO. In the following, test problems and performance measures are introduced first. Then the parameter settings of the algorithms are briefly described. Experimental results are shown next.

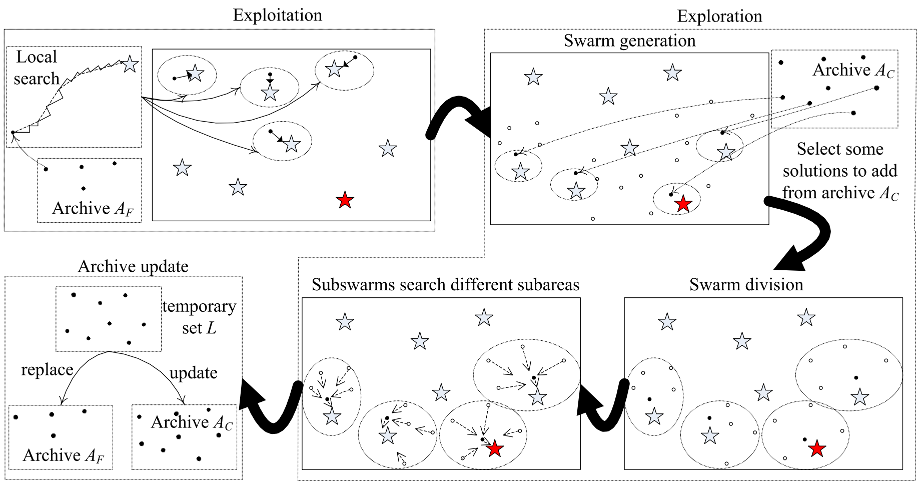


Fig. 1. Example of the DPSO implementation in an environment of a DOP with 8 peaks, where the solid rectangles represent the search space, the stars are the peak positions and the red one is the global optimum, the dash circles represent subareas, the solid circles represent solutions from archives, and the hollow circles are randomly generated solutions.

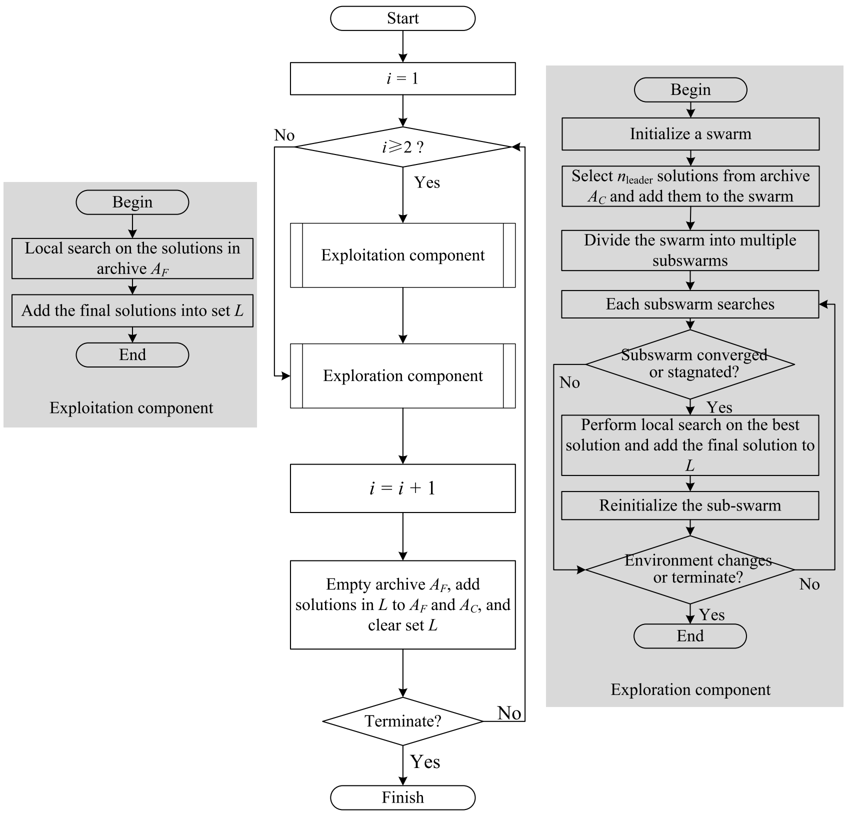

Start
i = 1
i⩾2 ?
No
Yes
Exploitation component
Exploration component
i = i + 1
Empty archive A_F, add solutions in L to A_F and A_C, and clear set L
Terminate?
No
Yes
Finish
Begin
Local search on the solutions in archive A_F
Add the final solutions into set L
End
Exploitation component
Begin
Initialize a swarm
Select n_{leader} solutions from archive A_C and add them to the swarm
Divide the swarm into multiple subswarms
Each subswarm searches
Subswarm converged or stagnated?
No
Yes
Perform local search on the best solution and add the final solution to L
Reinitialize the sub-swarm
Environment changes or terminate?
No
Yes
End
Exploration component

Fig. 2. Flowchart of the proposed DPSO.

4.1. Experimental setting

(1) *Test problem:* The well-known Moving Peaks Benchmark (MPB) [19] is adopted in the experiments. The MPB consists of several peaks. The height, weight, and position of the peaks dynamically change with time. The fitness function of MPB is defined as

$$F(\boldsymbol{x}, t) = \max_{i=1,\dots,P} \frac{H_i(t)}{1 + W_i(t)\sum_{j=1}^{D}(x_j(t) - X_{ij}(t))^2} \tag{9}$$

where t is the index of the environment, $\boldsymbol{x} = [x_1, \dots, x_D]$ is a vector in the D-dimensional decision space, $H_i(t)$, $W_i(t)$, and $\boldsymbol{X}_i(t)$ are the height, weight, and position of peak i in environment t, respectively. Through time, both the height $H_i(t)$ and weight $W_i(t)$ vary with a random Gaussian variable $\sigma \sim$ Normal(0,1) and the position $\boldsymbol{X}_i(t)$ changes with a vector $\boldsymbol{v}_i$ by

$$H_i(t) = H_i(t-1) + s_H \cdot \sigma \tag{10}$$

$$W_i(t) = W_i(t-1) + s_W \cdot \sigma \tag{11}$$

$$\boldsymbol{X}_i(t) = \boldsymbol{X}_i(t-1) + \boldsymbol{v}_i(t) \tag{12}$$

$$\boldsymbol{v}_i(t) = \frac{s_X}{|(1-\lambda)\boldsymbol{r} + \lambda\boldsymbol{v}_i(t-1)|}((1-\lambda)\boldsymbol{r} + \lambda\boldsymbol{v}_i(t-1)) \tag{13}$$

Table 1
Specification setting of the MPB.

Parameter	Value
Number of peaks (P)	50
Dimension of search space (D)	5
Change frequency (u)	5000
Height severity (s_H)	[1, 10]
Width severity (s_W)	[0.1, 1.0]
Peak shape	Cone
Basic function	No
Shift length (s_X)	1
H	[30, 70]
W	[1, 12]

where s_H, s_W, and s_X are severity parameters, $\boldsymbol{r}$ is a random vector, and λ is the correlated parameter that indicates the correlation of the position movements between different environments. The specification setting of the MPB is given in Table 1. In the experiments, we construct multiple MPB instances with different numbers of peaks, shift lengths, correlated parameters, and change frequencies. Note that the change frequency represents the function evaluation times in each environment, which is set as 5000 by default.

(2) *Performance Measure:* Several quality indicators have been suggested to measure the performance of the algorithms for DOPs. In this paper, we adopt two widely used indicators, i.e., offline error (E_o) and best-before-change error (E_B) [16]. The offline error E_o is the average of the best error found in every sampling time as

$$E_o = \frac{1}{num\_sample} \sum_{j=1}^{num\_sample} E_j \tag{14}$$

where *num_sample* is the number of samples in each run and E_j is the best error found at the jth sampling time. Herein, a sampling frequency of two FEs is adopted following [16]. The offline error E_o reflects the response ability of the algorithm to environmental changes. The best-before-change error E_B is the average of the best error obtained before a change occurs

$$E_B = \frac{1}{num\_change} \sum_{j=1}^{num\_change} E_j^{Best} \tag{15}$$

where E_j^{Best} is the best error found in the environment j and *num_change* is the number of environments. The E_B demonstrates the global search ability of an algorithm in all environments.

(3) *Parameter Settings:* For the proposed DPSO, the population size is set as 50 and the number of subswarms C is set as 10 following [15,49]. The acceleration coefficients c_1 and c_2 are set as 1.7 and the inertia weight ω is set as 0.6 following [60]. The maximum size of archive A_F is set as 100. For fair comparisons, all algorithms independently run 20 times following [52] and the average results are reported.

4.2. Effects of the components of DPSO

In this section, we investigate the effects of the two optimization components in DPSO, i.e., *e*xpl*o*itation of historical information (Ei) and *e*xplo*r*ation of new subareas (Er). Two variants of DPSO without the *e*xplo*i*tation component or without the *e*xplo*r*ation component, named noEi and noEr, respectively, are compared with the DPSO. Particularly, in the noEi variant, only the Er component is performed. However, in the noEr variant, since the Ei component needs initial solutions as starting points to perform local search, the Er component needs to be executed once in the first environment and only the Ei component is performed in the following environments. In the experiments, we construct 10 MPB instances with different numbers of peaks varying from 10 to 200, shift length of 1, and correlated parameter λ of 0. The results of DPSO and its two variants noEi, noEr are reported in Table 2.

From Table 2, we can see that the DPSO performs better than both noEi and noEr on all the instances. This shows that both the two components are important in the proposed DPSO. In particular, the exploitation component Ei helps to accelerate the convergence towards superior solutions by precisely using the historical information of the previous environments. Reversely, the exploration component Er conduces to dig up new information and acquire new high-quality solutions. Without the exploration component, the algorithm (i.e., the variant noEr) is easily trapped in local optima.

When comparing noEi with noEr, it is interesting to find that noEi performs better than noEr in terms of metric E_B but worse in terms of metric E_o on almost all the instances. This further shows the Ei and Er components are important to the acceleration of convergence and the improvement of solution accuracy, respectively. Thus, both the exploitation component and the exploration component are essential in DPSO.

To further test the effect of the archive A_C in the exploration component, we also compared the DPSO with two other variants: one variant without archive A_C, named noAc, which does not introduce any solution into the swarm from archive A_C; and one variant eachAc that expects to select one solution to each subswarm from archive A_C, and it sets $n_{\text{leader}} = C$. The results are reported in Table 2. We can see that although noAc performs slightly better on the instances whose number of peaks varies from 20 to 80, it is beaten by the other two algorithms that use archive A_C, i.e., the DPSO and its variant eachAc, on other instances with peak numbers from 100 to 200. Thus, the archive A_C takes effect especially on the instances with a large number of peaks. Indeed, since solutions in the archive A_C are collected from all the past environments, the archive A_C can provide efficient guidance for the proposed DPSO so as to quickly search different subareas under limited computation resources. Additionally, the DPSO performs better than eachAc on 7 out 10 instances in terms of metric E_B, showing that setting the number of archived solutions into swarm as a random integer in the range $1\ldots C$ is better than as a fixed value C, since the former one has higher chance to random search while reusing historical solutions. In general, the use of archive A_C in the exploration component makes a great difference to the DPSO.

4.3. Comparison with other algorithms

To further evaluate the performance of the proposed algorithm, five typical state-of-the-art algorithms for DOPs are selected for comparisons. They are tested on multiple instances with different numbers of peaks, different shift lengths, different correlated parameters, and different change frequencies to evaluate their response abilities. The convergence behavior of each algorithm is also presented.

4.3.1. Algorithms in comparison

Five typical algorithms for DOPs are selected for comparisons: clustering PSO (CPSO) [49], clustering PSO without change detection (CPSOR) [60], adaptive multipopulation framework with PSO (AMP/PSO) [16], dynamic differential evolution with Brownian and quantum individuals (DDEBQ) [11], and neighbor-based learning particle swarm optimization with short-term and long-term memory (NLPSO) [15]. These five algorithms adopt different techniques for DOPs. CPSO restarts when the environment changes; CPSOR maintains population diversity along the whole

Table 2
Results of DPSO and its variants noEi, noEr, noAc, and eachAc.

P	Error	DPSO	noEi	noEr	noAc	eachAc
10	E_o	**2.73(0.73)**	9.23(1.59)	6.80(3.18)	2.90(0.59)	2.96(0.54)
	E_B	**0.62(0.72)**	5.27(1.42)	6.10(3.26)	0.64(0.58)	0.81(0.54)
20	E_o	2.60(0.48)	7.53(1.00)	6.31(2.35)	**2.55(0.24)**	2.59(0.51)
	E_B	0.77(0.36)	4.35(0.84)	5.60(2.45)	**0.64(0.28)**	0.72(0.37)
30	E_o	2.52(0.33)	7.20(1.25)	5.91(1.52)	2.47(0.33)	**2.42(0.34)**
	E_B	0.78(0.27)	4.24(1.13)	5.24(1.50)	**0.74(0.25)**	0.82(0.39)
40	E_o	2.21(0.26)	5.99(0.57)	5.39(1.73)	**2.15(0.22)**	2.55(0.33)
	E_B	0.65(0.19)	3.41(0.46)	4.65(1.79)	**0.59(0.19)**	0.98(0.30)
50	E_o	2.21(0.24)	5.71(0.65)	5.17(1.29)	**2.07(0.23)**	2.29(0.31)
	E_B	0.74(0.22)	3.32(0.66)	4.31(1.33)	**0.55(0.17)**	0.82(0.28)
80	E_o	**1.96(0.20)**	4.67(0.41)	4.29(1.03)	1.99(0.23)	1.98(0.15)
	E_B	0.66(0.15)	2.52(0.34)	3.43(1.03)	**0.65(0.17)**	0.70(0.14)
100	E_o	**1.86(0.13)**	4.67(0.44)	4.32(0.73)	1.89(0.14)	1.89(0.16)
	E_B	**0.62(0.13)**	2.62(0.43)	3.36(0.72)	0.66(0.11)	0.69(0.11)
120	E_o	**1.77(0.12)**	4.22(0.32)	4.28(1.32)	1.84(0.14)	1.86(0.15)
	E_B	**0.62(0.10)**	2.24(0.23)	3.34(1.38)	0.67(0.10)	0.70(0.08)
150	E_o	**1.74(0.10)**	4.03(0.29)	3.91(0.66)	1.78(0.12)	1.75(0.13)
	E_B	0.67(0.06)	2.18(0.27)	2.95(0.73)	0.69(0.11)	**0.66(0.09)**
200	E_o	1.72(0.11)	3.69(0.21)	3.38(0.67)	1.79(0.12)	**1.65(0.14)**
	E_B	0.73(0.08)	1.92(0.20)	2.37(0.62)	0.79(0.11)	**0.70(0.13)**
Best		**8**	**0**	**0**	**8**	**4**

evolutionary process and adds new individuals when the diversity decreases below a predefined threshold; AMP/PSO adopts multiple populations to track multiple moving peaks and adaptively controls the population number according to heuristic information; DDEBQ adopts Brownian individuals, quantum individuals, and aging mechanism to maintain population diversity; and NLPSO adopts short-term memory and long-term memory to assist PSO. These typical algorithms make the comparison more comprehensive and convincing.

The parameters of the compared algorithms are set as follows referring to the original papers. The population sizes for CPSO, AMP/PSO, DDEBQ, and NLPSO are set as 100, 100, 60, and 100, respectively, and the population size of CPSOR is set as $300\times(1-e^{-0.33\times P^{0.5}})$, where P is the number of peaks. For the PSO parameters, the inertia weight is set as 0.6, 0.7298, and 0.1 in CPSOR, AMP/PSO, and NLPSO, respectively, while linearly decreases from 0.6 to 0.3 in CPSO; and the acceleration coefficients c_1 and c_2 are set as 1.7 in CPSO and CPSOR while are set as 1.496 for AMP/PSO. Only one acceleration coefficient $c = 2.0$ is used in NLPSO. In contrast, DDEBQ adopts DE operators (i.e., mutation and crossover) with a crossover rate of 0.9 and an adaptive scale factor. Note that the number of FE times in each environment is equal to the change frequency, which is kept the same in all compared algorithms.

4.3.2. Experimental results

Multiple MPB instances are derived to test the performance of the proposed DPSO. On each instance, the Wilcoxon rank-sum test is performed between the DPSO and each competitor at a 0.05 level of significance in terms of both E_o and E_B. The "+" and "−" indicate that the results of the DPSO are significantly better and significantly worse than the algorithm in comparison, respectively, while "=" indicates that there is no significant difference between the two algorithms. Besides, the average rank (AvgRank) of each algorithm in all instances is also listed to show the overall ranking relationships between algorithms.

4.3.2.1. MPB with different numbers of peaks. In this subsection, we test and compare the performance of the algorithms on 10 instances with different numbers of peaks, i.e., from 10 to 200. In all instances, the shift length is set as 1, the correlated parameter is set as 0, and the change frequency is set as 5000. The results of all algorithms are reported in Table 3.

From Table 3, we can see that the proposed DPSO can achieve competitive performance compared with other algorithms in both metrics E_o and E_B. In detail, the DPSO performs the best on 6 out of 10 instances in metrics E_o and E_B, followed by AMP/PSO that wins on 3 out of 10 instances, while all the other three algorithms CPSO, CPSOR, and DDEBQ do not perform the best on any instances. Specially, DPSO performs worse than AMP/PSO on instances with peak numbers in the range of [10,40] but is better on the instances with a larger number of peaks, i.e., from 50 peaks to 200 peaks in terms of both convergence speed and solution quality. Thus, the proposed DPSO is competitive on instances with large numbers of peaks. Besides, the DPSO obtains the smallest value of AvgRank, showing the overall advantage of the DPSO on other algorithms.

4.3.2.2. MPB with different shift lengths. To test the performance of the DPSO on different environment change severity, we compare the algorithms on instances with different shift lengths s_X, i.e., 1, 2, 3, 4, and 5. The number of peaks is set as two values of 50 and 150, the correlated parameter is set as 0, and the change frequency is set as 5000. The results of the DPSO and the compared algorithms are reported in Table 4.

From Table 4, we can see that the DPSO performs the best on all the 10 instances. Moreover, from the significance test results, the DPSO is significantly better than all the compared algorithms on all instances. Thus, no matter whether the environment changes slightly or severely, the DPSO can find better solutions faster. This is because the exploitation component allows the DPSO to find the moving optima quickly in small changes and the exploration component helps DPSO to find new optima faster in big changes. Therefore, compared with other algorithms, the DPSO has stronger response ability to dynamic environments.

4.3.2.3. MPB with different correlated parameters. To test the performance of the proposed DPSO on MPB instances with different correlation parameters for environmental changes, we construct 12 instances with different correlation parameters for evaluation. The correlation parameter λ is set in the range of [0,1] with a step size of 0.2, the number of peaks is set as 50 and 150, the shift length is set as 1, and the change frequency is set as 5000.

Table 3
Results of all algorithms on MPB with different numbers of peaks.

P	Error	DPSO	CPSO		CPSOR		AMP/PSO		DDEBQ		NLPSO	
10	E_o	2.73(0.73)	6.16(0.87)	+	1.47(0.07)	−	**1.18(0.09)**	−	5.71(0.89)	+	4.48(0.44)	+
	E_B	0.62(0.72)	2.28(0.51)	+	0.14(0.07)	−	**0.05(0.03)**	−	4.34(0.83)	+	1.75(0.42)	+
20	E_o	2.60(0.48)	5.43(0.55)	+	2.39(0.35)	−	**1.83(0.08)**	−	5.32(0.57)	+	4.50(0.54)	+
	E_B	0.77(0.36)	2.39(0.41)	+	1.48(0.32)	+	**0.33(0.12)**	−	3.81(0.54)	+	2.25(0.50)	+
30	E_o	2.52(0.33)	4.87(0.65)	+	2.44(0.27)	=	**1.78(0.07)**	−	4.58(0.71)	+	4.10(0.33)	+
	E_B	0.78(0.27)	2.14(0.50)	+	1.38(0.25)	+	**0.42(0.06)**	−	3.47(0.66)	+	1.83(0.26)	+
40	E_o	2.21(0.26)	4.74(0.42)	+	2.47(0.29)	=	**2.16(0.07)**	−	5.94(0.68)	+	4.37(0.40)	+
	E_B	**0.65(0.19)**	2.20(0.33)	+	1.49(0.28)	+	0.69(0.06)	=	4.68(0.62)	+	1.94(0.36)	+
50	E_o	**2.21(0.24)**	4.45(0.42)	+	2.93(0.26)	+	2.42(0.07)	+	6.34(0.73)	+	3.81(0.37)	+
	E_B	**0.74(0.22)**	2.09(0.34)	+	1.88(0.27)	+	0.77(0.05)	+	4.95(0.67)	+	1.67(0.28)	+
80	E_o	**1.96(0.20)**	3.83(0.25)	+	3.62(0.27)	+	2.72(0.07)	+	5.08(0.26)	+	3.19(0.25)	+
	E_B	**0.66(0.15)**	1.74(0.19)	+	2.40(0.28)	+	1.03(0.07)	+	3.90(0.27)	+	1.41(0.21)	+
100	E_o	**1.86(0.13)**	3.86(0.25)	+	3.10(0.17)	+	2.38(0.09)	+	5.01(0.45)	+	3.18(0.23)	+
	E_B	**0.62(0.13)**	1.78(0.22)	+	2.08(0.19)	+	0.98(0.06)	+	3.86(0.45)	+	1.47(0.20)	+
120	E_o	**1.77(0.12)**	3.59(0.19)	+	3.25(0.19)	+	2.55(0.06)	+	5.13(0.40)	+	3.00(0.22)	+
	E_B	**0.62(0.10)**	1.68(0.18)	+	2.18(0.19)	+	1.07(0.05)	+	3.82(0.37)	+	1.34(0.20)	+
150	E_o	**1.74(0.10)**	3.46(0.26)	+	3.29(0.35)	+	2.66(0.07)	+	4.54(0.28)	+	3.01(0.18)	+
	E_B	**0.67(0.06)**	1.57(0.22)	+	2.19(0.32)	+	1.22(0.05)	+	3.38(0.29)	+	1.33(0.15)	+
200	E_o	**1.72(0.11)**	3.24(0.13)	+	3.27(0.22)	+	2.49(0.15)	+	4.28(0.45)	+	2.76(0.17)	+
	E_B	**0.73(0.08)**	1.47(0.10)	+	2.16(0.23)	+	1.17(0.09)	+	3.12(0.39)	+	1.18(0.14)	+
Total		+	20		15		12		20		20	
		=	0		2		1		0		0	
		−	0		3		7		0		0	
Best		**13**	0		0		7		0		0	
AvgRank		**1.55**	4.85		3.60		1.65		5.85		3.50	

Table 4
Results of all algorithms on MPB instances with different shift lengths.

P	s_X	Error	DPSO	CPSO		CPSOR		AMP/PSO		DDEBQ		NLPSO	
50	1	E_o	**2.21(0.24)**	4.45(0.42)	+	2.93(0.26)	+	2.42(0.07)	+	6.34(0.73)	+	3.81(0.37)	+
		E_B	**0.74(0.22)**	2.09(0.34)	+	1.88(0.27)	+	0.77(0.05)	+	4.95(0.67)	+	1.67(0.28)	+
	2	E_o	**2.49(0.22)**	5.73(0.57)	+	3.92(0.25)	+	3.35(0.12)	+	7.26(0.75)	+	4.92(0.40)	+
		E_B	**0.72(0.20)**	2.38(0.35)	+	2.16(0.22)	+	1.19(0.06)	+	5.27(0.75)	+	1.82(0.32)	+
	3	E_o	**2.80(0.31)**	6.11(0.55)	+	4.24(0.22)	+	3.97(0.10)	+	7.76(0.65)	+	5.59(0.45)	+
		E_B	**0.81(0.21)**	2.32(0.41)	+	2.13(0.22)	+	1.32(0.10)	+	5.30(0.70)	+	1.84(0.30)	+
	4	E_o	**2.93(0.30)**	6.57(0.71)	+	4.77(0.31)	+	4.39(0.16)	+	8.44(0.63)	+	6.38(0.38)	+
		E_B	**0.79(0.20)**	2.24(0.38)	+	2.32(0.27)	+	1.56(0.15)	+	5.40(0.61)	+	2.07(0.26)	+
	5	E_o	**3.29(0.31)**	6.97(0.61)	+	4.70(0.18)	+	4.75(0.13)	+	9.19(0.66)	+	6.86(0.40)	+
		E_B	**0.96(0.19)**	2.28(0.40)	+	2.00(0.14)	+	1.61(0.10)	+	5.71(0.66)	+	2.01(0.32)	+
150	1	E_o	**1.74(0.10)**	3.46(0.26)	+	3.29(0.35)	+	2.66(0.07)	+	4.54(0.28)	+	3.01(0.18)	+
		E_B	**0.67(0.06)**	1.57(0.22)	+	2.19(0.32)	+	1.22(0.05)	+	3.38(0.29)	+	1.33(0.15)	+
	2	E_o	**2.21(0.17)**	4.16(0.30)	+	3.99(0.39)	+	3.30(0.09)	+	5.31(0.34)	+	3.82(0.19)	+
		E_B	**0.87(0.15)**	1.62(0.21)	+	2.18(0.33)	+	1.34(0.06)	+	3.64(0.31)	+	1.42(0.16)	+
	3	E_o	**2.42(0.12)**	4.66(0.24)	+	4.82(0.28)	+	3.55(0.10)	+	5.83(0.43)	+	4.33(0.25)	+
		E_B	**0.91(0.08)**	1.67(0.17)	+	2.43(0.28)	+	1.39(0.06)	+	3.65(0.38)	+	1.42(0.19)	+
	4	E_o	**2.69(0.18)**	5.04(0.26)	+	5.10(0.46)	+	3.85(0.11)	+	6.43(0.46)	+	4.90(0.26)	+
		E_B	**1.03(0.10)**	1.70(0.20)	+	2.45(0.44)	+	1.43(0.09)	+	3.83(0.37)	+	1.55(0.20)	+
	5	E_o	**2.79(0.13)**	5.35(0.28)	+	5.42(0.42)	+	4.02(0.13)	+	6.99(0.37)	+	5.28(0.24)	+
		E_B	**1.00(0.09)**	1.71(0.22)	+	2.62(0.34)	+	1.41(0.09)	+	3.93(0.35)	+	1.54(0.15)	+
Total			+	20		20		20		20		20	
			=	0		0		0		0		0	
			−	0		0		0		0		0	
Best			**20**	0		0		0		0		0	
AvgRank			**1.00**	4.55		4.10		2.05		6.00		3.30	

The results of DPSO and the compared algorithms are reported in Table 5.

From Table 5, it can be seen that the proposed DPSO performs significantly better than each compared algorithm on almost all

Table 5
Results of all algorithms on MPB with different correlated parameters.

P	λ	Error	DPSO	CPSO		CPSOR		AMP/PSO		DDEBQ		NLPSO	
50	0	E_o	**2.21(0.24)**	4.45(0.42)	+	2.93(0.26)	+	2.42(0.07)	+	6.34(0.73)	+	3.81(0.37)	+
		E_B	**0.74(0.22)**	2.09(0.34)	+	1.88(0.27)	+	0.77(0.05)	+	4.95(0.67)	+	1.67(0.28)	+
	0.2	E_o	**2.29(0.19)**	4.53(0.51)	+	3.00(0.24)	+	2.39(0.11)	+	6.23(0.78)	+	3.89(0.23)	+
		E_B	**0.77(0.14)**	2.11(0.39)	+	1.94(0.26)	+	0.84(0.06)	=	4.85(0.70)	+	1.69(0.19)	+
	0.4	E_o	**2.16(0.28)**	4.34(0.47)	+	3.12(0.34)	+	2.39(0.10)	+	6.34(0.55)	+	4.11(0.40)	+
		E_B	**0.72(0.21)**	1.96(0.36)	+	2.02(0.35)	+	0.89(0.09)	+	4.94(0.52)	+	1.79(0.35)	+
	0.6	E_o	**2.37(0.31)**	4.48(0.45)	+	3.00(0.28)	+	2.82(0.09)	+	6.32(0.63)	+	4.16(0.37)	+
		E_B	**0.84(0.24)**	2.14(0.43)	+	1.98(0.30)	+	1.06(0.10)	+	4.89(0.61)	+	1.75(0.34)	+
	0.8	E_o	**2.23(0.25)**	4.33(0.27)	+	2.53(0.26)	+	2.68(0.12)	+	6.15(0.69)	+	4.40(0.44)	+
		E_B	**0.71(0.25)**	2.03(0.26)	+	1.66(0.25)	+	1.18(0.08)	+	4.74(0.63)	+	1.87(0.33)	+
	1.0	E_o	**2.19(0.21)**	4.50(0.59)	+	2.73(0.30)	+	3.14(0.17)	+	6.36(0.75)	+	4.29(0.48)	+
		E_B	**0.66(0.15)**	2.12(0.50)	+	1.79(0.28)	+	1.39(0.18)	+	4.93(0.74)	+	1.86(0.43)	+
150	0	E_o	**1.74(0.10)**	3.46(0.26)	+	3.29(0.35)	+	2.66(0.07)	+	4.54(0.28)	+	3.01(0.18)	+
		E_B	**0.67(0.06)**	1.57(0.22)	+	2.19(0.32)	+	1.22(0.05)	+	3.38(0.29)	+	1.33(0.15)	+
	0.2	E_o	**1.78(0.13)**	3.49(0.25)	+	3.25(0.35)	+	2.71(0.08)	+	4.69(0.44)	+	3.06(0.20)	+
		E_B	**0.71(0.10)**	1.61(0.22)	+	2.15(0.33)	+	1.22(0.04)	+	3.51(0.45)	+	1.31(0.19)	+
	0.4	E_o	**1.75(0.12)**	3.37(0.25)	+	3.24(0.32)	+	2.67(0.13)	+	4.70(0.47)	+	3.14(0.22)	+
		E_B	**0.67(0.11)**	1.53(0.20)	+	2.15(0.30)	+	1.21(0.08)	+	3.49(0.42)	+	1.30(0.16)	+
	0.6	E_o	**1.77(0.13)**	3.33(0.18)	+	3.16(0.28)	+	2.57(0.10)	+	4.73(0.40)	+	3.38(0.27)	+
		E_B	**0.68(0.11)**	1.49(0.15)	+	2.12(0.25)	+	1.23(0.08)	+	3.48(0.33)	+	1.43(0.22)	+
	0.8	E_o	**1.71(0.10)**	3.42(0.21)	+	3.06(0.27)	+	2.73(0.11)	+	4.57(0.36)	+	3.59(0.28)	+
		E_B	**0.65(0.09)**	1.58(0.18)	+	2.10(0.23)	+	1.25(0.09)	+	3.40(0.31)	+	1.58(0.24)	+
	1.0	E_o	**1.74(0.14)**	3.49(0.23)	+	3.08(0.34)	+	2.69(0.09)	+	4.52(0.34)	+	3.47(0.32)	+
		E_B	**0.68(0.10)**	1.57(0.18)	+	2.09(0.33)	+	1.23(0.09)	+	3.32(0.28)	+	1.53(0.26)	+
Total			+	24		24		23		24		24	
			=	0		0		1		0		0	
			−	0		0		0		0		0	
Best			**20**	0		0		0		0		0	
AvgRank			**1.00**	4.54		3.79		2.04		6.00		3.62	

instances in terms of both metrics E_o and E_B. The results show that the performance of the proposed DPSO is not sensitive to the pattern of the environment changes. The reason may be due to that the exploitation and exploration components do not depend on the real movement directions of changes and can track the moving peaks no matter whether the environmental factors change randomly or following a certain pattern. Thus, the proposed DPSO can handle MPB problems with different correlated parameters.

4.3.2.4. MPB with different change frequencies. In this subsection, multiple MPB instances with different change frequencies are constructed to test the response ability of the algorithms. The change frequency u is set as 3000, 5000, and 10 000. The number of peaks is set as two values, i.e., 50 and 150, the shift length is fixed as 1, and the correlated parameter is set as 0. The results of DPSO and the compared algorithms are reported in Table 6.

From Table 6, it can be seen that the proposed DPSO can find better results in most instances. As far as the AvgRank is concerned, the proposed DPSO is slightly worse than the AMP/PSO because of its poor performance on the one instance with $P = 150$ and $u = 3000$. However, the DPSO becomes better on the other instances and it performs the best in 7 cases. When considering the metrics E_o and E_B simultaneously, the proposed DPSO performs significantly better than CPSO, CPSOR, AMP/PSO, DDEBQ, and NLPSO in 10, 10, 6, 10, and 10 out of 12 cases, respectively, while worse in only 2, 2, 4, 1, and 2 cases. Thus, the DPSO can perform well on MPB instances with different change frequencies.

4.3.2.5. Convergence behavior on MPB instances. To analyze the convergence behavior of the algorithms on MPB instances, we take two MPB instances with $P = 50$ and 100 as examples. In the two instances, we set $s_X = 1$, $\lambda = 0$, and $u = 5000$. The convergence curves (online errors) of the algorithms along 10 environments are illustrated in Fig. 3.

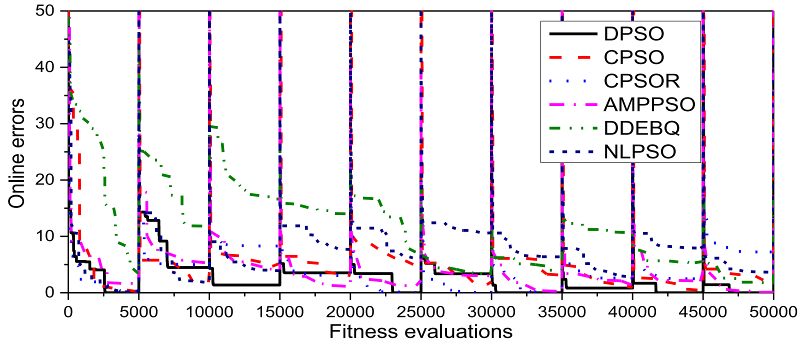
It can be seen that the DPSO converges faster to better solutions in most environments. In the first environment, the exploration component enables the DPSO to find better solutions than all compared algorithms. In the early stages of the following environments, taking advantage of the solutions found in the previous environments, the exploitation component in DPSO can quickly find better solutions afterwards. In the later stages, the DPSO may stagnate for a short time but its convergence curve is always able to drop finally since the exploration component helps to find new subareas. It is interesting to find that the DPSO converges to better solutions more quickly, especially in the later environments, i.e., the last four environments in Fig. 3(a) and the last three environments in Fig. 3(b). This is because that the archives have accumulated more search experience as time goes by, which can better assist the optimization of new environments. Generally, the DPSO can converge fast in dynamic environments.

5. Discussion

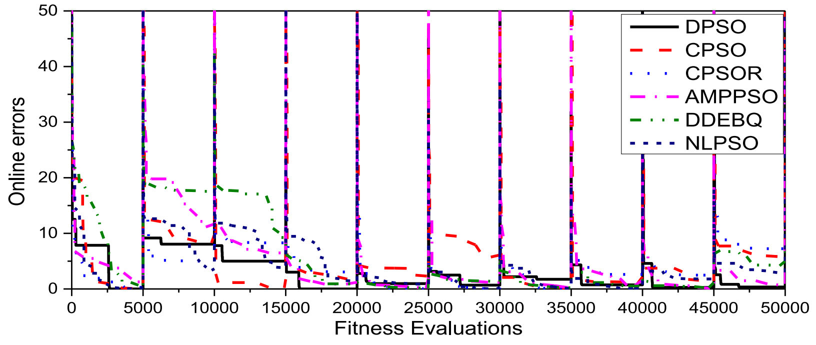
From the experiments in Section 4, we can see that the proposed algorithm can deal with multimodal DOPs and performs well on most tested instances, especially when there are large numbers of peaks. This is because that the reuse of the solutions in the two archives greatly improves the search efficiency in

Table 6
Results of all algorithms on MPB with different change frequencies.

P	*u*	Error	DPSO	CPSO		CPSOR		AMP/PSO		DDEBQ		NLPSO	
50	3 000	E_o	3.13(0.28)	5.85(0.52)	+	3.73(0.27)	+	**3.10(0.11)**	=	7.41(0.62)	+	5.37(0.38)	+
		E_B	**1.17(0.23)**	3.28(0.38)	+	2.12(0.24)	+	1.28(0.08)	=	6.01(0.59)	+	2.95(0.36)	+
	5 000	E_o	**2.21(0.24)**	4.45(0.42)	+	2.93(0.26)	+	2.42(0.07)	+	6.34(0.73)	+	3.81(0.37)	+
		E_B	**0.74(0.22)**	2.09(0.34)	+	1.88(0.27)	+	0.77(0.05)	+	4.95(0.67)	+	1.67(0.28)	+
	10 000	E_o	1.67(0.22)	3.15(0.29)	+	2.92(0.30)	+	**1.49(0.05)**	–	5.47(0.65)	+	2.55(0.27)	+
		E_B	0.60(0.19)	1.60(0.25)	+	1.75(0.31)	+	**0.34(0.04)**	–	4.41(0.61)	+	1.31(0.26)	+
150	3 000	E_o	6.16(0.64)	4.53(0.26)	–	3.96(0.32)	–	**3.46(0.11)**	–	5.84(0.43)	=	4.15(0.19)	–
		E_B	5.00(0.60)	2.40(0.19)	–	2.34(0.29)	–	**1.65(0.06)**	–	4.59(0.40)	–	2.20(0.13)	–
	5 000	E_o	**1.74(0.10)**	3.46(0.26)	+	3.29(0.35)	+	2.66(0.07)	+	4.54(0.28)	+	3.01(0.18)	+
		E_B	**0.67(0.06)**	1.57(0.22)	+	2.19(0.32)	+	1.22(0.05)	+	3.38(0.29)	+	1.33(0.15)	+
	10 000	E_o	**1.29(0.13)**	2.40(0.15)	+	3.24(0.32)	+	1.91(0.05)	+	3.79(0.27)	+	2.00(0.17)	+
		E_B	**0.45(0.10)**	1.20(0.15)	+	2.04(0.29)	+	0.73(0.05)	+	2.91(0.22)	+	1.08(0.16)	+
Total			+	10		10		6		10		10	
			=	0		0		2		1		0	
			–	2		2		4		1		2	
Best			**7**	0		0		5		0		0	
AvgRank			2.08	4.50		3.83		**1.58**		5.83		3.16	



(a) Online errors of the 6 algorithms over fitness evaluations on MPB with $P = 50$, $s_X = 1$, $\lambda = 0$, $u = 5000$.



(b) Online errors of the 6 algorithms over fitness evaluations on MPB with $P = 150$, $s_X = 1$, $\lambda = 0$, $u = 5000$.

Fig. 3. Convergence curves of DPSO, CPSO, CPSOR, AMP/PSO, DDEBQ, and NLPSO.

dynamic environments. The coarse-grained archive is beneficial to locate different subareas in the search space and reduce the risk of blind search. However, when the number of peaks is small, the proposed algorithm performs slightly worse than some other algorithms. This may be due to the dispersion of the computational resources on the exploration component, which further results in the incomplete local search of the solutions in the fine-grained archive. Indeed, when there are a small number of peaks, it is easier to track all the peaks and the fine-grained archive is likely to include all the peak positions. In such cases, more computational resources should be leaned towards the exploitation component. In general, the proposed algorithm has some weaknesses on DOPs with small numbers of peaks but performs well on DOPs with large numbers of peaks.

6. Conclusion and future work

In DOPs, the environment dynamically changes through time, but a new environment is often related to its past environments. Motivated by this, this paper proposes to combine the exploitation of historical environments with the exploration of new environments, and develops a new dual-archive-based particle swarm optimization (DPSO) method. It adopts two archives to store detailed and systemic information that will together assist the optimization in new environments. Experiments are performed on the famous Moving Peaks Benchmark and the algorithms are evaluated on two metrics concerning of convergence speed and solution quality, respectively. The experimental results show that, compared with several typical state-of-the-art algorithms, the proposed DPSO can find better solutions.

In future work, we will study new strategies for resource assignment to better balance exploitation and exploration. Also, we will investigate the performance of evolutionary algorithms on large-scale DOPs.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.asoc.2019.105876.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (61773410, 61673403).

References

[1] J.M. Moore, M. Small, Estimating dynamical dimensions from noisy observations, Inform. Sci. 462 (2018) 55–75, http://dx.doi.org/10.1016/j.ins.2018.06.011.

[2] H. Modares, F.L. Lewis, W. Kang, A. Davoudi, Optimal synchronization of heterogeneous nonlinear systems with unknown dynamics, IEEE Trans. Automat. Control 63 (1) (2018) 117–131, http://dx.doi.org/10.1109/TAC.2017.2713339.

[3] Y. Park, N. Paine, S. Oh, Development of force observer in series elastic actuator for dynamic control, IEEE Trans. Ind. Electron. 65 (3) (2018) 2398–2407, http://dx.doi.org/10.1109/TIE.2017.2745457.

[4] P. Zeng, H. Li, H. He, S. Li, Dynamic energy management of a microgrid using approximate dynamic programming and deep recurrent neural network learning, IEEE Trans. Smart Grid 10 (4) (2019) 4435–4445, http://dx.doi.org/10.1109/TSG.2018.2859821.

[5] J. Shi, Z. Yang, H. Xu, M. Chen, B. Champagne, Dynamic resource allocation for lte-based vehicle-to-infrastructure networks, IEEE Trans. Veh. Technol. 68 (5) (2019) 5017–5030, http://dx.doi.org/10.1109/TVT.2019.2903822.

[6] D.-C. Dang, T. Jansen, P.K. Lehre, Populations can be essential in dynamic optimisation, in: Proc. Genet. Evol. Comput. Conf., 2015, pp. 1407–1414. http://dx.doi.org/10.1145/2739480.2754808.

[7] A. Prakasam, N. Savarimuthu, Novel local restart strategies with hyper-populated ant colonies for dynamic optimization problems, Neural Comput. Appl. 31 (Suppl. 1) (2019) 63–76, http://dx.doi.org/10.1007/s00521-018-3638-3.

[8] R. Dang-Nhu, T. Dardinier, B. Doerr, G. Izacard, D. Nogneng, A new analysis method for evolutionary optimization of dynamic and noisy objective functions, in: Proc. Genet. Evol. Comput. Conf., Kyoto, Japan, 2018, pp. 1467–1474. http://dx.doi.org/10.1145/3205455.3205563.

[9] X.-F. Liu, Z.-H. Zhan, J. Zhang, Neural network for change direction prediction in dynamic optimization, IEEE Access 6 (2018) 72649–72662, http://dx.doi.org/10.1109/ACCESS.2018.2881538.

[10] A.G. Bari, A. Gaspar, DynTLBO-a teaching learning-based dynamic optimization algorithm, IEEE Congr. Evol. Comput. (2018) 1–8, http://dx.doi.org/10.1109/CEC.2018.8477702.

[11] S. Das, A. Mandal, R. Mukherjee, An adaptive differential evolution algorithm for global optimization in dynamic environments, IEEE Trans. Cybern. 44 (6) (2014) 966–978, http://dx.doi.org/10.1109/TCYB.2013.2278188.

[12] M. Mavrovouniotis, C. Li, S. Yang, A survey of swarm intelligence for dynamic optimization: algorithms and applications, Swarm Evol. Comput. 33 (2017) 1–17, http://dx.doi.org/10.1016/j.swevo.2016.12.005.

[13] H. Nakano, M. Kojima, A. Miyauchi, An artificial bee colony algorithm with a memory scheme for dynamic optimization problems, in: Proc. IEEE Congr. Evol. Comput., 2015, pp. 2657–2663. http://dx.doi.org/10.1109/CEC.2015.7257217.

[14] X. Yu, X. Wu, A multi-point local search algorithm for continuous dynamic optimization, in: Proc. IEEE Congr. Evol. Comput., 2016, pp. 2736–2743. http://dx.doi.org/10.1109/CEC.2016.7744134.

[15] L. Cao, L. Xu, E.D. Goodman, A neighbor-based learning particle swarm optimizer with short-term and long-term memory for dynamic optimization problems, Inform. Sci. 453 (2018) 463–485, http://dx.doi.org/10.1016/j.ins.2018.04.056.

[16] C. Li, T.T. Nguyen, M. Yang, M. Mavrovouniotis, S. Yang, An adaptive multipopulation framework for locating and tracking multiple optima, IEEE Trans. Evol. Comput. 20 (4) (2016) 590–605, http://dx.doi.org/10.1109/TEVC.2015.2504383.

[17] S.K. Nseef, S. Abdullah, A. Turky, G. Kendall, An adaptive multi-population artificial bee colony algorithm for dynamic optimisation problems, Knowl.-Based Syst. 104 (2016) 14–23, http://dx.doi.org/10.1016/j.knosys.2016.04.005.

[18] P. Rakshit, A. Konar, S. Das, Noisy evolutionary optimization algorithms — A comprehensive survey, Swarm Evol. Comput. 33 (2017) 18–45, http://dx.doi.org/10.1016/j.swevo.2016.09.002.

[19] J. Branke, Memory enhanced evolutionary algorithms for changing optimization problems, in: Proc. IEEE Congr. Evol. Comput., 1999, pp. 1875–1882. http://dx.doi.org/10.1109/CEC.1999.785502.

[20] Y.G. Woldesenbet, G.G. Yen, Dynamic evolutionary algorithm with variable relocation, IEEE Trans. Evol. Comput. 13 (3) (2009) 500–513, http://dx.doi.org/10.1109/TEVC.2008.2009031.

[21] S.A.G. Van der Stockt, A.P. Engelbrecht, Analysis of selection hyper-heuristics for population-based meta-heuristics in real-valued dynamic optimization, Swarm Evol. Comput. 43 (2018) 127–146, http://dx.doi.org/10.1016/j.swevo.2018.03.012.

[22] J. Lewis, E. Hart, G. Ritchie, A comparison of dominance mechanisms and simple mutation on non-stationary problems, in: Proc. 4th Int. Conf. Parallel Problem Solving From Nature, 1998, pp. 139–148. https://doi.org/10.1007/BFb0056857.

[23] D. Yazdani, J. Branke, M.N. Omidvar, T.T. Nguyen, X. Yao, Changing or keeping solutions in dynamic optimization problems with switching costs, in: Proc. Genet. Evol. Comput. Conf., 2018, pp. 1095–1102. http://dx.doi.org/10.1145/3205455.3205484.

[24] S.X. Yang, X. Yao, Population-based incremental learning with associative memory for dynamic environments, IEEE Trans. Evol. Comput. 12 (5) (2008) 542–561, http://dx.doi.org/10.1109/TEVC.2007.913070.

[25] Z. Zhu, L. Chen, C. Yuan, C. Xia, Global replacement-based differential evolution with neighbor-based memory for dynamic optimization, Appl. Intell. 48 (10) (2018) 3280–3294, http://dx.doi.org/10.1007/s10489-018-1147-9.

[26] T. Zhu, W. Luo, L. Yue, Combining multipopulation evolutionary algorithms with memory for dynamic optimization problems, in: Proc. IEEE Congr. Evol. Comput., 2014, pp. 2047–2054. http://dx.doi.org/10.1109/CEC.2014.6900492.

[27] M. Mavrovouniotis, F. Neri, S. Yang, An adaptive local search algorithm for real-valued dynamic optimization, in: Proc. IEEE Congr. Evol. Comput., 2015, pp. 1388–1395. http://dx.doi.org/10.1109/CEC.2015.7257050.

[28] W. Luo, J. Sun, C. Bu, H. Liang, Species-based Particle Swarm Optimizer enhanced by memory for dynamic optimization, Appl. Soft Comput. 47 (2016) 130–140, http://dx.doi.org/10.1016/j.asoc.2016.05.032.

[29] S. Yang, Memory-based immigrants for genetic algorithms in dynamic environments, in: Proc. Genet. Evol. Comput., 2005, pp. 1115–1122. http://dx.doi.org/10.1145/10680091068196.

[30] C. Cruz, J.R. González, D.A. Pelta, Optimization in dynamic environments: a survey on problems, methods and measures, Soft Comput. 15 (7) (2011) 1427–1448, http://dx.doi.org/10.1007/s00500-010-0681-0.

[31] J.K. Kordestani, H.A. Firouzjaee, M. Reza Meybodi, An adaptive bi-flight cuckoo search with variable nests for continuous dynamic optimization problems, Appl. Intell. 48 (1) (2017) 97–117, http://dx.doi.org/10.1007/s10489-017-0963-7.

[32] X. Li, K.H. Dam, Comparing particle swarms for tracking extrema in dynamic environments, in: Proc. IEEE Congr. Evol. Comput., 2003, pp. 1772–1779. http://dx.doi.org/10.1109/CEC.2003.1299887.

[33] S. Janson, M. Middendorf, A hierarchical particle swarm optimizer for noisy and dynamic environments, Genet. Program. Evol. Mach. 7 (4) (2006) 329–354, http://dx.doi.org/10.1007/s10710-006-9014-6.

[34] T.M. Blackwell, P.J. Bentley, Dynamic search with charged swarms, in: Proc. Genet. and Evol. Comput., 2002, pp. 19–26.

[35] L. Liu, S. Yang, D. Wang, Particle swarm optimization with composite particles in dynamic environments, IEEE Trans. Evol. Comput. 40 (6) (2010) 1634–1648, http://dx.doi.org/10.1109/TSMCB.2010.2043527.

[36] R. Mendes, A.S. Mohais, DynDE: a differential evolution for dynamic optimization problems, in: Proc. IEEE Congr. Evol. Comput., 2005, pp. 2808–2815. http://dx.doi.org/10.1109/CEC.2005.1555047.

[37] F.B. Ozsoydan, A. Baykasoğlu, Quantum firefly swarms for multimodal dynamic optimization problems, Expert Syst. Appl. 115 (2019) 189–199, http://dx.doi.org/10.1016/j.eswa.2018.08.007.

[38] R.-I. Chang, H.-M. Hsu, S.-Y. Lin, C.-C. Chang, J.-M. Ho, Query-based learning for dynamic particle swarm optimization, IEEE Access 5 (2017) 7648–7658, http://dx.doi.org/10.1109/access.2017.2694843.

[39] D. Yazdani, M.N. Omidvar, J. Branke, T.T. Nguyen, X. Yao, Scaling up dynamic optimization problems: a divide-and-conquer approach, IEEE Trans. Evol. Comput. (2019) http://dx.doi.org/10.1109/TEVC.2019.2902626.

[40] W. Luo, R. Yi, B. Yang, P. Xu, Surrogate-assisted evolutionary framework for data-driven dynamic optimization, IEEE Trans. Emerg. Top. Comput. Intell. 3 (2) (2019) 137–150, http://dx.doi.org/10.1109/TETCI.2018.2872029.

[41] T.M. Blackwell, P. Bentley, Don't push me! Collision-avoiding swarms, in: Proc. IEEE Congr. Evol. Comput., 2002, pp. 1691–1696. http://dx.doi.org/10.1109/CEC.2002.1004497.

[42] H. Ben-Romdhane, E. Alba, S. Krichen, A new evolutionary approach using pre-post testing to trigger exploration and exploitation in DOPs, in: Proc. Genet. Evol. Comput. Conf., Berlin, Germany, 2017, pp. 99–100. http://dx.doi.org/10.1145/3067695.3076022.

[43] R. Lung, D. Dumitrescu, Collaborative evolutionary swarm optimization with a gauss chaotic sequence generator, in: Innovations in Hybrid Intelligent Systems, in: Advances in Soft Computing, 44, 2007, pp. 207–214, http://dx.doi.org/10.1007/978-3-540-74972-1_28.

[44] A. Turky, S. Abdullah, A. Dawod, A dual-population multi operators harmony search algorithm for dynamic optimization problems, Comput. Ind. Eng. 117 (2018) 19–28, http://dx.doi.org/10.1016/j.cie.2018.01.003.

[45] 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, http://dx.doi.org/10.1109/TEVC.2005.859468.

[46] I. Schoeman, A. Engelbrecht, Niching for dynamic environments using particle swarm optimization, in: T.-D. Wang, et al. (Eds.), Simulated Evolution Learning, in: LNCS, 4247, Springer, Berlin, Germany, 2006, pp. 134–141, http://dx.doi.org/10.1007/11903697_18.

[47] T. Weise, S. Niemczyk, R. Chiong, M. Wan, A framework for multi-model EDAs with model recombination, in: Applications of Evolutionary Computation, EvoApplications 2011, in: Lecture Notes in Computer Science, 6624, 2011, pp. 304–313, http://dx.doi.org/10.1007/978-3-642-20525-5_31.

[48] W. Zhang, W. Zhang, G.G. Yen, H. Jing, A cluster-based clonal selection algorithm for optimization in dynamic environment, Swarm Evol. Comput. (2018) http://dx.doi.org/10.1016/j.swevo.2018.10.005, in press.

[49] 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, http://dx.doi.org/10.1109/TEVC.2010.2046667.

[50] U. Halder, S. Das, D. Maity, A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments, IEEE Trans. Cybern. 43 (3) (2013) 881–897, http://dx.doi.org/10.1109/TSMCB.2012.2217491.

[51] M.C. du Plessis, A.P. Engelbrecht, Differential evolution for dynamic environments with unknown numbers of optima, J. Global Optim. 55 (1) (2013) 73–99, http://dx.doi.org/10.1007/s10898-012-9864-9.

[52] C. Li, S. Yang, M. Yang, An adaptive multi-swarm optimizer for dynamic optimization problems, Evol. Comput. 22 (4) (2014) 559–594, http://dx.doi.org/10.1162/EVCO_a_00117.

[53] J.K. Kordestani, A.E. Ranginkaman, M.R. Meybodi, P. Novoa-Hernández, A novel framework for improving multi-population algorithms for dynamic optimization problems: A scheduling approach, Swarm Evol. Comput. 44 (2019) 788–805, http://dx.doi.org/10.1016/j.swevo.2018.09.002.

[54] M. Kamosi, A.B. Hashemi, M.R. Meybodi, A hibernating multi-swarm optimization algorithm for dynamic environemts, in: Proc. World Congr. Nat. Biol. Inspir. Comput., 2010, pp. 363–369. http://dx.doi.org/10.1109/NABIC.2010.5716372.

[55] R. Liaw, C. Ting, Incorporating fitness inheritance and k-nearest neighbors for evolutionary dynamic optimization, IEEE Congr. Evol. Comput. (2018) http://dx.doi.org/10.1109/CEC.2018.8477703.

[56] T. Blackwell, J. Branke, Multiswarms, exclusion, and anti-convergence in dynamic environments, IEEE Trans. Evol. Comput. 10 (4) (2006) 459–472, http://dx.doi.org/10.1109/TEVC.2005.857074.

[57] H.H. Rosenbrock, Some general implicit processes for the numerical solution of differential equations, Comput. J. 5 (4) (1963) 329–330, http://dx.doi.org/10.1093/comjnl/5.4.329.

[58] G. Arfken, Gram–Schmidt orthogonalization, in: Mathematical Methods for Physicists, third ed., Academic Press, Orlando, FL, 1985, pp. 516–520.

[59] S. Hui, P.N. Suganthan, Ensemble and arithmetic recombination-based speciation differential evolution for multimodal optimization, IEEE Trans. Cybernet. 46 (1) (2016) 64–74, http://dx.doi.org/10.1109/TCYB.2015.2394466.

[60] 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, http://dx.doi.org/10.1109/TEVC.2011.2169966.