AAAS
Research
Volume 2020, Article ID 1762107, 13 pages
https://doi.org/10.34133/2020/1762107

Research Article

Flexible Wolf Pack Algorithm for Dynamic Multidimensional Knapsack Problems

Husheng Wu[1] **and Renbin Xiao**[2]

[1]*School of Equipment Management and Support, Armed Police Force Engineering University, Xi'an 710086, China*
[2]*School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan 430074, China*

Correspondence should be addressed to Renbin Xiao; rbxiao@hust.edu.cn

Received 4 October 2019; Accepted 18 January 2020; Published 18 February 2020

Optimization problems especially in a dynamic environment is a hot research area that has attracted notable attention in the past decades. It is clear from the dynamic optimization literatures that most of the efforts have been devoted to continuous dynamic optimization problems although the majority of the real-life problems are combinatorial. Moreover, many algorithms shown to be successful in stationary combinatorial optimization problems commonly have mediocre performance in a dynamic environment. In this study, based on binary wolf pack algorithm (BWPA), combining with flexible population updating strategy, a flexible binary wolf pack algorithm (FWPA) is proposed. Then, FWPA is used to solve a set of static multidimensional knapsack benchmarks and several dynamic multidimensional knapsack problems, which have numerous practical applications. To the best of our knowledge, this paper constitutes the first study on the performance of WPA on a dynamic combinatorial problem. By comparing two state-of-the-art algorithms with the basic BWPA, the simulation experimental results demonstrate that FWPA can be considered as a feasibility and competitive algorithm for dynamic optimization problems.

1. Introduction

Most research in evolutionary computation focuses on static problems where the entire problem-related data remains stationary through optimization procedure [1–3]. However, numerous real-world optimization problems arising from the uncertainty of future events indeed have a dynamic nature. Changes in dynamic optimization problems (DOPs) may occur in the decision variables, constraints, and objective function [4, 5]. This requires optimization algorithms to not only detect and respond to the change of optima as quickly as possible but also keep track of the changing optima dynamically. Hence, the capability of continuously adapting the solution to a changing environment is necessary for optimization approaches [3, 6]. Therefore, DOPs are more challenging to address than stationary optimization problems.

DOPs can be generally divided into two major fields as combinatorial and continuous [7–9]. Typical combinatorial DOPs include dynamic travelling salesman problem (DTSP) [10], dynamic vehicle routing problem (DVRP) [11], dynamic job-shop scheduling problem (DJSSP) [12], and dynamic knapsack problem (DKP) [13–15]. In fact, many practical problems can be abstracted as a specific type of dynamic multidimensional knapsack problem (DMPK) when multiple dynamic constraints are needed to be tackled, such as task allocation, investment decision, cargo loading, and budget management [9, 16]. Given their wild application and complexity, DMKPs have important theoretical and practical value. Evolutionary algorithms (EAs) and swarm intelligence-based algorithms are expected to perform well on solving both combinatorial and continuous DOPs since evolutionary dynamics in nature also take place in a highly uncertain environment [8, 17, 18].

Wolf pack algorithm (WPA) [19] is a relatively new and promising member of swarm intelligence-based algorithms that model the cooperative hunting behavior of wolf pack. It has been proved an efficient optimizer for solving many nonlinear and complex optimization problems by successful applications in image processing [20], power system control [21], robot path planning [22], and static MKPs [23]. Many derivative versions of WPA also have been designed for solving different problems, such as binary WPA (BWPA) for 0-1

ordinary knapsack problem [24], improved binary WPA (IBWPA) for MKPs [23], and discrete WPA (DWPA) for TSP [25]. In [26], an integer coding wolf pack algorithm (ICWPA) is proposed to cope with the combat task allocation problems of aerial swarm. In [27], the improved WPA (IWPA) is proposed to solve VRP. Despite its high efficiency of binary WPA (BWPA) in solving static MKPs, WPA has not been introduced into the area of DMKPs.

The key issue of handling DOPs using EAs is how to avoid population diversity loss problem and maintain population diversity while tracking the changing global optima [5, 8, 9, 28]. In this regard, a flexible population updating strategy which is capable of introducing and maintaining diversity during execution is designed for BWPA to address the DMKPs in this study. Moreover, the flexible population updating strategy that generates new individuals by making use of the memory of previously found good solutions can be viewed as an explicit memory scheme [29, 30].

Compared with static extensions, there are relatively far less reported publications about DMKPs. It is necessary to develop new solution approaches for addressing DMKPs more efficiently as DMKPs have numerous practical implications. This is one of the main motivations of this study. Secondly, to the best of our knowledge, this is the first study that investigates the performance of BWPA and its improved version (as proposed in this paper) on MKPs in dynamic environments.

The reminder of this paper is arranged as follows: Section 2 provides the literature review and related concepts of DMKPs. The original BWPA and its variant FWPA are discussed in detail in Section 3. While Section 4 conducts the simulation experiment and analyzes the results. Finally, conclusions and some future research issue are given in Section 5.

2. Problem Definition and Related Work

In this section, we outline the necessary concepts of DMKPs and overview the related work about the MKPs in dynamic environments.

2.1. Definition of the Dynamic Multidimensional Knapsack Problem. MKP is a NP-hard problem and has been wildly used as a combinatorial benchmark problem of EAs and swarm intelligence-based algorithms [31, 32]. The MKP depends on the values of the profits p_j, resource consumptions w_{kj}, and the resource constraints c_k. As the generalization of the ordinary knapsack problem, MKP is more representative of real-world scenarios because multiple constrains are concerned [33]. The static MKP can be generally formulated as follows [34]:

$$\max f = \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{s.t.} \begin{cases} \sum_{j=1}^{n} w_{kj} x_j \le c_k, & k \in \mathbf{M} = \{0, 1, \cdots, m\} \\ x_j = \{0, 1\}, & j \in \mathbf{N} = \{0, 1, \cdots, n\}, \end{cases} \tag{2}$$

where n is the number of items and m is the number of knapsack constrains with capacities c_k for $k = 1, 2, \cdots, m$. Each item $j \in \boldsymbol{N}$ requires w_{kj} units of resource consumption in the k^{th} knapsack and yields p_j units of profit upon inclusion. The goal of MKP is to find a subset of all items that yield maximum profit without exceeding the multidimensional resource capacities [34]. All entries are naturally nonnegative. More precisely, without loss of generality, it can be assumed that the following constraints, as defined by (3), are satisfied. If this is not the case, one or more variables could be fixed to 0 or 1.

$$\max \left\{ w_{kj} : j \in \mathbf{N} \right\} \le c_k < \sum_{j}^{n} w_{kj}, \quad \forall k \in \mathbf{M}. \tag{3}$$

Dynamic instances of knapsack problems have been proposed before. However, these studies are mainly focused on either only one dimension problem or a cyclic change of the resource constraint [35, 36]. Inspiration from [13, 37], we construct the dynamic MKP by updating all parameters of w_{kj}, p_j, and c_k after a predefined simulation time unit using a normally distributed random distribution with zero mean and standard deviation θ:

$$\begin{aligned} p_j^+ &= p_j\left(1 + N\left(0, \theta_p\right)\right), \\ w_{kj}^+ &= w_{kj}(1 + N(0, \theta_w)), \\ c_k^+ &= c_k(1 + N(0, \theta_c)). \end{aligned} \tag{4}$$

In formula (4), p_j^+, w_{kj}^+, and c_k^+ denote the updated parameters of MKP when a change occurs after a predefined simulation time units, respectively. The less number of simulation time units yield to more frequent changes and vice versa [13, 37–39]. The number of iterations allocated for each environment is usually adopted as the frequency of changes.

2.2. Related Work on DMKPs. In recent years, DMKPs have attracted growing interest from the optimization community with its wide applications and challenging solutions. The related research on DMKPs can be generally summarized as follows:

(1) Various dynamic benchmark generators for DMKPs: many generators have been proposed to generate changing environments for MKPs and then translate a well-known static MKP into a dynamic version using specialized procedures. Branke et al. [37] designed a dynamic version of MKP by using a normal distribution to update each parameter of a MKP when a change occurs, as shown in formula (4). Yang and Yao [39] formalized a well-known dynamic problem generator to create required dynamics for a given static combinatorial problem using the bitwise exclusive-or (XOR) operator. This generator is also available for MKPs. Based on a XOR DOP generator, Li and Yang [40] proposed a generalized dynamic benchmark generator (GDBG)

that can be instantiated into the binary space, real space, and combinatory space. In addition, the GDBG can present a set of different properties to test algorithms by tuning some control parameters. Rohlfshagen and Yao [38] proposed a new benchmark problem for dynamic combinatorial optimization by taking both the underlying dynamics of the problem and the distances between successive global optima into consideration; the parameters of MKP can be changed over time by some set of difference equations

(2) Effects of solution representation techniques for DMKPs: the effects of different solution representations (i.e., weight coding, binary representation, and permutation representation) were compared with a set of DMKPs in [37]. Simulation results revealed that the solution representation affects the algorithms' performance greatly when solving DMKPs and the binary representation performs relatively poor

(3) Extensions of DMKPs: there are various versions of DMKPs in terms of the changed parameters of MKPs. In [41], a stochastic 0/1 KP where the value of the items p_j are deterministic but the unit resource consumptions w_{kj}^+ are randomly distributed was studied. He et al. [42] proposed a more generalized time-varying KP (TVKP) called randomized TVKP (RTVKP) where all parameters of MKPs p_j^+, w_{kj}^+, and c_k^+ change dynamically in a random way. Moreover, the dynamic version of MKPs that change its parameters p_j^+, w_{kj}^+, and c_k^+ using normal distribution is used as the dynamic benchmark problem of MKPs most wildly [13, 14, 37, 38]

(4) Different solution approaches for DMKPs: both EAs and swarm intelligence-based algorithms have been applied to solve DMKPs by adding some strategies to improve their adaptability to dynamic environments. In [42], the elitists model-based genetic algorithm (EGA) was integrated with greedy optimization algorithm (GOA) to handle RTVKPs; the GOA is capable of avoiding infeasible solutions and improving the convergence rate. Ünal [43] adopted the random immigrant-based GA and memory-based GA to solve the DMKPs, respectively. Compared with the random immigrant-based GA, the memory-based GA was proved to be more effective to adapt to the changing environments for DMKPs. Afterward, Ünal and Kayakutlu [14] tested different partial random restarting approaches of parthenogenetic algorithm (PGA) [44] by solving a set of MKPs in dynamic environments. When solving the DMKPs using ant colony algorithm (ACO), Randall [45] updated the pheromone trails indirectly according to the changes made to the solutions during the solution repair period; therefore, partial knowledge of the previous environment is preserved and the adaptability to dynamic environments is enhanced. Baykasoğlu and Ozsoydan [13] proposed an improved firefly algorithm (FA) that introduces population diversity by partial random restarts and the adaptive move procedure. The simulation results showed that the improved FA was a very powerful algorithm for solving both static and dynamic MKPs

3. Overview of Binary Wolf Pack Algorithm

Wolf pack algorithm (WPA) is a relatively new swarm intelligence-based optimizer which simulate the collaborative hunting behavior of wolf pack [19]. The basic WPA was originally designed for continuous optimization problems. Due to its simple implementation, robustness, and competitive global convergence performance for high-dimension multimodal functions [19–21], WPA has attracted increasing attention and its various derivative versions for solving discrete problems have been developed in recent years. In [24], Wu et al. proposed a binary WPA (BWPA) based on binary coding of solution to solve the classic 0-1 KPs. Afterward, they modified the BWPA by adding a trying-loading solution repair operator to handle MKPs [23].

Inspired by social hierarchy of biological wolves, individuals in WPA are divided as artificial lead wolf, scout wolves, and ferocious wolves according to their roles during searching optimum. The optimization process of WPA can be generally summarized as scouting, calling, and besieging behavior. In each iteration, the lead wolf can be replaced by other wolves that dynamically own better fitness and the whole population is updated in order to increase diversity. The main operation procedures of the BWPA are summarized as follows:

Step 1. Initialize the parameters of algorithm step coefficient S, distance determinant coefficient d_{near}, maximum number of repetitions in scouting behavior $T_{\max}$, and population renewing proportional coefficient β. Randomly initialize the position of artificial wolves in $N \times n$ Euclidean space, where N is the number of wolves and n is the number of variables, the position of artificial wolf i is $X_i = \{x_{i1}, x_{i2,} \cdots, x_{ij}, \cdots, x_{\text{in}}\}$. As for MKP, X_i is a n bit binary string and represents a potential solution. $Y_i = f(X_i)$ denote the objective function value of the wolf i. The wolf X_{lead} with best objective function value $Y_{\text{lead}} = \max\{Y_i\}$ is selected as the lead wolf of the first generation.

Step 2. Scouting behavior models the board search of prey in wolf pack's hunting behavior under the command of lead wolf. Except the lead wolf, the rest $n-1$ wolves act as the scout wolves to take the scouting behavior by implementing the moving operator Θ, respectively, until $Y_i > Y_{\text{lead}}$ or the scouting repetition number T reaches $T_{\max}$, then go to Step 3.

If $Y_i > Y_{\text{lead}}$, the scout wolf i replaces the role of previous lead wolf and acts as the new lead wolf; Elseif $Y_i \le Y_{\text{lead}}$, the scout wolf i, respectively, takes a step towards h different directions and move to the best direction $p*$ (i.e., $Y_{ip*} = \max\{Y_{ip}\}$). h is a positive integer that is randomly selected in the interval of $[h_{\min}, h_{\max}]$. After taking a step towards

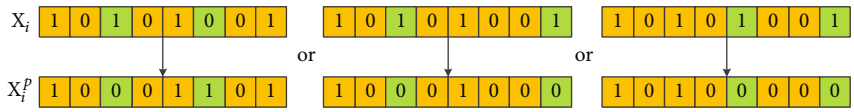

Figure 1: An illustration of moving operator.

the p^{th} scouting direction ($p \in \mathbf{H}, \mathbf{H} = \{1, 2, \cdots, h\}$), the position of the scout wolf i is updated by

$$\mathbf{X}_i^p = \Theta(\mathbf{X}_i, \mathbf{M}_a, \text{step}_a), \tag{5}$$

where X_i and step_a denote the position and step size of the scout wolf i, respectively. $M_a = \{1, 2, \cdots, m\}$. The function of moving operator $\Theta(\mathbf{X}_i, \mathbf{M_a}, \text{step}_a)$ is updating the X_i by reversing the step_a bits values which are randomly selected from M_a.

Assuming that $X_i = \{1, 0, 1, 0, 1, 0, 0, 1\}$, $M_a = \{3, 6, 8\}$, and $\text{step}_a = 2$, the reserving from X_i to $\mathbf{X}_i^p$ by moving operator $\Theta(\mathbf{X}_i, \mathbf{M_a}, \text{step}_a)$ can be illustrated as Figure 1.

Step 3. Except the lead wolf, the rest $n - 1$ wolves secondly act as the ferocious wolves in calling behavior. In order to hunt the prey, the lead wolf commands the ferocious wolves to gather towards its position X_{lead} by howling. The position of the ferocious wolf i is updated by

$$\mathbf{X}_i^{\text{new}} = \Theta(\mathbf{X}_i, \mathbf{M}_b, \text{step}_b), \tag{6}$$

where $\mathbf{X}_i^{\text{new}}$ and step_b denote the updated position and step size of the ferocious wolf i, respectively. $\mathbf{M}_b$ is the set of bits with different values between X_{lead} and X_i. Θ is the same moving operator as defined in Step 2.

If $Y_{i\text{new}} \geq Y_{\text{lead}}$, the ferocious wolf i replaces the previous lead wolf and restarts the calling behavior; otherwise, the ferocious wolf i continues running until $d_{\text{is}} \leq d_{\text{near}}$, then go to Step 4, where d_{is} indicates the distance between X_{lead} and X_i.

Step 4. After calling behavior, the wolves approach and surround the prey, then the whole wolf pack attack and capture the prey successfully. The position of the wolf i is updated by

$$\mathbf{X}_i^{\text{new}} = \Theta(\mathbf{X}_i, \mathbf{M_c}, \text{step}_c), \tag{7}$$

where $\mathbf{X}_i^{\text{new}}$ and step_c denote the updated position and besieging step size of the wolf i, respectively. $\mathbf{M}_c$ and Θ are the same as that defined in Step 3. The relationships between step_a, step_b, and step_c are described as follows:

$$\begin{cases} \text{step}_a = \text{rand int } [\text{step}_c, S], \\ \text{step}_b = \text{rand int } [\text{step}_c, 2S], \end{cases} \tag{8}$$

where step_c is commonly set to 1; rand int indicates a randomly selected integer in this interval.

Step 5. Update the position of wolf pack with population renewing proportional coefficient β.

Step 6. Output the position and function value of lead wolf (i.e., the optimal solution) when termination condition is satisfied, otherwise go to Step 2. The pseudocode of BWPA is illustrated in Algorithm 1.

4. Proposed Flexible Wolf Pack Algorithm

Flexibility is the ability to respond to changing environments effectively. Flexible wolf pack algorithm (FWPA) does not pursue the ultimate convergence of the population, but to maintain the diversity of the population throughout the evolution process, that is, to maintain a strong ability to open up new solution space, which of course should be combined with elite retention strategies. In this section, a flexible population updating strategy based on convergence situation is designed for FWPA to develop its capability of adapting to changing environments.

4.1. Original Population Updating Strategy in BWPA. For the BWPA, there are two cases of updating the population: population updating in normal situation and catastrophic situation.

Population updating in normal situation indicates that $R(R = \text{rand int } [N/(2\beta), N/\beta])$ artificial wolves with worst objective function values are deleted, while R new wolves are generated near the lead wolf by

$$\mathbf{X}_{\text{new}} = \Theta(\mathbf{X}_{\text{lead}}, M, \lfloor L_1 \rfloor), \quad L_1 = \frac{1}{4} \cdot \frac{\exp(z(g)) - \exp(-z(g))}{0.1 \exp(z(g)) + 2 \exp(-z(g))}, \tag{9}$$

where X_{lead} denotes the position of the lead wolf, $M = \{1, 2, \cdots, n\}$, and $z(g) = 10g/\text{MaxGen} - 5$; $\lfloor \cdot \rfloor$ represents that rounding down of L_1 to an integer.

Population updating in catastrophic situation indicates that R artificial wolves are randomly selected and deleted from the whole population when the best objective function value is not updated in $t_{\max}$ continuous iterations, then R new wolves are reproduced by

$$\mathbf{X}_{\text{new}}^* = \Theta(\mathbf{X}_i, \mathbf{M}, \lceil L_2 \rceil), \quad L_2 = \begin{cases} \dfrac{k_1(f(\mathbf{X}_{\text{lead}}) - f(\mathbf{X}_i))}{f(\mathbf{X}_{\text{lead}}) - f_{\text{avg}}}, & f(\mathbf{X}_i) \geq f_{\text{avg}}, \\ k_2, & f(\mathbf{X}_i) < f_{\text{avg}}, \end{cases} \tag{10}$$

where $\lceil \cdot \rceil$ represents that rounding up of L_2 to an integer, f_{avg} is the average fitness value of whole population. k_1 and k_2 are commonly set to 2 and 4, respectively.

Input: the parameters of BWPA including step coefficient S, distance determinant coefficient d_{near}, maximum number of repetitions in scouting behavior $T_{\max}$, and population renewing proportional coefficient β
Output: the best objective value
1. Generate initial population of wolves $X_i = \{x_{i1}, x_{i2}, \ldots, x_{ij}, \ldots, x_{in}\}$
2. Evaluate the fitness of whole population and select the initial lead wolf $Y_{lead} = max\{Y_i\}$
3. Set iteration counter for initial population g:=0
4. **while** g < MaxGen **do**
5. Scouting behavior
6. **if** $Y_i > Y_{lead}$ **then**
7. renew the lead wolf
8. **else if** $Y_i \le Y_{lead}$ & scouting repetition number $T < T_{\max}$ **then**
9. takes a step towards h different directions with the moving operator $\Theta(\boldsymbol{X}_i, \boldsymbol{M_a}, step_a)$
10. **else**
11. update the position of wolves as in Eq. (5)
12. **end if**
13. Calling behavior
14. **if** $Y_i > Y_{lead}$ **then**
15. renew the lead wolf and restart the calling behavior
16. **else if** $Y_i \le Y_{lead}$ & $d_{\text{is}} \le d_{near}$ **then**/* d_{is} is the distance between X_{lead} and X_i */
17. wolves move to the lead wolf with the moving operator $\Theta(\boldsymbol{X}_i, \boldsymbol{M}_b, step_b)$
18. **else**
19. update their positions as in Eq. (6)
20. **end if**
21. Besieging behavior
22. **if** $Y_i > Y_{lead}$ **then**
23. renew the lead wolf
24. **else if** $Y_i \le Y_{lead}$
25. wolves move to the lead wolf with the moving operator $\Theta(\boldsymbol{X}_i, \boldsymbol{M}_c, step_c)$
26. **else**
27. update their positions as in Eq. (7)
28. **end if**
29. Population updating
30. g++
31. Restart the scouting, calling, and besieging behaviors
32. **end while**

Algorithm 1: Binary wolf pack algorithm.

4.2. Flexible Population Updating Strategy. The original population updating strategy helps to increase the population diversity to some degree; however, it may also lead to two problems: (1) L_1, L_2, and the average fitness value of whole population f_{avg} are evaluated in each generation so that the computational cost increases. (2) In the catastrophic situation, the best objective function value is not updated after $t_{\max}$ continuous generations, which can be judged that the lead wolf has fallen into a local optimum. Generating R new wolves based on the previous randomly selected wolves has a tiny effect on jumping out the current local optima, because the updated wolves may gather to the previous lead wolf with a large probability. Therefore, the original population updating strategy is ineffective to introduce or maintain population diversity in a catastrophic situation.

Based on the above analysis, we design a simpler and efficient population updating strategy using the Cauchy distribution random number. Cauchy distribution is a well-known continuous probability distribution. Its probability density function and distribution function are presented by formulas (11) and (12), respectively.

$$f(x, z, \tau) = \frac{1}{\pi\tau} \cdot \frac{1}{1 + ((x - z)/\tau)^2}, -\infty < x < +\infty, \tau > 0, \tag{11}$$

$$F(x, z, \tau) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{x - z}{\tau}\right), \tag{12}$$

where z is the positional parameter and τ is the scaling or shape parameter. Cauchy distribution is named the standard Cauchy distribution C (0,1) when $\tau = 1$ and $z = 1$.

The Cauchy random numbers can be obtained by converting formula (12) to its inverse function (13), where $F(x) \in u(0, 1)$.

$$x = z + \tau \tan(\pi(F(x) - 0.5)). \tag{13}$$

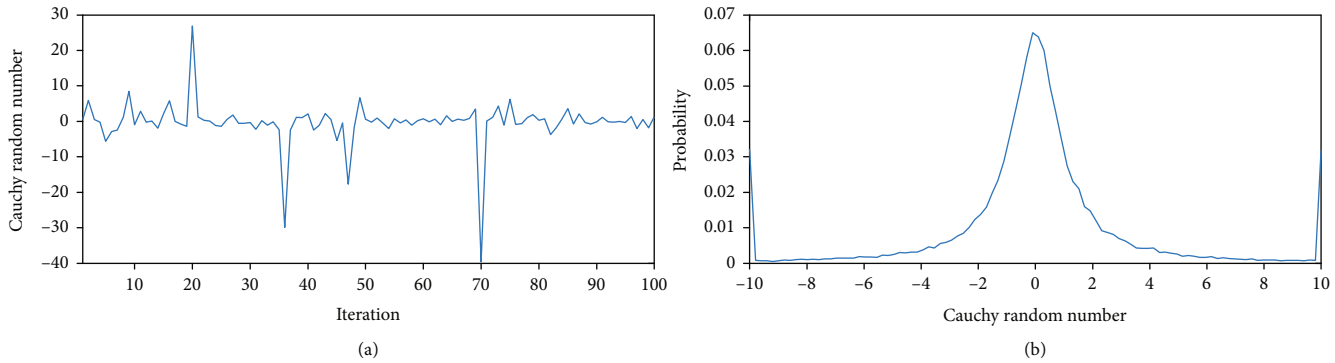

Figure 2: (a) Distribution of Cauchy random number with iterations. (b) Probability of Cauchy random number.

Input: $N, X, Y, X_{lead}, Y_{lead}, R, t, t_{\max}$
Output: new_X, new_Y
Generate initial population of wolves $X_i = \{x_{i1}, x_{i2}, \ldots, x_{ij}, \ldots, x_{in}\}$
Rank $Y = (Y_1, Y_2, \cdots, Y_N)$ and $X = (X_1, X_2, \cdots, X_N)$ based on objective function values Set iteration counter for initial population
  if $t \le t_{\max}$ **then**
    for $i = R + 1 : N$
      new_$X_i = \Theta(X_{lead}, M, C_2)$
      Replace X_i with new_X_i
    end
  else if $t > t_{\max}$ **then**
    for $i = 1 : R$
      new_$X_i = \Theta(X_{lead}, M, C_1)$
      Replace X_i with new_X_i
    end
  end if
  Evaluate new_Y

Algorithm 2: Flexible population updating strategy.

The distribution of the Cauchy distribution random numbers with iterations is shown as Figure 2.

As can be seen from Figures 2(a) and 2(b), the Cauchy random numbers consist of a few mutation numbers and many smoothly fluctuating numbers. Such distribution property is available for generating a few mutant wolves when updating their position in the search process. Therefore, the flexible population updating strategy can be formulated as

$$X_{\text{new}} = \begin{cases} \Theta(X_{\text{lead}}, M, C_1), t > t_{\max}, \\ \Theta(X_{\text{lead}}, M, C_2), t \le t_{\max}, \end{cases} \quad (14)$$

where $C_1 = \lceil |x| \rceil$ and $C_1 = C_2/\mu$, X_{new} denotes the position of new generated wolves, and μ is the correlation coefficient that binds the population updating in both normal and catastrophic situation together.

The flexible population updating strategy can be described as follows: in normal situation $t \le t_{\max}$, similar to original population updating strategy, R worst wolves are deleted and then R new wolves are generated based on the position of lead wolf. In catastrophic situation $t > t_{\max}$, contrary to normal situation, R current best wolves are deleted. The pseudocode of flexible population updating strategy is shown as Algorithm 2.

In fact, C_1 and C_2 can be viewed as the distance between the reinitialized wolf and the previous lead wolf. The larger C_1 and C_2 yield to new wolves that are more different from the previous lead wolf. Therefore, C_1 is larger than C_2 when μ subjects to (0,1). The new generated wolves are close to the previous lead wolf in normal situation, and the convergence rate can be accelerated because the positive individual informant is reused, while the new generated wolves are relatively far away from the previous lead wolf in catastrophic situation, so that the negative informant is deleted and the population diversity is increased consequently. The idea of this dynamic population updating strategy compromises the merits of *Partial restart* [46–48] and *Memory scheme* [49, 50]. The dynamic population updating strategy is inferior to the original ones in terms of increasing population diversity and previous informant reusing. The capability of adapting the dynamic environments of BWPA is developed.

4.3. Adapting in Changing Environments. All swarm intelligence-based algorithms are initially designed for converge to the optima quickly and precisely. However, when

Input: the parameters of BWPA
Output: the best objective value
1 Generate initial population and select the initial lead wolf
2 Set iteration counter for initial population g:=0
3 **while** g < MaxGen **do**
5 Scouting behavior
6 Calling behavior
7 Besieging behavior
8 Population updating based on the flexible population updating strategy
9 g++
10 Restart the scouting, calling, and besieging behaviors
11 **end while**

Algorithm 3: Flexible binary wolf pack algorithm.

Table 1: Parameters of the algorithms.

Algorithm	Main parameters
PGA	Elitism rate 0.01, insert rate 0.3, reserve rate 0.3, swap rate 0.3.
CBPSOTVAC	Inertial weight $w_{\max} = 1.5$, $w_{\min} = 0.5$, acceleration coefficient $c_{1i} = c_{2f} = 2.5$, $c_{2i} = c_{1f} = 0.5$, $V_{\min}$=4.
BWPA	Step coefficient $S = 2$, distance determinant coefficient $d_{\text{near}} = 4$, maximum number of repetitions in scouting behavior $T_{\max} = 10$, population renewing proportional coefficient $\beta = 2$.
FWPA	The parameters are the same as those of BWPA, $\mu = 0.5/0.75/1/2$.

solving DOPs, the capability of adapting to changing environments (i.e., detecting and tracking the changing optima quickly) is necessary. The efficient approach of increasing/-maintaining population diversity is significant for enhancing the adaptation capability. However, too high level of diversity will not always lead to better performance for an algorithm. The *knowledge transfer* and *diversity maintenance* should be well balanced.

In this study, the proposed flexible population updating strategy is capable of generating new wolves at each generation, so the diversity loss problem can be well addressed. After generating the new wolves, the fitness values of the whole population are reevaluated at each generation; the changed optima can be detected and tracked when all parameters of an MKP change. Moreover, the population is updated with the use of previous positive information, which is also beneficial for converging to the new optima quickly. Therefore, any other dynamic change detecting method is required.

4.4. Design of the FWPA for DMKP. The pseudocode of proposed FWPA is shown as Algorithm 3. To some extent, all dynamic methods try to make balance between diversification (global search), intensification (local search), and the balance between accuracy and speed. FWPA shows a good performance on both of them.

5. Simulation Experiments

To verify the performance of FWPA, we conduct both the static and dynamic experiments using a set of MKP benchmarks.

5.1. Experimental Data Set. As for stationary environment, we select 9 different benchmark problems with different difficulty levels available on the OR-LIBRARY website (http://people.brunel.ac.uk/mastjjb/jeb/orlib/files). The items of these instances, n, range from 100 to 500, and the constraints, m, vary from 5 to 30. These problems were also previously used in [14, 51–53]. We express these instances by the notation $m.n.i$ which indicates the ith instance with m constraints, n items. For example, 10.250.00 is the first instance of *mknapcb5.txt* with 10 constraints, 250 items, and tightness ratio of 0.25.

As for the dynamic environment, similar to [13, 37], dynamic instances of MKP are designed by updating the parameters after a predefined simulation time units as defined in Section 2.1. The instance of 10.250.00 was adopted here as the initial and basic environment to generate the changing environments. After the change occurs, the parameters are updated by formula (4).

5.2. Experimental Setup and Parameter Setting. For static experiments, the correlation coefficient μ is set to 0.5, 0.75, 1, and 2, respectively, to measure its effect on the performance of the improved BWPA. Two state-of-the-art algorithms that have been used to solve the MKPs are used for comparisons; they are chaotic binary particle swarm optimization with time-varying acceleration coefficients (CBPSOTVAC) [51] and parthenogenetic algorithm (PGA) [14]. The parameters of the algorithms were set as shown in Table 1. For each algorithm and each problem, 30 independent runs with 1000 iterations are implemented; the population sizes of all algorithms are equal to 100. The best

Table 2: Experimental results on stationary environments for MKPs.

Inst. (best known)		PGA	CBPSO TVAC	BWPA	FWPA			
					$\mu = 0.5$	$\mu = 0.75$	$\mu = 1$	$\mu = 2$
5.500.0 (120148)	Best	117365	118242	119406	119567	**119748**	119540	119421
	Avg.	116554	118104.6	119297.4	119333.6	**119413.6**	119361	119219.4
	Std	646.30	311.97	**89.90**	180.66	222.94	445.88	184.88
5.500.14 (218966)	Best	217404	217237	218130	218257	**218474**	218444	218325
	Avg.	216717	216815.6	21783.6	21792.5	**218163**	21832.6	218104.6
	Std	567.87	257.11	435.19	352.54	239.09	**107.92**	221.07
10.100.0 (23064)	Best	22947	23055	22961	22925	**23057**	23055	22961
	Avg.	22879.6	**22958.2**	22843.8	22830.8	22850.8	22926.4	22886.2
	Std	55.76	194.35	93.14	**83.60**	124.41	109.04	68.20
10.100.14 (41884)	Best	41572	41646	41727	41748	**41791**	41767	41737
	Avg.	41491	41525	41655.6	41643.4	**41704**	41688	41655.2
	Std	82.76	112.91	74.56	66.98	**50.90**	86.52	79.41
10.250.0 (59187)	Best	57943	58338	58846	58577	**58904**	58736	58714
	Avg.	57582.8	58132.8	58670.4	58333	**58564.8**	58522.6	58472.6
	Std	323.03	**139.42**	151.10	265.13	236.87	133.49	258.26
10.250.14 (108485)	Best	107369	107546	107932	108081	**108142**	108090	107853
	Avg.	107118.4	107067	107698	107914	**107958.2**	107859.2	10770.7
	Std	183.70	486.99	168.55	159.78	**108.08**	223.03	143.45
10.500.0 (117821)	Best	114842	115067	116159	116218	**116840**	116574	116389
	Avg.	114250.8	114852.4	116066.6	115896.2	**116631.4**	116134.6	116126.6
	Std	645.53	357.39	247.15	268.64	**150.82**	334.27	308.00
30.100.14 (41058)	Best	40866	40917	40954	40957	**41058**	40912	40922
	Avg.	40751.8	40747.8	40857.8	40817.6	**40920.4**	40797.6	40843.6
	Std	93.86	144,72	**58.17**	91,75	100.65	74.92	80.97
30.250.0 (56842)	Best	55374	55921	56194	55851	**56266**	56060	56057
	Avg.	54827	55719.4	55916.4	55554	**56069.4**	55818.4	55827.8
	Std	445.88	**209.81**	219.60	294.70	234.40	300.21	223.32

solution found (Best), the mean of the solutions (Avg.), and the standard deviation of all solutions (Std) overruns are used as the performance measures.

For dynamic experiments, the standard deviation σ of normal distributions of each parameter is assumed to be equal. σ reflects the severity of dynamic changes, so two different values of $\sigma_p = \sigma_w = \sigma_c = 0.05$ and $\sigma_p = \sigma_w = \sigma_c = 0.1$ are set, respectively, to test the proposed algorithm's capability of adapting to the different dynamic environments. In this study, for each algorithm and each problem, 30 independent runs with 2000 iterations are implemented, and a series of 200 iterations is adopted as the frequency of changes. Therefore, 10 different environments were generated using the basic environment (i.e., the instance of 10.250.00). The average best-of-generation was used to measure the algorithm's ability of finding a better solution at each generation in dynamic environments.

Both static and dynamic experiments were executed using the MATLAB software with a personal computer bundled with Intel i7 1.6 GHZ processor and 8 GB RAM.

5.3. Results on Stationary Environment. Results on stationary environment are shown in Table 2. The best results of each instance achieved by algorithms are denoted in bold.

According to the results presented in Table 2, FWPA proves inferior to the other three approaches in the majority of the problems in terms of Best, Avg., and Std. With the introduction of dynamic population updating strategy, the proposed algorithm enables to maintain the population diversity and enhance the capability of jumping out of the local optima. Therefore, FWPA can find better solutions and the efficiency of the proposed strategy is proved.

From the comparison of the different versions of FWPA that own different values of μ, it can be seen that the FWPA with $\mu = 0.75$ performs best, and the performance of the algorithms with $\mu = 0.5$, 1, and 2 is similar to BWPA. For each test instance, the FWPA with $\mu = 0.75$ achieves best results in terms of Best and Avg. Therefore, the parameter μ might affect the performance of FWPA crucially. In the following dynamic experiments, the parameter μ is set to 0.75.

Table 3: Experimental results on dynamic environments for MKPs.

	Env.	PGA	CBPSOTVAC	BWPA	FWPA ($\mu = 0.75$)
$\sigma = 0.05$	1	58131.4	58612.1	58667.6	**58736.8**
	2	59507.5	60451.9	60540.2	**60693.6**
	3	58393.3	59075.9	59028.5	**59283.7**
	4	57377.3	57825.1	57885.6	**58205.5**
	5	57536.6	58152.1	58292.3	**58508.9**
	6	57352.9	57978.0	58187.3	**58267.2**
	7	59151.5	60019.2	60208.9	**60357.7**
	8	57425.0	57968.8	57968.4	**58169.5**
	9	59327.7	59781.1	59907.0	**60002.7**
	10	58601.6	59199.0	59195.0	**59399.0**
$\sigma = 0.1$	1	56192.1	57558.4	58371.0	**58613.6**
	2	56070.7	58026.1	59257.0	**59422.1**
	3	56935.1	58986.9	60776.0	**61069.1**
	4	51729.2	52506.8	54635.9	**55060.1**
	5	58376.2	61149.4	63197.6	**63560.1**
	6	53055.8	55420.5	57063.8	**57213.6**
	7	51145.4	53753.6	55662.7	**55840.1**
	8	58366.9	61091.3	62909.5	**63215.7**
	9	53742.7	56579.1	57636.9	**57824.0**
	10	53224.3	56022.8	57159.0	**57435.7**

For Std, the FWPA with $\mu = 0.75$ achieves the better results than the compared algorithms in Inst. 10.100.14, 10.250.14, 10.500.0, and 30.100.0, which shows that the proposed algorithm has a good stability.

5.4. Results on Dynamic Environment. Results of average best-of-generation on dynamic environments that are generated by the instance 10.250.00 are shown in Table 3. An efficient algorithm is expected to quickly adapt to new environments and track the moving optima. From the results presented in Table 3, it can be seen that the proposed algorithm outperforms the compared algorithms for $\sigma = 0.05$ and 0.1. By partially restarting new wolves based on the memory of previous stored informant, the proposed algorithm is capable of tracking the changing optima quickly by efficiently maintaining/introducing the population diversity.

By comparing the results when $\sigma = 0.05$ and 0.1, which reflect the severity of the change between two dynamic environments, it can be seen that the differences of two consecutive environments become larger with the increase of σ. The proposed algorithm is capable of tracking the changing optima quickly and find better results than the other algorithms; this situation can attribute to the powerful capability of opening up new solution space using the dynamic population updating strategy.

Convergence graphs of the four algorithms when $\sigma = 0.05$ and 0.1 are presented in Figures 3 and 4, respectively. For each change, the FWPA can achieve best results. It is apparent from the figures that the proposed algorithm has more efficient capability of adapting the dynamic environments.

5.5. Statistical Verification. The statistical results of comparing algorithms by a one-tailed test with 98 degrees of freedom at a 0.05 level of significance are given in Table 4. In Table 4, the t-test result regarding FWPA, BWPA, PGA, and CBPSOTVAC is shown as "+," "~," and "-" when one algorithm is insignificantly better than, insignificantly worse than, significantly better than, and significantly worse than the other one, respectively.

From the statistical verification presented by Table 4, we can conclude that FWPA outperforms the other three algorithms for both dynamic and stationary environments. This result demonstrates the effectiveness of the dynamic population updating strategy.

6. Conclusions and Future Work

This paper presents a flexible BWPA (FWPA) by designing a novel and simpler flexible population updating strategy. The proposed flexible population updating strategy aims at addressing the problem of lack diversity for the WPA during the procedure of solving dynamic optimization problems. In fact, the flexible population updating strategy is a hybridization of *Partial restart* and *Memory scheme* strategy. The simulation experiments on a set of static MKPs prove the effectiveness of the proposed algorithm. Moreover, the simulation experiments on dynamic MKP instances demonstrate that the FWPA is capable of tracking the changing optima quickly and converge to a good solution.

To the best of our knowledge, this paper constitutes the first paper on the combinatorial dynamic optimization

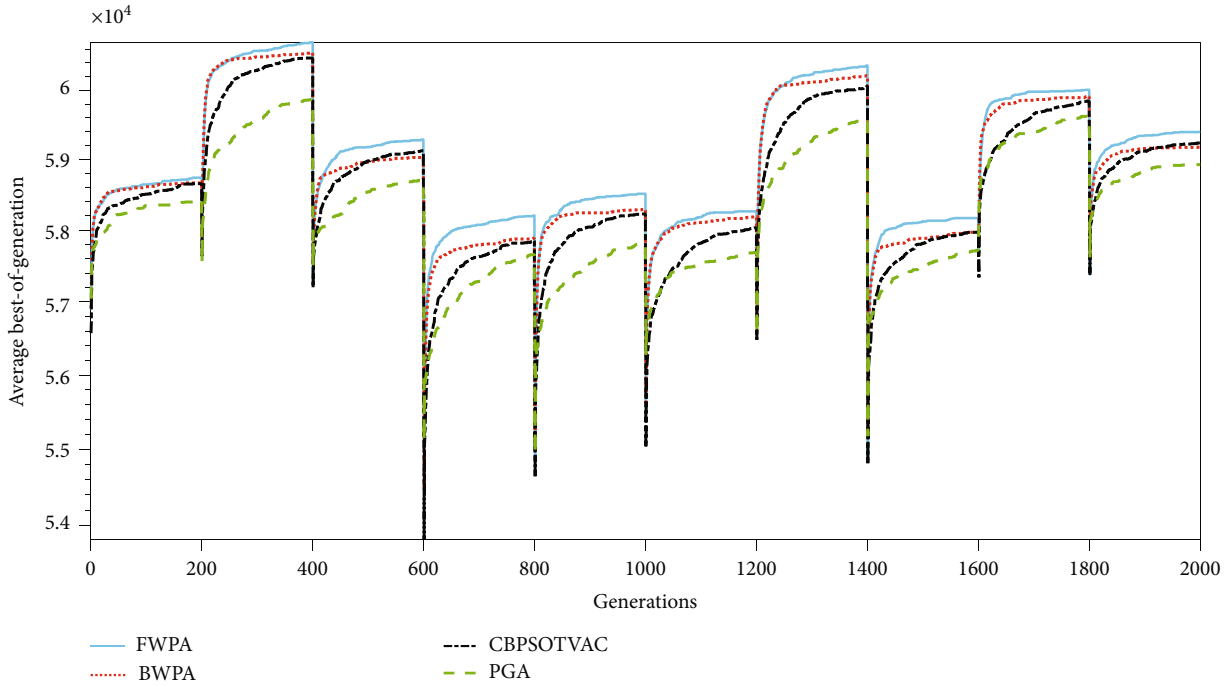

Figure 3: Convergence of algorithms on dynamic environments when $\sigma = 0.05$.

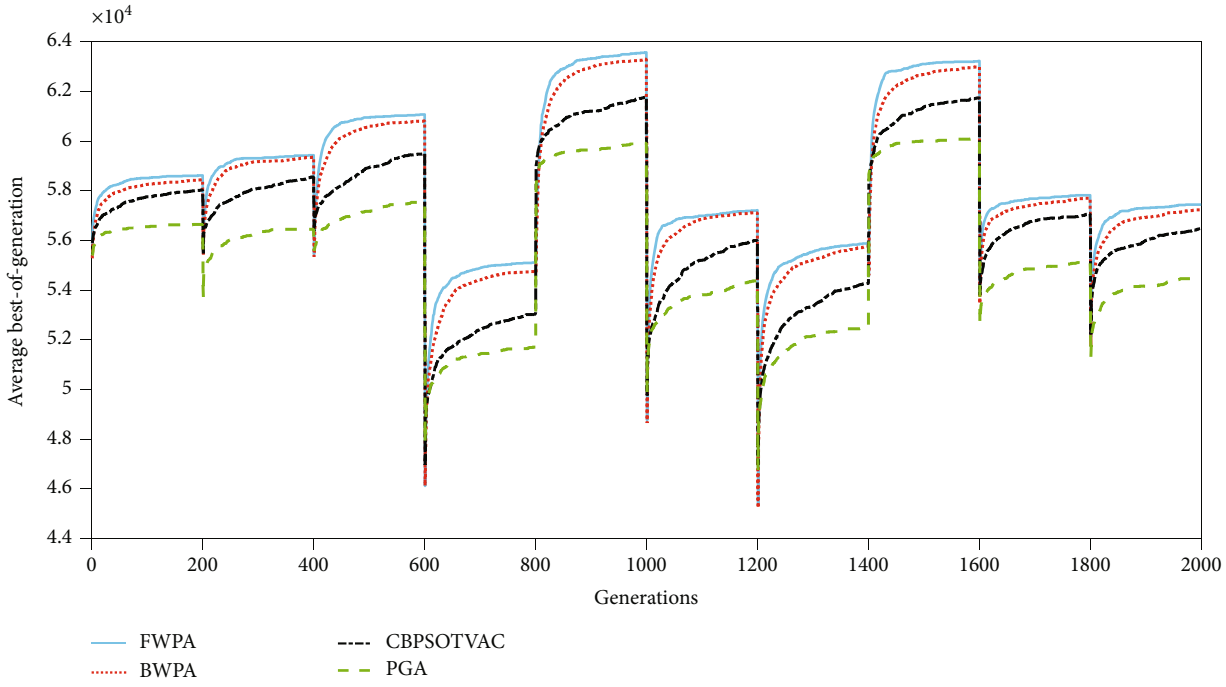

Figure 4: Convergence of algorithms on dynamic environments when $\sigma = 0.1$.

problems of WPA. Another contribution of the study is extending the family of approaches for dynamic optimization.

One of the future work is conducting a comparative study with the advanced algorithms such as jDE, SaDE, and Hyper-Mutation GA which were designed particularly for dynamic optimization problems. Moreover, the FWPA will be applied on various combinatorial dynamic optimization problems such as DTSP, DVRP, and DJSSP. Continuous dynamic optimization is also an expected research issue for WPA. Besides, as a relatively new metaheuristic algorithm, WPA has room for developing the performance of solving dynamic optimization problems in the long run. The issues of addressing

Table 4: t-test results of the algorithms.

t-test result	Static									Dynamic	
	1	2	3	4	5	6	7	8	9	$\sigma = 0.05$	$\sigma = 0.1$
FWPA-BWPA	+	+	+	+	+	+	+	+	+	+	+
FWPA-CBPSOTVAC	+	+	+	+	+	+	+	+	+	+	+
FWPA-PGA	+	+	+	+	+	+	+	+	+	+	+
BWPA-CBPSOTVAC	+	+	+	+	+	+	+	+	+	~	+
BWPA-PGA	+	+	+	+	+	+	+	+	+	+	+
PGA-CBPSOTVAC	-	-	-	-	-	-	-	-	-	-	-

the dynamic optimization problems more efficient for WPA can be summarized as follows:

(1) A more powerful capability of detecting and tracking the dynamic events

(2) Faster convergence rate along with the capability of being stuck in local optima

(3) Taking advantages of gathered evolutionary information to decrease computational cost and adapting the changing environment more efficiently

(4) Self-adaptive parameters tuning performance to decrease the difficulty of implementation

Conflicts of Interest

We declared that we have no conflicts of interest to this work. We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

Acknowledgments

This work is supported by the National Science and Technology Innovation 2030 Major Project of the Ministry of Science and Technology of China (Grant No. 2018AAA0101200) and the National Natural Science Foundation of China (Grant No. 61502534).

References

[1] J.-P. Chiou and F.-S. Wang, "Hybrid method of evolutionary algorithms for static and dynamic optimization problems with application to a fed-batch fermentation process," *Computers & Chemical Engineering*, vol. 23, no. 9, pp. 1277–1291, 1999.

[2] S. X. Yang, "Evolutionary computation for dynamic optimization problems," in *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference - GECCO Companion '15*, Madrid, Spain, July 2015.

[3] A. Zhou, Y. Jin, and Q. Zhang, "A population prediction strategy for evolutionary dynamic multiobjective optimization," *IEEE Transactions on Cybernetics*, vol. 44, no. 1, pp. 40–53, 2014.

[4] S. K. Nseef, S. Abdullah, A. Turky, and G. Kendall, "An adaptive multi-population artificial bee colony algorithm for dynamic optimisation problems," *Knowledge-Based Systems*, vol. 104, no. C, pp. 14–23, 2016.

[5] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[6] C.-H. Li and S.-X. Yang, "Fast multi-swarm optimization for dynamic optimization problems," in *2008 Fourth International Conference on Natural Computation*, pp. 627-628, Jinan, China, October 2008.

[7] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: a survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 6, no. 10, pp. 1–24, 2012.

[8] A. Baykasoğlu and F. B. Ozsoydan, "Evolutionary and population-based methods versus constructive search strategies in dynamic combinatorial optimization," *Information Sciences*, vol. 420, no. 12, pp. 159–183, 2017.

[9] M. Mavrovouniotis, C. Li, and S. Yang, "A survey of swarm intelligence for dynamic optimization: algorithms and applications," *Swarm and Evolutionary Computation*, vol. 33, no. 4, pp. 1–17, 2017.

[10] M. Mavrovouniotis and S. Yang, "Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors," *Applied Soft Computing*, vol. 13, no. 10, pp. 4023–4037, 2013.

[11] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, 2013.

[12] N. Kundakcı and O. Kulak, "Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem," *Computers & Industrial Engineering*, vol. 96, no. 6, pp. 31–51, 2016.

[13] A. Baykasoğlu and F. B. Ozsoydan, "An improved firefly algorithm for solving dynamic multidimensional knapsack problems," *Expert Systems with Applications*, vol. 41, no. 8, pp. 3712–3725, 2014.

[14] A. N. Ünal and G. Kayakutlu, "A partheno-genetic algorithm for dynamic 0-1 multidimensional knapsack problem," *RAIRO-Operations Research*, vol. 50, no. 1, pp. 47–66, 2016.

[15] Y. Feng, G.-G. Wang, and L. Wang, "Solving randomized time-varying knapsack problems by a novel global firefly algorithm," *Engineering with Computers*, vol. 34, no. 3, pp. 621–635, 2018.

[16] A. J. Page, T. M. Keane, and T. J. Naughton, "Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system," *Journal of Parallel and Distributed Computing*, vol. 70, no. 7, pp. 758–766, 2010.

[17] R. Mendes and A. S. Mohais, "DynDE: a differential evolution for dynamic optimization problems," in *2005 IEEE Congress on Evolutionary Computation*, Edinburgh, Scotland, UK, September 2005.

[18] J. Brest, P. Korošec, J. Šilc, A. Zamuda, B. Bošković, and M. S. Maučec, "Differential evolution and differential ant-stigmergy on dynamic optimisation problems," *International Journal of Systems Science*, vol. 44, no. 4, pp. 663–679, 2013.

[19] H.-S. Wu, F. M. Zhang, and L. S. Wu, "New swarm intelligence algorithm-wolf pack algorithm," *Systems Engineering and Electronics*, vol. 35, no. 11, pp. 2430–2438, 2013.

[20] R. Menassel, B. Nini, and T. Mekhaznia, "An improved fractal image compression using wolf pack algorithm," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 30, no. 3, pp. 429–439, 2018.

[21] S. Gupta and K. Saurabh, "Modified artificial wolf pack method for maximum power point tracking under partial shading condition," in *2017 International Conference on Power and Embedded Drive Control (ICPEDC)*, Chennai, India, March 2017.

[22] L. Zhang, L. Zhang, S. Liu, J. Zhou, and C. Papavassiliou, "Three-dimensional underwater path planning based on modified wolf pack algorithm," *IEEE Access*, vol. 5, pp. 22783–22795, 2017.

[23] H.-S. Wu, F. M. Zhang, and R. J. Zhan, "An improved binary wolf pack algorithm for solving multidimensional knapsack problem," *Systems Engineering and Electronics*, vol. 37, no. 5, pp. 1084–1091, 2015.

[24] H. S. Wu, F. M. Zhang, R. Zhan, S. Wang, and C. Zhang, "A binary wolf pack algorithm for solving 0-1 knapsack problem," *Systems Engineering and Electronics*, vol. 36, no. 8, pp. 1660–1667, 2014.

[25] W. S. Wu, F. M. Zhang, and H. Li, "Discrete wolf pack algorithm for traveling salesman problem," *Control and Decision*, vol. 30, no. 10, pp. 1861–1867, 2015.

[26] H. Li, R. B. Xiao, and H. S. Wu, "Modelling for combat task allocation problem of aerial swarm and its solution using wolf pack algorithm," *International Journal of Innovative Computing and Applications*, vol. 7, no. 1, pp. 50–59, 2016.

[27] Q. K. Cao, K. W. Yang, and X. Y. Ren, "Vehicle routing optimization with multiple fuzzy time windows based on improved wolf pack algorithm," *Advances in Production Engineering & Management*, vol. 12, no. 4, pp. 401–411, 2017.

[28] N. P. Bakas, "Numerical solution for the extrapolation problem of analytic functions," *Research*, vol. 2019, Article ID 3903187, 10 pages, 2019.

[29] S. Yang, "Genetic algorithms with memory-and elitism-based immigrants in dynamic environments," *Evolutionary Computation*, vol. 16, no. 3, pp. 385–416, 2008.

[30] B. Nasiri, M. R. Meybodi, and M. M. Ebadzadeh, "History-driven particle swarm optimization in dynamic and uncertain environments," *Neurocomputing*, vol. 172, no. 1, pp. 356–370, 2016.

[31] H. Kellerer, U. Pferschy, D. Pisinger, and D. Knapsack Problems, "Multidimensional knapsack problems," in *Knapsack Problems*, pp. 235–283, Springer, Berlin, Heidelberg, 2004.

[32] A. Fréville, "The multidimensional 0-1 knapsack problem: an overview," *European Journal of Operational Research*, vol. 155, no. 1, pp. 1–21, 2004.

[33] J. Langeveld and A. P. Engelbrecht, "Set-based particle swarm optimization applied to the multidimensional knapsack problem," *Swarm Intelligence*, vol. 6, no. 4, article 73, pp. 297–342, 2012.

[34] G. R. Raidl, "An improved genetic algorithm for the multiconstrained 0-1 knapsack problem," in *1998 IEEE International Conference on Evolutionary Computation*, pp. 207–211, Anchorage, AK, USA, May 1998.

[35] S. K. Basu and A. K. Bhatia, "A naive genetic approach for non-stationary constrained problems," *Soft Computing*, vol. 10, no. 2, pp. 152–162, 2006.

[36] V. Roostapour, A. Neumann, and F. Neumann, "On the performance of baseline evolutionary algorithms on the dynamic knapsack problem," in *Parallel Problem Solving from Nature – PPSN XV. PPSN 2018. Lecture Notes in Computer Science, vol 11101*, A. Auger, C. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, Eds., Springer, Cham, 2018.

[37] J. Branke, M. Orbayı, and Ş. Uyar, "The role of representations in dynamic knapsack problems," in *Applications of Evolutionary Computing. EvoWorkshops 2006. Lecture Notes in Computer Science, vol 3907*, F. Rothlauf, Ed., Springer, Berlin, Heidelberg, 2006.

[38] P. Rohlfshagen and X. Yao, "The dynamic knapsack problem revisited: a new benchmark problem for dynamic combinatorial optimisation," in *Applications of Evolutionary Computing. EvoWorkshops 2009. Lecture Notes in Computer Science, Vol 5484*, M. Giacobini, Ed., Springer, Berlin, Heidelberg, 2009.

[39] S. Yang and X. Yao, "Experimental study on population-based incremental learning algorithms for dynamic optimization problems," *Soft Computing*, vol. 9, no. 11, pp. 815–834, 2005.

[40] C.-H. Li and S. X. Yang, "A generalized approach to construct benchmark problems for dynamic optimization," in *Simulated Evolution and Learning. SEAL 2008, Lecture Notes in Computer Science, vol 5361*, X. Li, Ed., Springer, Berlin, Heidelberg, 2008.

[41] B. C. Dean, M. X. Goemans, and J. Vondrák, "Approximating the stochastic knapsack problem: the benefit of adaptivity," *Mathematics of Operations Research*, vol. 33, no. 4, pp. 945–964, 2008.

[42] Y. He, X. Zhang, W. Li, X. Li, W. Wu, and S. Gao, "Algorithms for randomized time-varying knapsack problems," *Journal of Combinatorial Optimization*, vol. 31, no. 1, pp. 95–117, 2016.

[43] A. N. Ünal, "A genetic algorithm for the multiple knapsack problem in dynamic environment," in *Proceedings of the World Congress on Engineering and Computer Science 2013 Vol II WCECS 2013*, p. 2, San Francisco, CA, USA, October 2013.

[44] R. Shuai, W. Jing, and X. Zhang, "Research on chaos parthenogenetic algorithm for TSP," in *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, Taiyuan, China, October 2010.

[45] M. Randall, "A dynamic optimisation approach for ant colony optimisation using the multiple knapsack problem," in *2th Australian Conference on Artificial Life*, Sydney, Australia, November 2005.

[46] R. Zhou, H. P. Lee, and A. Y. C. Nee, "Applying ant colony optimisation (ACO) algorithm to dynamic job shop scheduling problems," *International Journal of Manufacturing Research*, vol. 3, no. 3, pp. 301–320, 2008.

[47] M. R. Khouadja, L. Jourdan, and E. Talbi, "Adaptive particle swarm for solving the dynamic vehicle routing problem," in *ACS/IEEE International Conference on Computer Systems and Applications-AICCSA 2010*, pp. 1–8, Hammamet, Tunisia, May 2010.

[48] M. R. Khouadjia, B. Sarasola, E. Alba, L. Jourdan, and E. G. Talbi, "A comparative study between dynamic adapted PSO and VNS for the vehicle routing problem with dynamic requests," *Applied Soft Computing*, vol. 12, no. 4, pp. 1426–1439, 2012.

[49] S. Yang, "Memory-based immigrants for genetic algorithms in dynamic environments," in *7th annual conference on Genetic and evolutionary computation,*, pp. 1115–1122, Washington. DC, USA, June2005.

[50] H. Wang, D. Wang, and S. Yang, "Triggered memory-based swarm optimization in dynamic environments," in *Applications of Evolutionary Computing. EvoWorkshops 2007. Lecture Notes in Computer Science, vol 4448*, M. Giacobini, Ed., Springer, Berlin Heidelberg, 2007.

[51] M. Chih, C. J. Lin, M. S. Chern, and T. Y. Ou, "Particle swarm optimization with time-varying acceleration coefficients for the multidimensional knapsack problem," *Applied Mathematical Modelling*, vol. 38, no. 4, pp. 1338–1350, 2014.

[52] S. Boussier, M. Vasquez, Y. Vimont, S. Hanafi, and P. Michelon, "A multi-level search strategy for the 0-1 multidimensional knapsack problem," *Discrete Applied Mathematics*, vol. 158, no. 2, pp. 97–109, 2010.

[53] S. Hanafi and C. Wilbaut, "Improved convergent heuristics for the 0-1 multidimensional knapsack problem," *Annals of Operations Research*, vol. 183, no. 1, pp. 125–142, 2011.