Neural Computing and Applications
https://doi.org/10.1007/s00521-023-09075-x

S.I.: HYBRID APPROACHES TO NATURE-INSPIRED OPTIMIZATION ALGORITHMS AND THEIR APPLICATIONS

An adaptive grey wolf optimization with differential evolution operator for solving the discount {0–1} knapsack problem

Zijian Wang[1] · Xi Fang[1] · Fei Gao[1] · Liang Xie[1] · Xianchen Meng[1]

Received: 21 September 2022 / Accepted: 15 September 2023

Abstract

The discount {0–1} knapsack problem (D {0–1} KP) is a new variant of the knapsack problem. It is an NP-hard problem and also a binary optimization problem. As a new intelligent algorithm that imitates the leadership function of wolves, the grey wolf optimizer (GWO) can solve NP problems more effectively than accurate algorithms. At the same time, the GWO has fewer parameters, faster calculations, and easier implementation than other intelligent algorithms. This paper introduces a method of adaptively updating the prey position of wolves and a differential evolution operator with a scaling factor that adaptively changes according to the number of iterations, and selects which operator to use for iteration by the value of the search agent parameter. Finally, it combines the improved greedy repair operator based on D {0–1} KP to form the adaptive grey wolf optimization with differential evolution operator (de-AGWO). The experimental results of the standard test function prove that the algorithm in this paper has a significant improvement in function optimization performance. And the experimental results of D {0–1} KP shows that the proposed algorithm yields superior solution outcomes, except for unrelated datasets, and exhibits significant advantages when solving strongly correlated datasets. Finally, it is verified that more than 80% of the iterations utilize the grey wolf evolution operator, highlighting that the core of the algorithm remains the GWO.

Keywords D {0–1} · KP · NP-hard · Grey wolf optimizer · Differential evolution operator · Greedy repair operator

1 Introduction

The discount {0–1} knapsack problem (D {0–1} KP) is a more complex variant of the knapsack problem (KP), which has many practical applications in many fields such as project selection and budget control. The D {0–1} KP was first proposed by Guldan [1], and was solved by dynamic programming in his paper.

The concept of discount is taken from the commercial sector, and companies usually provide discounts to consumers. A D {0–1} KP involves $\{\{x_1, x_2, x_3\}, \{x_4, x_5, x_6\}, \ldots\ldots, \{x_{3n-2}, x_{3n-1}, x_{3n}\}\}$ items, and the weights and profits of the items are $\{\{w_1, w_2, w_3\}, \{w_4, w_5, w_6\}, \ldots\ldots, \{w_{3n-2}, w_{3n-1}, w_{3n}\}\}$ and $\{\{p_1, p_2, p_3\}, \{p_4, p_5, p_6\}, \ldots\ldots, \{p_{3n-2}, p_{3n-1}, p_{3n}\}\}$. Each group $i \in N, N = \{1, 2, \ldots\ldots, n\}$ consists of three items, which are two original price items x_{3i-2} and x_{3i-1} one discount item x_{3i} respectively. In order to achieve the purpose of bundling sales, the discount item satisfies $w_{3i} < w_{3i-2} + w_{3i-1}$, that is, the weight of purchasing two original price items at the same time is smaller than the weights sum of the two original price items alone. It should be noted that the discount item is not a real

✉ Xi Fang
fangxi@whut.edu.cn

Zijian Wang
wangzijian@whut.edu.cn

Fei Gao
gaof@whut.edu.cn

Liang Xie
whutxl@hotmail.com

Xianchen Meng
2757491290@qq.com

[1] School of Science, Wuhan University of Technology, Wuhan, China

Published online: 03 October 2023

commodity, so each item group can only choose one item to pack into a backpack with a capacity of C.

1. Select x_{3i-2} means to select the first product of the $i-th$ item.
2. Select x_{3i-1} means to select the second product of the $i-th$ item.
3. Select a discount item x_{3i} means to choose to purchase two original-priced products of the $i-th$ item at the same time.

Therefore, the mathematical model of the D {0–1} KP problem can be expressed as follows:

$$\max f(X) = \max \sum_{i=1}^{n} (x_{3i-2}p_{3i-2} + x_{3i-1}p_{3i-1} + x_{3i}p_{3i}) \tag{1}$$

Subject to:

$$x_{3i-2} + x_{3i-1} + x_{3i} \le 1, \quad \forall i \in N \tag{2}$$

$$\sum_{i=1}^{n} (x_{3i-2}w_{3i-2} + x_{3i-1}w_{3i-1} + x_{3i}w_{3i}) \le C \tag{3}$$

$$x_{3i}, x_{3i-1}, x_{3i-2} \in \{0, 1\}, \forall i \in N \tag{4}$$

Constraint (2) ensures that only one item in each group is selected. Constraint (3) ensures that the total weight of the items in the backpack is not greater than C. Constraint (4) indicates that each variable $x_i, i \in N$ has a value of 0 or 1. 0 means the item x_i is not selected, and 1 means it is selected.

Since D {0–1} KP is an NP-complete problem, accurate algorithms such as dynamic programming cannot meet the solution requirements after increasing the data set. Later, He et al. [2] proposed the second mathematical model of the D {0–1} KP, designed two elite genetic algorithms to solve the D {0–1} KP, and proposed two greedy repair strategies to repair and optimize individuals, which is defined as follows:

$$\max f(X) = \max \sum_{i=1}^{n} \lceil x_i/3 \rceil p_{3i-3+x_i} \tag{5}$$

Subject to:

$$\sum_{i=1}^{n} \lceil x_i/3 \rceil w_{3i-3+x_i} \le C \tag{6}$$

$$x_i \in \{0, 1, 2, 3\}, i \in N \tag{7}$$

x is a top function. An integer variable $x_i, i \in N$ indicates whether there are $i-th$ items in the group that need to be loaded into the backpack. x_i = 0, it means that no items in the $i-th$ group are loaded into the backpack. x_i = 1, it means the $3i-2$ item is loaded into the backpack. x_i = 2, which means the $3i-1$ item is loaded into the backpack. x_i = 3, which means the $3i$ item is loaded into the backpack. Integer vector $X = [x_1, x_2, \ldots\ldots, x_{n-1}, x_n] \in \{0, 1, 2, 3\}^n$ is a potential solution to the D {0–1} KP. Obviously, the second mathematical model has only one constraint (6), but considering the unsatisfactory results of reference [2] and the difficulties in the selection of quaternary effective evolution operators. This paper will use the first mathematical model to solve problems.

Based on the work of reference [2], Researchers use a variety of intelligent algorithms to solve the {0–1} KP. Even so, most intelligent algorithms need to select or control more complex parameters. And the grey wolf optimizer (GWO) [3], as a new meta-heuristic algorithm proposed by imitating the hunting behavior of grey wolves, has the advantages of few parameters and easy implementation, and has attracted wide attention from researchers. The GWO simulates the group behavior of grey wolves and has a very strict social hierarchy. The α wolf at the top of the pyramid is responsible for making decisions. The β wolves at the second level assist the wolves in decision-making. Other scout wolves and sentry wolves are called δ wolves, and the lowest ranked wolf in the population is called ω wolves. The δ wolf obeys the α and β wolves' dispatch, and dominate the ω wolf's behavior together.

The hunting behavior of wolves around their prey is simulated by the following equation:

$$x_j(t+1) = x_j^p(t) - g\left| r x_j^p(t) - x_j(t) \right| \tag{8}$$

where the number of iteration t gradually increases to a pre-determined maximal number G; $x_j(t)$ is the value of solution in the j - th dimension at the t - th iteration; $x_j^p(t)$ represents the current global optimal solution of the problem; random number r is uniformly distributed in $[0,2][0,2]$. $g \in [-2*(1-t/G), 2*(1-t/G)]$, $|g|<1$ means to attack the prey, the algorithm will perform a local search; $|g| > 1$ means to find a new prey, the algorithm will perform a global search.

From a mathematical point of view, the grey wolf algorithm assumes that α, β and δ have a better understanding of the potential location of the prey, and use the location of these leader wolves $x^\alpha, x^\beta, x^\delta$ to replace the location of the prey. Using Eq. (8), a new position of the prey is iterated according to the position of the prey replaced by the wolf, and then the real new position of the prey is updated by the average value. See Eqs. (9)–(12) for specific operations:

$$y_1 = x_j^\alpha(t) - g_1\left| r_1 x_j^\alpha(t) - x_j^i(t) \right| \tag{9}$$

$$y_2 = x_j^\beta(t) - g_2\left| r_2 x_j^\beta(t) - x_j^i(t) \right| \tag{10}$$

$$y_3 = x_j^{\delta}(t) - g_3\left|r_3 x_j^{\delta}(t) - x_j^{i}(t)\right| \tag{11}$$

$$x_j^k(t+1) = (y_1 + y_2 + y_3)/3 \tag{12}$$

where $k = 1, 2, \ldots\ldots, s$, s is the size of the population. The iterative operation of the GWO algorithm is simple and easy. The position of each wolf in the population is updated through the above iterative formula until the number of iterations exceeds a given number or other stopping criteria are met. The grey wolf optimizer can be briefly described as follows:

initialization
repeat
Update the position of each wolf by eqs. (9)-(12)
Evaluate the new positions of all wolves
Update the wolfs of α, β, δ
until the stopping criterion reached.

In the following research, Zhu [4] proposed a discrete differential evolution algorithm (DE) with few parameters and its two variants were proposed to solve the D {0–1} KP. The application of differential evolution greatly optimizes the calculation results. We find that the GWO is effective in avoiding local convergence, and the DE is excellent in global search. At the same time, both algorithms have the advantages of few parameters and easy implementation, so this paper will introduce differential evolution operator to improve the GWO to solve D {0–1} KP.

From Eq. (8), we can know that the parameter g determines whether the algorithm performs a global search or a local search. We know that the global search ability of differential evolution operator is excellent, and the local search ability of GWO is also better than other intelligent algorithms. Therefore, in order to integrate the search capabilities of the two algorithms, we will choose different algorithms according to the value of g.

We introduce a simple difference operator as the iterative method of the global search when $g > 1$. The form of the traditional differential evolution operator is as follows:

$$x_j(t+1) = x_j^p(t) + F(x_{r_1}(t) - x_{r_2}(t)) \tag{13}$$

where x_{r_1} and x_{r_2} represent two random individuals except x_j in the population. F represents the scaling factor, which is generally 0.5. Since there is no selection operator involved, the crossover probability cr will not be introduced here.

From Eq. (13) we can change F into the adaptively changing parameter g to dynamically adjust the scaling factor. The formula for obtaining the parameter g is as follows:

$$a = 1 - \frac{t}{T} \tag{14}$$

$$g = 2ra - a$$

where T represents the maximum number of iterations, and $r \in (0, 1)$ represents a random number. We use the improved Eq. (15) of the differential evolution operator as the iterative form of the grey wolf algorithm when $g > 1$.

$$x_j(t+1) = x_j^p(t) - g|x_{r_1}(t) - x_{r_2}(t)| \tag{15}$$

The present article adopts a top-down approach for organizing its content. Section 1 describes the background and mathematical model of the D {0–1} KP, and introduces the incorporation of differential evolution operators to enhance the GWO for solving the D {0–1} KP. Section 2 presents an overview of related research on KP and D {0–1} KP, including prior algorithms and methodologies. The generation and construction process of the proposed algorithm are discussed in detail in Sect. 3. Experimental results and analysis comparing the performance of different algorithms are presented in Sect. 4. Section 5 further discusses the proposed algorithm, analyzing its advantages and limitations. Lastly, Sect. 6 covers the main contributions of this study and outlines future research directions.

2 Related work

The knapsack problem (KP) is one of the classic combinatorial optimization problems [5]. It has an important application background in real life. For example, Azad [6] solves the problem of quadratic knapsack problems in finance and telecommunications through the binary artificial fish swarm algorithm; some projects applications like cutting stock and cargo loading problems can also be transformed into multi-dimensional knapsack problems and solved well [7]. The knapsack problem is also a classic NP-hard problem. It has many variations. For example, Li et al. [8] proposed a tabu search algorithm to solve the bounded knapsack problem with a general upper bound constraint; Poirriez [9] solved the unbounded knapsack problem by sparse dynamic programming method; Lai et al. [10] proposed a variable neighborhood quantum particle swarm algorithm to solve the multi-dimensional knapsack problem; Babukarthik et al. [11] proposed the utilization of the Cluster Particle Swarm Optimization (CPSO) algorithm to address the Multiple-Constraint Knapsack Problem (MCKP) by preserving multiple local optima; Xie et al. [12] searches on a structural landscape of the problem through the guided generate-and-test behavior under the law of socially biased individual learning to solve the quadratic knapsack problem; Lin et al. [13] studied the random knapsack problem in switch-over policies and dynamic pricing; Dizdar et al. [14] studied the application of dynamic knapsack problem in tax maximization.

The D {0–1} KP, as a more complex variant of the KP, has received considerable research attention due to its wide application in practical domains such as budget control. In 2012, Rong [15] used the core mechanism to divide the D {0–1} KP into several easier sub-problems to solve. In 2016, He [16] proposed an accurate algorithm for solving the D {0–1} KP based on dynamic programming by deriving a recurrence formula. The above method is still an accurate algorithm in essence. Since D {0–1} KP is an NP-complete problem, He et al. [2] proposed the second mathematical model of the D {0–1} KP, designed two elite genetic algorithms to solve the D {0–1} KP, and proposed two greedy repair strategies to repair and optimize individuals. Based on such above research, Feng proposed a multi-strategy Overlord Butterfly optimization algorithm [17] and a binary moth search algorithm based on multiple mutation operators [18] to solve the D {0–1} KP. After that, Zhu [4] proposed a discrete differential evolution algorithm with few parameters and its two variants were proposed to solve the D {0–1} KP, and the feasibility and effectiveness of the algorithm were verified by the method of code conversion. The experimental results of differential evolution algorithm in D {0–1} KP have been significantly improved. Table 1 gives the detailed information of the researches on D {0–1} KP.

Nowadays, as an intelligent algorithm with few parameters and easy implementation like the differential evolution algorithm, the GWO has gradually entered the researcher's field of vision. The GWO is a new meta-heuristic algorithm proposed by imitating the hunting behavior of gray wolf. Because of its unique perspective of leadership, this algorithm has attracted more and more attention, and it has a wide range of application prospects in the fields of parameterized string similarity metrics [19], traffic network dispatch [20], and unmanned combat aircraft path planning [21].Reference [22] solved the feature selection problem by introducing the binary gray wolf algorithm. Subsequently, there are many applications of binary gray wolf algorithm to discrete optimization problems, such as the unit commitment problem of power generation systems [23] and the multidimensional knapsack problem [24]. While some studies [25, 26] have indicated that optimization techniques inspired by bestial metaphors can be misleading, we believe that the widespread research on GWO is primarily due to its excellent algorithmic performance rather than its origin. This is because the fundamental nature of these intelligent algorithms is similar, and their effectiveness is evaluated based on their ability to solve complex optimization problems rather than solely on their metaphorical inspiration.

Table 1 Researches on D {0–1} KP

References	Main contribution	Algorithm	Pros	Cons
[1]	First propose the D {0–1} KP	Dynamic programming	–	–
[15]	Promote the attention of D {0–1} KP	Core mechanism	Divide the D {0–1} KP into several easier sub-problems	Only suitable for small data volume
[16]	Researsh the D {0–1} KP more systematically	Derive a recurrence formula and use a binary particle swarm optimization	Based on the principle of minimizing the total weight with the given sum of value coefficients	Still an accurate algorithm, and the research on approximate algorithm is not deep enough
[2]	Establish two new mathematical models of the D {0–1} KP	Genetic algorithm with elitist reservation strategy	New mathematical model simplifies the D {0–1} KP and lays a foundation for the following research	–
[17, 18]	Start using intelligent algorithm to solve the D {0–1} KP	Multi-strategy overlord butterfly optimization algorithm and binary moth search algorithm based on multiple mutation operators	The performance of the algorithm is greatly improved	Complex parameters
[4]	Use a new intelligent algorithm	Discrete differential evolution algorithm	Few parameters and excellent solving performance	–

3 The binary GWO for the D {0–1} KP

3.1 Population initialization and evaluation criteria

We know that an excellent initial population can increase the efficiency of the heuristic algorithm, but it should be considered based on the actual situation of the problem.

A solution of D {0–1} KP is a binary vector $x = [x_1, x_2, \ldots, x_j, \ldots, x_{3n-1}, x_{3n}]$, $x_j = 1$ means that j - th is selected, otherwise $x_j = 0$. A feasible solution of D {0–1} KP needs to meet the weight constraint and logic constraint. The weight constraint means that the maximum weight of the backpack cannot be exceeded, and the logic constraint means that each group can only select one item. Taking into account that the population must be repaired by the repair operator after initializing the population, it is difficult to improve the initialization of the D {0–1} KP population and is of little value. At present, most studies [2, 4, 17, 18] have no way to improve the initial population.

The evaluation criterion of D {0–1} KP is the fitness function calculation method using the first mathematical model [1], see Eq. (1), and related constraints see Eqs. (2)–(4).

3.2 Repair operator based on greedy strategy

When we solve constrained optimization problems, the common methods to deal with abnormal individuals are Penalty function approach [27], Repair approach [28] and Separatist approach [29]. These methods have their own advantages and disadvantages, and they are not universal. Michalewicz [30] found that the repair method based on the greedy strategy is better than the penalty function method in dealing with abnormal individuals through comparison. In fact, in the iterative process of the algorithm, if there are a large number of infeasible solutions, it is not appropriate to use the penalty function method.

The repair strategy proposed in reference [2] only sorts the value density. However, in some cases, items with large weight and value cannot be chosen into the backpack, leading to premature convergence of the algorithm. This paper improves the repair operator of reference [2] and introduces two options, taking into the account value density and the value itself.

The value density formula is as follow:

$$\rho_j = p_j / w_j, j = 1, 2, \ldots, 3n \tag{16}$$

where p_j is the value of the $j-th$ item and w_j is the weight of the $j-th$ item.

The improved greedy repair algorithm is as follow:

Algorithm 1.

```
Input: P,W,x,C,n
Output: x , f(x)
A = [P / W;1:3n];
T = -sortrows(-A',1)';
T = T(2,:);
H1 = find(sum(reshape(x,3,3n)) > 1);
[~,b] = size(H1);
for i = 1:b
    j = H1(i);
    x(3j-2) = x(3j-1) = 0;
    x(3*j) = 1;
end
for i = 3n :-1:1
    if x W' > C
        x(A(i)) = 0;
    end
end
H2 = find(sum(reshape(x,3,3n)) == 0);
for i = 1:3n
    if find(H2 == ceil(A(i) / 3))
        if xW' + W(A(i)) ≤ C
            x(A(i)) = 1;
        end
    elseif mod(A(i),3) == 0
        x' = x;
        x'(A(i)) = 1;
        x'(A(i)-2) = x'(A(i)-1) = 0;
        if x'W ≤ C
            x = x';
        end
    end
end
```

Algorithm 1. not only considers the value density, but also solves the problem that in some cases, those items with large weight and value cannot enter the backpack.

3.3 Feasible solution generation and algorithm iteration

In order to improve the detection capability of GWO, researchers have adopted various methods. Saremi [31] adopted the strategy of survival of the fittest to improve the median fitness of the population. Heidari and Pahlavani introduced Levy flight and greedy selection strategies in

[32] to enhance the algorithm, but the algorithm time complexity is higher. In reference [33], Luo proposed a new model to dynamically estimate the position of the prey, as follow:

$$x_j^p(t) = \omega_\alpha x_j^\alpha(t) + \omega_\beta x_j^\beta(t) + \omega_\delta x_j^\delta(t) + \varepsilon(t) \tag{17}$$

where ω_α, ω_β and ω_δ respectively denotes the adaptive weights of alpha wolf, beta wolf and delta wolf, and are obtained by the following formulas:

$$\omega_\alpha = \frac{f(x_\alpha)}{f(x_\alpha)+f(x_\beta)+f(x_\delta)} \tag{18}$$

$$\omega_\beta = \frac{f(x_\beta)}{f(x_\alpha)+f(x_\beta)+f(x_\delta)}$$

$$\omega_\delta = \frac{f(x_\delta)}{f(x_\alpha)+f(x_\beta)+f(x_\delta)}$$

$$\omega_\alpha + \omega_\beta + \omega_\delta = 1 \tag{19}$$

$$0 < \omega_\alpha, \omega_\beta, \omega_\delta \leq 1$$

Reference [31] has proved that the performance of GWO based on dynamic weight is better than the original algorithm.

Combining the above methods of generating new individuals, we obtain the iterative operator of this paper. When $g > 1$, we have Eq. (15) for global search; when $g < 1$, we have the following formula, mainly for local search:

$$y = x_j^k(t+1) = x_j^p(t) - g\left|2rx_j^p(t) - x_j^k(t)\right| \tag{20}$$

where, $r \in (0, 1)$ is a random number.

Since the individual wolves are discrete binary values, it is also necessary to introduce a transform function. The selection of the transform function should be based on the actual situation, where the discussion will be introduced in Sect. 5. From the above, the discretization coding formula of individual wolf is as follow:

$$z_j^k = \begin{cases} 1, & \text{if } rand < \varphi(y) \\ 0, & otherwise \end{cases} \tag{21}$$

During the comparative experiment in this paper, the transform function is $\varphi(y) = 1/(1 + e^{-10*(y-0.5)})$.

So we obtain the adaptive grey wolf optimization algorithm based on the search agent parameter g, called adaptive gray wolf optimization with differential evolution operator (de-AGWO), See Algorithm 2:

Algorithm 2.

Input: $P, W, x, C, popsize, n, MaxIt$

Output: X_{best}, f_{best}

1: Initialization: $x_\alpha, x_\beta, x_\delta$; $f_\alpha, f_\beta, f_\delta = 0$;
$WP = round(rand(popsize, 3n))$;
2: while $t < MaxIt$
3: for $i = 1: popsize$
4: repair $WP(i)$ by Algorithm 1;
5: $f(i) = WP(i) * P$;
6: if $f(i) > f_\alpha$
7: $f_\alpha = f(i)$ and $x_\alpha = WP(i)$;
8: elseif $f(i) < f_\alpha$ && $f(i) > f_\beta$
9: $f_\beta = f(i)$ and $x_\beta = WP(i)$;
10: elseif $f(i) < f_\beta$ && $f(i) > f_\delta$
11: $f_\delta = f(i)$ and $x_\delta = WP(i)$;
12: end
13: $f_{best}(t) = f_\alpha$;
14: end
15: use eqs. (17)-(19) to get x_p
16: use eqs. (14) to get g
17: for $i = 1: popsize$
18: for $j = 1: 3n$
19: if $g > 1$
20: use eq. (15) to get new $WP(i, j)$;
21: else
22: use eq. (20) to get new $WP(i, j)$;
23: end
24: end
25: end
26: $t = t + 1$;
27: use eq. (21) to discretize $WP(i, j)$;
28: end

The time complexity of step.3–14 is $O(MaxIt * (popsize * 3n))$, and the time complexity of step.17–25 is $O(MaxIt * (popsize * 3n))$, so the time complexity of the algorithm is $O(MaxIt * popsize * n)$.

Unlike reference [24], this paper does not retain the evolutionary mechanism of external archives composed of three best different historical solutions, but adopts an elite retention strategy. The elite retention strategy can better describe the nature of the leading wolves in the wolf pack that will continue to replace during the population iteration, and the amount of its calculation is smaller.

The flowchart of the algorithm is shown in Fig. 1.

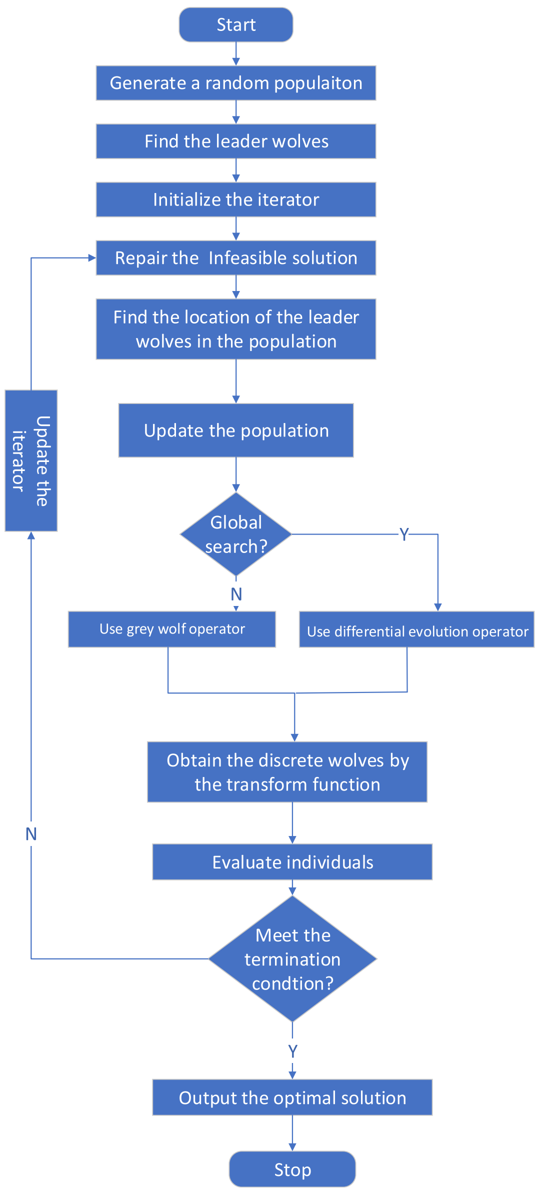


Fig. 1 The flowchart of de-AGWO

4 Experiments

The algorithms of this paper were written in MATLAB, and all the simulation platforms of them is Intel (R) Core (TM) i7-7700HQ CPU, 8.0 GB RAM, and the experimental environment is MATLAB2016a.

4.1 Function optimization experiments of de-AGWO

In Sect. 3, we have got the complete de-AGWO for solving the D {0–1} KP. In order to prove the performance of the algorithm in function optimization, we selected 15 standard test functions from reference [3] to conduct experiments and proved the superiority of the improved algorithm by comparing the traditional GWO and de-AGWO. The form of the standard test functions are shown in Table 2.

We selected 5 unimodal benchmark functions, 5 multimodal benchmark functions and 5 fixed-dimensional multimodal benchmark functions. We run 30 experiments on each function, the population size is fixed at 30, and the number of iterations from f_1 to f_{10} is fixed at 1000.

Because the multimodal benchmark functions need more iterations to see the difference, we choose the number of iterations to be 2000. The experimental results are as follow:

From Table 3, we can see that the solution performance of de-AGWO in function optimization has been significantly improved. It not only solves the defect that traditional GWO falls into local convergence when solving multimodal functions, but also greatly improves the accuracy of the algorithm as a whole. The experimental results prove that the algorithm in this paper is superior and feasible, which lays the foundation for us to solve practical problems later.

4.2 Experiments and analysis of the D {0–1} KP

In this section, we will compare 7 different intelligent algorithms through the results of the algorithm running on 40 general standard data sets. Among them, the running results of The FirEGA [2], MMBO [18], MS1 [4] and MDBBA [34] will directly quote the experimental results in the references.

The first four algorithm termination conditions in the comparison algorithm are the number of iterations, which are set to a constant equal to the dimension of the test set $(3 * n, n = 100, 200, \ldots, 1000)$. In order to ensure the fairness of the experiment, the population size of the algorithm is all 50, so the evaluation times of the fitness function are all greater than or equal to $50 * 3 * n$. Due to the obvious advantages of HBDE [4], BGWO and de-AGWO algorithms, the number of iterations of the algorithm here is a constant 100, so the number of function evaluations are all $50 * 100 = 5000$. It is worth mentioning that when the number of iterations *MaxIt* and the population size *popsize* are both linearly related to n, the time complexity of all algorithms in the comparative experiment is $O(n^3)$. It is worth explaining that the space complexity of the algorithms when solving the D {0–1} KP all defaults to the population size p * the capacity of the backpack n, i.e., $o(p*n)$, independently of the algorithm used.

Table 2 Benchmark functions

Function	Dim	Range	Min
$f_1(x) = \sum_{i=1}^{n} x_i^2$	30	[−100, 100]	0
$f_2(x) = \sum_{i=1}^{n} \lvert x_i\rvert + \Pi_{i=1}^{n}\lvert x_i\rvert$	30	[−10, 10]	0
$f_3(x) = \sum_{i=1}^{n} (\lvert x_i + 0.5\rvert)^2$	30	[−100, 100]	0
$f_4(x) = \max_i\{\lvert x_i\rvert, 1 \le i \le n\}$	30	[−100, 100]	0
$f_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	[−30, 30]	0
$f_6(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i\rvert})$	30	[−500, 500]	−418.9829 × 5
$f_7(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$	30	[−5.12, 5.12]	0
$f_8(x) = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{n} (x_i - 1)^2[1 + \sin^2(3\pi x_i + 1)] + (x_n - 1)^2[1 + \sin^2(2\pi x_n)]\}$	30	[−50, 50]	0
$+ \sum_{i=1}^{n} u(x_i, 5, 100, 4)$			
$f_9(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1$	30	[−600, 600]	0
$f_{10}(x) = \frac{\pi}{n}\{10\sin(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$	30	[−50, 50]	0
$y_i = 1 + \frac{x_i + 1}{4}$			
$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{cases}$			
$f_{11}(x) = (\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{j=1}^{2} (x_i - a_{ij})^6})^{-1}$	2	[−65, 65]	1
$f_{12}(x) = \sum_{i=1}^{11} [a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}]^2$	4	[−5, 5]	0.00030
$f_{13}(x) = \sum_{i=1}^{5} \lvert (X - a_i)(X - a_i)^T + c_i\rvert^{-1}$	2	[0, 10]	−10.1532
$f_{14}(x) = (x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos x_1 + 10$	2	[−5, 5]	0.398
$f_{15}(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$	2	[−2, 2]	3
$\times[30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$			

4.2.1 Algorithm comparison

Tables 4, 5, 6 and 7 shows the calculation results of 40 D {0–1} KP instances, and the experimental times are all 30. To ensure the fairness of the comparison experiment, we applied the same mathematical model (the first mathematical model) and the same repair mechanism. From the figure we can see the best value, the mean value, the worst value and the standard deviation (std) of the results of the algorithms. The optimal solution (opt) of the instance is shown in the first column. As part of the data in the table is quoted from different references, there may be missing. Some missing algorithm calculation results and std values have been shown in the table. The parameter settings of the algorithm are consistent with the references.

From the results of the four examples, it can be seen that in the UDKP and WDKP instances, the de-AGWO solution is slightly better than the HBDE algorithm; in the SDKP instances, the de-AGWO algorithm has significant advantages, indicating that the de-AGWO algorithm is suitable for data sets with strong correlation; HBDE only has a greater advantage in the IDKP instances. On the whole, we can also conclude that the de-AGWO algorithm has absolute advantages when the amount of data is small.

From the perspective of standard deviation, we find that the standard deviation of the de-AGWO algorithm is always slightly larger than that of HBDE, indicating that the algorithm in this paper has good population diversity in the search process, the search is more comprehensive, and the possibility of search stagnation is less.

4.2.2 Further analysis

In order to further compare the pros and cons of HBDE, BGWO and de-AGWO, we chose the run time and the Gap chart of the result as the basis for judgment.

Figure 2 shows that under the four data sets, the run time of HBDE is always much higher than that of GWO, indicating that GWO has a significant advantage in run time, although the time complexity of the two algorithms is approximately equal.

Table 3 Results of benchmark functions

Function	Algorithm	Mean	Std	Worst	Best
1	GWO	2.75E-22	1.91E-22	8.57E-22	1.11E-22
	de-AGWO	8.46E-33	2.07E-32	6.84E-32	5.66E-43
2	GWO	1.81E-12	5.21E-13	2.82E-12	8.6E-13
	de-AGWO	1.99E-21	3.02E-21	1.02E-20	7.6E-24
3	GWO	3.789854	0.6685	5.020351	1.772087
	de-AGWO	0.77773	0.425724	1.626108	8.95E-05
4	GWO	0.000281	0.000146	0.000671	6.61E-05
	de-AGWO	1.33E-06	2.09E-06	8.4E-06	1.89E-08
5	GWO	28.48963	0.514144	28.93085	27.13229
	de-AGWO	26.81932	0.967778	28.75627	25.65882
6	GWO	−5817.44	1023.341	−3904.11	−8651.28
	de-AGWO	−5697.36	65.66914	−5593.11	−5814.77
7	GWO	36.05262	11.00228	72.74691	18.86379
	de-AGWO	2.509096	8.879194	44.65311	0
8	GWO	2.248438	0.427678	3.133562	1.484606
	de-AGWO	0.793723	0.347946	1.406806	0.133093
9	GWO	0.005895	0.006509	0.015943	7.77E-16
	de-AGWO	0.00053	0.002902	0.015894	0
10	GWO	1.383046	0.466934	2.642859	0.429232
	de-AGWO	0.078053	0.104385	0.601156	0.011758
11	GWO	10.29463	5.256509	18.30431	0.998004
	de-AGWO	0.998004	3.18E-11	0.998004	0.998004
12	GWO	0.002401	0.006158	0.020942	0.000307
	de-AGWO	0.000674	0.000456	0.001223	0.000307
13	GWO	−7.98372	3.197686	−2.63047	−10.1532
	de-AGWO	−8.96656	2.187424	−5.0552	−10.1532
14	GWO	0.397887	5.16E-08	0.397888	0.397887
	de-AGWO	0.39789	1.13E-05	0.397949	0.397887
15	GWO	10.20001	21.19271	84	3
	de-AGWO	3.000004	6.42E-06	3.000028	3

The value of *best* is an important index to evaluate the performance of the algorithm. In order to further compare the performance of the three algorithms, this paper introduces an evaluation index *Gap*, the formula is as follow:

$$Gap = \frac{|opt - best|}{opt} \tag{22}$$

where *opt* is the theoretical optimal value and *best* is the mean values obtained by our experiments.

It can be seen from Fig. 3 that the de-AGWO of the UDKP instances is slightly worse than HBDE; in the other instances, de-AGWO is significantly better than HBDE; especially in the IDKP instances, de-AGWO can obtain the optimal solution for almost all cases. It is worth mentioning that de-AGWO is almost always better than HBDE when the amount of data is small. The average *Gap* graph analysis of the results is basically consistent with the analysis based on the tables.

4.3 Additional experiments and analysis

In the first two sections, we have proved the superiority of our algorithm through experiments. Now we will discuss the selection of the transformation function and the use ratio of two operators of de-AGWO. The former ensures the optimal transformation function of the algorithm, and the latter shows that the main iterative mode of de-AGWO still depends on the grey wolf evolution operator.

4.3.1 Transform function

Since the original GWO is a heuristic algorithm for solving continuous problems, the transform function plays a pivotal role in solving binary problems [24]. There are two types of transform functions: S-type and V-type. In order to find a suitable transform function for the D {0–1} KP, this paper

Table 4 Results on the best, mean, worst and std for UDKP instances

	FirEGA	MMBO	MS1	MDBBA	HBDE	BGWO	de-AGWO	
UDKP1 opt: 85,740	Best	80,101	82,703	84,200	78,939	85,624	85,503	**85,643**
	Mean	79,325.3	79,406.0	82,763.0	77,706.9	85,498.1	85,325.7	85,591.1
	Worst	78,499	75,624	81,131	76,370	85,451	85,230	85,437
	Std	–	–	–	–	52.7	73.5	56.3
UDKP2 opt: 163,744	Best	152,969	158,465	161,133	152,633	163,153	162,616	**163,325**
	Mean	151,045.2	155,976.0	158,503.0	150,162.6	162,964.7	162,368.6	162,924.9
	Worst	149,732	153,570	155,911	148,682	162,784	162,172	162,610
	Std	–	–	–	–	138.6	132.9	178.7
UDKP3opt: 269,393	Best	244,291	253,629	251,954	242,408	268,358	268,203	**268,532**
	Mean	241,061.2	246,651.0	249,646.0	238,121.1	268,227.1	267,916.0	268,330.7
	Worst	239,114	242,352	244,938	234,591	268,067	267,665	268,079
	Std	–	–	–	–	76.8	136.7	110.0
UDKP4opt: 347,599	Best	319,680	333,253	332,554	321,661	346,318	345,813	**346,414**
	Mean	316,503.4	329,155.0	320,776.0	318,714.3	346,214.2	345,602.5	346,007.5
	Worst	313,141	315,914	315,150	315,742	346,125	345,464	345,773
	Std	–	–	–	–	53.5	89.6	146.6
UDKP5opt: 442,644	Best	403,908	414,526	405,222	402,836	**439,458**	438,930	439,428
	Mean	399,525.2	403,898.0	400,653.0	398,440.6	439,237.6	438,452.5	439,023.5
	Worst	396,937	395,473	395,533	394,194	439,073	438,218	438,724
	Std	–	–	–	–	111.6	188.0	211.4
UDKP6opt: 536,578	Best	483,350	486,156	487,014	479,233	533,233	532,794	**533,520**
	Mean	478,779.5	480,552.0	481,401.0	474,065.1	533,103.1	532,477.3	533,075.1
	Worst	474,558	474,406	476,628	470,267	532,984	532,266	532,828
	Std	–	–	–	–	82.7	142.4	166.1
UDKP7: opt: 635,860	Best	564,656	615,617	618,146	569,559	**633,401**	632,636	633,092
	Mean	559,815.4	608,351.0	604,287.0	565,561.6	633,224.4	632,388.3	632,790.4
	Worst	555,763	599,086	588,175	560,807	632,969	632,229	632,355
	Std	–	–	–	–	121.7	97.1	150.9
UDKP8:opt: 650,206	Best	590,237	617,036	596,452	593,212	**646,959**	645,841	646,682
	Mean	584,264.3	610,379.0	581,196.0	589,469.9	646,834.9	645,387.9	646,209.4
	Worst	580,258	603,906	575,279	585,734	646,638	645,152	645,783
	Std	–	–	–	–	85.8	188.1	220.6
UDKP9opt: 718,532	Best	652,354	687,790	661,984	649,101	**715,353**	714,194	714,439
	Mean	646,592.2	683,032.0	652,572.0	646,157.2	714,915.4	713,755.2	714,056.6
	Worst	642,965	677,702	644,955	642,170	714,553	713,538	713,723
	Std	–	–	–	–	219.5	154.3	198.9
UDKP10opt: 779,460	Best	708,744	755,675	719,003	713,574	**774,075**	772,318	773,268
	Mean	703,947.8	748,568.0	713,858.0	707,078.1	773,847.5	772,140.5	772,765.9
	Worst	700,702	739,292	706,131	701,777	773,577	771,994	772,470
	Std	–	–	–	–	122.7	104.0	212.2

Result reaches the theoretical optimal solution for the benchmark are shown in bold

selects a total of 5 transform functions of two types, see Table 8.

The following comparison experiments are completed through a set of smaller test sets and a set of larger test sets in the SDKP and IDKP examples. The algorithm is run 30 times in total, and we get the box plot Fig. 4 of the program running results after 30 runs.

It can be seen from Fig. 4 that the V-type transformation function has a relatively stable performance in different instances and the size of the data set; the S-1 type has the best population diversity but the operating result is the

Table 5 Results on the best, mean, worst and std for WDKP instances

	FirEGA	MMBO	MS1	MDBBA	HBDE	BGWO	de-AGWO	
WDKP1opt: 83,098	Best	82,722	83,024	83,074	82,803	**83,096**	83,090	83,090
	Mean	82,539.6	82,524.0	82,947.0	82,755.3	83,086.0	83,061.2	83,078.53
	Worst	82,454	80,568	82,800	82,681	83,075	83,051	83,065
	Std	–	–	–	–	6.7	12.6	11.1
WDKP2opt: 138,215	Best	137,712	138,004	138,143	137,791	138,161	138,175	**138,194**
	Mean	137,225.8	137,747.0	137,989.0	137,728.4	138,147.1	138,106.0	138,160.6
	Worst	136,983	137,275	137,840	137,681	138,134	138,074	138,131
	Std	–	–	–	–	7.4	38.6	15.3
WDKP3opt: 256,616	Best	254,234	255,684	248,982	254,481	256,438	256,415	**256,507**
	Mean	253,294.4	254,214.0	248,318.0	254,290.9	256,447.9	256,325.4	256,439.7
	Worst	252,909	249,696	247,714	254,253	256,398	256,285	256,346
	Std	–	–	–	–	25.2	30.1	42.8
WDKP4opt: 315,657	Best	314,107	315,007	314,905	314,302	315,478	315,484	**315,530**
	Mean	312,343.1	314,621.0	314,612.0	314,271.5	315,454.1	315,452.3	315,473.8
	Worst	310,665	313,880	314,321	314,255	315,430	315,433	315,429
	Std	–	–	–	–	14.7	15.7	25.1
WDKP5opt: 428,490	Best	426,783	427,666	427,530	426,953	428,179	428,105	**428,195**
	Mean	424,384.2	427,038.0	427,173.0	426,810.3	428,114.1	428,062.4	428,097.3
	Worst	421,584	425,553	426,876	427,683	428,063	428,028	428,051
	Std	–	–	–	–	36.6	21.9	39.1
WDKP6opt: 466,050	Best	463,870	465,222	464,993	464,332	465,948	465,929	**465,984**
	Mean	460,750.4	464,299.0	464,701.0	464,285.5	465,907.5	465,892.8	465,920
	Worst	455,201	461,746	464,383	464,253	465,852	465,861	465,874
	Std	–	–	–	–	19.9	20.3	26.3
WDKP7: opt: 547,683	Best	544,059	546,716	545,607	544,861	547,498	547,481	**547,508**
	Mean	541,505.3	545,823.0	545,241.0	544,784.7	547,461.7	547,438.2	547,461.5
	Worst	535,551	544,933	544,912	544,776	547,409	547,407	547,413
	Std	–	–	–	–	20.8	17.6	27.8
WDKP8: opt: 576,959	Best	574,201	575,850	575,387	574,959	**576,752**	576,612	576,732
	Mean	571,594.9	575,297.0	575,126.0	574,882.8	576,684.2	576,568.0	576,662.8
	Worst	565,119	573,694	574,920	574,869	576,600	576,501	576,541
	Std	–	–	–	–	54.7	30.3	41.1
WDKP9opt: 650,660	Best	647,012	649,871	649,059	648,621	**650,446**	650,436	650,436
	Mean	644,298.2	649,600.0	648,813.0	648,607.2	650,417.0	650,376.6	650,385.3
	Worst	639,241	649,135	648,611	648,580	650,350	650,346	650,342
	Std	–	–	–	–	34.2	23.5	25.3
WDKP10opt: 678,967	Best	677,359	678,464	677,817	677,381	678,685	678,743	**678,791**
	Mean	643,776	678,216.0	677,581.0	677,375.2	678,664.3	678,685.4	678,701.1
	Worst	660,332	677,800	677,401	677,359	678,643	678,657	678,643
	Std	–	–	–	–	20.5	23.4	40.2

Result reaches the theoretical optimal solution for the benchmark are shown in bold

worst; on the whole, the S-2 type is the best and can meet the needs of the transformation function to solve the D {0–1} KP.

4.3.2 The use ratio of the two operators

In order to study the usage of the two evolutionary operators, we set two parameters respectively to record the usage times of the two kinds of operators.

Table 6 Results on the best, mean, worst and std for SDKP instances

	FirEGA	MMBO	MS1	MDBBA	HBDE	BGWO	de-AGWO	
SDKP1opt: 94,459	Best	93,316	93,686	94,030	93,401	94,416	94,332	**94,423**
	Mean	93,192.8	93,222.1	93,695.0	93,102.4	94,400.0	94,282.8	94,387.5
	Worst	93,064	92,371	93,376	92,927	94,388	94,242	94,361
	Std	–	–	–	–	10.5	26.7	15.3
SDKP2opt: 160,805	Best	159,116	159,881	159,019	159,265	160,518	160,436	**160,618**
	Mean	158,936.7	159,442.2	158,082.0	159,140.3	160,457.0	160,373.9	160,521.6
	Worst	158,798	158,433	155,574	159,088	160,389	160,299	160,394
	Std	–	–	–	–	37.5	45.1	57.9
SDKP3opt: 238,248	Best	235,372	236,896	236,634	235,623	**238,118**	237,943	**238,118**
	Mean	235,204.4	236,208.7	236,070.0	235,498.2	237,997.1	237,881.7	238,016.4
	Worst	235,015	232,114	235,765	235,474	237,865	237,854	237,882
	Std	–	–	–	–	57.6	32.7	52.4
SDKP4opt: 340,027	Best	336,369	338,392	337,954	336,813	339,239	338,939	**339,594**
	Mean	335,844.7	337,522.0	337,248.0	336,681.0	339,134.7	338,850.6	339,339.2
	Worst	335,524	336,733	336,834	336,631	339,067	338,776	339,143
	Std	–	–	–	–	47.1	59.7	102.3271
SDKP5opt: 463,033	Best	451,184	457,678	455,491	452,908	461,889	461,748	**462,187**
	Mean	447,335.9	454,344.0	454,026.0	450,961.1	461,847.2	46,162.7	462,037.3
	Worst	444,252	452,356	452,553	448,084	461,754	461,517	461,888
	Std	–	–	–	–	44.0	60.6	86.0
SDKP6opt: 466,097	Best	459,236	462,237	461,242	460,036	465,039	464,742	**465,179**
	Mean	458,746.1	460,603.0	460,729.0	459,904.2	464,904.0	464,655.9	464,949.7
	Worst	458,427	457,323	460,178	459,834	464,769	464,599	464,741
	Std	–	–	–	–	74.1	45.6	124.7
SDKP7: opt: 620,446	Best	607,200	614,167	609,852	607,838	619,337	619,173	**619,578**
	Mean	602,797.7	610,971.0	608,712.0	606,565.4	619,234.1	619,086.1	619,373.1
	Worst	600,496	606,124	604,763	603,387	619,154	618,997	619,205
	Std	–	–	–	–	43.4	45.2	91.3
SDKP8: opt: 670,697	Best	661,104	665,183	663,804	662,718	669,037	668,919	**669,377**
	Mean	659,844.6	663,766.0	663,103.0	662,565.9	668,981.1	668,836.2	669,174.8
	Worst	659,120	649,495	662,574	662,515	668,899	668,783	668,981
	Std	–	–	–	–	44.7	33.0	102.3
SDKP9opt: 739,121	Best	728,443	734,825	731,439	730,185	737,433	737,363	**737,682**
	Mean	727,364.5	733,517.0	730,654.0	730,048.4	737,334.6	737,251.7	737,386.6
	Worst	726,872	732,477	730,204	729,979	737,245	737,168	737,184
	Std	–	–	–	–	56.4	48.4	104.1
SDKP10opt: 765,317	Best	755,189	760,814	757,821	757,128	**763,846**	763,629	763,770
	Mean	752,931	759,625.0	757,466.0	756,903.3	763,674	763,577.8	763,654
	Worst	749,879	757,750	757,158	756,851	763,582	763,534	763,574
	Std	–	–	–	–	67.4	29.7	51.1

Result reaches the theoretical optimal solution for the benchmark are shown in bold

The data set used in the experiment is the IDKP instances with a small standard deviation. Each instance is calculated 30 times, the usage of the operator is recorded, and the average value is taken for analysis.

where the parameters *de* and *gw* respectively represent the number of times which the differential evolution operator and the grey wolf evolution operator are used in 100 iterations.

From Table 9, we can find that with the increase of the data set, the usage of the differential evolution operator increases, indicating that the proportion of the algorithm in

Table 7 Results on the best, mean, worst and std for IDKP instances

	FirEGA	MDBBA	HBDE	BGWO	de-AGWO	
IDKP1opt: 70,106	Best	**70,106***	70,077	**70,106***	**70,106***	**70,106***
	Mean	70,078.0	70,007.7	70,099.0	70,094.6	70,099.5
	Worst	70,022	69,841	70,090	70,090	70,077
	Std	–	–	7.3	7.0	7.4
IDKP2opt: 118,268	Best	118,034	118,268	**118,268***	**118,268***	**118,268***
	Mean	117,544.3	118,196.7	118,262.0	118,264.9	118,255.2
	Worst	117,249	117,988	118,232	118,232	118,232
	Std	–	–	11.6	9.7	14.0
IDKP3opt: 234,804	Best	234,508	234,802	**234,804***	234,803	**234,804***
	Mean	233,896.3	234,771.2	234,802.1	234,800.2	234,801.2
	Worst	233,447	234,658	234,799	234,799	234,789
	Std	–	–	1.6	1.5	3.1
IDKP4opt: 282,591	Best	281,804	282,583	**282,591***	**282,591***	**282,591***
	Mean	280,536.6	282,557.2	282,590.3	282,584.9	282,584.3
	Worst	278,179	282,474	282,582	282,576	282,577
	Std	–	–	2.0	4.8	4.5
IDKP5opt: 335,584	Best	335,068	335,580	**335,584***	**335,584***	**335,584***
	Mean	332,180.2	335,569.1	335,582.1	335,583.2	335,581.6
	Worst	328,661	335,514	335,580	335,580	335,580
	Std	–	–	1.7	1.6	1.9
IDKP6opt: 452,463	Best	451,498	452,457	452,462	**452,463***	**452,463***
	Mean	449,781	452,436.2	452,460.1	452,454.7	452,458.3
	Worst	446,456	452,337	452,448	452,452	452,445
	Std	–	–	4.1	3.0	4.3
IDKP7: opt: 489,149	Best	487,675	489,132	489,146	489,137	**489,149***
	Mean	484,305.8	489,104.9	489,142.0	489,134.9	489,135.1
	Worst	475,476	489,045	489,127	489,124	489,122
	Std	–	–	5.5	4.5	6.2
IDKP8: opt: 533,841	Best	531,872	533,833	**533,834**	533,830	**533,834**
	Mean	529,372.8	533,821.2	533,831.9	533,822.9	533,830.2
	Worst	524,404	533,702	533,829	533,717	533,817
	Std	–	–	1.7	6.2	4.3
IDKP9opt: 528,144	Best	525,460	528,133	529,138	528,139	**528,139**
	Mean	522,243.5	528,113.6	528,147.6	528,138.9	528,131
	Worst	501,428	528,051	528,129	528,136	528,119
	Std	–	–	2.7	0.7	6.8
IDKP10opt: 581,244	Best	578,897	581,238	**581,244***	581,225	**581,244***
	Mean	575,128.5	581,237.1	581,240.0	581,225	581,231.8
	Worst	551,772	581,225	581,228	581,225	581,225
	Std	–	–	4.9	0.0	5.1

Result reaches the theoretical optimal solution for the benchmark are shown in bold

the global search increases, which is consistent with the actual demand.

In general, GWO's operator accounts for about 84%, indicating that the grey wolf algorithm is still the main part.

5 Discuss

From the perspective of function optimization, our experimental results have shown that the de-AGWO has achieved a significant improvement in optimization capability across all benchmark sets. It effectively addresses the issue of local convergence and can be applied to

Table 8 The transform functions

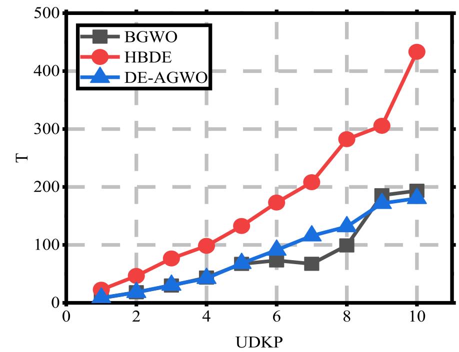
type	Transform function
$S-1$	$\frac{1}{e^{y}}$
$S-2$	${}^{1}/_{1\,+\,\mathrm{e}^{-10(\mathrm{y}-0.5)}}$
$V-1$	$\lvert\tanh y\rvert$
$V-2$	$\left\lvert {}^{y}\big/\sqrt{1+y^{2}} \right\rvert$
$V-3$	$\lvert\arctan \pi/2y\rvert$

challenging function optimization scenarios, including fixed-dimensional multimodal benchmark functions.

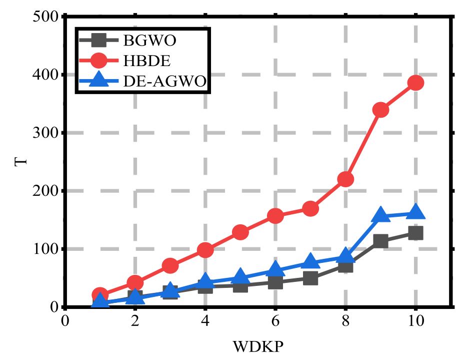
Meanwhile, a comparative analysis of specific instances reveals that the de-AGWO exhibits significantly superior experimental performance on strongly correlated datasets (SDKP) compared to other algorithms, rendering it more suitable for real-world scenarios, given that most practical datasets exhibit strong correlation. Moreover, de-AGWO demonstrates excellent performance on other datasets as well, particularly when dealing with small-scale data, presenting an absolute advantage for de-AGWO. To ensure the algorithm achieves optimal results, we meticulously select the most appropriate transformation function (S-2 type) through experimental evaluation. Importantly, the experimental results demonstrate that the core of the algorithm remains unchanged, with the grey wolf evolution operator still constituting the main iterative process, accounting for over 80% of the total number of iterations.

The de-AGWO proposed in this paper guarantees the status of the three leader wolves in the population (always actually exist in the population), and the algorithm is easy to implement. The most important improvement is that the algorithm in this paper uses two iterative operators (differential evolution operator and gray wolf evolution operator) for the value of search agent parameter g, and adopts an adaptive iterative formula for the position of the head wolf, which effectively makes up the defect that GWO's global search and local search cannot be considered at the same time. According to the experimental results, the proposed de-AGWO algorithm is better than other heuristic algorithms in comprehensive performance, and its framework is universal.

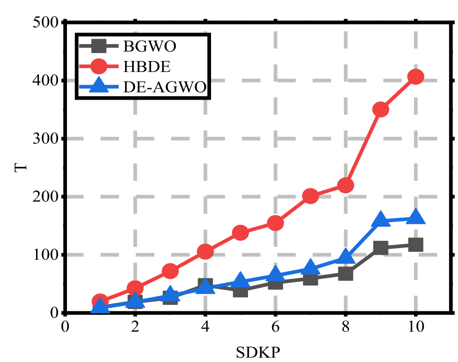
Nevertheless, our algorithm does have certain limitations. Like other heuristic algorithms, we are unable to mathematically prove the superiority of this algorithm. Furthermore, de-AGWO is specifically designed for the discounted {0–1} knapsack problem, and it needs to build a new model when applied to other specialized knapsack problems. Although GWO has the advantage of fewer parameters, the selection of transform function and the design of repair operator are still difficult problems to be faced when solving specific problems.



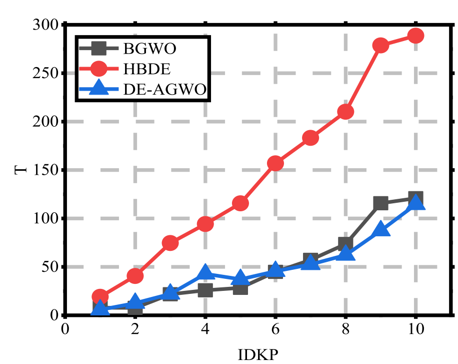
a) the run time of the UDKP instances



b) the run time of the WDKP instances

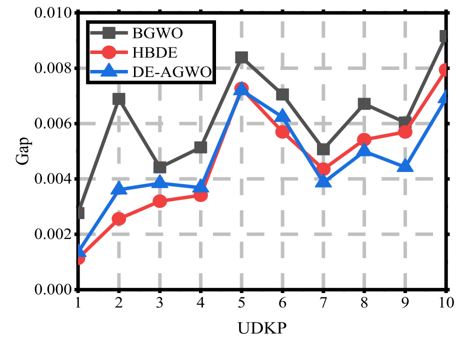


c) the run time of the SDKP instances

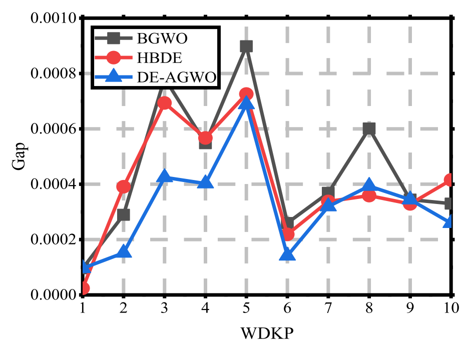


d) the run time of the IDKP instances

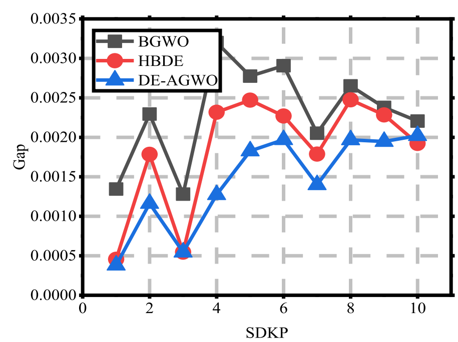
Fig. 2 The run time of three algorithms on four DKP instances



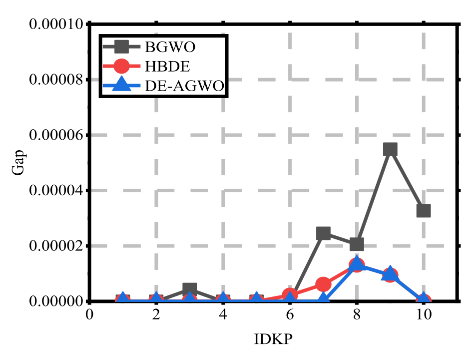
a) *Gap* for UDKP instances



b) *Gap* for WDKP instances

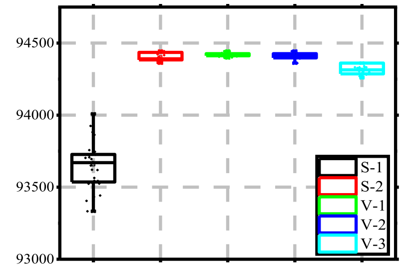


c) *Gap* for SDKP instances

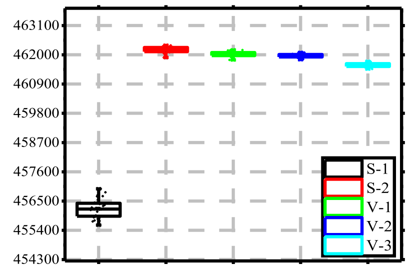


d) *Gap* for IDKP instances

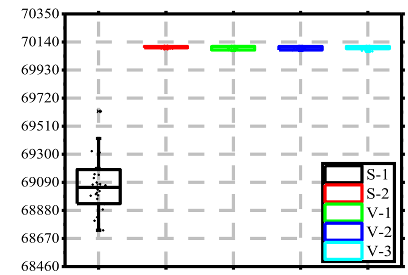
Fig. 3 *Gap* of three algorithms on two DKP instances



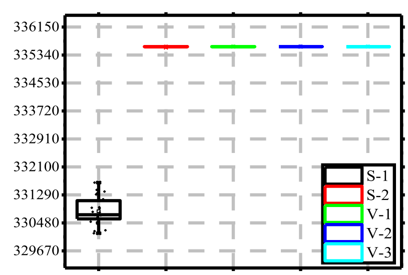
a) Boxplot of the five transform functions on SDKP1



b) Boxplot of the five transform functions on SDKP5



c) Boxplot of the five transform functions on IDKP1



d) Boxplot of the five transform functions on IDKP5

Fig. 4 The boxplot of the five transform functions on four DKP

6 Conclusion

This paper discussed a variant of KP. For the D {0–1} KP, BGWO is used for the first time, and we improve the wolf pack by using an adaptive wolf pack update method. The most important improvement is that we decide to use the differential evolution operator or the grey wolf evolution operator by the value of the search agent parameters, which

Table 9 The use ratio of the two operators

IDKP	*de*	*gw*
1	11.7	88.3
2	14.1	85.9
3	12.2	87.8
4	13.5	86.5
5	13.8	86.2
6	16.0	84.0
7	20.5	79.5
8	15.9	84.1
9	22.7	77.3
10	21.2	78.8
Avg	16.16	83.84

can make the algorithm have excellent performance in both global search and local search. This paper also introduced a greedy repair operator based on the D {0–1} KP. The proposed de-AGWO algorithm has the best comprehensive test performance on the four general data sets of the D {0–1} KP. Experiments show that de-AGWO has the richest population diversity, and the comprehensive consideration of average optimal value and run time is the best.

In addition, we compared and found the best suitable transform function for this experiment by setting different transform functions. We also discussed the proportion of the two operators used in the entire algorithm, and the results show that the grey wolf operator still occupies a dominant position. It is worth mentioning that, except that the binary discrete coding and repair operator are adjusted for D {0–1} KP, the overall framework of the algorithm is common to KP. When solving other binary discrete optimization problems, only need to redesign individual coding rules and repair operator.

The research on D {0–1} KP in this paper avoids the problem of how to generate an initial population of elites. This factor has a significant impact on the performance of the heuristic algorithm, which is also a challenge we will face in the future.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability The hyperlinks to the datasets used in this paper are as follows: https://github.com/whutcold/data.

References

1. Guldan B (2007) Heuristic and exact algorithms for discounted knapsack problems. Master thesis, University of Erlangen-Nu ̈rnberg, Germany.
2. He Y-C, Wang X-Z, Li W-B et al (2016) Research on genetic algorithms for the discounted 0–1 knapsack problem. Chinese J Comput 39(12):2614–2630
3. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
4. Zhu H, He Y, Wang X, Tsang ECC (2017) Discrete differential evolutions for the discounted 0–1 knapsack problem. Int J Bio-Inspired Comput 10(4):219
5. Kellerer H, Pferschy U, Pisinger D (2004) Knapsack problems. Springer, Berlin
6. Azad AK, Rocha AMAC, Fernandes EMGP (2014) A simplified binary artificial fish swarm algorithm for 0–1 quadratic knapsack problems'. J Comput Appl Mathematics 259(4):897–904
7. Babukarthik RG, Dhasarathan C, Kumar M, Shankar A, Thakur S, Cheng X (2021) A novel approach for multi-constraints knapsack problem using cluster particle swarm optimization. Comput Electric Eng 96:107399
8. Li VC, Curry GL (2005) Solving multidimensional knapsack problems with generalized upper bound constraints using critical event tabu search. Comput Oper Res 32(4):825–848
9. Poirriez V, Yanev N, Andonov R (2009) A hybrid algorithm for the unbounded knapsack problem. Discret Optim 6(1):110–124
10. Lai X et al. (2020) Diversity-preserving quantum particle swarm optimization for the multidimensional knapsack problem. Exp Syst Appl (2020):113310.
11. Syarif A et al (2020) Comparing various genetic algorithm approaches for multiple-choice multi-dimensional knapsack problem (mm-KP). Int J Intell Eng Syst 13:455–462
12. Xie X, Feng JL (20017) A mini-swarm for the quadratic knapsack problem. Swarm Intell Symposium IEEE, 2007:190–197.
13. Lin GY, Lu Y, Yao DD (2008) The stochastic knapsack revisited: switch-over policies and dynamic pricing. Oper Res 56(4):945–957
14. Dizdar D, Gershkov A, Moldovanu B (2011) Revenue maximization in the dynamic knapsack problem. Theor Econ 6(2):157–184
15. Rong A, Figueira JR, Klamroth K (2012) Dynamic programming based algorithms for the discounted 0–1 knapsack problem. Appl Math Comput 218(12):6921–6933
16. He YC, Wang XZ, He YL, Zhao SL, Li WB (2016) Exact and approximate algorithms for discounted 0–1 knapsack problem. Inf Sci 369:634–647
17. Feng Y, Wang GG, Li W, Li N (2017) Multi-strategy monarch butterfly optimization algorithm for discounted {0–1} knapsack problem. Neural Comput Applic: 1–18.
18. Feng YH, Wang GG (2018) Binary moth search algorithm for discounted 0–1 knapsack problem. IEEE Access 6(99):10708–10719
19. Cauteruccio F, Terracina G, Ursino D (2020) Generalizing identity-based string comparison metrics: framework and techniques. Knowledge-Based Syst, 187(Jan.), 104820.1–104820.17.
20. Jayabarathi T, Raghunathan T, Adarsh BR, Suganthan PN (2016) Economic dispatch using hybrid grey wolf optimizer. Energy 111:630–641
21. Zhang S, Zhou Y, Li Z, Pan W (2016) Grey wolf optimizer for unmanned combat aerial vehicle path planning. Adv Eng Softw 99:121–136
22. Chandramohan D, Dumka A, Dhilipkumar V, Loganathan J (2020) Data dissemination for green-vanets communication: an

opportunistic optimization approach. Int J Pervasive Comput Commun, ahead-of-print.

23. Panwar LK, Reddy S, Verma A, Panigrahi BK, Kumar R (2018) Binary grey wolf optimizer for large scale unit commitment problem, Swarm. Evol Comput 38:251–266
24. Luo K, Zhao Q (2019) A binary grey wolf optimizer for the multidimensional knapsack problem. Appl Soft Comput 83:105645
25. Camacho Villalón CL, Thomas S, Dorigo M (2020) Grey wolf, firefly and bat algorithms: three widespread algorithms that do not contain any novelty. Swarm Intelligence: 12th International Conference, ANTS 2020, LNCS 12421, Springer, 121–133. DOI: https://doi.org/10.1007/978-3-030-60376-2_10.
26. Camacho-Villalón CL, Dorigo M, Stützle T (2022) Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by bestial metaphors. Int Trans Operat Res. https://doi.org/10.1111/itor.13176
27. Deb K, Datta R (2010) A fast and accurate solution of constrained optimization problems using a hybrid bi-objective and penalty function approach.:1–8.
28. Alsuwaiyel MH (2009) Algorithms design techniques and analysis. World Scientific Publishing Company, Singapore
29. Coello, Carlos AC (2002) Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. Comput Methods in Appl Mech Eng 191. 11–12(2002):1245–1287.
30. Michalewicz Z (1996) Evolutionary algorithms for constrained parameter optimization problems. Evol Comput 4(1):1–32
31. Saremi S, Mirjalili SZ, Mirjalili SM (2015) Evolutionary population dynamics and grey wolf optimizer. Neural Comput Appl 26(5):1257–1263
32. Heidari AA, Pahlavani P (2017) An efficient modified grey wolf optimizer with Lévy flight for optimization tasks. Appl Soft Comput 60:115–134
33. Luo K (2019) Enhanced grey wolf optimizer with a model for dynamically estimating the location of the prey. Appl Soft Comput 77:225–235
34. Wu C, He Y, Chen Y et al (2017) Mutated bat algorithm for solving discounted 0–1 knapsack problem. J Comput Appl (China) 37(5):1292–1299