

Exact and approximate algorithms for discounted {0-1} knapsack problem



Yi-Chao He^a, Xi-Zhao Wang^{b,*}, Yu-Lin He^b, Shu-Liang Zhao^c, Wen-Bin Li^a

^a College of Information Engineering, Hebei GEO University, Shijiazhuang 050031, Hebei, China

^b Big Data Institute, College of Computer Science & Software Engineering, Shenzhen University, Shenzhen 518060, Guangdong, China

^c College of Mathematics and Information Science, Hebei Normal University, Shijiazhuang 050024, Hebei, China

ARTICLE INFO

Article history:

Received 30 March 2016

Revised 14 June 2016

Accepted 15 July 2016

Available online 16 July 2016

Keywords:

Discounted {0-1} knapsack problem

Exact algorithm

Approximate algorithm

Dynamic programming

Particle swarm optimization

ABSTRACT

The Discounted {0-1} Knapsack Problem (D{0-1}KP) is an extension of the classical 0-1 knapsack problem (0-1 KP) that consists of selecting a set of item groups where each group includes three items and at most one of the three items can be selected. The D{0-1}KP is more challenging than the 0-1 KP because four choices of items in an item group diversify the selection of the items. In this paper, we systematically studied the exact and approximate algorithms for solving D{0-1}KP. Firstly, a new exact algorithm based on the dynamic programming and its corresponding fully polynomial time approximation scheme were designed. Secondly, a 2-approximation algorithm for D{0-1}KP was developed. Thirdly, a greedy repair algorithm for handling the infeasible solutions of D{0-1}KP was proposed and we further studied how to use binary particle swarm optimization and greedy repair algorithm to solve the D{0-1}KP. Finally, we used four different kinds of instances to compare the approximate rate and solving time of the exact and approximate algorithms. The experimental results and theoretical analysis showed that the approximate algorithms worked well for D{0-1}KP instances with large value, weight, and size coefficients, while the exact algorithm was good at solving D{0-1}KP instances with small value, weight, and size coefficients.

© 2016 Elsevier Inc. All rights reserved.

1. Introduction

Knapsack Problem (KP) [1,6,10] is a classical NP-hard problem in computer science, which has found many applications in various areas, such as business management, computational complexity, cryptology, and applied mathematics, and so on. KP has some extended versions, e.g., the unbounded KP, multiple-choice KP, and quadratic KP, etc. These KP variants [22,27,29,30,33] have been well studied and successfully solved with different techniques and methods in the past few decades.

Discounted {0-1} Knapsack Problem (D{0-1}KP) a latest variant of classical 0-1 KP, which was firstly proposed by Guldán [12] in 2007 and used the concept of *discount* to reflect the sales promotion in real business activities. D{0-1}KP is a 0-1 integer programming problem with $n + 1$ inequality constraints including the *discount* constraints and knapsack capacity constraints, while there is only one inequality constraint in the classical 0-1 KP, where n is the number of item groups. Due to the better expressive capability to real business sales, D{0-1}KP has obtained a large number of applications in investment

* Corresponding author.

E-mail addresses: heyichao119@163.com (Y.-C. He), xzawang@szu.edu.cn, xizhaowang@ieee.org (X.-Z. Wang), dryulinhe@yahoo.com (Y.-L. He).

decision and resource allocation. However, the algorithmic study on how to solve D{0-1}KP in a more effective way is rare. In [12], Guldán presented an exact algorithm to solve D{0-1}KP based on dynamic programming [3] and discussed how to use the heuristic algorithm to solve D{0-1}KP. Rong et al. [32] in 2012 solved D{0-1}KP by combining the special kernel of D{0-1}KP with exact algorithm proposed in [12]. Currently, the study on how to solve D{0-1}KP mainly focuses on the exact algorithm. As far as we know, there is no work which uses the approximate and evolutionary algorithms to solve D{0-1}KP.

Motivated by designing the high-performance and low-complexity algorithms for solving D{0-1}KP, we systematically studied the exact and approximate algorithms for D{0-1}KP in this article. The main contributions of this article included the following four aspects: (1) proposing a **New Exact** algorithm for D{0-1}KP (NE-DKP) with lower complexity than algorithm studied in [12] when the sum of value coefficients is less than knapsack capacity; (2) developing a fully **Polynomial-time** approximate scheme (Poly-DKP) to simplify the aforementioned exact algorithm NE-DKP; (3) presenting a **2-Approximation** algorithm (App-2-DKP) for D{0-1}KP based on greedy strategy; and (4) providing a **Particle Swarm Optimization** based **Greedy Repair** algorithm for D{0-1}KP (PSO-GRDKP). On four kinds of well-known instances from real applications, we tested the practical performances of proposed exact/approximate algorithms and analyzed their computation complexities and approximation rates. The experimental results and theoretical analysis showed that the approximate algorithms, i.e., Poly-DKP, App-2-DKP, and PSO-GRDKP, work well for the large scale D{0-1}KPs, while the exact algorithm, i.e., NE-DKP, is good at solving the small scale D{0-1}KPs.

The remainder of this article is organized as follows. In Section 2, we provide a preliminary of D{0-1}KP. In Section 3, we describe the new exact algorithm for D{0-1}KP. Sections 4 depicts three approximate algorithms for D{0-1}KP, respectively. In Section 5, we report experimental comparisons that demonstrate the feasibility and effectiveness of proposed exact and approximate algorithms. Finally, we give our conclusions and suggestions for further research in Section 6.

2. Preliminary

In this section, the definition, mathematical model, and existing exact algorithm of discounted {0-1} knapsack problem (D{0-1}KP) are described.

Definition 1 ((Discounted {0-1} knapsack problem) [12,32]). Given n item groups having 3 items and one knapsack with capacity C , where the items in the i -th ($i = 0, 1, \dots, n-1$) item group are denoted as $3i$, $3i+1$, and $3i+2$. The value coefficients of $3i$, $3i+1$, and $3i+2$ are p_{3i} , p_{3i+1} , and $p_{3i+2} = p_{3i} + p_{3i+1}$, respectively. The weight coefficients of $3i$, $3i+1$, and $3i+2$ are w_{3i} , w_{3i+1} , and w_{3i+2} , where w_{3i+2} is the discounted weight, $w_{3i} + w_{3i+1} > w_{3i+2}$, $w_{3i+2} > w_{3i}$, and $w_{3i+2} > w_{3i+1}$. D{0-1}KP is to maximize the total value of items which can be put into the knapsack, where at most one item is selected from each item group and the sum of weight coefficients is less than knapsack capacity C .

Without loss of generality, we assume that the value coefficient p_k , weight coefficient w_k ($k = 0, 1, \dots, 3n-1$), and knapsack capacity C are the positive integers, and $w_{3i+2} \leq C$ ($i = 0, 1, \dots, n-1$), $\sum_{i=0}^{n-1} w_{3i+2} > C$, then the mathematical model of D{0-1}KP is defined as [12]:

$$\max \sum_{i=0}^{n-1} (x_{3i}p_{3i} + x_{3i+1}p_{3i+1} + x_{3i+2}p_{3i+2}) \quad (1)$$

$$\text{s.t. } x_{3i} + x_{3i+1} + x_{3i+2} \leq 1, i = 0, 1, \dots, n-1, \quad (2)$$

$$\sum_{i=0}^{n-1} (x_{3i}w_{3i} + x_{3i+1}w_{3i+1} + x_{3i+2}w_{3i+2}) \leq C, \quad (3)$$

$$x_{3i}, x_{3i+1}, x_{3i+2} \in \{0, 1\}, i = 0, 1, \dots, n-1, \quad (4)$$

where, x_{3i} , x_{3i+1} , and x_{3i+2} represent whether the items $3i$, $3i+1$, and $3i+2$ are put into the knapsack: $x_k = 0$ indicates the item k ($k = 0, 1, \dots, 3n-1$) is not in knapsack, while $x_k = 1$ indicates the item k is in knapsack. It is worth noting that a binary vector $X = (x_0, x_1, \dots, x_{3n-1}) \in \{0, 1\}^{3n}$ is a potential solution of D{0-1}KP. Only if X meets both Eqs. (2) and (3), it is a feasible solution of D{0-1}KP.

D{0-1}KP obviously has the properties of optimal substructure and overlapping subproblem, thus the dynamic programming is an appropriate method to solve D{0-1}KP. Guldán in [12] provided the first dynamic programming based exact algorithm to solve D{0-1}KP. In order to distinguish Guldán's exact algorithm with our new exact algorithm, we abbreviate it as OE-DKP (Old Exact algorithm for D{0-1}KP). The recursion formula and computational complexity analysis of OE-DKP are as follows.

Let $P = \{p_k | k = 0, 1, \dots, 3n-1\}$ and $W = \{w_k | k = 0, 1, \dots, 3n-1\}$ be the value set and weight set, respectively. For each item group $i' \in \{0, 1, \dots, i\}$, at most one item is selected to put into knapsack. For all items in knapsack, $G[i, j]$ is the

maximum of total value when the sum of weight coefficients is less than the knapsack capacity j :

$$G[i, j] = \begin{cases} G[i-1, j], & \text{if } 0 \leq j < w_{3i} \\ \max[G[i-1, j], G[i-1, j-w_{3i}] + p_{3i}], & \text{if } w_{3i} \leq j < w_{3i+1} \\ \max \begin{bmatrix} G[i-1, j], G[i-1, j-w_{3i}] + p_{3i}, \\ G[i-1, j-w_{3i+1}] + p_{3i+1} \end{bmatrix}, & \text{if } w_{3i+1} \leq j < w_{3i+2} \\ \max \begin{bmatrix} G[i-1, j], G[i-1, j-w_{3i}] + p_{3i}, \\ G[i-1, j-w_{3i+1}] + p_{3i+1}, G[i-1, j-w_{3i+2}] + p_{3i+2} \end{bmatrix}, & \text{if } w_{3i+2} \leq j \leq C \end{cases} \quad (5)$$

The initial $G[0, j]$ is defined as

$$G[0, j] = \begin{cases} 0, & \text{if } 0 \leq j < w_0 \\ p_0, & \text{if } w_0 \leq j < w_1 \\ p_1, & \text{if } w_1 \leq j < w_2 \\ p_2, & \text{if } w_2 \leq j \leq C \end{cases} \quad (6)$$

Let Opt_1 denote the optimal value of OE-DKP. According to Eqs. (5) and (6), we can get $Opt_1 = G[n-1, C]$. OE-DKP is an exact algorithm with pseudo polynomial time and its computational complexity of OE-DKP is $O(nC)$.

3. NE-DKP: the new exact algorithm for D{0-1}KP

The recursion formula Eq. (5) of OE-DKP is designed based on “maximizing the total value with the given sum of weight coefficients”. This section presents a new exact algorithm for D{0-1}KP based on dynamic programming by deriving the recursion formula with principle of “minimizing the total weight with the given sum of value coefficients”. The new exact algorithm is abbreviated as NE-DKP.

The weight minimization principle is to minimize the sum of weight coefficients corresponding to items satisfying the constraint conditions (2) and (3) when the total value is given. Let $E[i, j]$ be the minimum of total weight when the sum of value coefficients is j , where $i = 1, 2, \dots, n-1$, $j = 0, 1, \dots, S$, $S = \sum_{i=0}^{n-1} p_{3i+2}$. When there is no the subset of items of which the sum of value coefficients is j , $E[i, j] = +\infty$. In addition, $E[0, j]$ satisfies $E[0, 0] = 0$, $E[0, p_0] = w_0$, $E[0, p_1] = w_1$, $E[0, p_2] = w_2$, and $E[0, j'] = +\infty$ ($j' \in \{0, 1, \dots, S\}$ and $j' \neq 0, p_0, p_1, p_2$). Assume $E[i-1, j]$ is known and $p_{3i} \leq p_{3i+1}$, $E[i, j]$ can be calculated as following procedures.

1. When $0 \leq j < p_{3i}$, because $p_{3i} \leq p_{3i+1} < p_{3i+2}$, the value coefficients of items $3i$, $3i+1$, and $3i+2$ are all larger than j . It indicates that these three items can not be put into knapsack, then we get $E[i, j] = E[i-1, j]$.
2. When $p_{3i} \leq j < p_{3i+1}$, there is only one item $3i$ whose value coefficient is less than j . The item $3i$ can be considered to put into knapsack, then we can get $E[i, j] = \min[E[i-1, j], E[i-1, j-p_{3i}] + w_{3i}]$.
3. When $p_{3i+1} \leq j < p_{3i+2}$, the value coefficients of items $3i$ and $3i+1$ are all less than j . They can all be put into knapsack, then we can get $E[i, j] = \min[E[i-1, j], E[i-1, j-p_{3i}] + w_{3i}, E[i-1, j-p_{3i+1}] + w_{3i+1}]$.
4. When $p_{3i+2} \leq j \leq S$, the value coefficients of three items in item group i are all less than j . All items can be put into knapsack, then we can get $E[i, j] = \min[E[i-1, j], E[i-1, j-p_{3i}] + w_{3i}, E[i-1, j-p_{3i+1}] + w_{3i+1}, E[i-1, j-p_{3i+2}] + w_{3i+2}]$.

Based on the above-mentioned analysis, the recursion formula of NE-DKP is derived as

$$E[i, j] = \begin{cases} E[i-1, j], & \text{if } 0 \leq j < p_{3i} \\ \min[E[i-1, j], E[i-1, j-p_{3i}] + w_{3i}], & \text{if } p_{3i} \leq j < p_{3i+1} \\ \min \begin{bmatrix} E[i-1, j], E[i-1, j-p_{3i}] + w_{3i}, \\ E[i-1, j-p_{3i+1}] + w_{3i+1} \end{bmatrix}, & \text{if } p_{3i+1} \leq j < p_{3i+2} \\ \min \begin{bmatrix} E[i-1, j], E[i-1, j-p_{3i}] + w_{3i}, \\ E[i-1, j-p_{3i+1}] + w_{3i+1}, E[i-1, j-p_{3i+2}] + w_{3i+2} \end{bmatrix}, & \text{if } p_{3i+2} \leq j \leq S \end{cases} \quad (7)$$

The initial $E[0, j]$ is defined as

$$E[0, j] = \begin{cases} 0, & \text{if } j = 0 \\ w_0, & \text{if } j = p_0 \\ w_1, & \text{if } j = p_1 \\ w_2, & \text{if } j = p_2 \\ +\infty, & \text{otherwise} \end{cases} \quad (8)$$

The optimal value of NE-DKP is $Opt_2 = \max_{j=1,2,\dots,S} [j | E[n-1, j] \leq C]$ which represents the total value of items in knapsack is j and the sum of weight coefficients is less than C .

Now, we discuss how to get the optimal solution $X = (x_0, x_1, \dots, x_{3n-1}) \in \{0, 1\}^{3n}$ corresponding to Opt_2 by using Eqs. (7) and (8). Without loss of generality, we assume we have selected the items from item groups $n-1, n-2, \dots, i+1$. According to Eq. (7) and the value of $E[i, j]$, we discuss whether there is an item which should be selected from item group i .

1. If $E[i, j] = E[i-1, j]$, $E[i, j] \neq E[i-1, j-p_{3i}] + w_{3i}$, $E[i, j] \neq E[i-1, j-p_{3i+1}] + w_{3i+1}$, and $E[i, j] \neq E[i-1, j-p_{3i+2}] + w_{3i+2}$ hold simultaneously. This indicates that there is no item belonging to item group i which can be put into knapsack, then we can get $x_{3i} = x_{3i+1} = x_{3i+2} = 0$.
2. If $E[i, j] \neq E[i-1, j]$, this reflects that there is at least one of inequalities which holds.
 - If $E[i, j] = E[i-1, j-p_{3i}] + w_{3i}$, the item $3i$ is put into knapsack. Then, we can get $x_{3i} = 1$ and $x_{3i+1} = x_{3i+2} = 0$.
 - If $E[i, j] = E[i-1, j-p_{3i+1}] + w_{3i+1}$, the item $3i+1$ is put into knapsack. Then, we can get $x_{3i+1} = 1$ and $x_{3i} = x_{3i+2} = 0$.
 - If $E[i, j] = E[i-1, j-p_{3i+2}] + w_{3i+2}$, the item $3i+2$ is put into knapsack. Then, we can get $x_{3i+2} = 1$ and $x_{3i} = x_{3i+1} = 0$.

According to the aforementioned analysis, we can determine the selected item group and the corresponding item. From Eq. (8), we can know whether there is an item which is selected from item group 0. Thus, we get an optimal solution of NE-DKP. The detailed procedure of NE-DKP algorithm is summarized in Algorithm 1.

Algorithm 1 NE-DKP: the new exact algorithm for D{0-1}KP.

```

1: Input: The value set  $P = \{p_k | k = 0, 1, \dots, 3n-1\}$ , weight set  $W = \{w_k | k = 0, 1, \dots, 3n-1\}$ , and knapsack capacity  $C$ .
2: Output: The optimal solution  $X = (x_0, x_1, \dots, x_{3n-1}) \in \{0, 1\}^{3n}$  and optimal value  $Opt_2$ .
3:  $S \leftarrow \sum_{i=0}^{n-1} p_{3i+2}$ ,  $E[0, 0] \leftarrow 0$ ,  $E[0, p_0] \leftarrow w_0$ ,  $E[0, p_1] \leftarrow w_1$ , and  $E[0, p_2] \leftarrow w_2$ ;
4: for  $j = 1$  to  $S$  do
5:   if  $j \neq p_0$  and  $j \neq p_1$  and  $j \neq p_2$  then
6:      $E[0, j] \leftarrow +\infty$ ;
7:   end if
8: end for
9: for  $i = 1$  to  $n-1$  do
10:  for  $j = 0$  to  $S$  do
11:     $E[i, j] \leftarrow E[i-1, j]$ ;
12:    if  $p_{3i} \leq j < p_{3i+1}$  then
13:       $E[i, j] \leftarrow \min[E[i, j], E[i-1, j-p_{3i}] + w_{3i}]$ ;
14:    else if  $p_{3i+1} \leq j < p_{3i+2}$  then
15:       $E[i, j] \leftarrow \min[E[i, j], E[i-1, j-p_{3i}] + w_{3i}, E[i-1, j-p_{3i+1}] + w_{3i+1}]$ ;
16:    else if  $p_{3i+2} \leq j$  then
17:       $E[i, j] \leftarrow \min[E[i, j], E[i-1, j-p_{3i}] + w_{3i}, E[i-1, j-p_{3i+1}] + w_{3i+1}, E[i-1, j-p_{3i+2}] + w_{3i+2}]$ ;
18:    end if
19:  end for
20: end for
21:  $Opt_2 \leftarrow \max_{j=1,2,\dots,S} [j | E[n-1, j] \leq C]$ .
22: for  $i = 0$  to  $3n-1$  do
23:   $x_i \leftarrow 0$ ;
24: end for
25: while  $i > 0$  and  $j > 0$  do
26:  if  $E[i, j] = E[i-1, j-p_{3i}] + w_{3i}$  then
27:     $x_{3i} \leftarrow 1$ ,  $j \leftarrow j - p_{3i}$ ;
28:  else if  $E[i, j] = E[i-1, j-p_{3i+1}] + w_{3i+1}$  then
29:     $x_{3i+1} \leftarrow 1$ ,  $j \leftarrow j - p_{3i+1}$ ;
30:  else if  $E[i, j] = E[i-1, j-p_{3i+2}] + w_{3i+2}$  then
31:     $x_{3i+2} \leftarrow 1$ ,  $j \leftarrow j - p_{3i+2}$ ;
32:  end if
33:   $i \leftarrow i - 1$ ;
34: end while
35: if  $i = 0$  and  $j = p_0$  then
36:   $x_0 \leftarrow 1$ ;
37: else if  $i = 0$  and  $j = p_1$  then
38:   $x_1 \leftarrow 1$ ;
39: else if  $i = 0$  and  $j = p_2$  then
40:   $x_2 \leftarrow 1$ ;
41: end if
42: Return  $(Opt_2, X)$ .

```

In Algorithm 1, the time complexities of calculating the optimal value Opt_2 and optimal solution X are $O(nS)$ and $O(S)$, respectively. Thus, the computational complexity of NE-DKP is $O(nS)$, where $S = \sum_{i=0}^{n-1} p_{3i+2}$. NE-DKP is also an exact algorithm with pseudo polynomial time. By comparing the computational complexity of NE-DKP with OE-DKP, the following Theorem 1 can be obtained.

Theorem 1. *The solving speeds of OE-DKP and NE-DKP respectively depend on C and S : when $C < S$, OE-DKP is faster than NE-DKP; when $C > S$, OE-DKP is slower than NE-DKP. The time ratio of these two algorithms is $\frac{C}{S}$ when using them to solving a same $D\{0-1\}KP$.*

4. Three approximate algorithms for $D\{0-1\}KP$

Because OE-DKP and NE-DKP are all the exact algorithms with pseudo polynomial time, their computational complexities are extremely high when using them to solve the large scale $D\{0-1\}KP$. In fact, the approximate solution of $D\{0-1\}KP$ is always more desired by many NP-hard problems. It is more important for practical applications to get the approximate solutions with the fast speed [7,11]. This section provides three approximate algorithms for $D\{0-1\}KP$.

4.1. Poly-DKP: the fully polynomial-time approximation scheme for $D\{0-1\}KP$

A fully polynomial-time approximation scheme for $D\{0-1\}KP$ is given in this subsection based on the exact algorithm NE-DKP. We firstly introduce the concept of fully polynomial-time approximation scheme.

Definition 2. (Fully polynomial-time approximation scheme) [7,11]. Assume A_ε is a polynomial approximation scheme for the optimization problem I . If its performance ratio $r(A_\varepsilon) \leq 1 + \varepsilon$ (for any $0 < \varepsilon < 1$) and computational complexity is a polynomial function of problem scale and $\frac{1}{\varepsilon}$, then A_ε is a fully polynomial-time approximation scheme for I .

I is an instance of $D\{0-1\}KP$ with the value set $P = \{p_k | k = 0, 1, \dots, 3n-1\}$, weight set $W = \{w_k | k = 0, 1, \dots, 3n-1\}$, and knapsack capacity C . II is another instance of $D\{0-1\}KP$ with the value set $Q = \{q_k | q_k = \lfloor \frac{p_k}{K} \rfloor, k = 0, 1, \dots, 3n-1\}$, weight set W , and knapsack capacity C , where $K = \max \left[\frac{\varepsilon p}{2n}, 1 \right]$, $p = \max [p_{3i+2} | i = 0, 1, \dots, n-1]$, $0 < \varepsilon < 1$. The optimal solution of II is $Y = (y_0, y_1, \dots, y_{3n-1}) \in \{0, 1\}^{3n}$, then Y is a fully polynomial-time approximation solution of I and its approximate value is App_1 . The detailed procedure of Poly-DKP algorithm is summarized in Algorithm 2. From Algorithm 2, we can know that the approximate value of instance I is $App_1 = \sum_{i=0}^{n-1} (p_{3i}y_{3i} + p_{3i+1}y_{3i+1} + p_{3i+2}y_{3i+2})$.

Theorem 2. *Poly-DKP is a fully polynomial-time approximation scheme for $D\{0-1\}KP$, its computational complexity is $O\left(\frac{n^3}{\varepsilon}\right)$.*

Proof. Because $p = \max [p_{3i+2} | i = 0, 1, \dots, n-1]$, when $\frac{\varepsilon p}{2n} \geq 1$, we can derive

$$\sum_{i=0}^{n-1} q_{3i+2} = \sum_{i=0}^{n-1} \left\lfloor \frac{q_{3i+2}}{K} \right\rfloor \leq \sum_{i=0}^{n-1} \frac{q_{3i+2}}{K} \leq \sum_{i=0}^{n-1} \frac{p}{K} = \frac{np}{K} \stackrel{K = \frac{\varepsilon p}{2n}}{=} \frac{2n^2}{\varepsilon};$$

Algorithm 2 Poly-DKP: the fully polynomial-time approximation scheme for $D\{0-1\}KP$.

- 1: **Input:** The value set $P = \{p_k | k = 0, 1, \dots, 3n-1\}$, weight set $W = \{w_k | k = 0, 1, \dots, 3n-1\}$, and knapsack capacity C for instance I ; the any real number $0 < \varepsilon < 1$.
 - 2: **Output:** The approximate solution $Y = (y_0, y_1, \dots, y_{3n-1}) \in \{0, 1\}^{3n}$ and approximate value App_1 .
 - 3: $p \leftarrow \max [p_{3i+2} | i = 0, 1, \dots, n-1]$;
 - 4: $K \leftarrow \max \left[\frac{\varepsilon p}{2n}, 1 \right]$;
 - 5: **for** $i = 0$ to $n-1$ **do**
 - 6: $q_{3i} \leftarrow \text{int}\left(\frac{p_{3i}}{K}\right)$; (int(r) is the rounding operation to real number r .)
 - 7: $q_{3i+1} \leftarrow \text{int}\left(\frac{p_{3i+1}}{K}\right)$;
 - 8: **if** $q_{3i} = q_{3i+1}$ **then**
 - 9: $q_{3i} \leftarrow q_{3i} - 1$;
 - 10: **end if**
 - 11: $q_{3i+2} \leftarrow q_{3i} + q_{3i+1}$;
 - 12: **end for**
 - 13: $Q \leftarrow \{q_k | k = 0, 1, \dots, 3n-1\}$;
 - 14: $(App_1, Y) \leftarrow \text{NE-DKP}(Q, W, C)$; (NE-DKP is Algorithm 1.)
 - 15: $App_1 \leftarrow 0$;
 - 16: **for** $i = 0$ to $3n-1$ **do**
 - 17: $App_1 \leftarrow App_1 + p_i y_i$;
 - 18: **end for**
 - 19: **Return** (App_1, Y) .
-

when $\frac{\varepsilon p}{2n} < 1$, we can get

$$\sum_{i=0}^{n-1} q_{3i+2} \leq \frac{np}{K} \stackrel{K=1}{=} np \leq \frac{2n^2}{\varepsilon}.$$

From Algorithm 2, we can find that the computational complexity of Poly-DKP depends on NE-DKP. Thus, the time complexity of Algorithm 2 is $O(n \sum_{i=0}^{n-1} q_{3i+2}) = O\left(\frac{n^3}{\varepsilon}\right)$.

Assume $Opt(\mathbf{I})$ and $X = (x_0, x_1, \dots, x_{3n-1}) \in \{0, 1\}^{3n}$ are the optimal value and solution of instance $\mathbf{I}$, respectively. Let $S^* = \{i | x_i = 1, i = 0, 1, \dots, 3n-1\}$ denote the set of items in knapsack. The items in S^* are determined by X . Then, we can get $Opt(\mathbf{I}) = \sum_{i \in S^*} p_i$. Assume $Opt(\mathbf{II})$ and $Y = (y_0, y_1, \dots, y_{3n-1}) \in \{0, 1\}^{3n}$ are the optimal value and solution of instance $\mathbf{II}$, respectively. Let $S = \{i | y_i = 1, i = 0, 1, \dots, 3n-1\}$ denote the set of items in knapsack. The items in S are determined by Y . Then, we can get $Opt(\mathbf{II}) = \sum_{i \in S} q_i \geq \left\lfloor \frac{p}{K} \right\rfloor$. Because Y is the approximate solution of instance $\mathbf{I}$, $App(\mathbf{I}) = \sum_{i \in S} p_i$ can be derived. Because $q_{3i} \leq \left\lfloor \frac{p_{3i}}{K} \right\rfloor$, $q_{3i+1} \leq \left\lfloor \frac{p_{3i+1}}{K} \right\rfloor$, and $p_{3i+2} = p_{3i} + p_{3i+1}$, we can derive

$$q_{3i+2} = q_{3i} + q_{3i+1} \leq \left\lfloor \frac{p_{3i+2}}{K} \right\rfloor.$$

Furthermore, we can get

$$\sum_{i \in S} q_i \leq \sum_{i \in S} \left\lfloor \frac{p_i}{K} \right\rfloor \leq \sum_{i \in S} \frac{p_i}{K},$$

i.e., $K \sum_{i \in S} q_i \leq \sum_{i \in S} p_i$. Because $n \gg \varepsilon$, we get

$$\frac{p}{2} \leq \left(1 - \frac{\varepsilon}{2n}\right)p = K\left(\frac{p}{K} - 1\right) \leq K\left\lfloor \frac{p}{K} \right\rfloor \leq K \sum_{i \in S} q_i \leq \sum_{i \in S} p_i.$$

Because $\frac{p_i}{K} - 1 \leq q_i \leq \frac{p_i}{K}$, $\sum_{i \in T} p_i - K|T| \leq K \sum_{i \in T} q_i \leq \sum_{i \in T} p_i$ holds for set T which includes the arbitrary items. Thus, we can derive

$$Opt(\mathbf{I}) - K|S^*| = \sum_{i \in S^*} p_i - K|S^*| \leq K \sum_{i \in S^*} q_i \leq K Opt(\mathbf{II}) = K \sum_{i \in S} q_i \leq \sum_{i \in S} p_i = App(\mathbf{I})$$

for sets S^* and S . Because $|S^*| \leq n$, we can further get

$$App(\mathbf{I}) \geq Opt(\mathbf{I}) - Kn = Opt(\mathbf{I}) - \frac{\varepsilon p}{2} \geq Opt(\mathbf{I}) - \varepsilon \sum_{i \in S} p_i = Opt(\mathbf{I}) - \varepsilon App(\mathbf{I}),$$

i.e., $\frac{Opt(\mathbf{I})}{App(\mathbf{I})} \leq 1 + \varepsilon$. Hereto, we know that Poly-DKP is a fully polynomial-time approximation scheme for D{0-1}KP according to Definition 2. $\square$

4.2. App-2-DKP : the 2-approximation algorithm for D{0-1}KP

For the instance $\mathbf{I}$ of D{0-1}KP and arbitrarily small threshold $\varepsilon > 0$, the approximate ratio of Poly-DKP is $\frac{Opt(\mathbf{I})}{App(\mathbf{I})} \leq 1 + \varepsilon$. We can find that the computational complexity of Poly-DKP depends on $\frac{1}{\varepsilon}$ and increases with the decrease of ε . In fact, we can try to eliminate the negative effects of ε on the computational complexity of approximate algorithm and then give an approximate algorithm with computational complexity $O(n \log_2 n)$. The approximate ratio of such an approximate algorithm is $\frac{Opt(\mathbf{I})}{App(\mathbf{I})} \leq 2$.

Lemma 1. For items $3i$, $3i+1$, and $3i+2$ in item group i , the relationship among $\frac{p_{3i}}{w_{3i}}$, $\frac{p_{3i+1}}{w_{3i+1}}$, and $\frac{p_{3i+2}}{w_{3i+2}}$ can be induced as one of the following four cases:

1. $\frac{p_{3i}}{w_{3i}} \neq \frac{p_{3i+1}}{w_{3i+1}}$, $\frac{p_{3i}}{w_{3i}} \neq \frac{p_{3i+2}}{w_{3i+2}}$, and $\frac{p_{3i+1}}{w_{3i+1}} \neq \frac{p_{3i+2}}{w_{3i+2}}$;
2. $\frac{p_{3i}}{w_{3i}} = \frac{p_{3i+1}}{w_{3i+1}} < \frac{p_{3i+2}}{w_{3i+2}}$;
3. $\frac{p_{3i}}{w_{3i}} = \frac{p_{3i+2}}{w_{3i+2}} > \frac{p_{3i+1}}{w_{3i+1}}$;
4. $\frac{p_{3i+1}}{w_{3i+1}} = \frac{p_{3i+2}}{w_{3i+2}} > \frac{p_{3i}}{w_{3i}}$.

Proof. We only prove the case 2, because the proving methods of cases 3 and 4 are same as case 2. When $\frac{p_{3i}}{w_{3i}} = \frac{p_{3i+1}}{w_{3i+1}}$, let $\frac{w_{3i+1}}{w_{3i}} = \frac{p_{3i+1}}{p_{3i}} = k$, then we can get $w_{3i+1} = kw_{3i}$ and $p_{3i+1} = kp_{3i}$. According to $p_{3i+2} = p_{3i} + p_{3i+1}$ and $0 < w_{3i+2} < w_{3i} + w_{3i+1}$, we can derive

$$\frac{p_{3i+2}}{w_{3i+2}} = \frac{p_{3i} + p_{3i+2}}{w_{3i+2}} > \frac{p_{3i} + p_{3i+2}}{w_{3i} + w_{3i+1}} = \frac{(k+1)p_{3i}}{(k+1)w_{3i}} = \frac{p_{3i}}{w_{3i}}.$$

Then, we get that the case 2 holds for $\frac{p_{3i}}{w_{3i}}$, $\frac{p_{3i+1}}{w_{3i+1}}$, and $\frac{p_{3i+2}}{w_{3i+2}}$. $\square$

Algorithm 3 App-2-DKP : the 2-approximation algorithm for D{0-1}KP.

```

1: Input: The value set  $P = \{p_k | k = 0, 1, \dots, 3n - 1\}$ , weight set  $W = \{w_k | k = 0, 1, \dots, 3n - 1\}$ , and knapsack capacity  $C$  for instance I.
2: Output: The approximate solution  $X = (x_0, x_1, \dots, x_{3n-1}) \in \{0, 1\}^{3n}$  and approximate value  $App_2$ .
3: Sorting  $\frac{p_j}{w_j}$  ( $j = 0, 1, \dots, 3n - 1$ ) in descending order. Put the item  $3i + 2$  in front of the items  $3i$  and  $3i + 1$  when  $\frac{p_{3i}}{w_{3i}} = \frac{p_{3i+2}}{w_{3i+2}}$  or  $\frac{p_{3i+1}}{w_{3i+1}} = \frac{p_{3i+2}}{w_{3i+2}}$ . Storing the subscripts of ordered  $\frac{p_j}{w_j}$  in array  $H(0, 1, \dots, 3n - 1)$ .
4:  $X(0, 1, \dots, 3n - 1) \leftarrow (0, 0, \dots, 0)$ ;  $B(0, 1, \dots, n - 1) \leftarrow (0, 0, \dots, 0)$ ;  $T \leftarrow C$ ;  $App_2 \leftarrow 0$ ;  $i \leftarrow 0$ .
5: while  $T > 0$  and  $i \leq 3n - 1$  do
6:   if  $w_{H(i)} \leq T$  and  $B\left[\text{int}\left(\frac{H(i)}{3}\right)\right] == 0$  then
7:      $x_{H(i)} \leftarrow 1$ ;  $T \leftarrow T - w_{H(i)}$ ;  $B\left[\text{int}\left(\frac{H(i)}{3}\right)\right] \leftarrow 1$ ;  $App_2 \leftarrow App_2 + p_{H(i)}$ ;
8:   end if
9:    $i \leftarrow i + 1$ 
10: end while
11:  $p_{3k+2} \leftarrow \max[p_{3j+2} | j = 0, 1, \dots, n - 1]$ ;
12: if  $p_{3k+2} > App_2$  then
13:    $App_2 \leftarrow p_{3k+2}$ ;  $X \leftarrow \begin{matrix} x_0, & \dots, & x_{3k+1}, & x_{3k+2}, & x_{3k+3}, & \dots, & x_{3n-1} \\ (0, & \dots, & 0, & 1, & 0, & \dots, & 0) \end{matrix}$ ;
14: end if
15: Return  $(App_2, X)$ .
```

According to the cases 3 and 4 in Lemma 1, we know that $\frac{p_{3i}}{w_{3i}} = \frac{p_{3i+2}}{w_{3i+2}}$ or $\frac{p_{3i+1}}{w_{3i+1}} = \frac{p_{3i+2}}{w_{3i+2}}$ may exist for an instance of D{0-1}KP. If the item $3i + 2$ can be put into knapsack, the approximate solution is more desired, because it maximizes the sum of value coefficients for the residual capacity of knapsack. We give the following Example 1 to explain this fact.

Example 1. Assume there exist an instance **I** with scale $3n = 9$. The value set and weigh set of **I** are

$$[(3, 6, 9), (2, 4, 6), (4, 6, 10)] \text{ and } [(2, 5, 6), (4, 2, 5), (4, 3, 5)],$$

respectively. The knapsack capacity is $C=14$. Sorting $\frac{p_j}{w_j}$ ($j = 0, 1, \dots, 8$) in descending order, we can get the subscript sequence $\{4, 7, 8, 0, 2, 5, 1, 6, 3\}$ and $\frac{p_0}{w_0} = \frac{p_2}{w_2} = \frac{3}{2}$ which is the case 3 of Lemma 1. Then, the item 0 is put into the knapsack rather than item 2. The approximate solution determined based on the greedy strategy is $X_1 = (1, 0, 0, 0, 1, 0, 0, 1, 0)$ and the approximate value is 13. If we consider putting the item 2 into knapsack, we can get the approximate solution is $X_2 = (0, 0, 1, 0, 1, 0, 0, 0, 1)$ and the approximate value is 23. X_2 is the optimal solution of instance **I**.

Algorithm 3 shows the detailed procedure of App-2-DKP algorithm. From Algorithm 3, we can see that the computational complexity of App-2-DKP is $O(n \log_2 n)$ which mainly depends on the sorting operation. Hereinafter, we give the theoretical proof to the 2-approximation ratio of App-2-DKP.

Theorem 3. The performance ratio of App-2-DKP is less than or equal to 2, i.e., $\frac{Opt(\mathbf{I})}{App(\mathbf{I})} \leq 2$, where $Opt(\mathbf{I})$ and $App(\mathbf{I})$ are the optimal and approximate values of instance **I**, respectively.

Proof. Let $\frac{b_i}{a_i}$ denote the maximum of $\left(\frac{p_{3i}}{w_{3i}}, \frac{p_{3i+1}}{w_{3i+1}}, \frac{p_{3i+2}}{w_{3i+2}}\right)$. When $\frac{p_{3i}}{w_{3i}} = \frac{p_{3i+2}}{w_{3i+2}}$ or $\frac{p_{3i+1}}{w_{3i+1}} = \frac{p_{3i+2}}{w_{3i+2}}$, we can get $\frac{b_i}{a_i} = \frac{p_{3i+2}}{w_{3i+2}}$ ($i = 0, 1, \dots, n - 1$). For simplicity, we assume $\frac{b_0}{a_0} \geq \frac{b_1}{a_1} \geq \dots \geq \frac{b_{n-1}}{a_{n-1}}$. Then, there exists $k \in \{0, 1, \dots, n - 1\}$ which satisfies the inequations $\sum_{i=0}^k a_i \leq C$, $w_{3k+3} + \sum_{i=0}^k a_i > C$, $w_{3k+4} + \sum_{i=0}^k a_i > C$, and $w_{3k+5} + \sum_{i=0}^k a_i > C$. Thus, we can get $\sum_{i=0}^k b_i \leq Opt(\mathbf{I}) < \sum_{i=0}^{k+1} b_i$. It indicates that the item group $k + 1$ is the first one of which there is no any item that can be put into knapsack. Because there may be one item group in $(k + 2, k + 3, \dots, n - 1)$ of which an item will be put into knapsack, $App(\mathbf{I}) \geq \sum_{i=0}^k b_i$ holds. because $App(\mathbf{I}) \geq \max[p_{3i+2} | 0 = 1, 2, \dots, n - 1] \geq \max[p_{3(k+1)}, p_{3(k+1)+1}, p_{3(k+1)+2}] \geq b_{k+1}$, we can get

$$App(\mathbf{I}) \geq \frac{1}{2} \left(\sum_{i=0}^k b_i + b_{k+1} \right) \geq \frac{1}{2} \left(\sum_{i=0}^{k+1} b_i \right) > \frac{1}{2} Opt(\mathbf{I}),$$

so we can derive $\frac{Opt(\mathbf{I})}{App(\mathbf{I})} < 2$. That is to say, App-2-DKP is the 2-approximation algorithm for D{0-1}KP. $\square$

4.3. PSO-GRDKP: the particle swarm optimization based approximation algorithm for D{0-1}KP

In the above-mentioned descriptions, we present two approximate algorithms, i.e., Poly-DKP and App-2-DKP, based on the dynamic programming and greedy strategy, respectively. In the practical applications [8,9,23,24,26,31,34,36,37,40,43], we

Algorithm 4 GR-DKP: the greedy repair algorithm for D{0-1}KP.

```

1: Input: A binary vector  $X = (x_0, x_1, \dots, x_{3n-1}) \in \{0, 1\}^{3n}$  and array  $H(0, 1, \dots, 3n-1)$ .
2: Output: The repaired binary vector  $X = (x_0, x_1, \dots, x_{3n-1}) \in \{0, 1\}^{3n}$ .
3:  $fweight \leftarrow 0$ ;  $k \leftarrow 3n-1$ ;
4: for  $i = 0$  to  $n-1$  do
5:   if  $x_{3i} + x_{3i+1} + x_{3i+2} > 1$  then
6:      $x_{3i} \leftarrow 0$ ;  $x_{3i+1} \leftarrow 0$ ;  $x_{3i+2} \leftarrow 0$ ;
7:      $j = \arg \max_{j=3i, 3i+1, 3i+2} \left( \frac{p_j}{w_j} \right)$ 
8:      $x_j \leftarrow 1$ ;
9:      $fweight \leftarrow fweight + w_j$ ;
10:   end if
11: end for
12: while  $fweight > C$  do
13:   if  $x_{H(k)} == 1$  then
14:      $x_{H(k)} \leftarrow 0$ ;
15:      $fweight \leftarrow fweight - w_{H(k)}$ ;
16:   end if
17:    $k \leftarrow k-1$ ;
18: end while
19: for  $j = 0$  to  $3n-1$  do
20:    $i \leftarrow \text{int}\left(\frac{H(j)}{3}\right)$ ;
21:   if  $x_{3i} == 0$  and  $x_{3i+1} == 0$  and  $x_{3i+2} == 0$  and  $fweight + w_{H(j)} \leq C$  then
22:      $x_{H(j)} \leftarrow 1$ ;
23:      $fweight \leftarrow fweight + w_{H(j)}$ ;
24:   end if
25: end for
26: Return  $X$ .

```

find that it is very successful to use the evolutionary algorithm (e.g., particle swarm optimization [20,42]) to solve the optimization problems. However, the infeasible solutions will exist because D{0-1}KP is a constrained optimization problem. The commonly-used strategies to eliminate the infeasible solutions are to use greedy strategy to repair and optimize these infeasible solutions. Thus, we firstly propose a **Greedy Repair** algorithm for D{0-1}KP (GR-DKP) and then design the PSO based greedy repair algorithm for GR-DKP (PSO-GRDKP) in this subsection.

4.3.1. GR-DKP: the greedy repair algorithm for D{0-1}KP

When using PSO to solve the constrained optimization problem D{0-1}KP, two kinds of mostly-used methods to deal with the infeasible solutions are penalty function and repair [28], where the repair method is more effective than penalty function method [4,5,17]. Hereinafter, we present a greedy repair algorithm when PSO is used to solve the constrained optimization problem D{0-1}KP.

For any binary vector $X = (x_0, x_1, \dots, x_{3n-1}) \in \{0, 1\}^{3n}$, it will be a feasible solution of D{0-1}KP only if it satisfies the constrained conditions Eqs. (2) and (3). Thus, we firstly check whether X satisfies Eq. (2). If there exists x_{3i} , x_{3i+1} , and x_{3i+2} which makes $x_{3i} + x_{3i+1} + x_{3i+2} > 1$. It indicates that at least two items in item group i can be put into knapsack. Here, $x_j = 1$ ($j = 3i, 3i+1, 3i+2$) corresponding to the maximum of $\left(\frac{p_{3i}}{w_{3i}}, \frac{p_{3i+1}}{w_{3i+1}}, \frac{p_{3i+2}}{w_{3i+2}} \right)$ is not changed and other x_j s are assigned as 0. Then, we check whether X satisfies Eq. (3). If not, we change $x_{H(j)} = 1$ into $x_{H(j)} = 0$ according to the order of $H(3n-1), H(3n-2), \dots, H(0)$ until Eq. (3) holds. Finally, we check whether the updated X of which $x_j = 0$ is assigned to 1 satisfies Eqs. (2) and (3). Based on the aforementioned descriptions, we give the detailed procedure of GR-DKP in Algorithm 4, where $\frac{p_j}{w_j}$ ($j = 0, 1, \dots, 3n-1$) are sorted in descending order and the subscripts of ordered $\frac{p_j}{w_j}$ are stored in the array $H(0, 1, \dots, 3n-1)$. Obviously, the computational complexity of GR-DKP is $O(n)$.

4.3.2. PSO-GRDKP: the PSO based approximation algorithm for D{0-1}KP

We design a PSO based greedy repair algorithm for solving D{0-1}KP in this subsection. Binary PSO (BPSO) [21] is employed in our study due to its simplicity and availability when handling the 0-1 knapsack problems [18,25]. Assume $S(t) = \{[X_i(t), V_i(t)] | X_i(t) \in \{0, 1\}^{3n}, V_i(t) \in [L, U]^{3n}, i = 1, 2, \dots, N\}$ is the t -th generation population, where

$$X_i(t) = (x_{i0}(t), x_{i1}(t), \dots, x_{i,3n-1}(t))$$

is the position of i -th particle in the t -th generation population, which is a potential solution of GR-DKP;

$$V_i(t) = (v_{i0}(t), v_{i1}(t), \dots, v_{i,3n-1}(t))$$

Algorithm 5 PSO-GRDKP: the PSO based approximation algorithm for D{0-1}KP.

```

1: Input: The instance I of D{0-1}KP, population size  $N$ , iteration number  $T$ , lower bound  $L$ , upper bound  $U$ , acceleration constants  $c_1$  and  $c_2$ .
2: Output: The approximate solution  $G(T)$  and approximate value  $f[G(T)]$  of I.
3: Sorting  $\frac{p_j}{w_j}$  ( $j = 0, 1, \dots, 3n - 1$ ) in descending order. Storing the subscripts of ordered  $\frac{p_j}{w_j}$  in array  $H(0, 1, \dots, 3n - 1)$ .
4: Generating the initial population  $S(0) = \{[X_i(0), V_i(0)] | i = 1, 2, \dots, N\}$  randomly;
5: for  $j = 1$  to  $N$  do
6:    $X_i(0) \leftarrow \text{GR-DKP}[X_i(0), H(0, 1, \dots, 3n - 1)]$ ; (GR-DKP is Algorithm 4.)
7: end for
8: Determining  $P_i(0)$  and  $G(0)$  according to  $f[X_i(0)]$ ;  $t \leftarrow 0$ ;
9: while  $t < T$  do
10:  for  $i = 1$  to  $N$  do
11:    for  $j = 0$  to  $3n - 1$  do
12:       $v_{ij}(t + 1) \leftarrow v_{ij}(t) + c_1 r_1 [p_{ij}(t) - x_{ij}(t)] + c_2 r_2 [g_j(t) - x_{ij}(t)]$ ;
13:      if  $r_3 \geq \text{sig}[v_{ij}(t + 1)]$  then
14:         $x_{ij}(t + 1) \leftarrow 0$ ;
15:      else
16:         $x_{ij}(t + 1) \leftarrow 1$ ;
17:      end if
18:    end for
19:     $X_i(t + 1) \leftarrow \text{GR-DKP}[X_i(t + 1), H(0, 1, \dots, 3n - 1)]$ ;
20:    if  $f[X_i(t + 1)] > f[P_i(t + 1)]$  then
21:       $P_i(t + 1) \leftarrow X_i(t + 1)$ ;
22:    end if
23:  end for
24:  Determining  $G(t + 1)$  according to  $f[P_i(t + 1)]$ ,  $i = 1, 2, \dots, N$ ;
25:   $t \leftarrow t + 1$ ;
26: end while
27: Return  $[f[G(T)], G(T)]$ .

```

is the velocity of i -th particle in the t -th generation population; N is the number of particles in the t -th generation population; $t > 0$ is the number of iterations; L and U are the lower and upper bounds of velocity, respectively. Let

$$P_i(t) = (p_{i0}(t), p_{i1}(t), \dots, p_{i,3n-1}(t)) \in \{0, 1\}^{3n}$$

and

$$G(t) = (g_0(t), g_1(t), \dots, g_{3n-1}(t)) \in \{0, 1\}^{3n}$$

denote the locally or personally optimal solution of the i -th particle and globally optimal solution of population.

The position and velocity of the i -th particle is updated according to the following rules:

$$v_{ij}(t + 1) = v_{ij}(t) + c_1 r_1 [p_{ij}(t) - x_{ij}(t)] + c_2 r_2 [g_j(t) - x_{ij}(t)], \quad (9)$$

and

$$x_{ij}(t + 1) = \begin{cases} 0, & \text{if } r_3 \geq \text{sig}[v_{ij}(t + 1)] \\ 1, & \text{otherwise} \end{cases} \quad (10)$$

where, $i = 1, 2, \dots, N$, $j = 0, 1, \dots, 3n - 1$, $c_1 = c_2 = 2$ are the acceleration constants, r_1, r_2, r_3 are the random numbers within interval $(0, 1)$, and $\text{sig}(x) = \frac{1}{1 + e^{-x}}$ is the sigmoid function. Algorithm 5 depicts the detailed procedure of PSO-GRDKP. In Algorithm 5, the computational complexities of sorting operation and GR-DKP algorithm are $O(n \log_2 n)$ and $O(n)$, respectively. Thus, the time complexity of PSO-GRDKP is $O(n \log_2 n) + 3O(nN) + T \times [O(nN) + O(N)]$. Because $N < n$ and $T < n$, the computational complexity of PSO-GRDKP is $O(n^3)$.

5. Experimental comparison on 4 instances of D{0-1}KP

This section conducts the exhaustive experiments to compare the performances of OE-DKP [12], NE-DKP, Poly-DKP, App-2-DKP, and PSO-GRDKP based on 4 kinds of large scale D{0-1}KP instances,¹ i.e., **uncorrelated** instances of D{0-1}KP (udkp), **weakly correlated** instances of D{0-1}KP (wdkp), **strongly correlated** instances of D{0-1}KP (sdkp), and **inverse strongly correlated** instances of D{0-1}KP (idkp) [22,32]. All algorithms in this article are implemented with C++ language running on a

¹ Available at <http://sncet.com/FourKindsOfLarge-scalePIInstances.rar>.

Table 1

Solving performances of 5 algorithms on 10 data sets of udkp.

Instance	Optimal	OE-DKP		NE-DKP		Poly-DKP		App-2-DKP		PSO-GRDKP				
	value	C	Time ₁	S	Time ₂	App ₁	Time ₃	App ₂	Time ₄	App _{Best}	App _{Worst}	App _{Mean}	App _{Std}	Time ₅
udkp1	289,761	505,592	0.798	316,225	0.516	289,761	0.078	239,760	0.0	289,746	289,157	289489.4	219.0	0.134
udkp2	510,131	769,649	2.316	676,296	2.078	510,131	0.218	472,233	0.0	509,981	509,503	509790.6	170.9	0.522
udkp3	817,713	1,396,990	6.368	955,123	4.328	817,713	0.453	676,286	0.0	817,300	816,431	816961.6	298.4	1.156
udkp4	1,122,074	1,965,668	11.913	1,284,269	7.611	1,122,068	0.813	907,900	0.0	1,121,812	1,121,469	1121632.0	109.9	2.025
udkp5	1,233,057	2,054,733	15.469	1,573,966	11.626	1,233,035	1.297	1,085,503	0.008	1,232,740	1,232,490	1232591.5	100.5	3.196
udkp6	1,399,458	2,168,210	19.653	1,921,342	16.868	1,399,458	1.922	1,256,462	0.015	1,398,909	1,398,707	1398767.6	73.2	4.659
udkp7	1,826,261	3,096,369	32.692	2,205,688	22.630	1,826,258	2.609	1,576,358	0.015	1,825,711	1,824,834	1825424.6	320.5	6.248
udkp8	1,920,409	3,016,327	36.476	2,552,592	30.579	1,920,409	3.343	1,743,346	0.015	1,919,810	1,918,752	1919359.4	395.6	8.241
udkp9	2,458,318	4,097,285	55.711	2,914,816	38.315	2,458,308	4.297	2,061,688	0.015	2,456,686	2,454,768	2455811.8	793.6	10.432
udkp10	2,886,506	5,092,146	76.961	3,217,412	47.674	2,886,491	4.984	2,292,272	0.016	2,884,139	2,880,678	2882292.0	974.3	12.794

Table 2

Solving performances of 5 algorithms on 10 data sets of wdkp.

Instance	Optimal	OE-DKP		NE-DKP		Poly-DKP		App-2-DKP		PSO-GRDKP				
	value	C	Time ₁	S	Time ₂	App ₁	Time ₃	App ₂	Time ₄	App _{Best}	App _{Worst}	App _{Mean}	App _{Std}	Time ₅
wdkp1	310,805	255,619	0.422	416,119	0.721	310,805	0.11	310,740	0.0	310,805	310,768	310790.2	18.1	0.137
wdkp2	504,177	389,984	1.218	860,530	2.746	504,177	0.297	504,022	0.0	504,177	504,044	504100.4	62.6	0.519
wdkp3	840,609	679,302	3.079	1,240,929	5.891	840,609	0.625	840,518	0.0	840,597	840,539	840573.8	28.4	1.149
wdkp4	1,041,019	799,741	4.797	1,667,887	10.298	1,041,010	1.093	1,041,010	0.0	1,041,019	1,041,010	1041011.8	3.6	2.003
wdkp5	1,606,341	1,302,412	9.704	2,059,185	15.797	1,606,341	1.672	1,606,332	0.005	1,606,332	1,606,332	1606332.0	0.0	3.106
wdkp6	1,875,732	1,519,313	13.845	2,503,869	22.954	1,875,732	2.407	1,875,604	0.010	1,875,717	1,875,657	1875685.6	21.0	4.519
wdkp7	1,726,671	1,340,978	13.969	2,928,683	30.923	1,726,662	3.297	1,726,556	0.010	1,726,662	1,726,636	1726648.2	11.7	6.134
wdkp8	2,589,429	2,139,106	25.829	3,343,326	40.393	2,589,425	4.297	2,589,336	0.010	2,589,411	2,589,376	2589394.8	16.3	7.891
wdkp9	2,551,957	2,027,672	27.501	3,705,219	50.252	2,551,940	5.344	2,551,897	0.015	2,551,928	2,551,908	2551918.4	8.5	10.203
wdkp10	2,718,419	2,159,925	32.424	4,176,473	62.487	2,718,419	6.672	2,718,305	0.015	2,718,388	2,718,367	2718383.0	8.1	12.559

Acer Aspire E1-570G PC with Windows 8 running on a Intel(R) Core(TM) i5-3337u 1.8 GHz processor with 4 GB DDR3 RAM. Each instance includes 10 different data sets with sizes $3n = 300, 600, \dots, 3000$. For each data set, the value coefficients, weight coefficients, and knapsack capacity are generated as follows [32].

1. udkp instance. $p_{3i}, p_{3i+1} \in [128, 3072]$, $p_{3i} < p_{3i+1}$, $p_{3i+2} = p_{3i} + p_{3i+1}$ ($i = 0, 1, \dots, n-1$); $w_{3i}, w_{3i+1} \in [256, 4098]$, $w_{3i} < w_{3i+1}$, $w_{3i+2} \in [w_{3i+1} + 1, w_{3i} + w_{3i+1} + 1]$; $C = r \sum_{i=0}^{n-1} w_{3i+2}$, where r is a random decimal within interval $[0.5, 0.75]$ and $x \in [A, B]$ denotes x is a random integer within interval $[A, B]$.
2. wdkp instance. $w_{3i}, w_{3i+1} \in [256, 4098]$, $w_{3i} < w_{3i+1}$, $w_{3i+2} \in [w_{3i+1} + 1, w_{3i} + w_{3i+1} + 1]$ ($i = 0, 1, \dots, n-1$); $p_{3i} \in [w_{3i} - 100, w_{3i} + 100]$, $p_{3i+1} \in [w_{3i+1} - 100, w_{3i+1} + 100]$, $p_{3i} < p_{3i+1}$, $p_{3i+2} = p_{3i} + p_{3i+1}$; $C = r \sum_{i=0}^{n-1} w_{3i+2}$.
3. sdkp instance. $w_{3i}, w_{3i+1} \in [256, 4098]$, $w_{3i} < w_{3i+1}$, $w_{3i+2} \in [w_{3i+1} + 1, w_{3i} + w_{3i+1} + 1]$ ($i = 0, 1, \dots, n-1$); $p_{3i} = w_{3i} + 100$, $p_{3i+1} = w_{3i+1} + 100$, $p_{3i+2} = p_{3i} + p_{3i+1}$; $C = r \sum_{i=0}^{n-1} w_{3i+2}$.
4. idkp instance. $p_{3i}, p_{3i+1} \in [256, 4098]$, $p_{3i} < p_{3i+1}$, $p_{3i+2} = p_{3i} + p_{3i+1}$ ($i = 0, 1, \dots, n-1$); $w_{3i} = p_{3i} + 100$, $w_{3i+1} = p_{3i+1} + 100$, $w_{3i+2} \in [w_{3i+1} + 1, w_{3i} + w_{3i+1} + 1]$; $C = r \sum_{i=0}^{n-1} w_{3i+2}$.

The exact algorithms, i.e., OE-DKP and NE-DKP, which can solve the optimal solution and value for D{0-1}KP, are time-consuming when handling instances with the large value and weight coefficients, thus they are suitable for the instances with smaller C or S . Our experiment mainly test the impact of C and S on the solving times of OE-DKP and NE-DKP. For Poly-DKP, App-2-DKP, and PSO-GRDKP, we don't guarantee these three approximate algorithms can always find the optimal solutions of instances. Thus, their solving performances are evaluated with the approximate ratio and solving time. The lower the approximate ratio is and the lower the solving time is, the better the performance of algorithm is. In addition, PSO-GRDKP is a random algorithm and thus often gets the different solving results for the independent running. In order to evaluate the performance of PSO-GRDKP more reliably, we calculate the approximate ratios of the best and worst approximate solutions with the optimal solution.

Theoretically, ε in Poly-DKP can be an arbitrary positive real number. However, if ε is extremely small, it will lead to the small K or $K = 1$ for $K = \max \left[\frac{\varepsilon p}{2n}, 1 \right]$. In this case, $\frac{\hat{S}}{K} = \sum_{i=0}^{n-1} \frac{p_{3i+2}}{K}$ is very large and then it causes the out of memory when Poly-DKP calls NE-DKP in Algorithm 2. In our experiments, we set $\varepsilon = \frac{20n}{p}$ and then we can get $K = 10$. For PSO-GRDKP, the parameters are $N = 30$, $L = -5$, $U = 5$, $c_1 = c_2 = 2$, and $T = n$, where n is the number of item groups. For each algorithm, we repeatedly run it 20 times on the identical data set and use the average value as the final result. Tables 1–4 summarize the calculated values and solving times corresponding to 5 different D{0-1}KP algorithms on instances of udkp, wdkp, sdkp, idkp, respectively, where C is the knapsack capacity, $S = \sum_{i=0}^{n-1} p_{3i+2}$, App_1 is the approximate value of the approximate algorithm Poly-DKP, App_2 is the approximate value of the approximate algorithm App-2-DKP, App_{Best} , App_{Worst} , App_{Mean} , and App_{Std} are the best value, worst value, mean value, and standard deviation of PSO-GRDKP for 20 times independent running.

Table 3

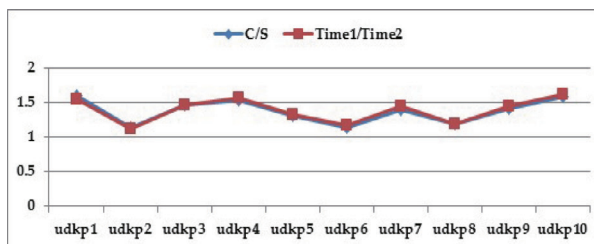
Solving performances of 5 algorithms on 10 data sets of sdkp.

Instance	Optimal value	OE-DKP		NE-DKP		Poly-DKP		App-2-DKP		PSO-GRDKP				
		C	Time ₁	S	Time ₂	App ₁	Time ₃	App ₂	Time ₄	App _{Best}	App _{Worst}	App _{Mean}	App _{Std}	Time ₅
sdkp1	352,019	265,233	0.421	456,879	0.781	351,992	0.078	351,140	0.0	352,003	351,928	351973.4	25.6	0.137
sdkp2	545,255	389,495	1.188	910,978	2.765	545,250	0.281	545,147	0.0	545,243	545,231	545235.5	4.5	0.504
sdkp3	986,019	741,444	3.359	1,379,036	6.140	986,008	0.625	984,581	0.0	985,905	985,602	985786.0	101.3	1.112
sdkp4	1,247,191	916,070	5.501	1,791,499	10.626	1,247,157	1.109	1,244,851	0.005	1,247,041	1,246,432	1246780.6	206.3	1.941
sdkp5	1,759,075	1,317,721	9.969	2,316,659	17.016	1,759,075	1.829	1,757,387	0.010	1,758,664	1,758,391	1758543.2	91.0	3.031
sdkp6	1,795,393	1,296,459	11.719	2,697,496	23.642	1,795,375	2.516	1,794,156	0.010	1,795,373	1,795,170	1795270.6	78.4	4.416
sdkp7	2,264,218	1,673,102	17.814	3,203,071	33.001	2,264,215	3.484	2,263,043	0.010	2,264,026	2,263,652	2263803.8	127.0	5.997
sdkp8	2,236,703	1,621,874	19.532	3,607,252	42.581	2,236,694	4.547	2,235,634	0.010	2,236,471	2,236,106	2236318.8	150.3	7.863
sdkp9	3,034,816	2,277,132	30.752	4,084,681	54.174	3,034,795	5.781	3,030,311	0.015	3,034,111	3,033,149	3033813.0	347.3	9.963
sdkp10	2,916,217	2,098,311	31.596	4,611,874	66.706	2,916,181	7.204	2,915,218	0.019	2,916,071	2,915,753	2915883.2	108.4	12.325

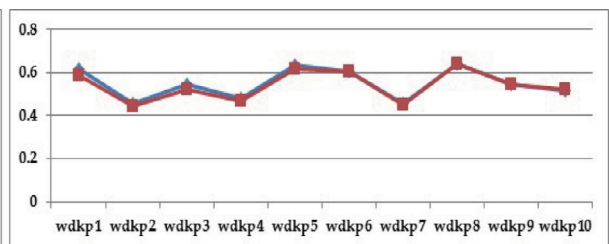
Table 4

Solving performances of 5 algorithms on 10 data sets of idkp.

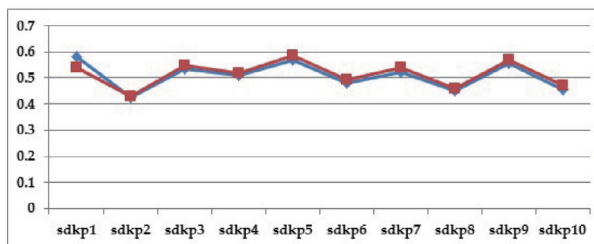
Instance	Optimal value	OE-DKP		NE-DKP		Poly-DKP		App-2-DKP		PSO-GRDKP				
		C	Time ₁	S	Time ₂	App ₁	Time ₃	App ₂	Time ₄	App _{Best}	App _{Worst}	App _{Mean}	App _{Std}	Time ₅
idkp1	277,642	225,675	0.359	435,982	0.696	277,633	0.109	277,581	0.0	277,642	277,633	277639.0	4.2	0.138
idkp2	541,724	428,422	1.313	880,465	2.753	541,724	0.313	541,604	0.0	541,724	541,653	541683.8	32.9	0.527
idkp3	1,016,524	823,711	3.765	1,313,736	6.150	1,016,524	0.688	1,016,457	0.0	1,016,518	1,016,464	1016500.4	22.5	1.137
idkp4	1,220,338	987,047	6.031	1,743,255	11.078	1,220,322	1.141	1,220,269	0.0	1,220,327	1,220,296	1220317.0	10.9	2.046
idkp5	1,342,480	1,051,968	8.016	2,159,116	16.938	1,342,474	1.812	1,342,175	0.005	1,342,454	1,342,443	1342446.6	4.5	3.184
idkp6	1,922,488	1,559,698	14.579	2,513,375	23.173	1,922,488	2.532	1,922,381	0.010	1,922,451	1,922,400	1922420.4	17.0	4.534
idkp7	2,190,780	1,775,465	18.923	3,011,804	32.439	2,190,763	3.453	2,190,681	0.015	2,190,759	2,190,709	2190733.2	18.6	6.128
idkp8	2,719,899	2,222,463	27.282	3,455,865	42.408	2,719,879	5.078	2,719,593	0.015	2,719,881	2,719,815	2719851.4	22.3	8.200
idkp9	2,377,631	1,852,115	25.095	3,911,824	53.253	2,377,619	5.656	2,377,328	0.015	2,377,631	2,377,526	2377586.6	35.5	10.281
idkp10	3,123,425	2,506,055	38.111	4,259,048	64.519	3,123,425	7.016	3,123,335	0.015	3,123,417	3,123,339	3123373.2	27.3	12.534



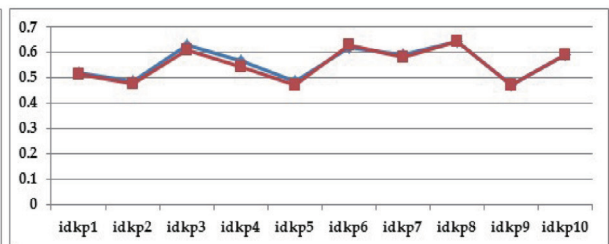
(a) On sdkp data sets



(b) On wdkp data sets



(c) On sdkp data sets



(d) On idkp data sets

Fig. 1. Comparison between $\frac{C}{S}$ and $\frac{Time_1}{Time_2}$ on 4 instances.

From Tables 1–4, we can easily find that the developed approximate algorithms (i.e., Poly-DKP, App-2-DKP, and PSO-GRDKP) based on the newly-designed exact algorithm (i.e., NE-DKP) obtain the better solving performances than the existing exact algorithm (OE-DKP): more accurate values and lower solving times.

Fig. 1 presents the comparison between $\frac{C}{S}$ and $\frac{Time_1}{Time_2}$ on 4 kinds of instances, where $S = \sum_{i=0}^{n-1} p_{3i+2}$. This comparison is to show the impact of C and S on the solving times of OE-DKP and NE-DKP. Tables 1–4 reflects that the solving time of OE-DKP is lower than NE-DKP for instances with $C < S$ and higher than NE-DKP for the instance with $C > S$. This indicates that OE-DKP is more suitable for instances with the smaller C and NE-DKP is more suitable for instances with the smaller

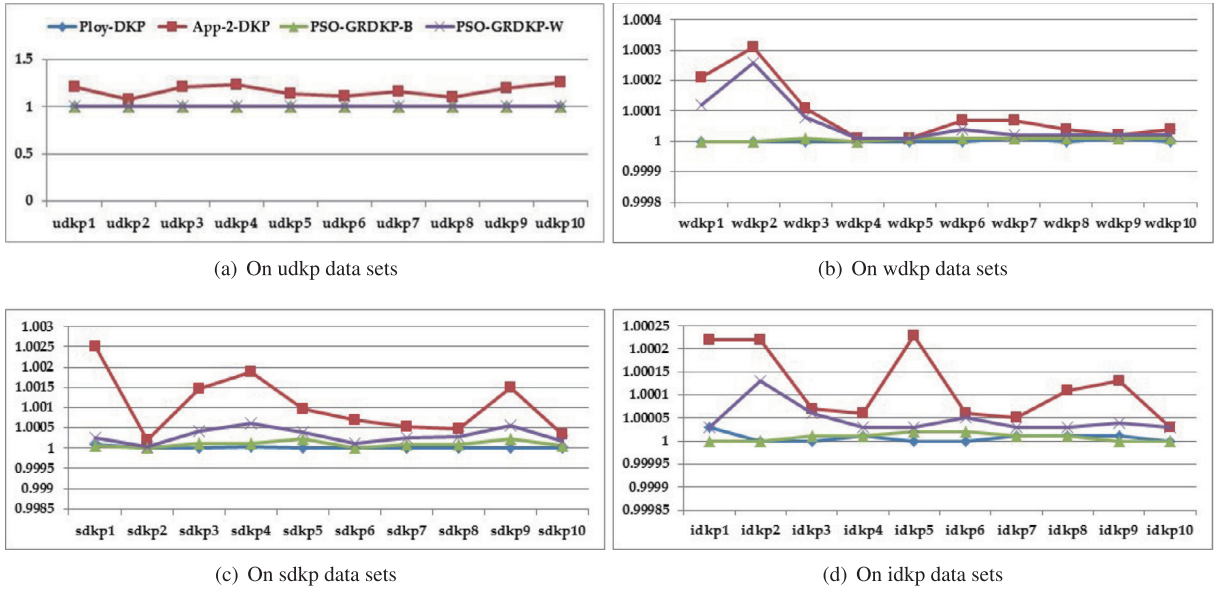


Fig. 2. Comparison of approximate ratios among Poly-DKP, App-2-DKP, and PSO-GRDKP on 4 instances.

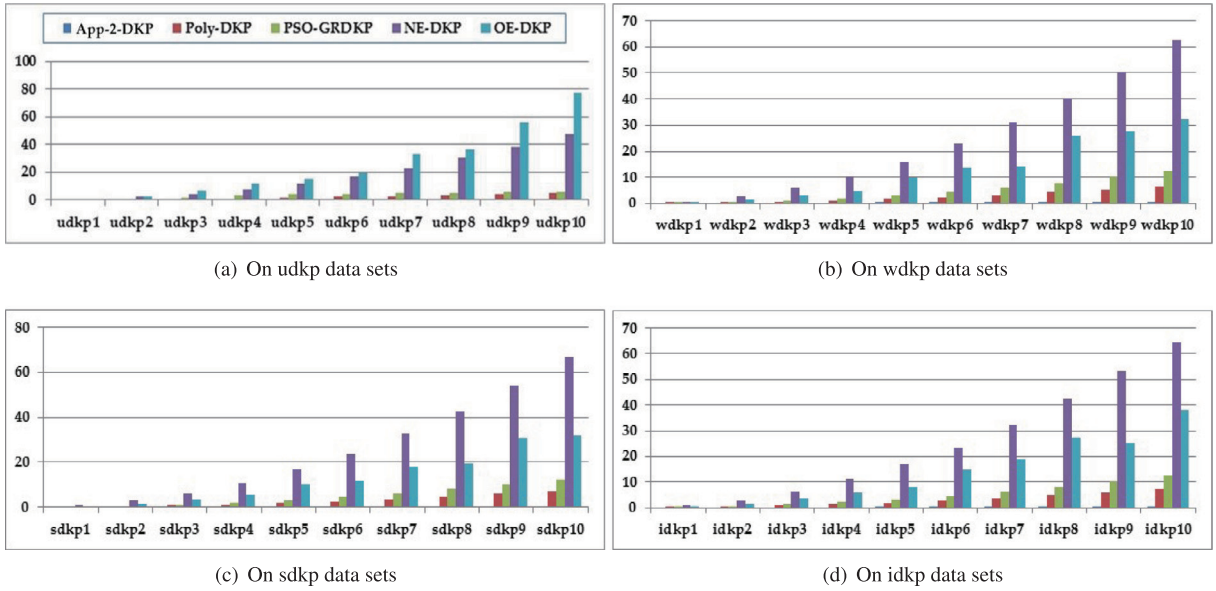


Fig. 3. Comparison of solving time among 5 algorithms on 4 instances.

S. In addition, Fig. 1 shows that the curve of $\frac{C}{S}$ is completely identical to the curve of $\frac{Time_1}{Time_2}$, where $Time_1$ and $Time_2$ are the solving times of OE-DKP and NE-DKP, respectively. This indicates that the time ratio of OE-DKP and NE-DKP is indeed $\frac{C}{S}$ as shown in Theorem 1. Fig. 2 lists the comparative results of approximate ratios among Poly-DKP, App-2-DKP, and PSO-GRDKP on 4 kinds of instances. From Fig. 2, we can see that Poly-DKP has the better results, because its approximate ratios are nearly equal to 1 on each data set. App-2-DKP obtains the worse results compared with Poly-DKP and PSO-GRDKP. Fig. 3 gives the comparison of solving time among 5 algorithms on each data set. When C and S are all very large, we can get $O(n \log_2 n) < O(n^3) < O\left(\frac{n^3}{\varepsilon}\right) < O(nC)$ or $O(n \log_2 n) < O(n^3) < O\left(\frac{n^3}{\varepsilon}\right) < O(nS)$. This reflects that App-2-DKP has the fastest solving speed and OE-DKP and NE-DKP are slowest. The solving speeds of Poly-DKP and PSO-GRDKP are slower than App-2-DKP and faster than OE-DKP and NE-DKP. The experimental results in Tables 1–4 and Fig. 3 confirm this conclusion. Even for the D[0-1]KP instance with scale $3n = 3000$, the solving times of App-2-DKP are lower than 0.02 s. The solving times of Poly-DKP and PSO-GRDKP are unrelated to value and weight coefficients, which increase with the increase of instance scale.

OE-DKP and NE-DKP has the highest solving times. Especially for the instances with large value and weight coefficients, the solving speeds of OE-DKP and NE-DKP are very slow.

Above all, we conclude the experimental results and analysis as follows. For the small scale D{0-1}KP instance with small value and weight coefficients, OE-DKP is the first choice when $C < S$ and NE-DKP is more excellent when $C > S$. For the large scale D{0-1}KP instance with large value and weight coefficients, Poly-DKP or PSO-GRDKP should be used to obtain a fast solving, where PSO-GRDKP is more appropriate to the instances with extremely large scales. If there is not the higher requirement to result, App-2-DKP is the best choice, because this approximate algorithm has the fastest solving speed among all approximate algorithms.

6. Conclusions and further works

This article studied one exact and three approximate algorithms for solving Discounted {0-1} Knapsack Problem (D{0-1}KP). Unlike the existing exact algorithm, The new exact algorithm (NE-DKP) was firstly designed based on the principle of minimizing the total weight with the given sum of value coefficients. Then, three approximate algorithms were developed based on the designed exact algorithm: fully **P**olynomial-time approximation scheme (Poly-DKP), **2-App**roximation algorithm (App-2-DKP), and **PSO** based **G**reedy **R**epair algorithm (PSO-GRDKP). On four kinds of well-known real applications of D{0-1}KP, we finally tested and analyzed the solving performances of proposed exact and approximate algorithms and provided the useful instructions for the application scenarios of proposed D{0-1}KP algorithms.

Our future research will be addressed as the following two issues. First, we will study the uncertainty measure criterion for a D{0-1}KP instance and further use the uncertainty minimization theory [2,13–15,38,39,44] to design the D{0-1}KP algorithm. Second, we will design the differential evolution [16,35] and artificial bee colony optimization [19,41] based on GR-DKP to solve D{0-1}KP.

Acknowledgments

We thank Editor-in-Chief and anonymous reviewers whose valuable comments and suggestions help us significantly improve this article. The first author and corresponding authors contributed equally the same to this article which was supported by Basic Research Project of Knowledge Innovation Program in Shenzhen (JCYJ20150324140036825), China Post-doctoral Science Foundations (2015M572361 and 2016T90799), National Natural Science Foundations of China (61503252 and 71371063), Scientific Research Project Program of Colleges and Universities in Hebei Province (ZD2016005), and [Natural Science Foundation of Hebei Province \(F2016403055\)](#).

References

- [1] M.H. Alsuwaiyel, *Algorithms Design Techniques and Analysis*, World Scientific Publishing Company, Singapore, 2009.
- [2] R.A.R. Ashfaq, X.Z. Wang, J.Z.X. Huang, H. Abbas, Y.L. He, Fuzziness based semi-supervised learning approach for intrusion detection system, in: *Information Sciences*, 2016. In press, doi: [10.1016/j.ins.2016.04.019](#).
- [3] R. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, 1957.
- [4] M.C. Chih, Self-adaptive check and repair operator-based particle swarm optimization for the multidimensional knapsack problem, *Appl. Soft Comput.* 26 (2015) 378–389.
- [5] P.C. Chu, J.E. Beasley, A genetic algorithm for the multidimensional knapsack problem, *J. Heuristics* 4 (1998) 63–86.
- [6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms*, The MIT Press, Cambridge, 2009.
- [7] P.W. David, B.S. David, *The Design of Approximation Algorithms*, Cambridge University Press, Cambridge, 2011.
- [8] B.P. De, R. Kar, D. Mandal, S.P. Ghoshal, Optimal selection of components value for analog active filter design using simplex particle swarm optimization, *Int. J. Mach. Learn. Cybern.* 6 (4) (2015) 621–636.
- [9] B.P. De, R. Kar, D. Mandal, S.P. Ghoshal, An efficient design of CMOS comparator and folded cascode op-amp circuits using particle swarm optimization with an aging leader and challengers algorithm, *Int. J. Mach. Learn. Cybern.* 7 (2016) 325–344.
- [10] D.Z. Du, K.I. Ko, *Theory of Computational Complexity*, Wiley-Interscience, New York, 2000.
- [11] D.Z. Du, K.I. Ko, X.D. Hu, *Design and Analysis of Approximation Algorithms*, Springer Science Business Media LLC, Berlin, 2012.
- [12] B. Guldán, *Heuristic and exact algorithms for discounted knapsack problems*, University of Erlangen-Nürnberg, Germany, 2007 Master thesis.
- [13] Y.L. He, J.N.K. Liu, Y.H. Hu, X.Z. Wang, OWA operator based link prediction ensemble for social network, *Expert Syst. Appl.* 42 (1) (2015) 21–50.
- [14] Y.L. He, X.Z. Wang, J.Z.X. Huang, Fuzzy nonlinear regression analysis using a random weight network, *Inf. Sci.* (2016a). In press, doi: [10.1016/j.ins.2016.01.037](#).
- [15] Y.L. He, X.Z. Wang, J.Z.X. Huang, Recent advances in multiple criteria decision making techniques, *Int. J. Mach. Learn. Cybern.* (2016b). In press, doi: [10.1007/s13042-015-0490-y](#).
- [16] Y.C. He, X.Z. Wang, K.Q. Liu, Y.Q. Wang, Convergent analysis and algorithmic improvement of differential evolution, *J. Softw.* 21 (5) (2010) 875–885.
- [17] Y.C. He, X.L. Zhang, W.B. Li, X. Li, W.L. Wu, S.G. Gao, Algorithms for randomized time-varying knapsack problems, *J. Comb. Optim.* 31 (1) (2016) 95–117.
- [18] Y.C. He, L. Zhou, C.P. Shen, A greedy particle swarm optimization for solving knapsack problem, in: *Proceedings of 2007 International Conference on Machine Learning and Cybernetics*, 2007, pp. 995–998.
- [19] D. Karaboga, *An Idea Based on Honey Bee Swarm for Numerical Optimization*, Technical Report TR06, 2005.
- [20] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: *Proceedings of the IEEE International Conference on Neural Networks*, 1995, pp. 1942–1948.
- [21] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm optimization, in: *Proceedings of 1997 Conference on System, Man, and Cybernetics*, 1997, pp. 4104–4109.
- [22] H. Kellerer, U. Pferschy, D. Pisinger, *Knapsack Problems*, Springer, Berlin, 2004.
- [23] Z.P. Liang, R.Z. Song, Q.Z. Lin, Z.H. Du, J.Y. Chen, Z. Ming, J.P. Yu, A double-module immune algorithm for multi-objective optimization problems, *Appl. Soft Comput.* 35 (2015) 161–174.
- [24] Q.Z. Lin, Q.L. Zhu, P.Z. Huang, J.Y. Chen, Z. Ming, J.P. Yu, A novel hybrid multi-objective immune algorithm with adaptive differential evolution, *Comput. Oper. Res.* 62 (2015) 95–111.
- [25] J.Q. Liu, Y.C. He, Q.Q. Gu, Solving knapsack problem based on discrete particle swarm optimization, *Comput. Eng. Des.* 28 (13) (2007) 3189–3191. 3204.

- [26] W.M. Ma, M.M. Wang, X.X. Zhu, Improved particle swarm optimization based approach for bilevel programming problem-an application on supply chain model, *Int. J. Mach. Learn. Cybern.* 5 (2) (2014) 281–292.
- [27] S. Martello, *Knapsack Problem: Algorithms and Computer Implementations*, JohnWiley & Sons, New York, 1990.
- [28] Z. Michalewicz, M. Schoenauer, Evolutionary algorithms for parameter optimization problems, *Evol. Comput.* 4 (1) (1996) 1–32.
- [29] D. Pisinger, The quadratic knapsack problem-a survey, *Discrete Appl. Math.* 155 (2007) 623–648.
- [30] V. Poirriez, N. Yanev, R. Andonov, A hybrid algorithm for the unbounded knapsack problem, *Discrete Optim.* 6 (2009) 110–124.
- [31] A. Rajasekhar, A. Abraham, M. Pant, A Hybrid Differential Artificial Bee Colony Algorithm based tuning of fractional order controller for Permanent Magnet Synchronous Motor drive, *Int. J. Mach. Learn. Cybern.* 5 (3) (2014) 327–337.
- [32] A.Y. Rong, J.R. Figueira, K. Klamroth, Dynamic programming based algorithms for the discounted {0–1} knapsack problem, *Appl. Math. Comput.* 218 (2012) 6921–6933.
- [33] A. Sbihi, A best first search exact algorithm for the multiple-choice multidimensional knapsack problem, *J. Comb. Optim.* 13 (2007) 337–351.
- [34] D. Souravlias, K.E. Parsopoulos, Particle swarm optimization with neighborhood-based budget allocation, *Int. J. Mach. Learn. Cybern.* 7 (3) (2016) 451–477.
- [35] R. Storn, K.V. Price, *Differential Evolution-A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces*, Tech. Report TR-95–012, Institute of Company Secretaries of India, Chennai, Tamil Nadu, 1995.
- [36] N. Tian, Z.C. Ji, C.H. Lai, Simultaneous estimation of nonlinear parameters in parabolic partial differential equation using quantum-behaved particle swarm optimization with gaussian mutation, *Int. J. Mach. Learn. Cybern.* 6 (2) (2015) 307–318.
- [37] N. Tian, C.H. Lai, Parallel quantum-behaved particle swarm optimization, *Int. J. Mach. Learn. Cybern.* 5 (2) (2014) 309–318.
- [38] X.Z. Wang, Learning from big data with uncertainty – editorial, *J. Intell. Fuzzy Syst.* 28 (5) (2015) 2329–2330.
- [39] X.Z. Wang, R.A.R. Ashfaq, A.M. Fu, Fuzziness based sample categorization for classifier performance improvement, *J. Intell. Fuzzy Syst.* 29 (3) (2015) 1185–1196.
- [40] S.E. Wang, Z.Z. Han, F.C. Liu, Y.G. Tang, Nonlinear system identification using least squares support vector machine tuned by an adaptive particle swarm optimization, *Int. J. Mach. Learn. Cybern.* 6 (6) (2015) 981–992.
- [41] R.J. Wang, Y.J. Zhan, H.F. Zhou, Q.L. Cai, Application of artificial bee colony optimization algorithm in complex blind separation source, *Sci. China. Inf. Sci.* 44 (2) (2014) 199–220.
- [42] J.C. Zeng, J. Jie, Z.H. Cui, *Particle Swarm Optimization*, Science Press, Beijing, 2004.
- [43] Z.X. Zhu, J.R. Zhou, Z. Ji, Y.H. Shi, DNA sequence compression using adaptive particle swarm optimization-based memetic algorithm, *IEEE Trans. Evol. Comput.* 15 (5) (2011) 643–658.
- [44] X. Zhu, X. Li, Iterative search method for total flowtime minimization no-wait flowshop problem, *Int. J. Mach. Learn. Cybern.* 6 (5) (2015) 747–761.