Solving 0-1 knapsack problem by greedy degree and expectation efficiency

Jianhui Lv[a], Xingwei Wang[a,*], Min Huang[a], Hui Cheng[b], Fuliang Li[a]

[a]College of Information Science and Engineering, Northeastern University, Shenyang, 110819, China
[b]School of Computing and Mathematical Sciences, Liverpool John Moores University, Liverpool, L3 5UX, UK

Abstract

It is well known that 0-1 knapsack problem (KP01) plays an important role in both computing theory and many real life applications. Due to its NP-hardness, lots of impressive research work has been performed on many variants of the problem. Inspired by region partition of items, an effective hybrid algorithm based on greedy degree and expectation efficiency (GDEE) is presented in this paper. In the proposed algorithm, initially determinate items region, candidate items region and unknown items region are generated to direct the selection of items. A greedy degree model inspired by greedy strategy is devised to select some items as initially determinate region. Dynamic expectation efficiency strategy is designed and used to select some other items as candidate region, and the remaining items are regarded as unknown region. To obtain the final items to which the best profit corresponds, static expectation efficiency strategy is proposed whilst the parallel computing method is adopted to update the objective function value. Extensive numerical investigations based on a large number of instances are conducted. The proposed GDEE algorithm is evaluated against chemical reaction optimization algorithm and modified discrete shuffled frog leaping algorithm. The comparative results show that GDEE is much more effective in solving KP01 than other algorithms and that it is a promising tool for solving combinatorial optimization problems such as resource allocation and production scheduling.

Keywords: Economics; Region partition; Greedy degree; Expectation efficiency; Parallel computing

1. Introduction

Knapsack problem (KP) is known as a well-studied combinatorial optimization problem and it has been thoroughly studied in the past decades. Generally speaking, KP is classified into separable KP (SKP) in which the items can be split arbitrarily and 0-1 KP (KP01) in which the items cannot be split. Among them, KP01 is an important type of KP due to its NP-hardness [1] and it offers many practical applications such as capital budgeting, project selection, resource allocation, cutting stock, investment decision-making, etc [2]. Therefore, more and more researchers have paid attention to the problem of KP01 optimization. Especially, Martello gave a comprehensive review with further discussions on techniques commonly used in solving KP01 [3]. Since KP01 has been proven to be NP-hard, the methods employed to solve KP01 have been divided into three categories, i.e., exact methods with the exact solutions, meta-heuristic methods and heuristic methods with the approximate solutions.

About exact methods, some related research has been carried out. In [4], dynamic programming algorithm was proposed to solve KP01. And then, Rong [5] developed dynamic programming algorithm according to state reduction and Figueira [6] used many complementary dominance relations to improve dynamic programming algorithm. Kolesar [7] proposed branch and bound algorithm to solve KP01 in 1967. Later, Horowitz [8], Martello [9], and Pisinger [10] respectively expanded branch and bound algorithm in 1974, 1981, and 1993. In [11], the procedure based on linear mathematical programming formulation was proposed. In [12], a Lagrangian decomposition-based algorithm was proposed. In [13], an algorithm based on surrogate, Lagrangian, and continuous relaxations was proposed. In [14], an algorithm with a single continuous variable for KP01 was proposed. Although they can produce the optimal solutions in solving small-scale problems, these exact algorithms perform poorly when the scales come to be large.

In recent decades, many computational intelligence methods [15] have been developed and regarded as meta-heuristic algorithms to solve KP01. In [16], a global harmony search algorithm was proposed. In [17], Zhang proposed a harmony search algorithm while it adopted the parallel updating strategy rather than serial updating strategy. In [18], Kong proposed a simplified binary harmony search algorithm for large scale KP01. In [19], an algorithm based on amoeboid organism was proposed by transforming the longest path into the shortest path. In [20], a quantum-inspired artificial immune system for KP01 was proposed. In [21], an improved hybrid encoding cuckoo search algorithm was proposed. In [22], a shuffled frog leaping algorithm was proposed. In [23], a particle swarm-based algorithm was proposed. In [24], a human learning optimization algorithm was proposed. In [25], a DNA-based computing method was proposed with fast parallel molecular. Although meta-heuristics can effectively solve KP01, they have to un-

*Corresponding author. Tel.: +86 18512451179
E-mail address: lvjianhui2012@163.com (J. Lv), wangxw@mail.neu.edu.cn (X. Wang), mhuang@mail.neu.edu.cn (M. Huang), H.Cheng@ljmu.ac.uk (H. Cheng), lifuliang207@126.com (F. Li)

dergo the complex iteration process and set different number of populations with different instances. With that said, it will increase the difficulty of solving KP01, for example, the number of populations for a certain KP01 instance is difficult to set. Furthermore, the complex iteration process is not adapted to the optimization of engineering problem.

It becomes increasingly popular for the computer science researchers worldwide to apply heuristic techniques in the optimization problems [26]. Given this, there are a number of effective heuristics for solving KP01. In [27], a hybrid differential evolution approach was studied. In [28], a polyhedral study with disjoint cardinality constraints was proposed. In [29], a population-based incremental learning algorithm based on greedy strategy was presented. In [30], a chemical reaction optimization algorithm with greedy strategy was proposed. In [31], an efficient fully polynomial time approximate scheme for KP01 was proposed. In [32], an iterative rounding search-based algorithm was proposed to solve KP01. In [33], a soccer league competition algorithm was proposed. In [34], a thermondynamical selection-based discrete differential evolution method was presented. However, they still cost the high time complexity especially in [27], [28], [29], and [31]. Thus, an expectation efficiency based on economical model was studied [35] and the time complexity was $O(n)$, which suggested the expectation efficiency model for solving KP01 is more effective compared to other existing heuristics. In this paper, the expectation efficiency will be studied by combining with greedy degree.

Although a large number of KP01s have been resolved successfully by these existing algorithms, some new and more difficult KP01s are always hidden in the real world. Thus, the research on KP01 should be further improved and developed. Furthermore, especially in the industry, it usually focuses on finding the good approximate solution rather than spending a lot of time for the exact solution. Under this context, heuristic methods for KP01 will play more important role than exact methods and meta-heuristic methods. Given this, the heuristic methods should be encouraged so that the optimization and application of KP01 can be enhanced.

Given the above consideration, a hybrid algorithm based on greedy degree and expectation efficiency, called GDEE, is proposed which is inspired by region partition of items to solve KP01. The main contributions of this paper are summarized as follows. (a) A greedy degree model inspired by greedy strategy is designed to select the first some items to put into knapsack early and these items are never removed from knapsack in the following operations. (b) A dynamic expectation efficiency strategy is proposed to select some remaining items to put into knapsack, meanwhile, one candidate objective function value is obtained. (c) A static expectation efficiency strategy is also presented as the benchmark to update the candidate objective function value, as a result, a number of new objective function values are generated. (d) To accelerate the update speed of objective function value, the parallel computing method is adopted. (e) Experimental results based on a large number of instances reveal that GDEE is correct, feasible, effective, and stable.

The rest of this paper is organized as follows. Section 2 presents the design of GDEE in detail. In Section 3, experimental results based on a large number of instances are reported. Finally, Section 4 concludes this paper and suggests potential future work.

2. Design GDEE for KP01

2.1. KP01 description

Given n items, where item i owns weight w_i and profit p_i, and a knapsack that holds a fixed capacity cap, the goal of KP01 is to load some possible items into knapsack so that the total profit of the selected items has the maximal value while the total weight of the items is not greater than cap. Mathematically, KP01 can be described as follows.

$$\begin{cases} Maximize & optp = \sum_{i=1}^{n} p_i x_i \\ s.t. & \sum_{i=1}^{n} w_i x_i \le cap \end{cases} \tag{1}$$

where $optp$ means the objective function value and $x_i \in \{0, 1\}$. If x_i is 1, item i is in knapsack; otherwise, it is not in knapsack. Let $X = (x_1, x_2, \cdots, x_n)$ means the solution of KP01. Thus, the goal of KP01 is to find a possible X which maximizes $optp$.

2.2. Greedy degree

Let r_i denote the ratio of p_i and w_i. In this paper, n items are rearranged by r_i in descending order in the first place, which is also done under greedy strategy, backtracking algorithm, and dynamic programming algorithm. Thus, the arrangement will be executed before performing the following GDEE operations. It is well known that greedy algorithm is applied to solve KP01 at the beginning and the obtained solution is local optimal rather than the optimal. As a matter of fact, some items can be still loaded into knapsack from the perspective of greedy algorithm, in other words, the items are determinate while the remaining items are uncertain. Based on this consideration, the determinate items should be loaded into knapsack in advance and never be removed from knapsack. However, how many and which items are determinate? To answer this question, the concept of greedy degree is proposed as follows.

Definition 1. n items are rearranged, if the first m items can be always loaded into knapsack by greedy strategy, and then m is regarded as the size of greedy degree.

The objective function value, the total weight of items, and the number of items can be obtained by greedy algorithm. Let $Goptp$, GW, and Q represent them respectively, and the constraint conditions are shown as follows.

$$\begin{cases} \frac{\sum_{i=1}^{j} w_i}{GW} \wedge \frac{\sum_{i=1}^{j} p_i}{Goptp} \le \lambda \\ \frac{\sum_{i=1}^{j+1} w_i}{GW} \wedge \frac{\sum_{i=1}^{j+1} p_i}{Goptp} > \lambda \\ \frac{\sum_{i=1}^{n/2} w_i + \sum_{i=n/2+1}^{j} w_i}{GW} \wedge \frac{\sum_{i=1}^{n/2} p_i + \sum_{i=n/2+1}^{j} p_i}{Goptp} < \xi \\ \frac{\sum_{i=1}^{n/2} w_i + \sum_{i=n/2+1}^{j+1} w_i}{GW} \wedge \frac{\sum_{i=1}^{n/2} p_i + \sum_{i=n/2+1}^{j+1} p_i}{Goptp} \ge \xi \end{cases} \tag{2}$$

where λ and ξ are parameters, $\lambda > \xi$, $\lambda, \xi \in (0, 1)$ with setting in real KP01s; j is the serial number of item; $X \wedge Y \le Z$ means that $X \le Z$ and $Y \le Z$. Right after this, the greedy degree algorithm is designed and described in Algorithm 1.

Algorithm 1: Greedy degree algorithm
Input: W, P, n and cap
// $W = (w_1, w_2, \cdots, w_n)$, $P = (p_1, p_2, \cdots, p_n)$
Output: m
Run greedy algorithm;
Obtain $Goptp$, GW and Q;
while $Q < n/2$, **do**
  $n = n/2$;
end while
while inequality (2) is met, **do**
  $m = j$;
end while

In Algorithm 1, the first m items should be loaded into knapsack, and they constitute initially determinate region.

2.3. Dynamic expectation efficiency

The following work is to select some items from the remaining $n - m$ items as candidate region (*e.g.*, for ten items, among them, the first four items are loaded into knapsack in advance, and m is four. If the optimal solution consists of eight items, the following is to select four items from the remaining six items as candidate region). In economics, there is a classical expectation efficiency theory which was proposed in 1979 [36] and developed in 1992 by Kahneman and Tversky [37]. It can be applied to wide fields such as stock prediction, psychoanalysis, live-hood economy, etc. Western economics suggests that expectation efficiency can be used to solve the uncertain problem and the expectation results are acceptable [35]. In this paper, it will be treated as a heuristic method to solve KP01, in which the most basic idea is to expect the property of the next item by that of the current item. Let $f(i)$ represent the expectation efficiency of item i, and it is shown as follows.

$$f(i) = \frac{p_i\left(cap - \sum_{k=1}^{i-1} w_k\right)/w_i}{\left(cap - \sum_{k=1}^{i-1} w_k\right)p_{i-1}/w_{i-1}} = \frac{r_i}{r_{i-1}} \tag{3}$$

Here, $f(i)$ means the potential of item i that can be loaded into knapsack. And, the greater the item expectation efficiency value is, the higher the possibility of the item is loaded into knapsack is. To illustrate Eq. (3), an instance KP1 is given in Table 1.

In KP1, the first two items are loaded into knapsack early by Algorithm 1, and m is two. Then, $f(3)$, $f(4)$ and $f(5)$ can be obtained by Eq. (3), and they are 0.88, 0.9680 and 0.9783 respectively. Since $f(5)$ is the greatest of all, the fifth item should be loaded into knapsack. At last, $X = (1, 1, 0, 0, 1)$ and $optp = 92$ can be obtained. With that said, Eq. (3) may be a nice model.

If p_5 is 47 not 50 in KP1, then $f(5)$ is changed to 0.9196. In this case, $f(4)$ is the greatest of all, and the fourth item should be loaded into knapsack. In similar way, $X = (1, 1, 0, 1, 0)$ and $optp = 88$ can be obtained. However, the optimal solution is still $(1, 1, 0, 0, 1)$ and $optp$ is 89 rather than 88. With this said, Eq. (3) may be not a nice model.

The optimal solution of KP01 is related to n, w, p, cap, and i, which can be seen from Eq. (1); and it is also related to r, which can be seen from Eq. (3). Thus, a model involves the six factors should be designed to replace Eq. (3), as follows.

$$f(i, w, p, r, n, cap) \propto i, w, p, r, n, cap \tag{4}$$

The following work is to make a concrete model for Eq. (4) due to its abstraction. The remaining capacity of knapsack tends to zero more and more with the item loaded into knapsack one by one. If the item $(i - 1)$ is in knapsack and the remaining capacity of knapsack is greater than zero, then the expectation profit which the remaining capacity of knapsack references to r_{i-1} can be obtained. Let $optp'$ represent it, as follows.

$$\frac{p_{i-1}}{w_{i-1}} = \frac{optp'}{cap - \sum_{k=1}^{m} w_k - \sum_{k=m+1}^{i-1} w_k x_k} \tag{5}$$

$$optp' = r_{i-1}\left(cap - \sum_{k=1}^{m} w_k - \sum_{k=m+1}^{i-1} w_k x_k\right) \tag{6}$$

It is certain that $optp'$ is far greater than p_i. In order to balance the difference between them, let the average of $optp'$ subtract p_i. The result is Δp, as follows.

$$\Delta p = \frac{r_{i-1}\left(cap - \sum_{k=1}^{m} w_k - \sum_{k=m+1}^{i-1} w_k x_k\right)}{n - i + 1} - p_i \tag{7}$$

Given Eqs. (5)-(7), Eq. (4) is defined as follows.

$$\begin{aligned} &Df(i, w, p, r, n, cap) = \\ &\frac{r_i}{r_{i-1}} * \frac{r_{i-1}\left(cap - \sum_{k=1}^{m} w_k - \sum_{k=m+1}^{i-1} w_k x_k\right) - (n - i + 1)p_i}{\left(cap - \sum_{k=1}^{m} w_k - \sum_{k=m+1}^{i-1} w_k x_k\right) - (n - i + 1)w_i}, \\ &m + 1 \le i \le n \end{aligned} \tag{8}$$

Eq. (8) can be converted to Eq. (9) by symbolic substitution.

$$\begin{cases} Df(i, w, p, r, n, cap) = f(i) * \frac{Ar_{i-1} - B}{A - C}, m + 1 \le i \le n \\ A = cap - \sum_{k=1}^{m} w_k - \sum_{k=m+1}^{i-1} w_k x_k \\ B = (n - i + 1)p_i \\ C = (n - i + 1)w_i \end{cases} \tag{9}$$

Among them, $Df(i, w, p, r, n, cap)$ represents dynamic expectation efficiency function, A represents the remaining capacity of knapsack, Ar_{i-1} represents expectation profit that references to r_{i-1}, $(Ar_{i-1} - B)$ represents the difference between the average of the expectation profit and the profit of the current item, and $(A - C)$ represents the difference between the average of the remaining capacity of knapsack and the weight of the current item. And, $(Ar_{i-1} - B)$ and $(A - C)$ have the balance function on Eq. (8). For Eq. (8), $(n - m)$ dynamic expectation efficiency values can be obtained, and they are denoted by $Df(m + 1), Df(m + 2), \cdots, Df(n)$.

It must be noted that A should be still greater than or equal to zero. Otherwise, some items that are from items $(m + 1)$ to n should be removed from knapsack in turn until $A \ge 0$. Since Ar_{i-1} is always changing, the expectation process is dynamic. Based on Eq. (8), the candidate region of items can be obtained, as a result, the corresponding candidate objective function value can be also obtained, denoted by $Doptp$ which can be obtained by Algorithm 2 as follows.

Table 1: KP1 instance

cap	n	$optp$	w_i	p_i	r_i	X
50	5	92	(5,15,25,27,30)	(12,30,44,46,50)	(2.4,2.1,1.76,1.803,1.6667)	(1,1,0,0,1)

Algorithm 2: Dynamic expectation efficiency algorithm
Input: W, P, n, m and cap
Output: $Doptp$
Initially: $x_k = 1, 1 \le k \le n$
for $k = m + 1$ to n, **do**
  Obtain dynamic expectation efficiency values by Eq. (8);
  while $A < 0$, **do**
    $Df(l) = min\{Df(m+1), Df(m+2), \cdots, Df(k)\}$;
    $x_l = 0, m + 1 \le l \le k$;
    $Df(l) = min\{\{Df(m+1), Df(m+2), \cdots, Df(k)\}/Df(l)\}$;
    //$\{X, Y, Z\}/X$ means that X is removed from $\{X, Y, Z\}$
  end while
  Obtain dynamic expectation efficiency values by Eq. (8);
  if $A == 0$, **then**
    End Algorithm 2; Break;
  end if
end for
for $s = m + 1$ to n, **do**
  if $x_s == 1$, **then**
    Select $Df(s)$ and item s;
    Arrange $Df(m+1), Df(m+2), \cdots, Df(s)$ in descending order;
  else
    $Df(s) = 0$;
  end if
end for
for $j = 1$ to τ, **do**
//τ is the number of items loaded into knapsack from item $m + 1$ to item n
  Put item that owes the greater Df into knapsack;
  $Doptp = Moptp$ + the profit of the current item;
end for

In Algorithm 2, some items from $(m + 1)$ to n are selected as the candidate region, at the same time, the unknown region is also accomplished. If the candidate region contains u items, then the unknown region contains $(n - m - u)$ items.

2.4. Static expectation efficiency

$Doptp$ by Algorithm 2 cannot be regarded as the best profit, and thus a static expectation efficiency model will be proposed to update it. At first, the range of the best profit is presented and shown in Theorem 1.

Theorem 1. Let $Uoptp = Goptp + p_{Q+1}$, and the best profit is in $[Goptp, Uoptp)$.

Proof: Reduction to absurdity for the proof is adopted.

(a) $Goptp$ is one objective function value by greedy algorithm. It's obvious that it is less than or equal to the best profit.

(b)Suppose the best profit is equal to $Uoptp$. The item $(Q + 1)$ should be loaded into knapsack when greedy algorithm is run, and $Goptp$ should be equal to $Uoptp$, which conflicts with $Uoptp = Goptp + p_{Q+1}$.

(c) Suppose the best profit is greater than $Uoptp$. In similar way, the item $(Q + 1)$ should be loaded into knapsack when greedy algorithm is run. It's obvious that $Goptp \ge Uoptp$, which also conflicts with $Uoptp = Goptp + p_{Q+1}$.

To sum up (a)-(c), Theorem 1 is proved.

In Section 2.3, the dynamic expectation efficiency model has been introduced, in which Ar_{i-1} is always changing in Eq. (8) when expectation efficiency value is computed. $Doptp$ is one candidate objective function value and that $Doptp$ is equal to the best profit is uncertain, hence it needs to be updated. Given this, one static expectation efficiency model is proposed which is inspired by keeping Ar_{i-1} unchanged in Eq. (8), and it is defined as follows.

$$Sf(i, w, p, r, n, cap, t) = \frac{r_i}{r_{i-1}} * \frac{\left(optp(t) - \sum_{k=1}^{m} w_k - \sum_{k=m+1}^{i-1} w_k x_k\right) - (n - i + 1)p_i}{\left(cap - \sum_{k=1}^{m} w_k - \sum_{k=m+1}^{i-1} w_k x_k\right) - (n - i + 1)w_i}, \quad m + 1 \le i \le n \tag{10}$$

where $Sf(i, w, p, r, n, cap, t)$ represents static expectation efficiency function, and $optp(t)$ is defined as follows.

$$optp(t) = Doptp + t \tag{11}$$

The best profit is in $[Goptp, Uoptp)$ according to Theorem 1. And because the updating process of objective function value should provide the good efficiency, the objective function value that will be obtained by static expectation efficiency model shouldn't be less than $Doptp$. Based on this, the constraint conditions are shown as follows.

$$\begin{cases} Doptp \le optp(t) < Uoptp, & Doptp < Goptp \\ Goptp \le optp(t) < Uoptp, & Doptp \ge Goptp \end{cases} \tag{12}$$

Let Eq. (11) be into inequality (12), as follows.

$$\begin{cases} 0 \le t < Uoptp - Doptp, & Doptp < Goptp \\ Goptp - Doptp \le t < Uoptp - Doptp, & Doptp \ge Goptp \end{cases} \tag{13}$$

where t is a positive integer variable. Let it be into Eq. (10) and Eq. (11) with one step length, and many new objective function values will be obtained (please see section 2.5 for more details). If t keeps unchanged, then $optp(t)$ is certain.

For one new objective function value, denoted by $NSoptp$, it can be obtained by static expectation efficiency model, meanwhile, $(n - m)$ static expectation efficiency values can be obtained, denoted by $Sf(m + 1), Sf(m + 2), \cdots, Sf(n)$. Based on the above, $NSoptp$ can be obtained by Algorithm 3 as follows.

Algorithm 3: Static expectation efficiency algorithm
/*
The principle of Algorithm 3 is similar to Algorithm 2.
The differences are summarized as (a) the running equation and (b) the checking constraint condition.
(a) Eq. (8) is run in Algorithm 2, and Eq. (10) is run in Algorithm 3.
(b) A is checked in Algorithm 2, and T is checked in Algorithm 3.
*/

T is shown as follows.

$$T = optp(t) - \sum_{k=1}^{m} w_k - \sum_{k=m+1}^{i-1} w_k x_k \tag{14}$$

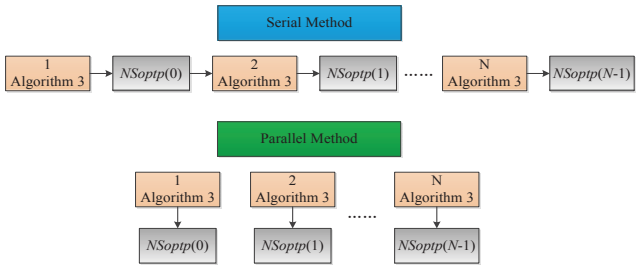

Figure 1: Process of updating objective function value with the serial method and the parallel method, where one output for the former and N outputs for the latter per running Algorithm 3.

In Algorithm 3, one new objective function value can be obtained to replace candidate objective function value by Algorithm 2.

2.5. Parallel computing

A number of new objective function values can be obtained when t changes. The number of new objective function values is denoted by N, and it is shown as follows.

$$N = \begin{cases} Uoptp - Doptp, & Doptp < Goptp \\ Uoptp - Goptp, & Doptp \geq Goptp \end{cases} \tag{15}$$

If these new objective function values are computed with the serial method, then a lot of time will be cost. Thus, the parallel computing method is adopted in this paper. In other words, the candidate objective function value is updated by Algorithm 3 with different t synchronously. The process of updating objective function value with the serial method and parallel method are depicted in Figure 1, where the update of objective function value is accomplished with the serial way by N rounds while that is accomplished with the parallel way by one round.

If $Doptp < Goptp$, then N new objective function values are $NSoptp(0), NSoptp(1), \cdots, NSoptp(Uoptp - Doptp - 1)$. The best profit is denoted by $Boptp$, and it can be obtained by Eq.(16). If $Doptp \geq Goptp$, then N new objective function values are $NSoptp(0), NSoptp(1), \cdots, NSoptp(Uoptp - Goptp - 1)$, and $Boptp$ can be obtained by Eq. (17).

$$Boptp = max\Big\{Doptp, u | u \in \{NSoptp(t), 0 \leq t < Uoptp - Doptp\}\Big\} \tag{16}$$

$$Boptp = max\Big\{Doptp, u | u \in \{NSoptp(t), 0 \leq t < Uoptp - Goptp\}\Big\} \tag{17}$$

If lu items can be loaded into knapsack by updating the candidate objective function value, then the finally determinate region can be accomplished, which contains $(m + lu)$ items.

According to the above statements, the process of GDEE for solving KP01 is depicted in Figure 2. Specifically, at first, n items are rearranged by the ratio of profit and weight in descending order. Secondly, the greedy degree model is used to load the first m items into knapsack and they are not never removed from knapsack. Thirdly, the dynamic expectation

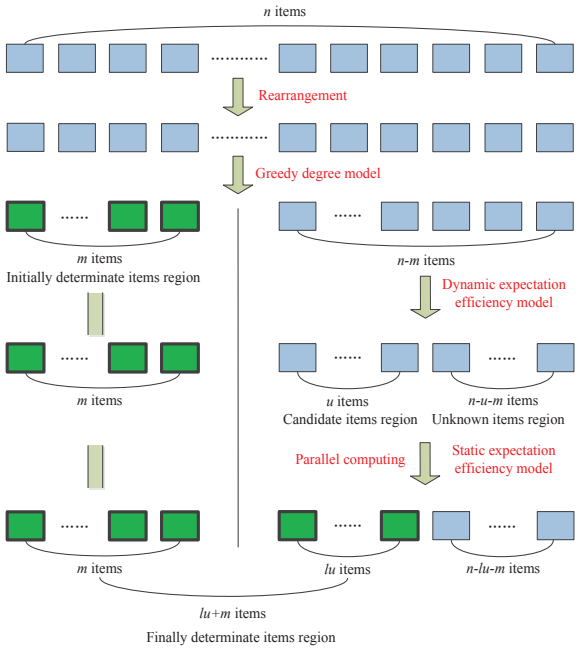

Figure 2: Process of GDEE for solving KP01 in this paper

efficiency strategy is used to load u items from the remaining $(n - m)$ items into knapsack and the last remaining $(n - u - m)$ items are regarded as unknown items region. Fourthly, the static expectation efficiency model with the parallel computing method is used to update the candidate objective function value and lu items are loaded into knapsack. At last, the first m items and the determined lu items constitute finally determinate region. The process can be described in Algorithm 4 as follows.

Algorithm 4: GDEE algorithm
Input: W, P, n and cap
Output: $Boptp$
Run Algorithm 1;
Obtain m;
Run Algorithm 2;
Obtain $Doptp$;
Run Algorithm 3 by parallel method;
if $Doptp < Goptp$, **then**
  Obtain $Boptp$ by Eq. (16);
else
  Obtain $Boptp$ by Eq. (17);
end if

2.6. Computational complexity analysis of GDEE

2.6.1. Time complexity analysis

Theorem 2. Algorithm 1 runs in $O(n)$.

Proof: Algorithm 1 consists of three parts, i.e., running greedy algorithm, checking whether $Q < n/2$ is met or not, and checking whether inequality (2) is met or not. As far as we know, greedy algorithm and the third part both run in $O(n)$. About the second part, $n/2$ items need to be searched when it begins to be run, and then $n/4$ items need to be searched. By that analogy, $n/2^k$ items need to be searched in the end. Right now, it is $k - th$ round search. Here, $n/2^k$ is a positive integer, according to the definition of lower limit function, inequalities are shown as follows.

$$1 \le n/2^k < 2 \tag{18}$$

$$(log_2 n) - 1 < k \le log_2 n \tag{19}$$

As can be seen from inequality (19), the second part runs in $O(logn)$. Since the three parts work in serial way, Algorithm 1 runs in $O(n)$. To sum up, Theorem 2 is proved.

Theorem 3. Algorithm 2 runs in $O(n)$.

Proof: The number of Eq. (8) is run in the dynamic expectation efficiency model is $(n - m)$, and thus this part runs in $O(n)$. In similar way, the other parts run in $O(n)$. There is no doubt that Algorithm 2 runs in $O(n)$.

Theorem 4. Algorithm 3 runs in $O(n)$.

Proof: The proof is similar to Theorem 3 since the principle of Algorithm 3 is similar to Algorithm 2.

Theorem 5. GDEE runs in $O(n)$.

Proof: GDEE consists of four parts, i.e., running Algorithm 1, running Algorithm 2, running Algorithm 3, and selecting $Boptp$ as the best profit by Eq. (16) or (17). Algorithms 1, 2 and 3 all run in $O(n)$ which can be found by Theorems 2, 3 and 4 respectively. In terms of part four, it needs to search $(Uoptp - Doptp)$ or $(Uoptp - Goptp)$ elements when $Boptp$ is selected as the best profit, and thus this part runs in $O(n)$. Since they work in serial way, GDEE also runs in $O(n)$.

2.6.2. Space complexity analysis

Theorem 6. Space complexity of GDEE is in $O(n^2)$.

Proof: At first, let the candidate objective function value $Doptp$ be into static expectation efficiency model, $(n-m)$ static expectation efficiency values are obtained by Eq. (10). And then, to use the method of parallel computing, N new objective function values will be generated by Eq. (15). In total, $N * (n - m)$ static expectation efficiency values are emerged, which needs $N * (n - m)$ space to store. Besides, $N = Uoptp - Doptp$ or $N = Uoptp - Goptp$ by Eq. (15), it keeps the same level with n (i.e., $N = n + \nu$, ν is a constant). In conclusion, the space complexity of GDEE is $O(n^2)$.

3. Experimental results and analysis

In this section, the performance of GDEE is extensively investigated by a large number of experimental studies with considering running time, best profit, worst profit, and the size of storage space as the indexes of performance evaluation. Since correctness, feasibility, effectiveness, and stability are very important for evaluating the performance of GDEE [38], four groups of simulation experiments according to sixty-five KP01 instances are presented. Firstly, a numerical instance is given to illustrate the computation process and correctness of GDEE. Secondly, fifteen instances are tested to demonstrate the feasibility of GDEE. Thirdly, GDEE is compared with chemical reaction optimization algorithm with greedy strategy in [30] called CROG in this paper and modified discrete shuffled frog leaping algorithm in [22] called MDSFL according to five instances to show the effectiveness of GDEE. Finally,

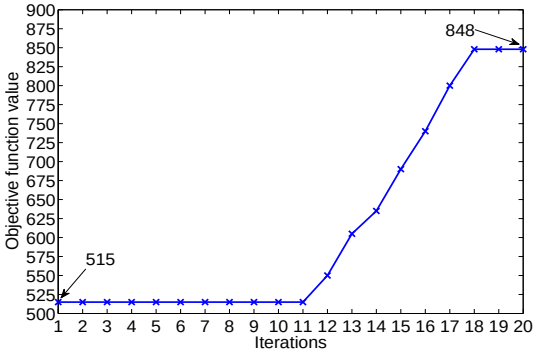

Figure 3: Detailed changing process of objective function value for KP2

GDEE is run on different test case libraries with forty-four instances to reveal the stability of GDEE. The simulation experiments are conducted with VC++ 6.0 in Intel (R) Core(TM) i7, 2.93GHZ CPU, 4G RAM. Simulation parameters are set as follows: $\lambda = 0.7$ and $\xi = 0.5$.

3.1. Significant test

3.1.1. Numerical instance for correctness test

Given one instance KP2: $cap = 620$, $n = 20$, W =(22,36,32, 18,35,26,44,50,45,44,48,50,12,52,24,52,60,28,55,38), P =(56, 60,48,25,72,40,55,55,50,37,30,48,18,78,30,60,45,35,70,48). The process of obtaining the best profit is presented as follows.

Step1: Rearrange 20 items. W =(22,35,36,26,12,32,52,18,55, 38,28,24,44,52,45,50,50,44,60,48), P =(56,72,60,40,18,48,78, 25,70,48,35,30,55,60,50,55,48,37,45,30).

Step2: Run Algorithm 1. $Goptp = 848$, $GW = 619$, $Q = 17$, and $m = 10$.

Step3: Run Algorithm 2. $Doptp = 848$.

Step4: Compute the range of the best profit according to Theorem 1. $Uoptp = 885$ and $Doptp = Goptp = 848$, and then $optp(t) \in [848, 885)$.

Step5: Obtain some new objective function values by parallel computing method. Let $optp(t)$ be into Eq. (10), and 37 new objective function values are 848, 849, $\cdots$, 884 respectively.

Step6: Obtain the best profit by Eq. (17) since $Doptp = Goptp$ =848 in ***Step4***. $Boptp=max\{848, NSoptp(0), NSoptp(1), \cdots, NSoptp(36)\}$ =848.

For KP2, the best profit is 848, and the detailed changing process of objective function value is depicted in Figure 3.

In Figure 3, the best profit can be obtained after 17 iterations. At the beginning, the objective function value is 515, which is the total profit of the first ten items. It keeps unchanged from the first iteration to 10th iteration since the size of greedy degree is ten. And then, dynamic expectation efficiency model, static efficiency model, and parallel computing method are run, and thus the objective function value begins to increase from the 11th iteration. In this process, the determinate items are loaded into knapsack one by one.

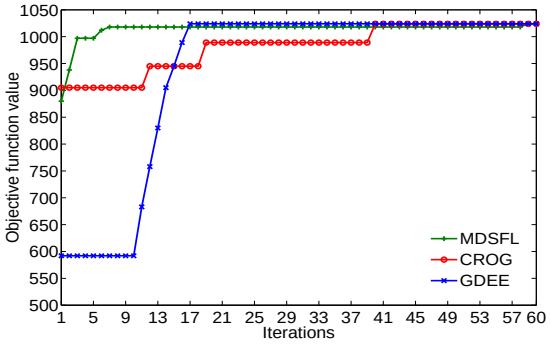

Figure 4: Detailed changing process of objective function value for f_2

3.1.2. Feasibility test

To demonstrate the feasibility of GDEE, fifteen instances from the Standard Test Case Libraries (STCL) of KP01 are given, and the sizes of items are 5, 8, 10, 10, 20, 23, 50, 50, 80, 80, 100, 100, 100, 200, and 200 respectively. The experimental results are composed of five indexes, i.e., the number of parallel update, best profit, worst profit, the size of storage space and running time, and they are shown in Table 2.

As can be seen in Table 2, a number of useful conclusions are presented as follows. (a) GDEE can solve KP01 and obtain the best profit. (b) Although it requires a lot of storage space to place the expectation efficiency values, the memory is cheap nowadays, which can be usually accepted by individuals and enterprises. (c) The running time increases with the increasing of number of items because it requires processing more items with spending much time in general. (d) The number of items is not proportional to the size of storage space since it is determined by the upper bound of the total profit ($Uoptp$) and the number of items. (e) For some KP01s (*e.g.*, KP4 and KP5), although their number of items are same, they have the different best profits since the best profit is determined by six factors (see Eq. (4) for more details). In short, the given fifteen instances can demonstrate the feasibility of GDEE.

3.2. Comparison among three algorithms

3.2.1. Effectiveness test

To further demonstrate the effectiveness of GDEE, it is compared with CROG and MDSFL including best profit, worst profit, difference between best profit and worst profit, population size/the number of parallel update (PS/NPU, i.e., PS and NPU are equivalent), and running time according to five instances which are from Table 1 in [22]. Comparison results among three algorithms are shown in Table 3, and the detailed changing process of the objective function value for f_2, f_4, f_6, f_8 and f_{10} are shown in Figures 4-8 respectively.

In Table 3, CROG obtains the best profit by setting PS for 20; MDSFL obtains the best profit by setting PS for 200; and GDEE obtains the best profit by setting NPU for 11, 8, 15, 486, and 11 respectively.

In Figure 4, the objective function value of CROG increases in stages and it begins to change on the 12th , 19th, and 40th iteration, and the best profit can be obtained on the 40th iteration; one of MDSFL increases continually in early stage and keeps unchanged from the 7th to 58th iteration, and the best profit can be obtained on the 59th iteration; one of GDEE keeps unchanged in early stage from the first iteration to 10th iteration and begins to increase continually until the best profit can be obtained on the 17th iteration.

In Figure 5, the objective function value of CROG increases in stages and it begins to change on the 6th and 9th iteration; one of MDSFL obtains the best profit only by one iteration; one of GDEE keeps unchanged on the first four iterations and begins to increase continually until the best profit can be obtained on the 6th iteration.

In Figure 6, the objective function value of CROG increases in stages and it begins to change on the 4th, 9th, 11th, and 15th iteration, and the best profit can be obtained on the 15th iteration; one of MDSFL obtains the best profit only by one iteration; one of GDEE keeps unchanged on the first five iterations and begins to increase continually until the best profit can be obtained on the 7th iteration.

In Figure 7, the objective function value of CROG increases in stages and it begins to change on the 6th, 14th, 17th, and 27th iteration, and the best profit can be obtained on the 27th iteration; one of MDSFL also increases in stages and it begins to change on the 9th and 22th iteration; one of GDEE keeps unchanged on the first seven iterations and begins to increase continually until the best profit can be obtained on the 13th iteration.

In Figure 8, the objective function value of CROG increases in stages and it begins to change on the 12th, 20th, and 41th iteration, and the best profit can be obtained on the 41th iteration; one of MDSFL increases continually in early stage from first iteration to 16th iteration and increases in stages from the 17th to 53th iteration, and the best profit can be obtained on the 53th iteration; one of GDEE keeps unchanged in early stage from the first iteration to 10th iteration and begins to increase continually until the best profit can be obtained on the 17th iteration.

As can be seen in Table 3 and Figures 4-8, some conclusions can be further presented as follows. (a) The best profit and the optimal solution can be obtained by CROG, MDSFL and GDEE. (b) Only for f_4 and f_6, MDSFL has the fastest convergence speed, but GDEE is still superior to CROG. (c) For f_2, f_8 and f_{10}, GDEE has faster convergence speed than CROG and MDSFL. (d) From the perspective of difference between best profit and worst profit, GDEE and MDSFL have better performance compared to CROG. (e) In terms of PS/NPU, for f_2, f_4, f_6 and f_{10}, GDEE requires less NPU than CROG and MDSFL. (f) From the perspective of running time, for f_4 and f_6, MDSFL spends less time in solving KP01 while GDEE spends less time in solving KP01 except f_4 and f_6. In summary, in comparison with a heuristic algorithm and a computational intelligence algorithm, GDEE is more effective.

3.2.2. Stability test

To demonstrate the stability of GDEE, three STCLs are considered for comparing CROG, MDSFL and GDEE. STCL1 is from [22] with 10 instances and the sizes of items are 10,

Table 2: Test instances for demonstrating feasibility

Instances	Items	*cap*	NPU	Best profit	Worst profit	The size of storage space	Running time(ms)
KP2	5	100	87	133	114	174	26.3015
KP3	8	200	6	334	334	12	26.3234
KP4	10	269	15	295	294	30	27.5272
KP5	10	300	59	388	388	177	28.2806
KP6	20	878	35	1024	1018	245	29.4753
KP7	23	10000	486	9767	9767	972	29.4961
KP8	50	300	11	1063	1060	88	31.0254
KP9	50	500	34	1153	1145	204	31.0108
KP10	80	800	16	2085	2085	112	32.7726
KP11	80	1000	13	2337	2334	65	32.4819
KP12	100	1000	24	2614	2613	240	33.9043
KP13	100	1000	15	2558	2558	225	35.6462
KP14	100	1000	43	2617	2610	860	37.3527
KP15	200	2000	20	5097	5089	760	40.2658
KP16	200	2000	26	5185	5185	494	39.7635

Table 3: Comparison results among GDEE, CROG, and MDSFL under five instances

Instances	Algorithms	Best profit	Worst profit	Difference	PS/NPU	Running time(ms)
	CROG	1024	1018	6	20	36.1007
f_2	MDSFL	1024	1018	6	200	38.6534
	GDEE	1024	1018	6	11	29.4753
	CROG	23	16	7	20	29.7146
f_4	MDSFL	23	23	0	200	19.3600
	GDEE	23	22	1	8	26.2014
	CROG	52	50	2	20	30.9173
f_6	MDSFL	52	52	0	200	20.4812
	GDEE	52	52	0	15	27.3208
	CROG	9767	9765	2	20	34.6059
f_8	MDSFL	9767	9767	0	200	31.5554
	GDEE	9767	9767	0	486	29.4961
	CROG	1025	1019	6	20	36.2556
f_{10}	MDSFL	1025	1019	6	200	37.8395
	GDEE	1025	1019	6	11	29.4768

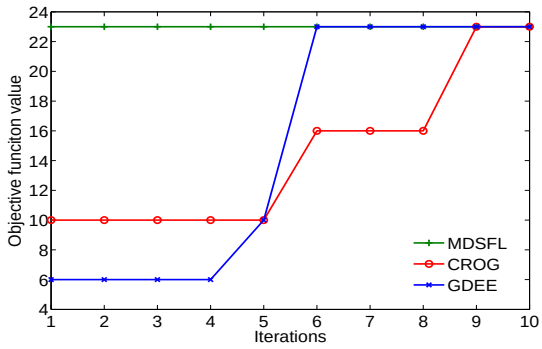

Figure 5: Detailed changing process of objective function value for f_4

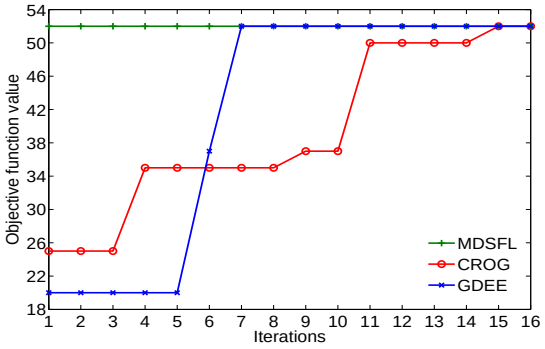

Figure 6: Detailed changing process of objective function value for f_6

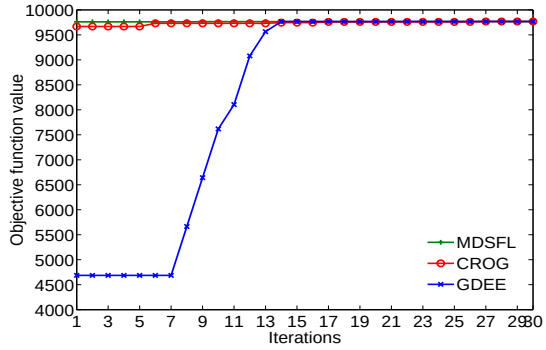

Figure 7: Detailed changing process of objective function value for f_8

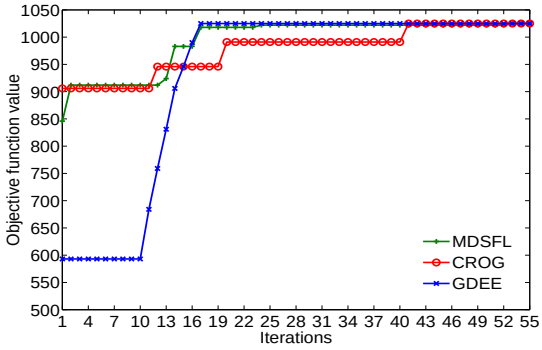

Figure 8: Detailed changing process of objective function value for f_{10}

Table 4: Comparison results among GDEE, CROG, and MDSFL under three STCLs

STCLs	Numbers	Algorithms	Average running time(ms)	Correct rate
		CROG	34.5615	100%
STCL1	10	MDSFL	25.4783	100%
		GDEE	27.2467	100%
		CROG	38.9465	100%
STCL2	20	MDSFL	41.7808	100%
		GDEE	29.3354	100%
		CROG	60.5491	100%
STCL3	14	MDSFL	37.4324	100%
		GDEE	35.6918	100%

20, 4, 4, 15, 10, 7, 23, 5, and 20 respectively; STCL2 is from http://lvjianhui.lingw.net/article-6224551-1.html with 20 instances and the sizes of items are all 20; and STCL3 is from http://lvjianhui.lingw.net/article-6224553-1.html with 14 instances and the sizes of items are 50, 50, 80, 80, 100, 100, 100, 100, 100, 100, 200, 200, and 200 respectively. Comparing CROG, MDSFL and GDEE, and the experimental results including average running time and correct rate (i.e., whether the KP01 can be solved or not) are shown in Table 4.

As can be seen in Table 4, about STCL1, the average running time of CROG is the largest while that of MDSFL is the smallest. About STCL2, the average running time of MDSFL is the largest while that of GDEE is the smallest. About STCL3, the average running time of CROG is the largest while that of GDEE is the smallest. In terms of the 44 instances from different STCLs, CROG, MDSFL and GDEE can all solve KP01 and obtain the best profit (the correct rate is 100%), which suggests that GDEE has a relative stability.

4. Conclusions and future work

KP01 has been widely applied in the real world applications such as capital budgeting, project selection, resource allocation, investment decision-making, etc. In this paper, a new hybrid heuristic algorithm based on greedy degree and expectation, called GDEE, is proposed for solving KP01. In the proposed GDEE, greedy degree model is presented to put the first some items into knapsack early. Furthermore, two expectation efficiency models are designed to generate the objective function value from the remaining items. Moreover, the parallel computing method is adopted to accelerate the update speed of objective function value. In addition, the time complexity of GDEE is analyzed and it runs in $O(n)$, which is appreciable. The space complexity of GDEE is also analyzed and it runs in $O(n^2)$, which is acceptable due to the cheap memory nowadays. The performance of GDEE is extensively investigated through a lot of instances in four groups of experiments, where one instance, fifteen instances, five instances and three STCLs demonstrate its correctness, feasibility, effectiveness, and stability respectively, which demonstrates the proposal of GDEE is a promising tool for solving combinatorial optimizations such as resource allocation and production scheduling.

Based on the whole in-depth study, the proposed GEDD has some distinguished advantages summarized as (a) the lower time complexity compared to other existing algorithms; (b) the fast convergence speed with parallel computing way instead of serial computing way; and (c) simple and comprehensible computation process without the complex iterations. Furthermore, GDEE has a potential for solving any KP01, and it may also be used to solve multiple-objective KP01 and other combinatorial optimization problems like resource allocation in different fields, production scheduling in industry, etc.

However, GDEE also has two limitations as follows. On one hand, it has the higher space complexity due to the parallel computing. On the other hand, it relies on greedy degree model too much, in which once the size of greedy degree is inaccurate, the optimal solution is hard to capture. Thus, as a new algorithm, GDEE needs to be further studied and improved. The future work can be carried out in the following directions. Firstly, a better model than greedy degree model in this paper should be designed to put some items into knapsack in advance. And then, both dynamic and static expectation efficiency models need to be modified to increase the convergence speed of the optimal solution. Last but not least, GDEE is expected to solve the real engineering problems such as resource allocation in cloud computing and job scheduling in industry to expand and enhance the application of GDEE. In total, GDEE is a hybrid and novel approach for solving KP01, and it can provide some valuable rationale for the optimization research of KP01.

Acknowledgements

We would like to thank the editors and all anonymous reviewers for helpful comments and suggestions, which have considerably improved and enhanced the quality of paper. This work is supported by the National Science Foundation for Distinguished Young Scholars of China (Grant No. 61225012 and No. 71325002), the Specialized Research Fund of the Doctoral Program of Higher Education for the Priority Development Areas (Grant No. 20120042130003), and the Fundamental Research Funds for the Central Universities (Grant No. N120104001).

References

[1] R. Merkle, M. Hellman, Hiding information and signatures in trapdoor knapsacks, IEEE Transactions on Information Theory 24 (1978) 525-530.

[2] G. Mavrotas, D. Diakoulaki, A. Kourentzis, Selection among ranked projects under segmentation, policy and logical constraints, European Journal of Operational Research 187 (2008) 177-192.

[3] S. Martello, D. Pisinger, P. Toth, New trends in exact algorithms for the 0-1 knapsack problem, European Journal of Operational Research 123 (2000) 325-332.

[4] P. Toth, Dynamic programming algorithm for zero-one knapsack problem, Computing, 25 (1980) 29-45.

[5] A. Rong, J.R. Figueira, Computational performance of basic state reduction based on dynamic programming algorithms for bi-objective 0-1 knapsack problems, Computers and Mathematics with Applications 63 (2012) 1462-1480.

[6] J.R. Figueira, L. Paquete, M. Simoes, Algorithm improvements on dynamic programming for the bi-objective 0-1 knapsack problem, Computational Optimization and Applications 56 (2013) 97-111.

[7] P.J. Kolesar, A branch and bound algorithm for the knapsack problem, Management Science 13 (1967) 723-735.

[8] E. Horowitz, S. Sahni, Computing partitions with applications to the knapsack problem, Journal of the Association for Computing Machinery 21 (1974) 277-292.
[9] S. Martello, P. Toth, A bound and bound algorithm for the zero-one multiple knapsack problem, Discrete Applied Mathematics 3 (1981) 275-288.
[10] D. Pisinger, An expanding-core algorithm for the exact 0-1 knapsack problem, European Journal of Operational Research 87 (1995) 175-187.
[11] F. Jolai, M.J. Rezaee, M. Rabbani, J. Razmi, P. Fattahi, Exact algorithm for bi-objective 0-1 knapsack problem, European Journal of Operational Research 194 (2007) 544-551.
[12] A. Billionnet, E. Soutif, An exact method based on Lagrangian decomposition for the 0-1 quadratic knapsack problem, European Journal of Operational Research 157 (2004) 565-575.
[13] S. Martello, P. Toth, An exact algorithm for the two-constraint 0-1 knapsack problem, Operations Research 51 (2003) 826-835.
[14] G. Lin, W. Zhu, M.M. Ali, An exact algorithm for the 0-1 linear knapsack problem with a single continuous variable, Journal of Global Optimization 50 (2011) 657-673.
[15] P. Vasant, Handbook of research on artificial intelligence techniques and algorithms (2 volumes), Information Science Reference, 2014.
[16] D. Zou, L. Gao, S. Li, J. Wu, Solving 0-1 knapsack problem by a novel global harmony search algorithm, Applied Soft Computing 11 (2011) 1556-1564.
[17] B. Zhang, Q. Pan, X. Zhang, P. Duan, An effective hybrid harmony search-based algorithm for solving multidimensional knapsack problems, Applied Soft Computing 29 (2015) 288-297.
[18] X. Kong, L. Gao, H. Ouyang, S. Li, A simplified binary harmony search algorithm for large scale 0-1 knapsack problems, Expert Systems with Applications 42 (2015) 5337-5355.
[19] X. Zhang, S. Huang, Y. Hu, Y. Zhang, S. Mahadevan, Y. Deng, Solving 0-1 knapsack problems based on amoeboid organism algorithm, Applied Mathematics and Computation 219 (2013) 9959-9970.
[20] J. Gao, G. He, R. Liang, Z. Feng, A quantum-inspired artificial immune system for the multiobjective 0-1 knapsack problem, Applied Mathematics and Computation 230 (2014) 120-137.
[21] Y. Feng, K. Jia, Y. He, An improved hybrid encoding cuckoo search algorithm for 0-1 knapsack problems, Computational Intelligence and Neuroscience 2014 (2014) 1-9.
[22] K.K. Bhattacharjee, S.P. Sarmah, Shuffled frog leaping algorithm and its application to 0/1 knapsack problem, Applied Soft Computing 19 (2014) 252-263.
[23] M. Chih, Self-adaptive check and repair operator-based particle swarm optimization for the multidimensional knapsack problem, Applied Soft Computing 26 (2015) 378-389.
[24] L. Wang, R. Yang, H. Ni, W. Ye, M. Fei, P.M. Pardalos, A human learning optimization algorithm and its application to multi-dimensional knapsack problems, Applied Soft Computing 34 (2015) 736-743.
[25] S.T. Tsai, Fast parallel molecular solution for DNA-based computing: the 0-1 knapsack problem, International Conference on Algorithms and Architectures for Parallel Processing, 2009, pp. 8-11.
[26] P. Vasant, G.W. Weber, N. Barsoum, V. NgocDieu, Power, control, and optimization, The Scientific World Journal 2015 (2015) 1-2.
[27] J. Shu, J. Li, Solving 0-1 knapsack problems via a hybrid differential evolution, International Symposium on Intelligent Information Technology Application, 2009, pp. 134-137.
[28] B. Zeng, J.P.P. Richard, A polyhedral study on 0-1 knapsack problems with disjoint cardinality constraints: facet-defining inequalities by sequential lifting, Discrete Optimization 8 (2011) 277-301.
[29] X. Fang, N. Chen, Y. Guo, A PBIL algorithm for solving 0-1 knapsack problem based on greedy strategy, Asia-Pacific Computational Intelligence and Information Technology Conference, 2013, pp. 664-672.
[30] K.T. Truong, K. Li, Y. Xu, Chemical reaction optimization with greedy strategy for the 0-1 knapsack problem, Applied Soft Computing 13 (2013) 1774-1780.
[31] C. Bazgan, H. Hugot, D. Vanderpooten, Implementing an efficient fptas for the 0-1 multi-objective knapsack problem, Europrean Journal of Operational Research 198 (2009) 47-56.
[32] M. Hiff, An iterative rounding search-based algorithm for the disjunctively constrained knapsack problem, Engineering Optimization 46 (2014) 1109-1122.
[33] N. Moosavian, Soccer league competition algorithm for solving knapsack problems, Swarm and Evolutionary Computation 20 (2015) 14-22.
[34] Z. Guo, X. Yue, K. Zhang, S. Wang, Z. Wu, A thermodynamical selection-based discrete differential evolution for the 0-1 knapsack problem, Entroy 16 (2014) 6263-6285.
[35] Y. Tian, J. Lv, L. Zheng, An algorithm of 0-1 knapsack problem based on economic model, Journal of Applied Mathematics and Physics 1 (2013) 31-35.
[36] D. Kahneman, A. Tversky, Prospect theory: an analysis of decision under risk, Econometrica 47 (1979) 263-291.
[37] A. Tversky, D. Kahneman, Advances in prospect theory: cumulative representation of uncertainty, Journal of Risk and Uncertainty 5 (1992) 297-323.
[38] P. Vasant, N. Barsoum, J. Webb, Innovation in power, control, and optimization: emerging energy technologies, IGI Global, 2011.

Jianhui Lv received the B.S. degree in mathematics and applied mathematics from the Jilin Institute of Chemical Technology, Jinlin, China in 2012, and the M.S. degree in computer science from the Northeastern University, Shenyang, China in 2014. He is currently working toward the Ph.D. degree in the Northeastern University, Shenyang, China. His research interests include routing in ICN and operational research, etc.

Xingwei Wang received the B.S., M.S., and Ph.D. degrees in computer science from the Northeastern University, Shenyang, China in 1989, 1992, and 1998 respectively. He is currently a Professor at the College of Information Science and Engineering, Northeastern University, Shenyang, China. His research interests include cloud computing and future Internet, etc.

Min Huang received the B.S. degree in automatic instrument, the M.S. degree in systems engineering, and Ph.D. degree in control theory from the Northeastern University, Shenyang, China in 1990, 1993, and 1999 respectively. She is currently a Professor at the College of Information Science and Engineering, Northeastern University, Shenyang, China. Her research interests include modeling and optimization for logistics and supply chain system, etc.

Hui Cheng received the B.S. and M.S. degrees in Computer Science from the Northeastern University, Shenyang, China, in 2001 and 2004 respectively, and the Ph.D. degree in Computer Science from the Hong Kong Polytechnic University, Hong Kong, China in 2007. He is currently a Senior Lecturer at the School of Computing and Mathematical Sciences, Liverpool John Moores University, Liverpool, UK. His research interests include wireless mobile networks, optical networks, artificial intelligence and dynamic optimization, etc.

Fuliang Li received the B.S. degree in computer science from the Northeastern University, Shenyang, China in 2009, and the Ph.D. degree in computer science from the Tsinghua University, Beijing, China in 2015. He is currently an assistant professor at the College of Information Science and Engineering, Northeastern University, Shenyang, China. His research interests include network management and measurement, software defined networking and network security, etc.