Knowledge-Based Systems 219 (2021) 104445

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Group theory-based optimization algorithm for solving knapsack problems

Yichao He [a], Xizhao Wang [b,*]

[a] College of Information and Engineering, Hebei GEO University, Shijiazhuang 050031, China
[b] College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China

ARTICLE INFO

Article history:
Received 3 February 2018
Received in revised form 27 July 2018
Accepted 31 July 2018
Available online 6 August 2018

Keywords:
Evolutionary algorithms
Combinatorial optimization
Additive group
Direct product
Knapsack problems

ABSTRACT

This paper proposes a group theory-based optimization algorithm (GTOA) for knapsack problems, which draws algebraic group operations into the evolution process. The key parts of GTOA are that the feasible solution of the knapsack problem is considered as an element of the direct product of groups and that the evolution process is implemented by multiplication and inverse operations of the direct product of groups. Based on the algorithms for handling infeasible solutions, GTOA is used to solve knapsack problems such as the Set-union knapsack problem, the Discounted {0-1} knapsack problem, and the Bounded knapsack problem. GTOA is validated to be an efficient algorithm for solving knapsack problems. A comparison between GTOA and existing evolutionary algorithms such as genetic algorithm, binary particle swarm optimization, binary artificial bee colony, and their improved variations is conducted and the comparative results show that GTOA has a better performance than other algorithms. In addition, GTOA is not only an efficient algorithm for solving knapsack problems but is also the first paradigm that applies group theory to directly design an evolutionary algorithm.

1. Introduction

Knapsack problems (KPs) [1,2] are classical NP-complete problems and important combinatorial optimization problems as well. Modeling and solving KPs has great theoretical significance and practical value in many fields such as resource allocations, capital budgets, investment decisions, industrial loading, economy and finance, and information security [3,4]. The classic knapsack problem is the 0-1 knapsack problem (0-1KP), and there exist many extensions of 0-1KP such as the bounded knapsack problem (BKP), the unbounded knapsack problem (UKP), the multidimensional knapsack problem (MDKP), the multiple knapsack problem (MKP), the quadratic knapsack problem (QKP), the set-union knapsack problem (SUKP), the discounted {0-1} knapsack problem (D{0-1}KP), the randomized time-varying knapsack problem (RTVKP), the quadratic multiple knapsack problem (QMKP), and the multiple-choice multidimensional knapsack problem (MMKP) [3,5–10]. Though there are many members of the KPs-family, they can be divided into three categories according to their representation of feasible solutions.

The feasible solution of the first category of KPs can be represented as an n-dimensional 0-1 vector $Y = (y_1, y_2, \ldots, y_n) \in \{0, 1\}^n$, where n is the number of items in KPs. Obviously, 0-1KP, MDKP, QKP, D{0-1}KP, SUKP, and RTVKP all belong to the first category of KPs. The second category of KPs use an n-dimensional integer vector $Y = (y_1, y_2, \ldots, y_n) \in \{0, 1, \ldots, m\}^n$ to represent a feasible solution, where n is the number of items in KPs, m is an integer and $m \geq 2$. MKP, QMKP, and D{0-1}KP all belong to the second category of KPs. Note that D{0-1}KP belongs to the first and second category of KPs simultaneously. The feasible solution of third category of KPs is an n-dimensional integer vector $Y = (y_1, y_2, \ldots, y_n) \in \{0, 1, \ldots, m_1\} \times \{0, 1, \ldots, m_2\} \times \cdots \times \{0, 1, \ldots, m_n\}$, where n is the number of items in KPs, and $m_1, m_2, \ldots, m_n$ are n positive integers in which at least two values are not equal. BKP, UKP, and MMKP all belong to the third category of KPs.

For solving KPs, there are many algorithms which can be mainly divided into two categories. One category consists of deterministic algorithms including dynamic programming, backtracking, and branch and bound [11,12]. The other category is non-deterministic algorithms which contains randomized algorithms, approximation algorithms, biological algorithms, and evolutionary algorithms [2,13–15]. Since there do not exist polynomial time deterministic algorithms to solve KPs, non-deterministic algorithms are more suited to quickly and approximately solve the KPs in practical applications.

Evolutionary algorithms (EAs) [15,16] have been well recognized as algorithms with swarm intelligence, which are essentially stochastic approximations. Their main advantage is that

* Corresponding author.
E-mail addresses: heyichao@hgu.edu.cn (Y. He), xizhaowang@ieee.org (X. Wang).

https://doi.org/10.1016/j.knosys.2018.07.045
0950-7051/

they do not need to calculate the derivatives and gradients of the objective function, do not require the objective function to be continuous, and do have the inherent implicit parallelism and the strong ability to search for global optimization as well. The most classic EAs include the genetic algorithm (GA) [17], particle swarm optimization (PSO) [18], differential evolution (DE) [19], harmony search algorithm (HS) [20], artificial fish swarm (AFS) [21], ant colony optimization (ACO) [22], and artificial bee colony (ABC) [23]. In recent years, EAs have had many successful applications in solving optimization problems, such as numerical optimization problems [24,25], knapsack problems [26], satisfiability problems [27], and set cover problems [28], which attract the great attention of scholars in many fields. Currently, by simulating behaviors of biological colonies in nature or learning from some phenomena of social activities, researchers have put forward many new EAs consecutively, such as the firefly algorithm (FFA) [29], the sine cosine algorithm (SCA) [30], the symbiotic organisms search (SOS) [31], the gray wolf optimizer (GWO) [32], and teaching–learning-based optimization (TLBO) [33]. Although nature provides us with endless inspiration, it is not perfect to design EAs solely based on bionic thought. In addition, the performance of many newly proposed EAs is not better than the classic ones, which accords with the No Free Lunch Theorem [34]. It is well known that, except for GA and ACO, almost existing EAs were initially put forward to solve numerical optimization problems and they generally could not be applied to directly solve combinatorial optimization problems. The methods proposed in the literature [35–37] that modify the evolution operator to fit to solve problems and in the literature [38,39] that map a real vector to a potential solution of problem are suitable only for the first category of KPs, not for the second and third ones.

Recently, the literatures [40,41] has used permutation group to improve the existing evolutionary algorithms, which are given an abstract algebraic differential mutation of DE and a vector operations of PSO, respectively, and used them to provide approaches to solving permutation problems. In fact, group theory not only can be used to improve the existing evolutionary algorithms, but also can be used to directly design a new evolutionary algorithm. In this paper, we will take the knapsack problems as an example to illustrate how to design a new evolutionary algorithm by using the direct product of groups which is generated by the additive group of $\boldsymbol{Z}/n\boldsymbol{Z}$, the integers modulo n, where $\boldsymbol{Z}$ is the integer set. From the point of view of algebra, we can see that no matter which category the KPs belong to, each dimensional component of its feasible solutions can be considered as an element in the additive group of $\boldsymbol{Z}/n\boldsymbol{Z}$ which can be simplified as $\boldsymbol{Z}_n = \{0, 1, \ldots, n-1\}$, where n is a positive integer and $n \geq 2$. Based on the abovementioned observation, an algebraic methodology for designing EAs is presented, and a Group Theory-based Optimization Algorithm (GTOA) is proposed by applying the direct product of groups. GTOA has a universal property and is suitable for solving KPs. The computations of SUKP, D{0-1}KP, and BKP illustrate that GTOA is not only easy to implement but also highly efficient. Moreover, GTOA is also a successful example of algorithm design based on group theory.

2. The proposed group theory-based optimization algorithm

2.1. Group and direct product of groups

The following is a brief introduction to the basic concepts and terminologies of the group and direct product of groups. More details can be found from literatures [42,43].

Let G be a nonempty set, and let $*$ be a binary operation on G, if

(1) $\forall a, b \in G, a * b \in G$;
(2) $\forall a, b, c \in G, (a * b) * c = a * (b * c)$;
(3) $\forall a \in G, \exists e \in G, a * e = e * a = a$;
(4) $\forall a \in G, \exists b \in G, a * b = b * a = e$, then $(G, *)$ is a group.

Henceforth, we shall write ab instead of $a * b$ and G instead of $(G, *)$. We call e the identity of G and, if $ab = ba = e$, then we call b the inverse of a, denoted by a^{-1}. It is clear that the inverse of e is itself.

Let G be a group, $a \in G$, and let i be an integer. Then, the ith power a^i of a is defined as following:

$$a^i = \begin{cases} a^{i-1}a, & \text{if } i > 0\ ; \\ e, & \text{if } i = 0\ ; \\ (a^{-1})^{-i}, & \text{if } i < 0\ . \end{cases} \tag{1}$$

Let $\boldsymbol{Z}_n = \{0, 1, \ldots, n-1\}$, n is a positive integer and $n \geq 2$. We define a binary operation $\bigoplus$ on $\boldsymbol{Z}_n$ as follows:

$$\forall a, b \in \boldsymbol{Z}_n, a \bigoplus b = (a + b)(mod n). \tag{2}$$

where $+$ is a common addition operator, $x(mod n)$ denotes the remainder when x is divided by n.

It is easy to see that $\boldsymbol{Z}_n$ is a group for operation $\bigoplus$, and its identity is 0. We denote $-a$ as the inverse of a in $\boldsymbol{Z}_n$. Then, -0=0, and, if $a \neq 0$ then $-a = n - a$.

If $G_1, G_2, \ldots, G_k$ are k groups, then their direct product, denoted by $G_1 \times G_2 \times \cdots \times G_k$, is the group with elements that are all ordered k tuple $(a_1, a_2, \ldots, a_k)$, where $a_i \in G_i (i = 1, 2, \ldots, k)$, and with operation

$$(a_1, \ldots, a_k)(b_1, \ldots, b_k) = (a_1 b_1, \ldots, a_k b_k). \tag{3}$$

It is easy to check that $G_1 \times G_2 \times \cdots \times G_k$ is a group: the identity is $(e_1, e_2, \ldots, e_k)$ where e_i is the identity of G_i $(i = 1, 2, \ldots, k)$ and the inverse $(a_1, a_2, \ldots, a_k)^{-1}$ is $(a_1^{-1}, a_2^{-1}, \ldots, a_k^{-1})$.

Let $\boldsymbol{Z}_{n_i} = \{0, 1, \ldots, n_i - 1\}$, where n_i is a positive integer and $n_i \geq 2$, $i = 1, 2, \ldots, k$. Then, $\boldsymbol{Z}_{n_1} \times \boldsymbol{Z}_{n_2} \times \cdots \times \boldsymbol{Z}_{n_k}$ is a direct product of groups, denoted by $\boldsymbol{Z}[n_1, n_2, \ldots, n_k]$. It is clear that $(0, 0, \ldots, 0)$ is the identity, and the inverse of $(a_1, a_2, \ldots, a_k)$ is $-(a_1, a_2, \ldots, a_k)$, and

$$-(a_1, a_2, \ldots, a_k) = (-a_1, -a_2, \ldots, -a_k). \tag{4}$$

where $-a_i$ is the inverse of a_i in $\boldsymbol{Z}_{n_i}$. $\forall (a_1, a_2, \ldots, a_k), (b_1, b_2, \ldots, b_k) \in \boldsymbol{Z}[n_1, n_2, \ldots, n_k]$, we have

$$(a_1, \ldots, a_k) \bigoplus (b_1, \ldots, b_k) = (a_1 \bigoplus b_1, \ldots, a_k \bigoplus b_k). \tag{5}$$

The ith power of $(a_1, a_2, \ldots, a_k)$ is $i(a_1, a_2, \ldots, a_k)$, where i is an integer.

It is easy to see that $\boldsymbol{Z}[2, 2, \ldots, 2] = \{0, 1\}^n$, $\boldsymbol{Z}[m+1, m+1, \ldots, m+1] = \{0, 1, \ldots, m\}^n$ and $\boldsymbol{Z}[m_1+1, m_2+1, \ldots, m_n+1] = \{0, 1, \ldots, m_1\} \times \{0, 1, \ldots, m_2\} \times \cdots \times \{0, 1, m_n\}$.

2.2. Group theory-based optimization algorithm

It is easy to see that the feasible solutions of the first category of KPs are the elements in the direct product of groups $\boldsymbol{Z}[2, 2, \ldots, 2]$; those of the second category are the elements in $\boldsymbol{Z}[m+1, m+1, \ldots, m+1]$; and those of the third category are the elements in $\boldsymbol{Z}[m_1+1, m_2+1, \ldots, m_n+1]$. We notice that $\boldsymbol{Z}[2, 2, \ldots, 2]$ and $\boldsymbol{Z}[m+1, m+1, \ldots, m+1]$ are special cases of $\boldsymbol{Z}[m_1+1, m_2+1, \ldots, m_n+1]$, respectively. Hence, the basic principle and the pseudo-code of GTOA will be described by applying the direct product $\boldsymbol{Z}[m_1+1, m_2+1, \ldots, m_n+1]$.

Suppose that $Y = (y_1, y_2, \ldots, y_n)$, $V = (v_1, v_2, \ldots, v_n)$, $W = (w_1, w_2, \ldots, w_n)$ are three different elements randomly selected from $\boldsymbol{Z}[m_1+1, m_2+1, \ldots, m_n+1]$. A new element $X = (x_1, x_2, \ldots, x_n) \in \boldsymbol{Z}[m_1+1, m_2+1, \ldots, m_n+1]$ is generated

by manipulating a group operation on Y, V, W according to the following equation:

$$X = Y \bigoplus (F(V \bigoplus (-W))). \tag{6}$$

where $x_j = y_j \bigoplus [f_j(v_j \bigoplus (m_j + 1 - w_j))]$, $j = 1, 2, \ldots, n$; $F = (f_1, f_2, \ldots, f_n)$ is an n-dimensional random vector in $\{-1, 0, 1\}^n$, which is called the Combinatorial Factor Vector. Since formula (6) is similar to the representation of a straight-line equation, we call it the Random Linear Combination Operator (RLCO).

It is worth to note that RLCO uses three different random elements to produce a new element on the space $\boldsymbol{Z}[m_1+1, m_2+1, \ldots, m_n+1]$, which indicates it is not only a random operation but has the characteristic of learning from other elements. Therefore, RLCO is a global random search operator and exhibits the global exploration ability by learning from others. However, it is not enough for an evolutionary algorithm to only have the global exploration ability. An excellent evolutionary algorithm should make a trade-off between exploration and exploitation, which are both indispensable factors for an evolutionary algorithm to work efficiently. In the following, we propose a local search operator over the space $\boldsymbol{Z}[m_1+1, m_2+1, \ldots, m_n+1]$ based on the inverse operation, which is named the Inversion and Random Mutation Operator (IRMO).

Let $X = (x_1, x_2, \ldots, x_n) \in \boldsymbol{Z}[m_1 + 1, m_2 + 1, \ldots, m_n + 1]$ and $m_i \geq 1$ $(i = 1, 2, \ldots, n)$, P_m be the mutation probability of IRMO. Then, the pseudo-code of IRMO is described as below.

Algorithm 1. IRMO

Input: $X = (x_1, x_2, \ldots, x_n)$, Mutation probability P_m.

Output: The mutated $X = (x_1, x_2, \ldots, x_n)$.

1 for $i \leftarrow 1$ to n do

2 if $(rand_1 < P_m)$ then

3 if $(rand_2 \leq 0.5 \wedge x_i \neq 0)$ then $x_i \leftarrow m_i + 1 - x_i$;

4 else $x_i \leftarrow rand([0, m_i] - \{x_i\})$;

5 end if

6 end for

In IRMO, $rand_1$ and $rand_2$ are two random numbers in the interval (0, 1); the mutation probability P_m satisfies $0 < P_m \leq 0.5$; $rand([0, m_i] - \{x_i\})$ represents a random integer that is not equal to x_i in $[0, m_i]$. It is obvious that the probabilities of obtaining the inverse mutation (Step3) and the random mutation (Step4) are equal. The time complexity of IRMO is $O(n)$.

In addition, in group $\boldsymbol{Z}_2 = \{0, 1\}$ under modulo 2, since $-0 = 0$ and $-1 = 1$, the IRMO does not work for 0-1 vectors in the search space $\boldsymbol{Z}[2, 2, \ldots, 2]$. Drawing lessons from the mutation operator of GA [17], a local search operator suitable for the space $\boldsymbol{Z}[2, 2, \ldots, 2]$ is proposed, we name it the Switch Mutation Operator (SMO).

Let $X = (x_1, x_2, \ldots, x_n) \in \boldsymbol{Z}[2, 2, \ldots, 2]$ and P_m be the mutation probability of SMO. Then, the pseudo-code of SMO is described as follow:

Algorithm 2. SMO

Input: $X = (x_1, x_2, \ldots, x_n)$, Mutation probability P_m.

Output: The mutated $X = (x_1, x_2, \ldots, x_n)$.

1 for $i \leftarrow 1$ to n do

2 if $(rand \leq P_m)$ then $x_i \leftarrow 1 - x_i$;

3 end for

In SMO, $rand$ is a random number in (0, 1), and the mutation probability P_m satisfies $0 < P_m \leq 0.5$. It is easy to see that the time complexity of SMO is $O(n)$.

Let $\boldsymbol{P}(t) = \{X_i(t) \mid 1 \leq i \leq NP\}$ be the tth generation population of GTOA, where $X_i(t) = (x_{i1}(t), x_{i2}(t), \ldots, x_{in}(t)) \in \boldsymbol{Z}[m_1 + 1, m_2 + 1, \ldots, m_n + 1]$ is the ith individual, NP is the size of the population, t is an integer and $t \geq 0$. $Fit(X_i(t))$ denotes the fitness of the individual $X_i(t)$ and its computation generally relies on the objective function. Let $B = (b_1, b_2, \ldots, b_n) \in \boldsymbol{Z}[m_1 + 1, m_2 + 1, \ldots, m_n + 1]$ be the best individual in $\boldsymbol{P}(MIT)$, MIT be the number of maximum iterations, and $Y = (y_1, y_2, \ldots, y_n) \in \boldsymbol{Z}[m_1 + 1, m_2 + 1, \ldots, m_n + 1]$ be an n-dimensional vector. For the KPs with the objective function $Maxf(X)$, the pseudo-code of GTOA is described as follows:

Algorithm 3. GTOA

Input: An instance of KPs; Parameters NP, MIT, m_1, m_2, …, m_n; Mutation probability P_m .

Output: An approximate solution (or optimal solution) B and $f(B)$.

1 Generate initial population $\boldsymbol{P}(0) = \{X_i(0) \mid 1 \leq i \leq NP\}$ randomly;

2 Compute $Fit(X_i(0))$, $1 \leq i \leq NP$, and $t \leftarrow 0$

3 while $(t < MIT)$ do

4 for $i \leftarrow 1$ to NP do

5 $Y \leftarrow X_{p_1}(t) \bigoplus (F(X_{p_2}(t) \bigoplus (-X_{p_3}(t))))$;

6 $Y \leftarrow \boldsymbol{IRMO}(Y, P_m)$; (or $Y \leftarrow \boldsymbol{SMO}(Y, P_m)$;)

7 if $Fit(Y) > Fit(X_i(t))$ then $X_i(t+1) \leftarrow Y$;

8 else $X_i(t+1) \leftarrow X_i(t)$;

9 end for

10 Generate $\boldsymbol{P}(t+1)$, and $t \leftarrow t+1$;

11 end while

12 Determine B in $\boldsymbol{P}(MIT)$;

13 return $(B, f(B))$.

In Step 5 of GTOA, a temporary individual $Y = (y_1, y_2, \ldots, y_n) \in \boldsymbol{Z}[m_1+1, m_2+1, \ldots, m_n+1]$ is generated by the RLCO operator, where $X_{p_1}(t)$, $X_{p_2}(t)$ and $X_{p_3}(t)$ are three different individuals randomly selected from the tth generation population $\boldsymbol{P}(t)$. The individual Y is mutated by SMO in Step 6 when solving the first category of KPs and by IRMO when solving those in the second and third categories. Noting that both MIT and NP are multiples of n, the time complexity of GTOA is $O(MIT * n * NP) = O(n^3)$, which indicates that GTOA is a stochastic approximation algorithm.

3. Solving knapsack problems by GTOA

To illustrate the effectiveness and wide application range of GTOA, we select three typical knapsack problems, i.e., SUKP, D{0-1}KP, and BKP, and use GTOA to solve them. Since these three problems are constrained optimization problems, their infeasible solutions will be unavoidably generated during GTOA's solving process. At present, there are several common methods for handling infeasible solutions, such as the Penalty function approach, the Repair approach, the Purist approach, the Separation approach, and the Hybrid approach [44–49]. Existing studies show that the greedy strategy-based repair and optimization methods are the best fit for KPs [9,15,44,45]. According to the mathematical model of KPs and the greedy strategy-based repair and optimization method, we will present the efficient approaches to solving KPs by using GTOA.

3.1. Solving set union knapsack problem by GTOA

The SUKP [7,50], belonging to the first category of KPs, has important application in investment decision-making, flexible manufacturing systems, the database system with primary, and secondary memories [3,7,51]. Goldschmidt et al. [7] first used hyper graph to propose a deterministic algorithm for solving SUKP based on dynamic programming, but its time complexity is exponential, which leads to very poor practicability. By means of the d-regular graph theory, Ashwin Arulselvan [50] proposed an approximation algorithm A-SUKP with the approximation ratio of $\frac{1}{(1-e^{\frac{-1}{d}})}$ based on the greedy strategy, where $d(d \geq 2)$ is the supremum of the frequencies of all elements. A-SUKP greatly improved the speed for solving SUKP. Yichao He et al. [44] proposed a novel binary artificial bee colony (BABC) to solve SUKP and conducted a comparison with A-SUKP showing the BABC not only has a better performance but is also faster.

Definition of SUKP: Given a set of elements $\boldsymbol{U} = \{1, 2, \ldots, m\}$ and a set of items $\boldsymbol{S} = \{1, 2, \ldots, n\}$, where each item $i \in \boldsymbol{S}$ corresponds to a nonempty subset $\boldsymbol{U}_i \subseteq \boldsymbol{U}$ and has a profit $p_i > 0$; every element $j \in \boldsymbol{U}$ has a weight $w_j > 0$. For any nonempty subset $\boldsymbol{A} \subseteq \boldsymbol{S}$, the profit of $\boldsymbol{A}$ is defined as $P(\boldsymbol{A}) = \sum_{i \in \boldsymbol{A}} p_i$, and the weight is $W(\boldsymbol{A}) = \sum_{j \in \bigcup_{i \in \boldsymbol{A}} \boldsymbol{U}_j} w_j \leq C$. For a given knapsack capacity C, The goal of SUKP is to find a subset of items $\boldsymbol{S}^* \subseteq \boldsymbol{S}$, such that $P(\boldsymbol{S}^*)$ is maximized and $W(\boldsymbol{S}^*) \leq C$.

Without loss of generality, we suppose that $p_i\ (i = 1, 2, \ldots, n)$, $w_j\ (j = 1, 2, \ldots, m)$ and C are positive integers, the set family $\{\boldsymbol{U}_1, \boldsymbol{U}_2, \ldots, \boldsymbol{U}_n\}$ is a cover of $\boldsymbol{U}$, where $\boldsymbol{U}_i \subseteq \boldsymbol{U}$ and $\boldsymbol{U}_i \neq \emptyset$; $W(\boldsymbol{S}) > C$, and for $i \in \boldsymbol{S}$, $\sum_{j \in U_i} w_j \leq C$. Let $X = (x_1, x_2, \ldots, x_n)$ be an n-dimensional 0-1 vector, and $\boldsymbol{A}_X = \{i | x_i \in X \wedge x_i = 1, 1 \leq i \leq n\} \subseteq \boldsymbol{S}$. For any $i\ (i = 1, 2, \ldots, n)$, $x_i = 1$ if and only if $i \in \boldsymbol{A}_X$. By the one-to-one relationship between X and $\boldsymbol{A}_X$, the mathematical model of SUKP is represented as:

$$Max\ \ f(X) = \sum_{i=1}^{n} x_i p_i \tag{7}$$

$$s.t.\ \ W(A_X) = \sum_{j \in \bigcup_{i \in A_X} U_i} w_j \leq C \tag{8}$$

When using GTOA to solve SUKP, every individual is an n-dimensional 0-1 vector on $\{0, 1\}^n$. It should be noted that any of the n-dimensional 0-1 vectors on $\{0, 1\}^n$ is only a potential solution of SUKP, and it is a feasible solution only when satisfying the constraint (8), otherwise it is an infeasible solution. Yichao He et al. [44] proposed a repair and optimization algorithm S-GROA to eliminate the infeasible solutions when solving SUKP by EAs. In S-GROA, $d_j\ (j = 1, 2, \ldots, m)$ denotes the frequency of the element $j (j \in \boldsymbol{U})$ in subsets $\boldsymbol{U}_1, \boldsymbol{U}_2, \ldots, \boldsymbol{U}_n$, and $T_i = \sum_{j \in \boldsymbol{U}_i} (w_j / d_j)$ $(i = 1, 2, \ldots, n)$. On the basis of using S-GROA to deal with infeasible solution, the algorithm for solving SUKP by GTOA can be described as follows:

Algorithm 4. GTOA for SUKP

Input: An instance of SUKP; Parameters NP, MIT, $m_1, m_2, \ldots, m_n$; Mutation probability P_m .

Output: An approximate solution (or optimal solution) B and $f(B)$.

1 Sorting $p_i/T_i\ (i = 1, 2, \ldots, n)$ in descending order. Storing the subscripts of ordered p_i/T_i in array $H[1 \ldots n]$;

2 Generate initial population $\boldsymbol{P}(0) = \{X_i(0) \in \{0, 1\}^n \mid 1 \leq i \leq NP\}$ randomly, and $t \leftarrow 0$;

3 $(X_i(t), f(X_i(t))) \leftarrow \boldsymbol{S-GROA}(X_i(t), H[1 \ldots n]),\ 1 \leq i \leq NP$;

4 while$(t < MIT)$ do

5 for $i \leftarrow 1$ to NP do

6 $Y \leftarrow X_{p1}(t) \bigoplus (F(X_{p2}(t) \bigoplus (-X_{p3}(t))))$;

7 $Y \leftarrow \boldsymbol{SMO}(Y, P_m)$;

8 $(Y, f(Y)) \leftarrow \boldsymbol{S-GROA}(Y, H[1 \ldots n])$;

9 if $f(Y) > f(X_i(t))$ then $X_i(t+1) \leftarrow Y$;

10 else $X_i(t+1) \leftarrow X_i(t)$;

11 end for

12 Generate $\boldsymbol{P}(t+1)$, and $t \leftarrow t+1$;

13 end while

14 Determine B in $\boldsymbol{P}(MIT)$;

15 return$(B,\ f(B))$.

Since the time complexity of the *SortAlgorithm* [12] and S-GROA [44]are $O(nlogn)$ and $O(nm)$, respectively, the time complexity of Algorithm 4 is $O(n^3 m)$.

3.2. Solving discounted $\{0-1\}$ knapsack problem by GTOA

Guldan [5] first proposed D{0-1}KP and gave its heuristic and deterministic algorithms. Aiying Rong et al. [6] defined the kernel of D{0-1}KP and discussed the validity of solving D{0-1}KP by deterministic algorithm. Yichao He et al. [45] presented two evolutionary algorithms FirEGA and SecEGA for solving D{0-1}KP based on GA. Furthermore, they studied deterministic algorithms and approximation algorithms of D{0-1}KP in the literature [49], and note that PSO-GRDKP is a high performance evolutionary algorithm for solving D{0-1}KP.

Definition of D{0-1}KP [5,6]: Given a set of n item groups, suppose that each group $i\ (i = 0, 1, \ldots, n-1)$ consists of three items: $3i$, $3i+1$ and $3i+2$. The item $3i$ has weight w_{3i} and profit p_{3i}, and the item $3i+1$ has weight w_{3i+1} and profit p_{3i+1}. The first two items $3i$ and $3i+1$ are paired to derive the third item $3i+2$ with profit $p_{3i+2} = p_{3i} + p_{3i+1}$ and the discounted weight w_{3i+2}, which satisfies $w_{3i+2} < w_{3i} + w_{3i+1}$, $w_{3i} < w_{3i+2}$ and $w_{3i+1} < w_{3i+2}$. In each group, at most one of the three items can be selected to be placed in the knapsack with capacity C. The problem is how to select items loaded into the knapsack such that the total profit is maximized under the condition that the total weight of the selected items does not exceed C.

Without losing generality, we suppose that p_j, $w_j\ (0 \leq j \leq 3n-1)$, and C are positive integers, $w_{3i+2} \leq C\ (0 \leq i \leq n-1)$, and $\sum_{i=0}^{n-1} w_{3i+2} > C$. Two mathematical models of D{0-1}KP are represented as follows.

The first mathematical model [5,6]:

$$Max\ \ f(X) = \sum_{i=0}^{n-1} (x_{3i} p_{3i} + x_{3i+1} p_{3i+1} + x_{3i+2} p_{3i+2}) \tag{9}$$

$$s.t.\ \ x_{3i} + x_{3i+1} + x_{3i+2} \leq 1, i = 0, 1, \ldots, n-1, \tag{10}$$

$$\sum_{i=0}^{n-1} (x_{3i} w_{3i} + x_{3i+1} w_{3i+1} + x_{3i+2} w_{3i+2}) \leq C, \tag{11}$$

$$x_{3i}, x_{3i+1}, x_{3i+2} \in \{0, 1\}, i = 0, 1, \ldots, n-1. \tag{12}$$

where $X = (x_0, x_1, \ldots, x_{3n-1}) \in \{0, 1\}^{3n}$, $x_j (0 \le j \le 3n-1)$ is used to indicate whether the item j is loaded into knapsack. The item j is loaded into knapsack if and only if $x_j = 1$.

The second mathematical model [45]:

$$Max\ \ f(X) = \sum_{i=0}^{n-1} \lceil \frac{x_i}{3} \rceil p_{3i+|x_i-1|} \tag{13}$$

$$s.t.\ \sum_{i=0}^{n-1} \lceil \frac{x_i}{3} \rceil w_{3i+|x_i-1|} \le C \tag{14}$$

$$x_i \in \{0, 1, 2, 3\}, i = 0, 1, \ldots, n-1. \tag{15}$$

where $X = (x_0, x_1, \ldots, x_{n-1}) \in \{0, 1, 2, 3\}^n$. The integer variable $x_i (0 \le i \le n-1)$ indicates whether there is an item of the item group i to be loaded into the knapsack. $x_i = 0$ indicates that no items of item group i is loaded into the knapsack; $x_i = 1$ expresses that the item $3i$ is loaded into the knapsack; $x_i = 2$ indicates that the item $3i+1$ is loaded into the knapsack; $x_i = 3$ expresses that the item $3i+2$ is loaded into the knapsack. $\lceil x \rceil$ is top function of x representing the smallest integer not less than x. Obviously, the length of the feasible solution in the second mathematical model of D{0-1}KP is 1/3 that of the first one. Therefore, when use GTOA to solve D{0-1}KP under the second mathematical model, it cannot only save storage space but also improve the algorithm's running speed.

To eliminate the infeasible solutions of D{0-1}KP, we refer to the skill in [49] to propose a repair and optimization algorithm D-GROA. Similar to SUKP, sort all items of the D{0-1}KP instance according to the order of p_j/w_j $(0 \le j \le 3n-1)$ from large to small, and store the subscripts of the items into the array $H[0 \ldots 3n-1]$ according to the order. Let $X = (x_0, x_1, \ldots, x_{n-1}) \in \{0, 1, 2, 3\}^n$ be a potential solution of the D{0-1}KP instance; $\lfloor x \rfloor$ is the bottom function representing the maximum integer not greater than x; $H[j](mod3)$ denotes the remainder when $H[j]$ is divided by 3. Then, the pseudo-code of D-GROA is described as follows:

Algorithm 5. $\boldsymbol{D-GROA}$

Input: A potential solution $X = (x_0, x_1, \ldots, x_{n-1}) \in \{0, 1, 2, 3\}^n$ and array $H[1 \ldots 3n]$.

Output: A feasible solution $X = (x_0, x_1, \ldots, x_{n-1})$ and its objective function value $f(X)$.

1 $R \leftarrow \sum_{i=0}^{n-1} \lceil \frac{x_i}{3} \rceil w_{3i+|x_i-1|}$; $j \leftarrow 3n-1$;
2 while $(R > C)$ do //Repair phase
3 $k \leftarrow \lfloor H[j]/3 \rfloor$; $r \leftarrow H[j](mod3)$;
4 if $(x_k = r+1)$ then $x_k \leftarrow 0$; $R \leftarrow R - w_{H[j]}$;
5 $j \leftarrow j-1$;
6 end while
7 for $j \leftarrow 0$ to $3n-1$ do //Optimizing phase
8 $k \leftarrow \lfloor H[j]/3 \rfloor$; $r \leftarrow H[j](mod3)$;
9 if $(x_k = 0) \wedge (R + w_{H[j]} \le C)$ then
10 $R \leftarrow R + w_{H[j]}$; $x_k \leftarrow r+1$;
11 end if
12 end for
13 return$(X, f(X))$.

In D-GROA, $f(X) = \sum_{i=0}^{n-1} \lceil \frac{x_i}{3} \rceil p_{3i+|x_i-1|}$ is used as the fitness of X, which is obtained in Step 13. The time complexity of D-GROA is $O(n)$.

On the basis of using D-GROA to eliminate infeasible solutions, the algorithm for solving D{0-1}KP by GTOA is given as follows.

Algorithm 6. $\boldsymbol{GTOA\ for\ D\{0-1\}KP}$

Input: An instance of D{0-1}KP; Parameters NP, MIT, $m_1, m_2, \ldots, m_n$; Mutation probability P_m .

Output: An approximate solution (or optimal solution) B and $f(B)$.

1 Sort p_j/w_j $(j = 0, 1, \ldots, 3n-1)$ in descending order. Store the subscripts of ordered p_j/w_j in array $H[0 \ldots 3n-1]$;
2 Generating initial population $\boldsymbol{P}(0) = \{X_i(0) \in \{0, 1, 2, 3\}^n \mid 1 \le i \le NP\}$ randomly, $t \leftarrow 0$;
3 $(X_i(t), f(X_i(t))) \leftarrow \boldsymbol{D-GROA}(X_i(t), H[0 \ldots 3n-1])$, $1 \le i \le NP$;
4 while$(t < MIT)$ do
5 for $i \leftarrow 1$ to NP do
6 $Y \leftarrow X_{p1}(t) \bigoplus (F(X_{p2}(t) \bigoplus (-X_{p3}(t))))$;
7 $Y \leftarrow \boldsymbol{IRMO}(Y, P_m)$;
8 $(Y, f(Y)) \leftarrow \boldsymbol{D-GROA}(Y, H[0 \ldots 3n-1])$;
9 if $f(Y) > f(X_i(t))$ then $X_i(t+1) \leftarrow Y$
10 else $X_i(t+1) \leftarrow X_i(t)$;
11 end for
12 Generate $\boldsymbol{P}(t+1)$, and $t \leftarrow t+1$;
13 end while
14 Determine B in $\boldsymbol{P}(MIT)$;
15 return$(B, f(B))$.

Obviously, the time complexity of Algorithm 6 is $O(n^3)$.

Obviously, the time complexity of Algorithm 6 is $O(n^3)$.

3.3. Solving the bounded knapsack problem by GTOA

The BKP [3] belongs to the third category of KPs. It is defined as follows. Let $N = \{1, 2, \ldots, n\}$ be a set of n items to be packed into a knapsack of size C. For $j = 1, 2, \ldots, n$, let p_j, w_j and b_j be the profit, weight and the number of identical copies available for the jth item, respectively, where C and p_j, w_j, b_j are all positive integers. The objective is to fill the knapsack with some items from N whose total weight is at most C and such that their total profit is maximized. The mathematical model of BKP is shown as follows.

$$Max\ \ f(X) = \sum_{j=1}^{n} p_j x_j \tag{16}$$

$$s.t\ \ \sum_{j=1}^{n} w_j x_j \le C \tag{17}$$

$$0 \le x_j \le b_j, x_j\ is\ integer, j = 1, 2, \ldots, n. \tag{18}$$

where $X = (x_1, x_2, \ldots, x_n) \in \boldsymbol{Z}[b_1+1, b_2+1, \ldots, b_n+1]$. None of the copies of the jth item is loaded into the knapsack when $x_j = 0$, otherwise x_j copies of the jth item are loaded. Obviously, 0-1KP is a special case of BKP when $b_j = 1 (j = 1, 2, \ldots, n)$. Without loss of generality, suppose there is at least one j which satisfies $b_j \ge 2$, $Max\{b_j w_j \mid 1 \le j \le n\} \le C$ and $\sum_{j=1}^{n} b_j w_j > C$.

Similar to SUKP and D{0-1}KP, we give a repair and optimization algorithm B-GROA to handle infeasible solution: Sort all the items in the BKP instance according to the order of p_j/w_j $(1 \le j \le n)$ from large to small, and store the subscripts sequentially into the array $H[1 \ldots n]$ based on the order. Let $X = (x_1, x_2, \ldots, x_n) \in \boldsymbol{Z}[b_1+1, b_2+1, \ldots, b_n+1]$ denote a potential solution of BKP. The pseudo-code of B-GROA is described as below.

Algorithm 7. $B-GROA$

Input: A potential solution $X=(x_1,x_2\ldots,x_n)\in \boldsymbol{Z}[b_1+1,\ldots,b_n+1]$ and array $H[1\ldots n]$.

Output: A feasible solution $X=(x_1,x_2\ldots,x_n)$ and its objective function value $f(X)$.

$R\leftarrow\sum_{j=1}^{n}w_jx_j;\quad j\leftarrow n;$
while $(R>C)$ do //Repair phase
  if $(x_{H[j]}>0)$ then
    $temp\leftarrow x_{H[j]};$
    $x_{H[j]}\leftarrow Max\{0,x_{H[j]}-\lceil(R-C)/w_{H[j]}\rceil\}$;
    $R\leftarrow R-w_{H[j]}*(temp-x_{H[j]});$
  end if
  $j\leftarrow j-1;$
end while
for $j\leftarrow 1$ to n do //Optimizing phase
  if $(x_{H[j]}<b_{H[j]})\wedge(R+w_{H[j]}\leq C)$ then
    $temp\leftarrow x_{H[j]};$
    $x_{H[j]}\leftarrow Min\{b_{H[j]},x_{H[j]}+\lfloor(R-C)/w_{H[j]}\rfloor\}$;
    $R\leftarrow R+w_{H[j])}*(x_{H[j]}-temp);$
  end if
end for
return$(X,\ f(X))$.

Obviously, the time complexity of B-GROA is $O(n)$.

When solving BKP by GTOA, we apply B-GROA to eliminate infeasible solution, which is similar to the procedure in SUKP and D{0-1}KP. To avoid duplication, we omit it.

4. Experimental results and comparative studies

As we all know, GA [17,46] is one of the most commonly used classic EAs, suitable for solving combinatorial optimization problems. BPSO [36] is a simple and effective EAs, and performs excellently when solving D{0-1}KP [49] and RTVKP [52]. Literature [44] is the only study that uses EAs to solve SUKP, in which the proposed BABC performs better than GA [17,46], binDE [35], ABCbin [38], and A-SUKP [50]. To verify the performance of GTOA for solving KPs, we experimentally conduct a comparison between GTOA and GA, BPSO, BABC, FirEGA [45], and SecEGA [45] for their computational performance.

All experiments are were carried out on Acer Aspire E1-570G with Intel(R) Core(TM)i5-3337u CPU-1.8 GHz, 4GB DDR3 (3.82 GB available); the operating system is Microsoft Windows 8. All algorithms are implemented using C ++, and the compilation environment is Visual C ++ 6.0. The figures are drawn by the MATLAB7.10.0.499 (R2010a).

4.1. Simulation results for the SUKP instances

4.1.1. Parameters setting of the algorithms

To determine the optimal value of P_m in GTOA, we solve three instances *sukp*300_285_0.10_0.75, *sukp*300_300_0.10_0.75, and *sukp*285_300_0.10_0.75 [44] by GTOA when $P_m=0.005$, 0.006, 0.007, 0.008, 0.009, 0.010, and 0.011. Each instance is independently calculated 100 times. According to the box-plots in Figs. 1–3, it is easy to see that $0.008\leq P_m\leq 0.009$ is the most reasonable choice.

When using GTOA,GA,BPSO, and BABC to solve three kinds of large-scale SUKP instances [44], the population size of all algorithms is set as $NP=20$, individual encoding is an n-dimensional 0-1 vector, the iteration number is set as $MIT=Max\{m,n\}$, n is the number of items and m is the number of elements, S-GROA [44] is used to deal with infeasible solutions. In GTOA, F is a random 0-1 vector, $P_m=0.008$. In GA [46], the single point crossover operator, the uniform mutation operator, and the fitness proportional model are used; the crossover probability is $P_c=0.8$ and the mutation probability is $P_m=0.01$. In BPSO [36], set $W=1.0$, $C_1=C_2=2.0$, and each dimensional component in the n-dimensional real vector is in $[-5.0,5.0]$. In BABC [44], $a=5.0$ and the threshold value is $limit=Max\{m,n\}/5$.

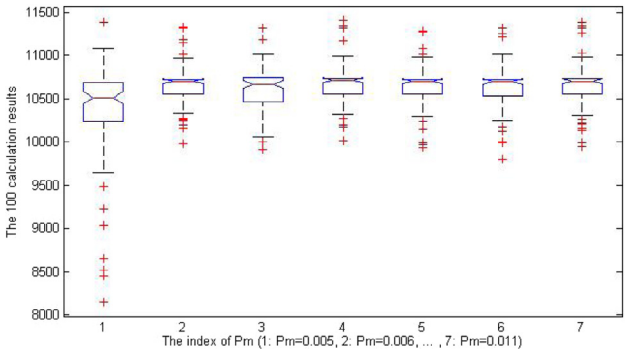

Fig. 1. Boxplots of instance sukp300_285_0.10_0.75 when parameter $P_m=0.005, 0.006, 0.007, 0.008, 0.009, 0.010,$ *and* 0.011 respectively.

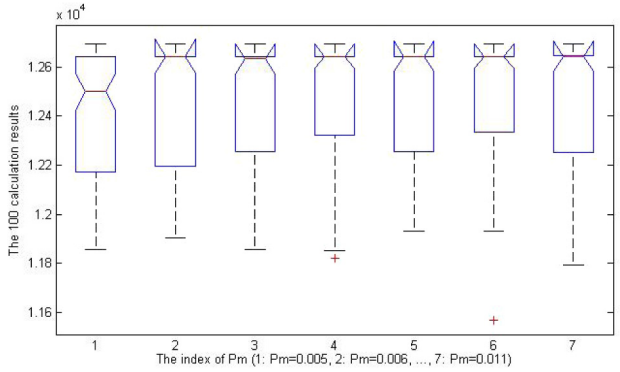

Fig. 2. Boxplots of instance sukp300_300_0.10_0.75 when parameter $P_m=0.005, 0.006, 0.007, 0.008, 0.009, 0.010,$ *and* 0.011 respectively.

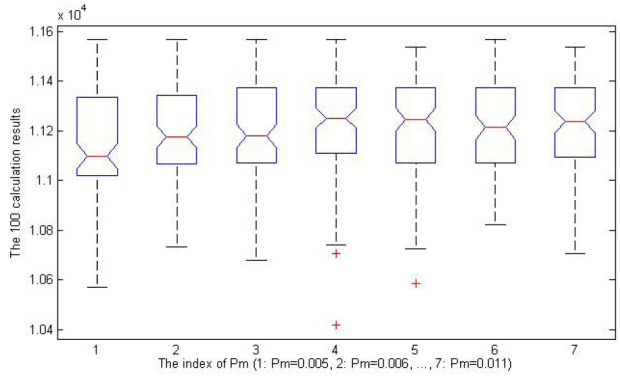

Fig. 3. Boxplots of instance sukp285_300_0.10_0.75 when parameter $P_m=0.005, 0.006, 0.007, 0.008, 0.009, 0.010,$ *and* 0.011 respectively.

4.1.2. Calculation and comparison

In Tables 1–3 (in Appendix), *CBEST* is the best result currently known for the instance. *Best* and *Worst* are the best value and the worst value, respectively, obtained by GTOA, GA, BPSO, and BABC when solving every instance 100 times independently. *Mean* and

Table 1
Comparison among GTOA, GA, BPSO and BABC for the 1st category of SUKP.

Index	Instance	*CBEST*	Algorithm	*Best*	*Mean*	*Worst*	*StD*	*Time*
1	*sukp*100_85	13283	GA	13044	12956.4	12596	130.66	0.112
	_0.10_0.75		BPSO	13082	12979.2	12497	147.28	0.199
			BABC	**13251**	**13028.5**	**12763**	92.63	0.210
			GTOA	**13251**	13025.4	**12763**	59.70	0.134
2	*sukp*100_85	12479	GA	12066	11546.0	11296	214.94	0.119
	_0.15_0.85		BPSO	12238	12089.0	11576	128.99	0.225
			BABC	12238	**12155.0**	**12066**	53.29	0.223
			GTOA	**12274**	12029.3	11483	127.25	0.150
3	*sukp*200_185	13405	GA	13064	12492.5	12596	320.03	1.013
	_0.10_0.75		BPSO	13241	12831.6	11687	434.69	1.640
			BABC	13241	13064.4	12808	99.57	1.562
			GTOA	**13405**	**13196.9**	**12932**	127.43	1.160
4	*sukp*200_185	14215	GA	13671	12802.9	12332	291.66	1.133
	_0.15_0.85		BPSO	14044	**13380.7**	12782	332.27	1.737
			BABC	13829	13359.2	**12881**	234.99	1.729
			GTOA	**14215**	13285.9	12763	276.35	1.277
5	*sukp*300_285	11413	GA	10553	9980.9	9640	142.97	3.608
	_0.10_0.75		BPSO	10869	10371.9	9708	240.20	5.759
			BABC	10428	9994.8	9661	154.03	5.281
			GTOA	**11407**	**10674.5**	**10032**	599.55	3.957
6	*sukp*300_285	12245	GA	11016	10349.8	9906	215.13	3.899
	_0.15_0.85		BPSO	**12245**	11034.1	10056	581.86	5.812
			BABC	12012	10902.9	9929	449.45	5.673
			GTOA	**12245**	**11533.4**	**10382**	466.27	4.144
7	*sukp*400_385	11435	GA	10083	9641.9	9370	168.94	9.779
	_0.10_0.75		BPSO	11230	10580.0	9915	329.43	13.375
			BABC	10766	10065.2	9515	241.45	12.976
			GTOA	**11435**	**10830.3**	**10051**	353.33	9.165
8	*sukp*400_385	10397	GA	9831	9326.8	8980	192.20	9.978
	_0.15_0.85		BPSO	9990	9500.4	9081	278.67	15.076
			BABC	9649	9136.0	8894	151.90	13.359
			GTOA	**10397**	**9894.6**	**9142**	290.02	9.973
9	*sukp*500_485	11716	GA	11031	10567.9	10288	123.15	18.198
	_0.10_0.75		BPSO	11473	10839.5	10322	311.32	26.049
			BABC	10784	10452.2	10257	114.35	25.372
			GTOA	**11716**	**11171.4**	**10591**	299.00	18.476
10	*sukp*500_485	9892	GA	9472	8692.7	8400	180.12	19.720
	_0.15_0.85		BPSO	9446	9012.9	8627	197.57	29.790
			BABC	9090	8857.9	8651	94.55	26.874
			GTOA	**9860**	**9262.0**	**8742**	153.89	20.053

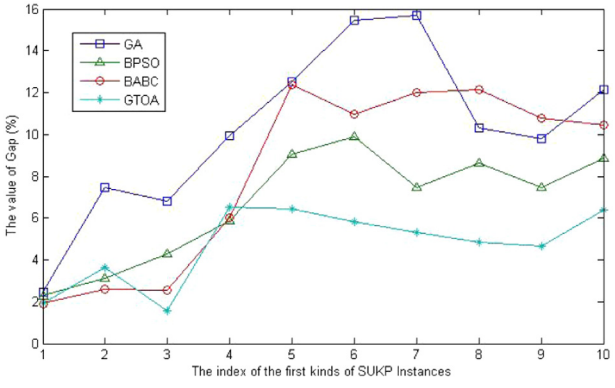

Fig. 4. *Gap* fitting curve of the 1st category of SUKP instances.

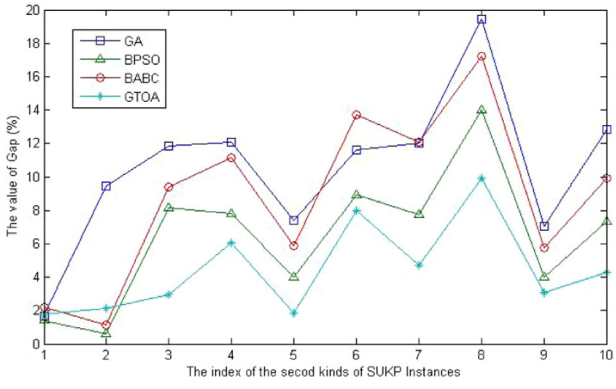

Fig. 5. *Gap* fitting curve of the 2nd category of SUKP instances.

StD denote the mean value and the standard deviation, respectively, and *Time* is the average running time that each algorithm takes for individually solving every instance.

Tables 1–3 illustrate that, among the 30 instances, GTOA can achieve the best result currently known of 22 instances, BPSO does that on 5 instances, BABC does that on 2 instances, while

Table 2
Comparison among GTOA, GA, BPSO and BABC for the 2nd category of SUKP.

Index	Instance	*CBEST*	Algorithm	*Best*	*Mean*	*Worst*	*StD*	*Time*
1	*sukp*100_100	14044	GA	**14044**	13806.0	13587	144.91	0.129
	_0.10_0.75		BPSO	**14044**	**13846.1**	**13664**	62.21	0.307
			BABC	13860	13734.9	13573	70.76	0.213
			GTOA	**14044**	13792.5	13561	90.57	0.161
2	*sukp*100_100	13508	GA	13145	12234.8	11582	388.66	0.143
	_0.15_0.85		BPSO	**13508**	**13428.9**	**13104**	115.87	0.270
			BABC	**13508**	13352.4	12837	155.14	0.244
			GTOA	**13508**	13220.5	11988	296.61	0.175
3	*sukp*200_200	12350	GA	11656	10888.7	10337	237.85	1.106
	_0.10_0.75		BPSO	12019	11344.8	10641	330.30	2.221
			BABC	11846	11194.3	10581	249.58	1.633
			GTOA	**12350**	**11983.4**	**10903**	326.08	1.302
4	*sukp*200_200	12317	GA	11792	10827.5	10304	334.43	1.183
	_0.15_0.85		BPSO	11821	11357.2	10607	381.88	1.922
			BABC	11521	10945.0	10436	255.14	1.819
			GTOA	**12317**	**11572.8**	**10957**	242.36	1.410
5	*sukp*300_300	12695	GA	12055	11755.1	11169	144.45	3.789
	_0.10_0.75		BPSO	12644	12187.6	11807	180.18	6.166
			BABC	12186	11945.8	11724	127.80	5.315
			GTOA	**12695**	**12464.1**	**11968**	231.68	4.190
6	*sukp*300_300	11425	GA	10666	10099.2	**9549**	337.42	4.106
	_0.15_0.85		BPSO	11007	10409.4	9463	304.99	6.707
			BABC	10382	9859.7	9476	177.02	6.019
			GTOA	**11425**	**10513.9**	9477	355.88	4.398
7	*sukp*400_400	11490	GA	10570	10112.4	9786	157.89	9.187
	_0.10_0.75		BPSO	11310	10600.5	10022	271.05	13.419
			BABC	10626	10101.1	9756	196.99	12.805
			GTOA	**11450**	**10951.9**	**10360**	264.34	9.695
8	*sukp*400_400	10915	GA	9235	8793.8	8501	169.52	9.830
	_0.15_0.85		BPSO	10404	9383.6	8597	411.48	15.103
			BABC	9541	9033.0	8553	194.18	12.953
			GTOA	**10915**	**9834.43**	**9145**	312.87	10.429
9	*sukp*500_500	10960	GA	10460	10185.4	9919	114.19	20.717
	_0.10_0.75		BPSO	10888	10522.4	**10139**	166.42	28.253
			BABC	10755	10328.5	**10139**	91.62	27.735
			GTOA	**10960**	**10626.1**	10048	152.96	18.592
10	*sukp*500_500	10194	GA	9496	8882.9	8577	158.21	20.379
	_0.15_0.85		BPSO	9840	9447.9	8731	202.16	32.389
			BABC	9318	9180.8	8833	84.91	27.813
			GTOA	**10194**	**9754.7**	**9044**	231.77	20.291

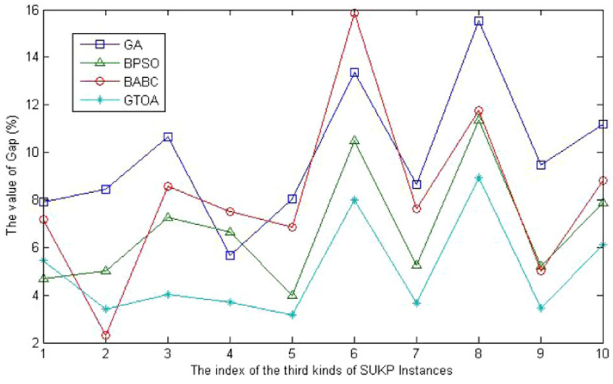

Fig. 6. *Gap* fitting curve of the 3rd category of SUKP instances.

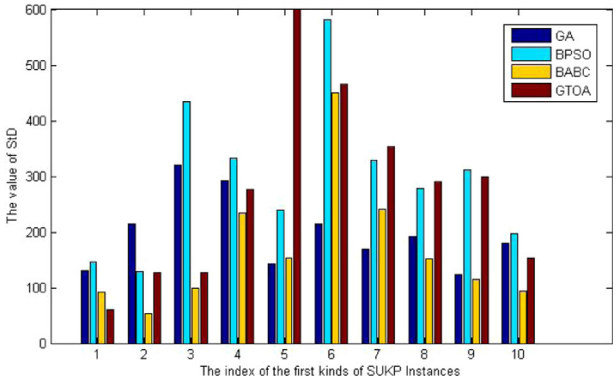

Fig. 7. *StD* histogram of the 1st category of SUKP instances.

GA gets the best result currently known on only one instance. Regarding the average running time, the solving speeds of GA and GTOA are almost equal and obviously faster than those of BPSO and BABC. The difference between the speeds of BABC and BPSO is small.

Table 3
Comparison among GTOA, GA, BPSO and BABC for the 3rd category of SUKP.

Index	Instance	*CBEST*	Algorithm	*Best*	*Mean*	*Worst*	*StD*	*Time*
1	*sukp*85_100	12045	GA	11454	11092.7	10749	171.22	0.113
	_0.10_0.75		BPSO	11710	**11482.6**	**11174**	189.17	0.190
			BABC	11664	11182.7	10897	183.57	0.188
			GTOA	**12045**	11388.3	11083	107.48	0.121
2	*sukp*85_100	12369	GA	12124	11326.3	10369	417.00	0.131
	_0.15_0.85		BPSO	**12369**	11750.7	11374	424.40	0.211
			BABC	**12369**	**12081.6**	**11710**	193.79	0.217
			GTOA	**12369**	11945.7	11251	400.84	0.126
3	*sukp*185_200	13696	GA	12841	12236.6	11843	198.18	1.231
	_0.10_0.75		BPSO	13497	12703.0	**12247**	382.50	1.572
			BABC	13047	12522.8	12011	201.35	1.502
			GTOA	**13647**	**13143.5**	12170	308.36	1.099
4	*sukp*185_200	11298	GA	10920	10351.5	9832	208.08	1.204
	_0.15_0.85		BPSO	10920	10242.5	9783	373.53	1.732
			BABC	10602	10150.6	**9900**	152.91	1.948
			GTOA	**10973**	**10566.1**	9790	292.99	1.177
5	*sukp*285_300	11568	GA	10994	10640.1	10304	126.84	3.827
	_0.10_0.75		BPSO	11538	11104.9	10419	190.23	5.612
			BABC	11158	10775.9	10584	116.80	5.450
			GTOA	**11568**	**11202.4**	**10734**	201.21	3.667
6	*sukp*285_300	11763	GA	11093	10190.3	9737	249.76	3.990
	_0.15_0.85		BPSO	**11377**	10529.8	9767	320.93	6.844
			BABC	10528	9897.9	9622	186.53	5.571
			GTOA	**11377**	**10821.9**	**10004**	319.23	3.947
7	*sukp*385_400	10326	GA	9799	9432.8	9137	163.84	9.325
	_0.10_0.75		BPSO	10252	9782.2	9089	222.64	12.985
			BABC	10085	9537.5	9202	184.62	13.012
			GTOA	**10326**	**9949.3**	**9501**	165.80	8.892
8	*sukp*385_400	10302	GA	9173	8703.7	8342	154.15	9.911
	_0.15_0.85		BPSO	**10302**	9131.5	8198	271.05	14.953
			BABC	9456	9090.0	8694	156.69	13.724
			GTOA	**10302**	**9381.3**	**8841**	286.24	9.887
9	*sukp*485_500	11037	GA	10311	9993.2	9799	117.73	18.708
	_0.10_0.75		BPSO	10923	10461.7	9929	232.03	26.590
			BABC	10823	10483.4	9965	228.34	27.227
			GTOA	**11037**	**10658.0**	**10206**	164.93	18.670
10	*sukp*485_500	9964	GA	9329	8849.5	8586	141.84	20.129
	_0.15_0.85		BPSO	9589	9180.1	8631	217.33	29.108
			BABC	9333	9085.6	8666	115.62	28.493
			GTOA	**9964**	**9356.3**	**8785**	205.87	18.896

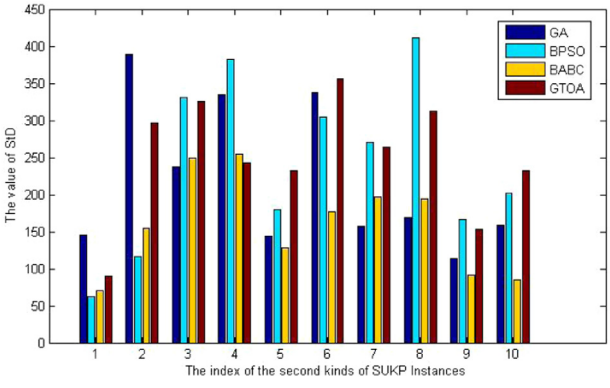

Fig. 8. *StD* histogram of the 2nd category of SUKP instances.

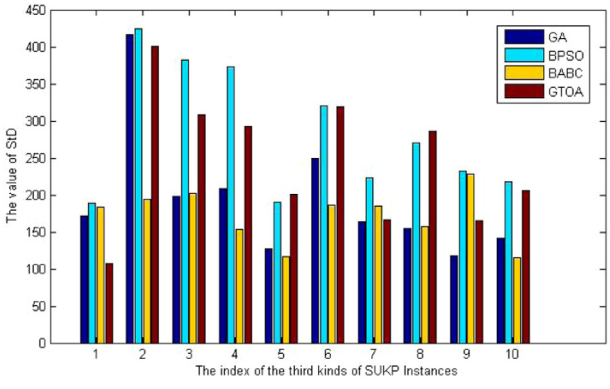

Fig. 9. *StD* histogram of the 3rd category of SUKP instances.

Since EAs are a type of stochastic approximation algorithms, in order to evaluate its performance, we also need to consider the statistical features of the average performance and stability of all algorithms. Comparisons on the average performance of all algorithms can be conducted by using the *Gap* fitting curves,

Table 4
Comparison among FirEGA, SecEGA, GPSO and GTOA for UDKP1∼UDKP10.

Index	Instance	*OPT*	Algorithm	*Best*	*Mean*	*Worst*	*StD*	*Gap*
1	UDKP1	85740	FirEGA	80593	79103.2	77935	690.01	7.740
			SecEGA	78287	76807.2	75156	798.95	10.420
			GPSO	**85740**	85669.3	85459	67.511	0.080
			GOTA	**85740**	**85684.8**	**85489**	53.8687	0.064
2	UDKP2	163744	FirEGA	155039	151662	149875	1044.95	7.380
			SecEGA	148043	145548	143833	883.434	11.110
			GPSO	**163744**	163710	163566	33.076	0.020
			GOTA	**163744**	**163727**	**163640**	23.1604	0.010
3	UDKP3	269393	FirEGA	246698	240886	237980	1491.97	10.580
			SecEGA	228823	225492	222486	1353.58	16.300
			GPSO	269340	269124	268504	129.523	0.100
			GOTA	**269393**	**269305**	**269148**	57.4767	0.033
4	UDKP4	347599	FirEGA	321605	317319	314486	1426.85	8.710
			SecEGA	305796	299978	297606	1435.46	13.700
			GPSO	347541	347267	346786	147.994	0.100
			GOTA	**347582**	**347535**	**347462**	27.6553	0.018
5	UDKP5	442644	FirEGA	405409	399620	395367	1692.23	9.720
			SecEGA	376147	370808	367574	1611.71	16.230
			GPSO	441693	440555	439151	464.361	0.479
			GOTA	**442605**	**442514**	**442332**	51.4972	0.029
6	UDKP6	536578	FirEGA	486556	478726	474015	2233.61	10.780
			SecEGA	447438	442499	438809	1765.28	17.530
			GPSO	534571	532997	531429	707.305	0.670
			GOTA	**536563**	**536177**	**535573**	215.415	0.075
7	UDKP7	635860	FirEGA	568119	560948	556938	2441.8	11.780
			SecEGA	529753	521401	518407	1813.04	18.000
			GPSO	632919	631497	629352	746.345	0.690
			GOTA	**635311**	**634648**	**633941**	323.171	0.191
8	UDKP8	650206	FirEGA	590137	585286	580684	2078.87	9.980
			SecEGA	550645	546678	543836	1449.36	15.920
			GPSO	646602	644282	641659	877.06	0.910
			GOTA	**649514**	**648719**	**647259**	416.966	0.229
9	UDKP9	718532	FirEGA	655172	649636	645012	2023.64	9.590
			SecEGA	613581	602215	605835	2003.75	16.190
			GPSO	712591	710039	707289	967.617	1.180
			GOTA	**717243**	**715941**	**714110**	533.289	0.361
10	UDKP10	779460	FirEGA	712270	706575	701545	2013.43	9.350
			SecEGA	665459	658908	655645	1723.8	15.470
			GPSO	773678	771246	768946	1027.4	1.050
			GOTA	**778576**	**777781**	**776820**	370.856	0.215

where the *Gap* metric is the relative difference between the optimal values *OPT* and the mean value *Mean*, whose formulation is given in (19). The closer the *Gap* curve is to the abscissa axis, the better the mean performance of the algorithm is. In addition, we can draw a histogram according to the value of *StD* and evaluate the stability of all algorithms by the distribution of columns.

$$Gap = \frac{|OPT - Mean|}{OPT} \times 100(\%) \tag{19}$$

Since *OPT* of SUKP instances is unknown, *OPT* is replaced by *CBEST* when calculating the *Gap* of GTOA, GA,BPSO, and BABC. The *Gap* curves for each algorithm are given in Figs. 4–6, and the histograms of *StD* are drawn in Figs. 7–9.

From Figs. 4–6, we can see that the performance of GTOA is best in the four algorithms since its average performance is much better than those of GA, BPSO, and BABC; BPSO and BABC rank second; and GA is the worst. Figs. 7–9 show that stability of all algorithms are basically equal.

4.2. Simulation results for the $D\{0-1\}$ KP instances

4.2.1. Parameters setting of the algorithms

Using the same method as Section 4.1.1, we find that $P_m = 0.008$ is a suitable value when solving D{0-1}KP by GTOA.

FirEGA and SecEGA [45] are the most classic two EAs for solving D{0-1}KP based on GA and different mathematical models. PSO-GRDKP [49] is the most efficient evolutionary algorithm for solving D{0-1}KP currently, which is proposed by using BPSO (we denote PSO-GRDKP as GPSO for convenience). In the following, we will use GTOA, FirEGA, SecEGA, and GPSO to solve four kinds of large-scale D{0-1}KP instances [45]: UDKP1∼UDKP10, WDKP1∼UDKP10, SDKP1∼UDKP10, IDKP1∼UDKP10, and evaluate the performance of GTOA by the comparative results.

FirEGA and GPSO use GR-DKP [49] to deal with infeasible solutions. GTOA and SecEGA apply D-GROA to handle infeasible solutions. The iteration number of all algorithms is set as $MIT = 10 * n$, where n is the number of item groups. In GTOA, the population size is set as $NP = 20$; $P_m = 0.008$. The population size of FirEGA and SecEGA are set as $NP = 50$; the others parameters are set to be equal to those in the literature [45].

Table 5
Comparison among FirEGA, SecEGA, GPSO and GTOA for WDKP1∼WDKP10.

Index	Instance	*OPT*	Algorithm	*Best*	*Mean*	*Worst*	*StD*	*Gap*
1	WDKP1	83098	FirEGA	82803	82693.2	82592	52.0424	0.487
			SecEGA	80014	79021.8	78096	473.674	4.905
			GPSO	**83098**	**83086.5**	**83058**	6.85054	0.014
			GOTA	**83098**	83083.9	**83058**	7.0420	0.017
2	WDKP2	138215	FirEGA	137704	137584	137356	63.228	0.457
			SecEGA	133315	132276	131337	415.619	4.297
			GPSO	**138215**	**138202**	138133	18.2569	0.009
			GOTA	**138215**	**138202**	**138157**	11.3193	0.009
3	WDKP3	256616	FirEGA	254120	253657	253307	173.011	1.153
			SecEGA	238331	235721	234025	873.581	8.143
			GPSO	**256616**	256573	256493	24.6037	0.017
			GOTA	**256616**	**256605**	**256575**	9.3001	0.004
4	WDKP4	315657	FirEGA	313966	312849	311998	484.755	0.890
			SecEGA	293640	290851	288764	950.06	7.859
			GPSO	315653	315605	315493	32.0495	0.017
			GOTA	**315657**	**315635**	**315596**	12.9266	0.007
5	WDKP5	428490	FirEGA	426311	424548	423058	798.53	0.920
			SecEGA	393617	390014	387992	1059.83	8.979
			GPSO	428484	428419	428303	34.5386	0.017
			GOTA	**428487**	**428469**	**428431**	12.6941	0.005
6	WDKP6	466050	FirEGA	463185	461672	457718	1107.57	0.939
			SecEGA	429208	425112	423269	1058.37	8.784
			GPSO	466019	465947	465828	45.2202	0.022
			GOTA	**466050**	**466036**	**466008**	8.3908	0.003
7	WDKP7	547683	FirEGA	544019	541949	538126	1224.68	1.047
			SecEGA	501557	496134	493845	1230.94	9.412
			GPSO	547565	547355	547138	87.7251	0.060
			GOTA	**547675**	**547647**	**547572**	17.000	0.007
8	WDKP8	576959	FirEGA	573427	571559	563253	1495.36	0.936
			SecEGA	530971	523203	520350	2157.09	9.317
			GPSO	576800	576597	576339	87.4878	0.063
			GOTA	**576954**	**576904**	**576820**	28.4346	0.010
9	WDKP9	650660	FirEGA	647477	644820	630086	2056.06	0.898
			SecEGA	598343	586770	583854	2315.5	9.819
			GPSO	650502	650259	649938	107.902	0.062
			GOTA	**650648**	**650596**	**650450**	33.5694	0.010
10	WDKP10	678967	FirEGA	675452	673008	668239	1441.96	0.878
			SecEGA	620230	606215	609964	3090.86	10.715
			GPSO	678862	678662	678401	91.1195	0.045
			GOTA	**678945**	**678857**	**678551**	63.5837	0.016

In GPSO [49], the population size is set as $NP = 20$; each dimensional component in the $3n$-dimensional real vector takes value in $[-5.0, 5.0]$; $W = 1.0$ and $C_1 = C_2 = 2.0$.

4.2.2. Calculation and comparison

All experimental results are based on 100 independent runs. In Tables 4–7 (in Appendix), *OPT* is the optimal value; *Best* and *Worst* represent the best and worst values among 100 calculated results respectively, and *Mean* and *StD* are the expectation and the standard deviation, respectively; *Gap* is the relative differences between *Mean* and *OPT* for each instance, which can be calculated by (19).

From the *Gap* in Tables 4–7 we can know that the *Gap* metrics for all algorithms are no more than 18.00, which illustrates that it is not only feasible but also highly efficient to solve D{0-1}KP by using GTOA, FirEGA, SecEGA, and GPSO. Furthermore, the comparison between the *Best* and *Worst* of all algorithms shows that, except the instance IDKP1, the *Worst* obtained by GTOA and GPSO compared to the other 39 instances are better than the *Best* gained by FirEGA and SecEGA, which indicates that the performances of GTOA and GPSO for solving D{0-1}KP are much better than those of FirEGA and SecEGA. Therefore, we only need to compare GTOA with GPSO in terms of the average performance and the stability to verify the efficient performance of GTOA.

The *Gap* fitting curves of GTOA and GPSO in Figs. 10–13 show that the maximum value of *Gap* for GTOA is no more than 0.4 and that for GPSO it is no more than 1.2, which illustrates that the average performances of both algorithms for solving D{0-1}KP are excellent. Moreover, for UDKP, WDKP and SDKP, it is obvious that the average performance of GTOA is much better than that of GPSO, but for IDKP, GPSO is slightly better than GTOA.

It can be known from the *StD* histograms of GTOA and GPSO in Figs. 14–17 that for UDKP, WDKP and SDKP, the algorithm stability of GTOA is better than that of GPSO, while for IDKP, GPSO is better.

The above comparative results indicate that the performances of both GTOA and GPSO are much better than those of FirEGA and SecEGA. Moreover, for UDKP, WDKP, and SDKP, GTOA has the best averaged performance and stability. It confirms that solving D{0-1}KP by GTOA is not only feasible but also highly efficient.

Table 6
Comparison among FirEGA, SecEGA, GPSO and GTOA for SDKP1∼SDKP10.

Index	Instance	*OPT*	Algorithm	*Best*	*Mean*	*Worst*	*StD*	*Gap*
1	SDKP1	94459	FirEGA	93235	93170.8	93070	42.147	1.365
			SecEGA	89769	88831.5	87463	594.911	5.958
			GPSO	94449	94260.7	93818	110.235	0.210
			GOTA	**94452**	**94425.5**	**94329**	22.8099	0.035
2	SDKP2	160805	FirEGA	159159	159004	158859	61.538	1.120
			SecEGA	153821	152059	150753	489.391	5.439
			GPSO	160777	160607	160253	95.830	0.123
			GOTA	**160805**	**160769**	**160701**	19.9885	0.022
3	SDKP3	238248	FirEGA	235454	235241	235043	79.858	1.262
			SecEGA	224997	223580	221918	543.378	6.157
			GPSO	238158	237900	237606	99.133	0.146
			GOTA	**238243**	**238204**	**238144**	21.1418	0.018
4	SDKP4	340027	FirEGA	336353	335963	335709	122.410	1.1952
			SecEGA	318510	315513	313747	851.135	7.2094
			GPSO	339830	339526	339156	131.691	0.1473
			GOTA	**340025**	**339995**	**339933**	17.6036	0.009
5	SDKP5	463033	FirEGA	452900	447587	444255	1974.990	3.3368
			SecEGA	420238	416964	413933	1291.650	9.949
			GPSO	462107	461566	460906	260.874	0.317
			GOTA	**462980**	**462868**	**462717**	48.4461	0.036
6	SDKP6	466097	FirEGA	459254	458893	458584	162.938	1.546
			SecEGA	430738	427304	425504	1031.120	8.323
			GPSO	465378	464856	464171	222.960	0.266
			GOTA	**466074**	**466010**	**465881**	33.8705	0.019
7	SDKP7	620446	FirEGA	599361	592279	579673	3949.030	4.540
			SecEGA	561224	556083	552007	1926.260	10.374
			GPSO	618753	617827	616602	342.714	0.422
			GOTA	**620219**	**619978**	**619623**	109.375	0.075
8	SDKP8	670697	FirEGA	661276	660104	659367	426.056	1.579
			SecEGA	611644	606263	603774	1446.940	9.607
			GPSO	668821	668107	667341	322.923	0.386
			GOTA	**670304**	**670071**	**669791**	124.218	0.093
9	SDKP9	739121	FirEGA	729135	727544	727064	343.670	1.566
			SecEGA	674885	667900	664580	1614.040	9.636
			GPSO	736589	735805	734871	349.522	0.449
			GOTA	**738458**	**738107**	**737667**	179.945	0.137
10	SDKP10	765317	FirEGA	756205	753394	750757	985.464	1.558
			SecEGA	708935	695557	691994	2956.08	9.115
			GPSO	762603	761980	761258	288.412	0.436
			GOTA	**764586**	**764159**	**763609**	197.927	0.151

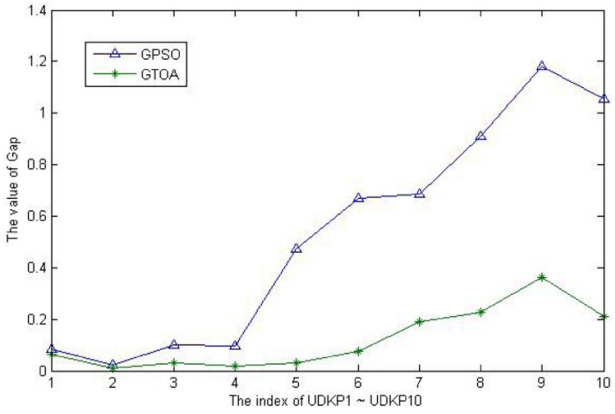

Fig. 10. *Gap* fitting curve of UDKP1∼10.

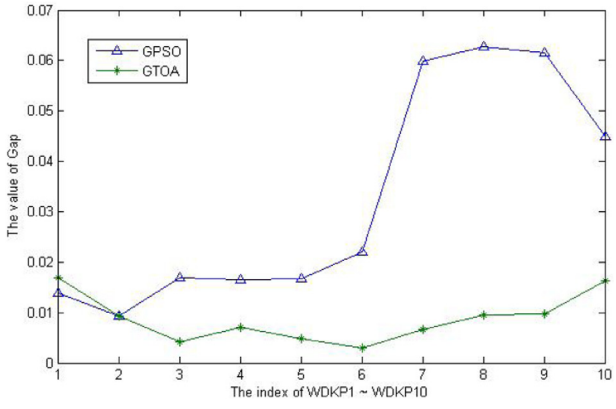

Fig. 11. *Gap* fitting curve of WDKP1∼10.

Table 7
Comparison among FirEGA, SecEGA, GPSO and GTOA for IDKP1∼IDKP10.

Index	Instance	*OPT*	Algorithm	*Best*	*Mean*	*Worst*	*StD*	*Gap*
1	IDKP1	70106	FirEGA	70106	70099.4	70090	7.234	0.009
			SecEGA	68663	67999.8	67369	328.441	3.004
			GPSO	**70106**	**70098.1**	**70077**	6.685	0.011
			GOTA	**70106**	70089.3	70048	15.1772	0.024
2	IDKP2	118268	FirEGA	118169	117869	117625	102.598	0.337
			SecEGA	114434	113385	112307	446.672	4.129
			GPSO	**118268**	**118253**	**118202**	15.687	0.013
			GOTA	**118268**	118239	118168	23.6627	0.025
3	IDKP3	234804	FirEGA	234497	233997	233666	175.423	0.344
			SecEGA	220096	217982	216313	835.828	7.164
			GPSO	**234804**	**234796**	**234759**	9.202	0.003
			GOTA	**234804**	234792	234731	12.5576	0.005
4	IDKP4	282591	FirEGA	282148	280695	278881	827.625	0.671
			SecEGA	263238	260425	258922	933.402	7.844
			GPSO	**282591**	**282578**	**282554**	9.553	0.005
			GOTA	282583	282560	282497	16.2573	0.011
5	IDKP5	335584	FirEGA	335004	333484	329621	1173.900	0.626
			SecEGA	309573	306878	304881	907.187	8.554
			GPSO	**335584**	**335580**	**335546**	7.344	0.001
			GOTA	335580	335549	335449	27.8969	0.010
6	IDKP6	452463	FirEGA	451680	449863	446704	1161.520	0.575
			SecEGA	414090	411367	408788	1099.310	9.083
			GPSO	**452463**	**452452**	**452425**	9.168	0.002
			GOTA	**452463**	452438	452372	18.5076	0.006
7	IDKP7	489149	FirEGA	488009	485592	476385	2294.280	0.727
			SecEGA	451528	444316	442133	1280.310	9.166
			GPSO	**489149**	**489133**	**489105**	8.268	0.003
			GOTA	489144	489109	489019	23.9663	0.008
8	IDKP8	533841	FirEGA	533035	529984	514196	2308.110	0.722
			SecEGA	490494	481831	478035	2215.660	9.743
			GPSO	**533839**	**533827**	**533808**	6.246	0.003
			GOTA	533834	533807	533762	14.4209	0.006
9	IDKP9	528144	FirEGA	526410	523982	511651	2216.130	0.788
			SecEGA	489661	477001	471848	3656.220	9.684
			GPSO	528140	**528131**	**528094**	9.466	0.002
			GOTA	**528144**	528103	527998	28.6709	0.008
10	IDKP10	581244	FirEGA	578903	576772	568903	1905.180	0.769
			SecEGA	535541	521604	516445	4265.070	10.261
			GPSO	**581244**	**581230**	**581194**	10.610	0.002
			GOTA	581234	581200	581086	24.9040	0.008

Table 8
Comparisons between GA and GTOA for solving instances UBKP1∼UBKP10.

Instance	*OPT*	GA				GTOA			
		Best	*Mean*	*Worst*	*StD*	*Best*	*Mean*	*Worst*	*StD*
UBKP1	201616	**201616**	201582.0	201499	26.904	**201616**	**201614.0**	**201590**	6.361
UBKP2	414114	**414114**	414107.0	414028	22.857	**414114**	**414114**	**414114**	0.0
UBKP3	594613	594610	594597.0	594574	9.181	**594613**	**594608.0**	**594587**	3.892
UBKP4	831629	**831629**	831612.0	831599	6.890	**831629**	**831628.0**	**831611**	4.107
UBKP5	1003643	1003642	1003632.7	**1003628**	4.249	**1003643**	**1003640.2**	1003626	3.820
UBKP6	1228085	**1228085**	**1228080.5**	**1228073**	3.220	**1228085**	1228079.7	**1228073**	2.965
UBKP7	1524770	**1524770**	1524764.8	1524755	5.717	**1524770**	**1524769.0**	**1524759**	3.056
UBKP8	1692853	1692846	1692838.5	1692829	4.493	**1692853**	**1692847.3**	**1692835**	4.917
UBKP9	1869142	1869138	1869133.5	1869126	3.025	**1869142**	**1869134.8**	**1869127**	3.385
UBKP10	2066060	**2066060**	2066057.2	**2066055**	1. 817	**2066060**	**2066058.2**	2066052	2.559

4.3. Simulation results for BKP instances

Among the current EAs, GA [17,46] is the only algorithm suitable for solving BKP since the feasible solutions of BKP are denoted as integer vectors in $\boldsymbol{Z}[b_1+1, b_2+1, \ldots, b_n+1]$, where $b_j(1 \leq j \leq n)$ is a positive integer. Therefore, we verify the performance of GTOA by experimentally comparing GTOA with GA for solving large-scale BKP instances.

Table 9
Comparisons between GA and GTOA for solving instances WBKP1∼WBKP10.

Instance	*OPT*	GA				GTOA			
		Best	*Mean*	*Worst*	*StD*	*Best*	*Mean*	*Worst*	*StD*
WBKP1	119312	**119312**	119305.0	119288	5.293	**119312**	**119310.0**	**119308**	1.521
WBKP2	297700	**297700**	**297700**	**297700**	0.0	**297700**	297699.8	297687	1.566
WBKP3	444156	**444156**	444150.0	444142	4.328	**444156**	**444151.0**	**444142**	3.593
WBKP4	605678	**605678**	605671.0	605664	2.917	**605678**	**605676.0**	**605671**	1.802
WBKP5	772191	**772191**	**772188.0**	**772183**	0.980	**772191**	772186.0	**72183**	1.708
WBKP6	890314	**890313**	890311.0	890309	0.774	**890313**	**890312.0**	**890310**	0.989
WBKP7	1045302	**1045302**	**1045299.0**	**1045298**	0.942	1045300	1045298.5	1045296	1.284
WBKP8	1210947	**1210946**	1210944.1	**1210944**	0.276	1210945	**1210944.6**	1210940	0.984
WBKP9	1407365	**1407365**	1407364.0	1407361	0.374	**1407365**	**1407364.1**	**1407364**	0.099
WBKP10	1574079	**1574079**	1574076.4	1574073	1.278	**1574079**	**1574077.6**	**1574073**	1.538

Table 10
Comparison between GA and GTOA for solving instances SBKP1∼SBKP10.

Instance	*OPT*	GA				GTOA			
		Best	*Mean*	*Worst*	*StD*	*Best*	*Mean*	*Worst*	*StD*
SBKP1	144822	**144822**	**144818.0**	**144807**	3.752	**144822**	144813.0	144799	6.135
SBKP2	259853	**259853**	**259852.0**	**259848**	1.050	**259853**	**259852.0**	259832	2.974
SBKP3	433414	**433414**	**433414.0**	433411	0.490	**433414**	**433414.0**	**433413**	0.099
SBKP4	493847	**493847**	**493847.0**	**493846**	0.218	**493847**	**493847.0**	493842	0.497
SBKP5	688246	**688246**	**688246**	**688246**	0.0	**688246**	**688246.0**	688239	0.829
SBKP6	849526	**849526**	**849526**	**849526**	0.0	**849526**	849518.0	849512	3.457
SBKP7	1060106	**1060106**	**1060106**	**1060106**	0.0	**1060106**	1060097.0	1060094	2.786
SBKP8	1171576	**1171576**	**1171576**	**1171576**	0.0	**1171576**	1171570.2	1171566	4.430
SBKP9	1263609	**1263609**	**1263609**	**1263609**	0.0	**1263609**	1263605.2	1263599	4.434
SBKP10	1412095	**1412095**	**1412095**	**1412095**	0.0	**1412095**	1412094.9	1412089	0.611

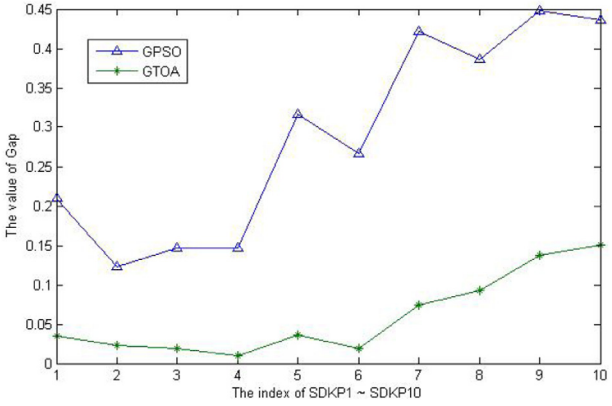

Fig. 12. *Gap* fitting curve of SDKP1∼10.

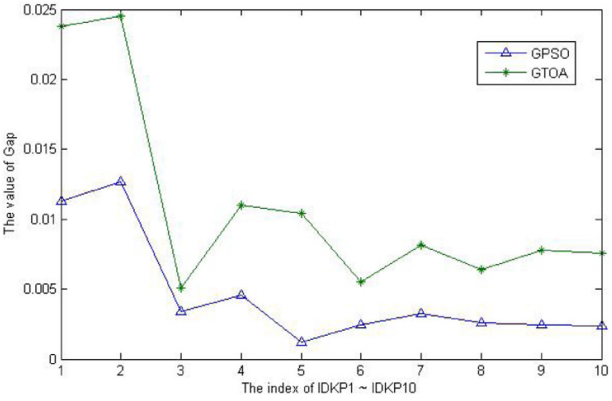

Fig. 13. *Gap* fitting curve of IDKP1∼10.

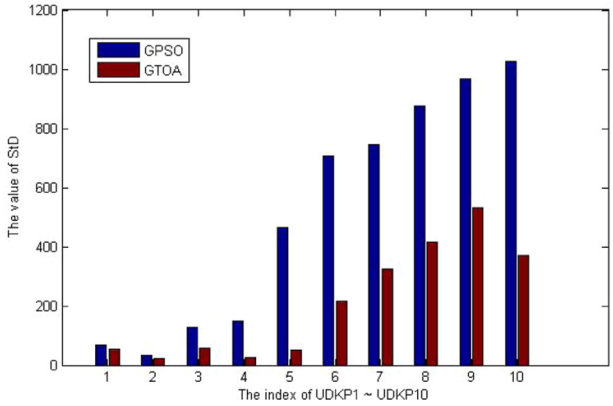

Fig. 14. *StD* histogram of UDKP1∼10.

So far there has not yet been a report of benchmark for BKP. We generate three kinds of large-scale BKP instances according to the method in [3].

(1) Uncorrelated instances of BKP(UBKP): p_j and w_j are randomly distributed in $[1, R]$, $1 \le j \le n$.

(2) Weakly correlated instances of BKP (WBKP): w_j randomly distributed in $[1, R]$, and p_j randomly distributed in $[w_j - \frac{R}{10}, w_j + \frac{R}{10}]$ such that $p_j \ge 1$.

(3) Strongly correlated instances of BKP (SBKP): w_j randomly distributed in $[1, R]$ and $p_j = w_j + 10$.

In all instances, $R = 1000$ and $C = 0.55 * \sum_{j=1}^{n} w_j b_j$, where $b_j (1 \le j \le n)$ is randomly distributed in interval [1,9]. For more details of the BKP instances, please refer to http://xxgc.hgu.edu.cn/uploads/heyichao/ThreekindsofBKPInstances.rar.

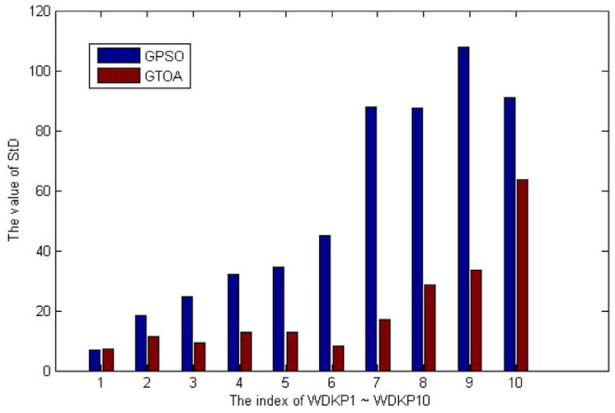

Fig. 15. *StD* histogram of WDKP1∼10.

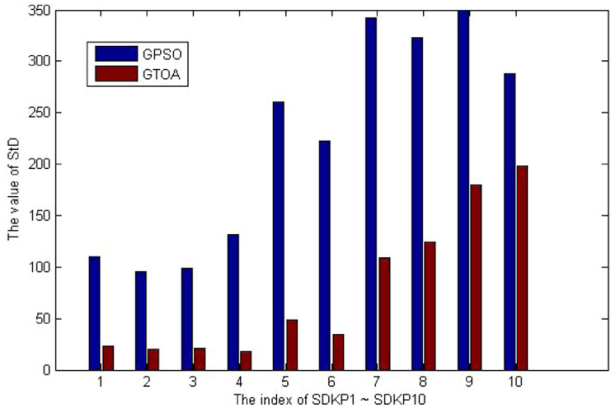

Fig. 16. *StD* histogram of SDKP1∼10.

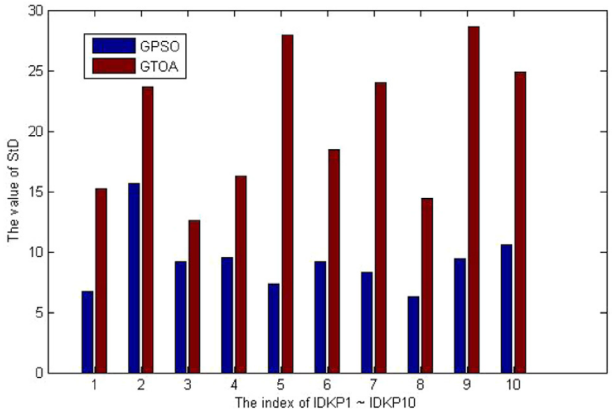

Fig. 17. *StD* histogram of IDKP1∼10.

When using GA and GTOA to solve BKP, the iteration number is set as $MIT = 2n$, n is the number of items, and we use B-GROA to eliminate infeasible solutions. In GTOA, the population size is $NP = 20$, and we use RLCO and IRMO to generate new individuals; $P_m = 0.001$. In GA, the population size is $NP = 50$; we use single-point crossover, uniform mutation operators, and fitness proportional model to generate new individuals; $P_c = 0.8$ and $P_m = 0.001$.

In Tables 8–10 (in Appendix), *OPT* is the optimal value of instance; *Best* and *Worst* represent the best and worst values, respectively, obtained by using GTOA and GA in 100 independent runs, and *Mean* and *StD* are the mean value and standard deviation, respectively.

Tables 8–10 illustrate that in terms of *Best* and *Worst*, the absolute error $|OPT - Worst|$ between *Worst* and *OPT* is no more than 117. In addition, for 4/5 of all BKP instances, the *OPT* can be attained by both algorithms, which indicates that the average performances of both GA and GTOA are excellent. Noting that the *StD* values of both GA and GTOA are no more than 27, we believe that both of GA and GTOA have very good stability.

Although both GA and GTOA perform well for solving BKP, the indexes in Tables 8–10 show that GTOA performs better when GTOA is used to solve the UBKP instances, and GA is better than GTOA for the SBKP instances. For the WBKP instances, both algorithms perform similarly. In summary, GTOA and GA have the same performance for solving BKP. They can efficiently solve BKP.

5. Conclusion

This paper presents an algebraic approach for designing evolutionary algorithms and proposes a new evolutionary algorithm GTOA by using the direct product of groups. To verify GTOA's effectiveness and wide range of applications, we apply it to solve different KPs, i.e., SUKP, D{0-1}KP, and BKP. In comparison with GA,BPSO,GPSO,BABC,FirEGA, and SecEGA for large-scale instances of KPs, we conclude that GTOA is not only easy to implement but also performs much better.

Some remarks and further investigations in this topic are listed: First, using the operations of a group to implement evolution search is a new and feasible evolution search mechanism. It is not only the innovation of algorithm design method but is also a successful paradigm for designing evolutionary algorithms by using the strict algebra theory. Second, the proposed GTOA is suitable not only for KPs but also for the others combinatorial optimization problems, for example, the satisfiability problem [27], the set cover problem [28,53], the decision making problem [54,55], and the machine scheduling problem [56]. Therefore, using GTOA to solve others combinatorial optimization problem is an important research topic. Finally, how to use GTOA to solve numerical optimization problems [57], topology optimization problem [58–60], feature selection problem [61], and semi-supervised clustering [62] is another topic which is worth researching in the future.

Declaration of competing interest

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

Acknowledgments

This study was partially supported by the Natural Science Foundation of China (61503252 and 71371063), the scientific Research Project Program of Colleges and Universities in Hebei Province (ZD2016005), and the Natural Science Foundation of Hebei Province (F2016403055).

Appendix

See Tables 1–10

References

[1] G.B. Dantzig, Discrete variable extremum problems, Oper. Res. 5 (2) (1957) 266–277.
[2] D.Z. Du, K.I. Ko, X.D. Hu, Design and Analysis of Approximation Algorithms, Springer Science Business Media LLC, Berlin, 2012.
[3] Hans Kellerer, Ulrich Pferschy, David Pisinger, Knapsack Problems, Springer, Berlin, 2004.
[4] Silvano Martello, Paolo Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons, Inc., New York, 1990.
[5] B. Guldan, Heuristic and Exact Algorithms for Discounted Knapsack Problems (Master thesis), University of Erlangen-Nrnberg, Germany, 2007.
[6] Aiying Rong, Jos Rui Figueira, Kathrin Klamroth, Dynamic programming based algorithms for the discounted {0-1} knapsack problem, Appl. Math. Comput. 218 (12) (2012) 6921–6933.
[7] O. Goldschmidt, D. Nehme, G. Yu, Note: On the set-union knapsack problem, Nav. Res. Logist. 41 (6) (1994) 833–842.
[8] Sara, Tugba, A. Sipahioglu, A genetic algorithm for the quadratic multiple knapsack problem, in: International Conference on Advances in Brain, Vision and Artificial Intelligence, Vol. 4729, Springer-Verlag, 2007, pp. 490–498.
[9] Yichao He, Xinlu Zhang, Wenbin Li, et al., Algorithms for randomized time-varying knapsack problems, J. Comb. Optim. 31 (1) (2016) 95–117.
[10] Zhi-gang Ren, Zu-ren Feng, Ai-min Zhang, Fusing ant colony optimization with Lagrangian relaxation for the multiple-choice multidimensional knapsack problem, Inform. Sci. 182 (1) (2012) 15–29.
[11] S. Martello, D. Pisinger, P. Toth, Dynamic programming and strong bounds for the 0-1 knapsack problem, Manage. Sci. 45 (3) (1999) 414–424.
[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., the MIT Press, Cambridge, 2001.
[13] Majid Darehmiraki, Hasan Mishmast Nehi, Molecular solution to the 0-1 knapsack problem based on DNA computing, Appl. Math. Comput. 187 (2) (2007) 1033–1037.
[14] Rajeev Motwani, Prabhakar Raghavan, Randomized Algorithms, Cambridge University Press, Cambridge, 1995.
[15] Z. Michalewicz, M. Schoenauer, Evolutionary algorithms for constrained parameter optimization problems, Evol. Comput. 4 (1) (1996) 1–32.
[16] D. Ashlock, Evolutionary Computation for Modeling and Optimization, Springer-Verlag, London, United Kingdom, 2006.
[17] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Longman Publishing Co., Inc., Boston, 1989.
[18] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ, 1995, pp. 1942–1948, IV.
[19] R. Storn, K. Price, Differential evolution- A simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (4) (1997) 341–359.
[20] Z.W. Geem, J.H. Kim, G.V. Loganathan, A new heuristic optimization algorithm: Harmony search, Simulation 76 (2) (2001) 60–68.
[21] Xiao-Lei Li, A New Intelligent Optimization Method – Artificial Fish Swarm Algorithm (Ph.D. thesis), Zhejiang University, 2003.
[22] M. Dorigo, T. Sttzle, Ant Colony Optimization, MIT Press, Cambridge, MA, 2004.
[23] Dervis Karaboga, Barbaros Basturk, A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm, J. Global Optim. 39 (3) (2007) 459–471.
[24] Xiaodong Li, Niching without niching parameters: Particle swarm optimization using a ring topology, IEEE Trans. Evol. Comput. 14 (1) (2010) 150–169.
[25] Zhi-Hui Zhan, Jun Zhang, Yun Li, Yu-Hui Shi, Orthogonal learning particle swarm optimization, IEEE Trans. Evol. Comput. 15 (6) (2011) 832–847.
[26] P.C. Chu, J.E. Beasley, A genetic algorithm for the multidimensional knapsack problem, J. Heuristics 4 (1) (1998) 63–86.
[27] Jens Gottlieb, Elena Marchiori, Claudio Rossi, Evolutionary algorithms for the satisifiability problem, Evol. Comput. 10 (1) (2002) 35–50.
[28] J.E. Beasley, P.C. Chu, A genetic algorithm for the set covering problem, European J. Oper. Res. 94 (2) (1996) 392–404.
[29] X.-S. Yang, Firefly algorithms for multimodal optimization, in stochastic algorithms: Foundations and applications, SAGA 2009, in: Lecture Notes in Computer Sciences, Vol. 5792, 2009, pp. 169–178.
[30] S. Mirjalili, SCA: A Sine cosine algorithm for solving optimization problems, Knowl.-Based Syst. 96 (15) (2016) 120–133.
[31] G.G. Tejani, V.J. Savsani, V.K. Patel, S. Mirjalili, Truss optimization with natural frequency bounds using improved symbiotic organisms search, Knowl.-Based Syst. 143 (12) (2018) 162–178.
[32] Seyedali Mirjalili, Seyed Mohammad Mirjalili, Andrew Lewis, Grey wolf optimizer, Adv. Eng. Softw. 69 (2014) 46–61.
[33] V.J. Savsani, G.G. Tejani, V.K. Patel, Truss topology optimization with static and dynamic constraints using modified subpopulation teaching-learning-based optimization, Eng. Optim. 48 (11) (2016) 1990–2006.
[34] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 1 (1) (1997) 67–82.
[35] A.P. Engelbrecht, G. Pampara, Binary differential evolution strategies, in: IEEE Congress on Evolutionary Computation, CEC 2007, 25-28 2007, Singapore, 2007, pp. 1942–1947.
[36] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm optimization, in: Proceedings of 1997 Conference on System, Man, and Cybernetices, 1997, pp. 4104–4109.
[37] Mina Husseinzadeh Kashan, Nasim Nahavandi, Ali Husseinzadeh Kashan, DisABC: A new artificial bee colony algorithm for binary optimization, Appl. Soft Comput. 12 (1) (2012) 342–352.
[38] Mustafa Servet Kiran, The continuous artificial bee colony algorithm for binary optimization, Appl. Soft Comput. 33 (2015) 15–23.
[39] Yichao He, Xizhao Wang, Yingzhan Kou, A binary differential evolution algorithm with hybrid encoding, J. Comput. Res. Dev. 44 (9) (2007) 1476–1484.
[40] M. Baioletti, A. Milani, V. Santucci, Algebraic Particle Swarm Optimization for the permutations search space, in: Evolutionary Computation, IEEE, 2017, pp. 1587–1594.
[41] V. Santucci, M. Baioletti, A. Milani, A differential evolution algorithm for the permutation flowshop scheduling problem with total flow time criterion, IEEE Trans. Evol. Comput. 20 (5) (2016) 682–694.
[42] Derek J.S. Robinson, A Course in the Theory of Groups, second ed., Springer-Verlag, NewYork, 2003.
[43] Joseph J. Rotman, A First Course in Abstract Algebra, third ed., Prentice Hall, New Jersey, 2008.
[44] Yichao He, Haoran Xie, Tak-Lam Wong, Xizhao Wang, A novel binary artificial bee colony algorithm for the set-union knapsack problem, Future Gener. Comput. Syst. 87 (1) (2018) 77–86.
[45] Yi-Chao He, Xi-Zhao Wang, Win-Bin Li, Xin-Lu Zhang, Yi-Ying Chen, Research on genetic algorithms for the discounted {0-1} knapsack problem, Chinese J. Comput. 38 (12) (2016) 2614–2630.
[46] Guo-liang Chen, Xi-fa Wang, et al., Genetic Algorithms and Its Applications, Science Press, Beijing, China, 2003.
[47] Carlos A. Coello Coello, Theoretial and numerical constraint-handling techniques used with evolutionary algorithm-a survey of the state of art, Comput. Methods Appl. Mech. Engrg. 191 (11–12) (2002) 1245–1287.
[48] T.P. Runarsson, X. Yao, Stochastic ranking for constrained evolutionary optimization, IEEE Trans. Evol. Comput. 4 (3) (2000) 284–294.
[49] Yi-Chao He, Xi-Zhao Wang, Yu-Lin He, Shu-Liang Zhao, Wen-Bin Li, Exact and approximate algorithms for discounted {0-1} knapsack problem, Inform. Sci. 369 (2016) 634–647.
[50] Ashwin Arulselvan, A note on the set union knapsack problem, Discrete Appl. Math. 169 (41) (2014) 214–218.
[51] Samir Khuller, Anna Moss, Joseph (Seffi) Naor, The budgeted maximum coverage problem, Inform. Process. Lett. 70 (1) (1999) 39–45.
[52] Yichao He, Xizhao Wang, et al., Exact algorithms and evolutionary algorithms for randomized time-varying knapsack problem, J. Softw. 28 (2) (2017) 185–202.
[53] Qingyuan Xu, Anhui Tan, Yaojin Lin, A rough set method for the unicost set covering problem, Int. J. Mach. Learn. Cybern. 8 (3) (2017) 781–792.
[54] I. Deli, Y. Subas, A ranking method of single valued neutrosophic numbers and its applications to multi-attribute decision making problems, Int. J. Mach. Learn. Cybern. 8 (4) (2017) 1309–1322.
[55] Jun Ye, Simplified neutrosophic harmonic averaging projection-based method for multiple attribute decision-making problems, Int. J. Mach. Learn. Cybern. 8 (3) (2017) 981–987.
[56] Yufu Ning, Xiumei Chen, Zhiyong Wang, Xiangying Li, An uncertain multiobjective programming model for machine scheduling problem, Int. J. Mach. Learn. Cybern. 8 (5) (2017) 1493–1500.
[57] Yaosheng Liang, Zhongping Wan, Debin Fang, An improved artificial bee colony algorithm for solving constrained optimization problems, Int. J. Mach. Learn. Cybern. 8 (3) (2017) 739–754.
[58] V.J. Savsani, G.G. Tejani, V.K. Patel, P. Savsani, Modified meta-heuristics using random mutation for truss topology optimization with static and dynamic constraints, J. Comput. Des. Eng. 4 (2) (2017) 106–130.
[59] G.G. Tejani, V.J. Savsani, S. Bureerat, V.K. Patel, Topology and size optimization of trusses with static and dynamic bounds by modified symbiotic organisms search, J. Comput. Civ. Eng. 32 (2) (2017) 1–11.
[60] G.G. Tejani, V.J. Savsani, V.K. Patel, Adaptive symbiotic organisms search (SOS) algorithm for structural design optimization, J. Comput. Des. Eng. 3 (3) (2016) 226–249.
[61] G. Aldehim, Wenjia Wang, Determining appropriate approaches for using data in feature selection, Int. J. Mach. Learn. Cybern. 8 (3) (2017) 915–928.
[62] A.K. Alok, S. Saha, A. Ekbal, Semi-supervised clustering for gene-expression data in multiobjective optimization framework, Int. J. Mach. Learn. Cybern. 8 (2) (2017) 421–439.