A Binary Differential Search Algorithm for 0-1 Multidimensional Knapsack Problem

Jianjun Liu[a], Changzhi Wu[b,*], Xiangyu Wang[b,c], Guoning Wu[a], Kok Lay Teo[d]

[a]*College of Science, China University of Petroleum, Beijing 102249, China*
[b]*Australasian Joint Research Centre for Building Information Modelling, School of Built Environment, Curtin University, Perth, WA 6845, Australia*
[c]*Department of Housing and Interior Design, Kyung Hee University, Seoul, Korea*
[d]*Department of Mathematics and Statistics, Curtin University, Perth, WA 6845, Australia*

Abstract

The Multidimensional Knapsack Problem (MKP) is known to be NP-hard in Operations Research. It has a wide range of applications in engineering and management. In this paper, we propose a binary differential search method to solve 0-1 MKPs in which the stochastic search is guided by a Brownian motion-like random walk. Our proposed method is composed of two main operations: discrete solution generating and feasible solution making. Discrete solution generating is realized through integrating a Brownian motion-like random search with an integer rounding operation. However, the rounded discrete variables may violate the constraints. To maintain the feasibility of the rounded discrete variables, a feasible solution making strategy is executed. To demonstrate the efficiency of our proposed algorithm, various 0-1 MKPs are solved by our proposed algorithm as well as by some of the existing meta-heuristic methods. The obtained numerical results indicate that our algorithm outperforms those existing meta-heuristic methods. Furthermore, it shows that our algorithm has the capability to solve large-scale 0-1 MKPs.

Keywords: Differential Search, MKP, Binary Optimization

1. Introduction

The multidimensional knapsack problem is a generalization of a well-known Knapsack Problem (KP) which has only one constraint. Even for the case with only one constraint, MKP is NP-hard [1]. It has wide applications in areas, such as capital budgeting problems [2], loading problems [3] and resource allocation [4]. A comprehensive review on practical applications and theoretical results of the MKPs can be found in [5].

0-1 MKP (for simplification, it is called MKP in following description) is to select a subset of given objects (or items) in such a way that the total profit of the selected objects is maximized while a set of knapsack constraints are satisfied. Let D be the number of objects, m be the number of knapsack constraints with capacities $C_j, (j = 1, 2, \cdots, m)$, p_i be the profit of the object i in the knapsack, x_i be a binary variable that $x_i = 1$, if the object i has been stored in the knapsack and

[☆]The authors gratefully acknowledge the financial support from Australian Research Council, the National Natural Science Foundation of China (Grant No. 61473326, 11371371) and the Natural Science Foundation of Chongqing (cstc2013jjB00001 and cstc2013jcyjA00029).

[*]Corresponding author
Email address: c.wu@curtin.edu.au (Changzhi Wu)

$x_i = 0$, if it remains out of the knapsack, and w_{ij} be the entry of the knapsack's constraints. Now the MKP can be formally stated as follows:

$$\max \quad f(x) = \sum_{i=1}^{D} p_i x_i \tag{1}$$

$$\text{subject to} \quad g_j(x) = \sum_{i=1}^{D} w_{ij} x_i \le C_j, \quad \forall j = 1, 2, \cdots, m \tag{2}$$

$$x_i \in \{0, 1\}, i = 1, 2, \cdots, D \tag{3}$$

Without loss of generality, we assume that p_i, w_{ij} and C_j are non-negative integers with $w_{ij} \le C_j$ and $\sum_{i=1}^{D} w_{ij} \ge C_j$.

The 0-1 multidimensional knapsack problem is widely studied in the literature, for example, [6, 7]. Some surveys on approaches to solving knapsack problems can be found in [8].

In optimization, there are two differential kinds of algorithms, deterministic algorithms [9] and heuristic algorithms [10]. Many deterministic algorithms have been developed to solve MKPs, see, for example, [11, 12]. A branch and bound algorithm is applied to solve MKP in [13]. An exact algorithm based on modifying the multi-criteria branch and bound algorithm is presented in [14] to solve multiobjective and multiconstraint (or multidimensional) knapsack problems (MOMCKPs). Meanwhile, dynamic programming is also widely applied to solve MKPs. A preprocessing procedure for the 0-1 multidimensional knapsack problem is presented in [15]. An approach based on dynamic programming is proposed to solve the 0-1 multi-objective knapsack problem in [16]. In [17], a method is proposed to solve exactly the knapsack sharing problem (KSP) based on dynamic programming through decomposition of the original KSP problem into a set of knapsack problems. In [7], an exact method based on a multi-level search strategy for solving the large-scale 0-1 Multidimensional Knapsack Problem. This search strategy is based on the reduced costs of the non-basic variables of the Liner Programming relaxation solution. Another algorithm which combines Linear Programming with an efficient tabu search is proposed in [13] and it is embedded into a variables fixing heuristic in the process of solving MKPs. However, all the methods mentioned above still have difficulties to solving large-scale knapsack problems.

In recent decades, many bio-inspired methods, such as Genetic Algorithms [18], Improved binary Artificial Fish Swarm algorithm [19], Particle Swarm Optimization (PSO) [20], Firefly Algorithm [21], Artificial Bee Colony (ABC) algorithm and Ant Colony Optimization [22, 23], have been proposed to solve complex problems with high complexity. In [24], a binary fruit fly optimization algorithm (bFOA) is proposed to solve the multidimensional knapsack problem (MKP). In the bFOA, binary string is used to represent the solution of the MKP, and three main search processes are designed to perform evolutionary search, including smell-based search process, local vision-based search process and global vision-based search process. In addition, a group generating probability vector is designed to produce new solutions. In [25], a hybrid artificially glowworm swarm optimization algorithm is presented to solve multidimensional 0-1 knapsack problem, which utilizes two important strategies: (i) how to select the item based on its unit volume value; and (ii) the binary glowworm swarm optimization algorithm.

Compared with the methods based on dynamic programming, bio-inspired methods appear to be more effective for solving 0-1 knapsack problems [26]. For this, new bio-inspired methods have been developed to solve knapsack problems. In [27], a modified discrete shuffled frog leaping algorithm (MDSFL), which combines the local search of the "particle swarm optimization" technique with the competitiveness mixing of information of the "shuffled complex evolution" technique, is applied to solve MKPs. In order to achieve better performance, fuzzy possibility and necessity approaches are used to obtain optimal solution with ant colony algorithm [23] and a quantum-inspired artificial immune system (MOQAIS) with two diversity schemes, suppression algorithm and truncation algorithm, are employed to improve the diversity of the population in [28].

The Differential Search (DS) [29, 30] algorithm is a new population-based meta-heuristic optimization algorithm that has been used with success to solve some continuous numerical optimization problems. In comparison with some other algorithms, DS algorithm shows better performance both in the global convergence and the convergence speed. In this paper, we propose a binary DS (BDS) to solve the knapsack problem.

The remaining of the paper is organized as follows. Section 2 briefly introduces DS. Section 3 details the binary Differential Search algorithm and how it solves MKP. Section 4 presents numerical experiment. Section 5 ends the paper with some concluding remarks.

2. Review of Differential Search Algorithm

Differential Search (DS) algorithm is originally introduced to solve the problem of transforming the geocentric Cartesian coordinates into geodetic coordinates in 2012. Comparison studies are carried out in [29] for continuous unconstrained optimization problems between the DS algorithm and 8 widely used algorithms, including PSO, ABC, Differential Evolutionary (DE) algorithm, and Gravitational Search Algorithm (GSA). The results show that DS algorithm in [29] is more powerful. DS algorithm simulates the Brownian motion-like random-walk movement carried out by an organism to migrate. In the migration movement, the migrating species of living beings constitute a superorganism, which contains a large number of individuals. Then, the superorganism starts to change its position by moving towards more fruitful areas. The movement of a superorganism is simulated by a *Brownian motion-like random walk* model. Therefore, the simulation of the Brownian motion is taken as a search strategy in DS.

2.1. Generation of the initial solution

In the process of solving optimization problem by DS, it is assumed that a population is made up of random outcomes of these artificial-superorganism migrations. An artificial-superorganism will migrate to the global minimum of the problem. During this migration, the artificial-superorganism examines whether some randomly selected positions are suitable temporary positions during the migration. If such a position is suitable to stopover temporarily during the migration, the members of the artificial-superorganism (i.e. artificial-organisms) that made such discovery immediately settle at the discovered position and continue their migration from this position.

Artificial-organisms (i.e., $X_i, i = 1, 2, \cdots, N$) making up an artificial-superorganism (i.e.,$S_g, g = 1, 2, \cdots, G$) contain elements (i.e., $x_{ij}, j = 1, 2, \cdots, D$) whose size is equal to the problem dimension. Here, N, G and D denote, respectively, the number of members in the superorganism, the number of maximum generations (or iterations) and the dimension of each problem. Each element of an artificial-organism in the initial position is defined as:

$$x_{ij} = l_j + r \cdot (u_j - l_j) \tag{4}$$

where r is a uniformly distributed random number. Set $L = (l_1, l_2, \cdots, l_D)$ and $U = (u_1, u_2, \cdots, u_D)$ as the lower and upper bounds for all the artificial-superorganism, where l_j and u_j are the lower and upper bounds of the j-th dimension of the ith artificial-organism X_i, respectively. In such a case, an artificial-organism can be defined as $X_i = [x_{ij}]$.

2.2. Searching strategy of DS

The mechanism of DS for finding a stopover site in the remaining areas between the artificial-organisms can be described by a *Brownian motion-like random walk* model (see Algorithm 1). These individuals which are randomly selected in the artificial-organisms move towards the targets of $donor = X_{\text{Random\_Shuffling}(i)}$ so as to discover stopover sites that are defined by

$$S = X_i + \gamma \cdot (donor - X_i) \tag{5}$$

where S is a stopover site position. The function of Random_Shuffling is to randomly change the order of the elements in the set of $i = \{1, 2, \cdots, N\}$. γ is a scale factor to control the scale of position change among their members. Its value is generated from the Gamma distribution, with parameters a and b, denoted by

$$\gamma = 1/\text{GamRnd}(a, b) \tag{6}$$

The members (i.e., *individuals*) X_i of the artificial-organisms among the superorganisms S_k to participate in the search process at stopover site are determined by a random process. This random process is given by Algorithm 1 in which the $rand_j$, $j = 1, \cdots, 5$ are the uniformly distributed random numbers and $Counter_k$, $k = 1, \cdots, 4$ denote four counters. The control parameters p_1 and p_2 are given as $p_1 = 0.3$ and $p_2 = 0.3$.

If one of the elements x_{ij} of X_i is moving beyond the limits of the habitat (i.e., the bound of the search space), it is randomly deferred to another position in the habitat. If the *individuals* of the artificial-organism discover a stopover site, which is more fertile than the sources owned by the artificial-organism, the artificial-organism moves to that stopover site. So a selection strategy in DS can be modelled by the following formula

$$S^* = \begin{cases} S, & \text{if } y(S) < y(S^*); \\ S^*, & \text{otherwise.} \end{cases} \tag{7}$$

where $y(S)$ and $y(S^*)$ are the evaluation of the stopover site and the current best optimum, respectively. This strategy is referred to as *greedy rule*. If the artificial-organisms change sites, the superorganism containing the artificial-organisms continues its migration towards a global optimum.

2.3. Balance mechanism of exploration and exploitation in DS

Exploration (or diversification) and exploitation (or intensification) are the two main components of any meta-heuristic algorithms [31]. Exploration is to generate diverse solutions so as to explore the search space on the global scale, while exploitation is to focus on the search in a local region by exploiting the information that a current good solution is found in this region. When designing a global optimization algorithm, we generally combine them together to select a better solution.

The exploration, in DS algorithm, is realized through *Brownian motion-like random walks* (see Algorithm 1) and the exploitation is realized through a greedy selection method as shown by Eq.(7). Despite the importance of a fine balance between the right amount of exploration and the right degree of exploitation, there is no practical guideline for achieving this balance [32]. However, the numerical experimental results in Section 4 show that the DS holds a good balance between exploration and exploitation by keeping a simple searching strategy in the solution space.

Algorithm 1 Matlab pseudo code: a Brownian motion-like random walk process searching for a stopover site

```
% $rand_1, \cdots rand_5$ are five random numbers in $(0,1)$
if $rand_1 < rand_2$ then
    if $rand_3 < p_1$ then
        %Generate a random matrix $R_{n \times D}$
        $R = rand(N, D)$
        for $Counter_1 = 1 : N$ do
            % Set each element in matrix $R_{n \times D}$ to be 0 or 1 according to comparison with $rand_4$
            $R(Counter_1, :) = R(Counter_1, :) < rand_4$
        end for
    else
        %Generate a random matrix $R_{n \times D}$ whose elements are 1
        $R = ones(N, D)$
        for $Counter_2 = 1 : N$ do
            % Set each element in matrix $R_{n \times D}$ to be 0 or 1 according to comparison with $rand_5$
            $R(Counter_2, randi(D)) = R(Counter_2, randi(D)) < rand_5$
        end for
    end if
else
    $R = ones(N, D)$
    for $Counter_3 = 1 : N$ do
        % Generate a vector $d_{1 \times \lceil p_2 \cdot D \rceil}$ whose elements are random integers in $(0, D)$
        $d = randi(D, 1, \lceil p_2 \cdot D \rceil)$
        for $Counter_4 = 1 : \text{size}(d)$ do
            $R(Counter_3, d(Counter_4)) = 0$
        end for
    end for
end if
% Set the value of $individuals_{I,J}$ to be the position index of $R_{I,J}$ whose value is greater than 0
$individuals_{I,J} \leftarrow R_{I,J} > 0 | I \in i, J \in [1, D]$
$S(individuals_{I,J}) = Superorganism(individuals_{I,J})$
```

3. Binary differential search to MKP

The original differential search algorithm mentioned above is designed to solve continuous optimization problems with bound constraints. In this section, we will extend it to solve 0-1 MKP.

3.1. Constraint transformation

By virtue of the penalty method proposed in [33, 34], the mathematical model of MKP formulated by Eqs. (1)-(3) is converted to the following unconstrained optimization problem:

$$F(x) = \max \left\{ \sum_{i=1}^{D} p_i x_i + \lambda \sum_{j=1}^{m} \left[\min \left\{ C_j - \sum_{i=1}^{D} w_{ji} x_i, 0 \right\} \right] \right\} \tag{8}$$

with

$$\lambda = \frac{1 + \sum_{i=1}^{D} p_i}{\max_{i,j} \{C_j - w_{ji}\}}$$

Note that the constant λ is such that if $x = (x_1, x_2, \cdots, x_D)$ satisfies all the inequality constraints (2), then

$$\sum_{j=1}^{m} \left[\min \left\{ C_j - \sum_{i=1}^{D} w_{ji} x_i, 0 \right\} \right] = 0$$

and

$$\sum_{i=1}^{D} p_i x_i \geq 0$$

Alternatively, if x does not satisfy all of the constraints in MKP problem, then

$$\sum_{j=1}^{m} \left[\min \left\{ C_j - \sum_{i=1}^{D} w_{ji} x_i, 0 \right\} \right] \leq - \left(1 + \sum_{i=1}^{D} p_i x_i \right)$$

and $F(x) \leq -1$. Through appending the constraints into the objective function as described in (8), the constrained MKP is transferred to an unconstrained discrete maximization with bound constraints.

3.2. Generate initial solutions

To establish a binary Differential Search algorithm to solve MKP, the basic component is to represent the solution of MKP and to generate the initial and new solutions. The initial solution is a random binary vector of size N which is the number of artificial organisms. Thus, the initial organism swarm consists of a random matrix as given below:

$$X = [X_1, X_2, \cdots, X_N]^T = \begin{pmatrix} 0 & 1 & 1 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 1 \\ & \cdots & & \cdots & \\ 1 & 1 & 0 & \cdots & 1 \end{pmatrix}_{N \times D} \tag{9}$$

3.3. Determine the stopover site

The *donor* in Eq. (5) can be constructed in two ways for BDS. One is that a random permutation of 0 and 1 is taken as a *donor* in

$$donor = \mathrm{randn}(1, D) \tag{10}$$

where $\mathrm{randn}(1, D)$ is a function to generate a binary vector, and the other is that an elastic superorganism (that means the superorganism with best fitness) is taken as a *donor* in

$$donor = \arg\max \{f(X_i)\} = \arg\max \{f(x_{i1}, \cdots, x_{iD})\}, i = 1, 2, \cdots, N. \tag{11}$$

where $f(x)$ is a transfer function. The *donor* given by Eq. (11) is called elitist.

By virtue of the transfer function, a real number can be transferred into a binary number 0 or 1 according to

$$x_{ij} = \begin{cases} 0, & \text{if } f(x_{ij}) < r_j, j = 1, 2, \cdots, D; \\ 1, & \text{otherwise.} \end{cases} \tag{12}$$

120 where r_j is random number generated between 0 and 1 to decide the variable as 0 or 1. If $f(x_{ij})$ is greater than r_j, then the variable of x_{ij} is 1, otherwise it is 0.

Two functions are often applied as transfer functions for the transfer of a real number into binary numbers 0 or 1. They are Tanh and Sigmoid [35, 36]. The Tanh function is expressed as

$$f(x) = \mathrm{Tanh}(x) = \frac{e^{\tau|x|} - 1}{e^{\tau|x|} + 1} \tag{13}$$

and the Sigmoid function is

$$f(x) = \mathrm{Sig}(x) = \frac{1}{1 + e^{-\tau x}} \tag{14}$$

where $f(x)$ denotes the probability of bit x equalling 1 (see Figure 1).

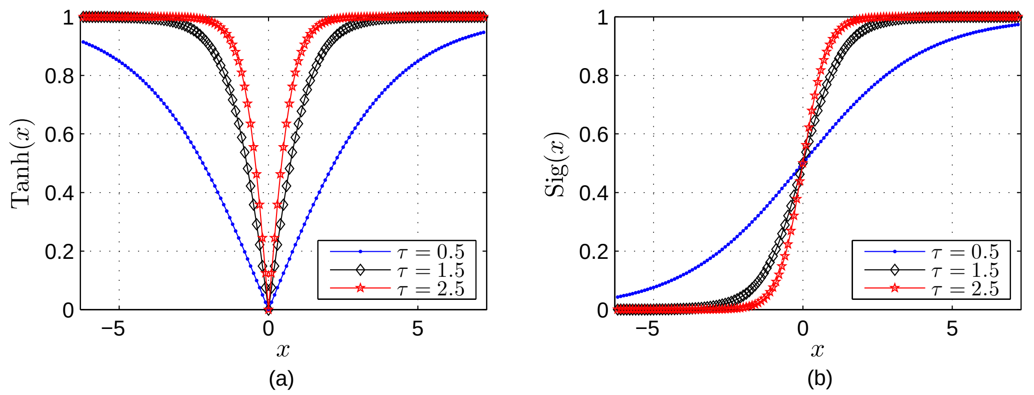
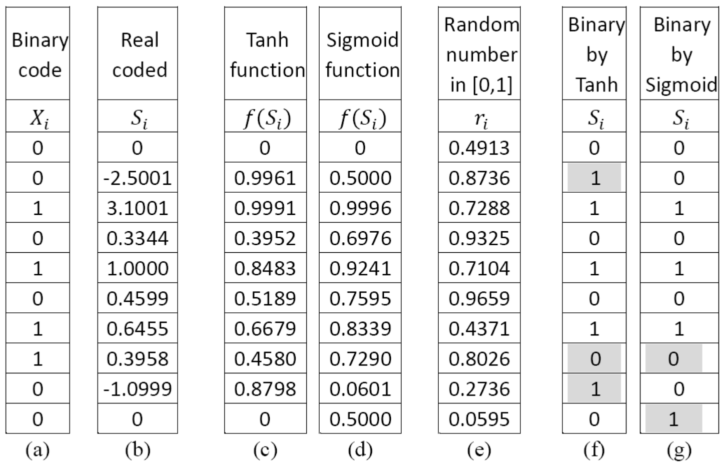


Figure 1: (a) Tanh function $\mathrm{Tanh}(x)$ and (b) Sigmoid function $\mathrm{Sig}(x)$ with $\tau = 0.5, 1.5$ and 2.5, respectively.

The procedure of a new binary solution generated by using of Eq. (12) for a stopover site position, $S_i = (x_{i1}, x_{i2}, \cdots, x_{i10})$ and the binary codes of S_i in BDS algorithm with respect to 125 Tanh and Sigmoid functions are also as shown in Figure 2. The randomly generated bits of X_i are listed in Figure 2(a). A stopover position S_i yielded by Eq. (5) is as showed in Figure 2(b). The $f(S_i)$ using Tanh and Sigmoid functions are given in Figure 2(c) and 2(d). Figure 2(e) is the



Binary code	Real coded	Tanh function	Sigmoid function	Random number in [0,1]	Binary by Tanh	Binary by Sigmoid
X_i	S_i	$f(S_i)$	$f(S_i)$	r_i	S_i	S_i
0	0	0	0	0.4913	0	0
0	-2.5001	0.9961	0.5000	0.8736	1	0
1	3.1001	0.9991	0.9996	0.7288	1	1
0	0.3344	0.3952	0.6976	0.9325	0	0
1	1.0000	0.8483	0.9241	0.7104	1	1
0	0.4599	0.5189	0.7595	0.9659	0	0
1	0.6455	0.6679	0.8339	0.4371	1	1
1	0.3958	0.4580	0.7290	0.8026	0	0
0	-1.0999	0.8798	0.0601	0.2736	1	0
0	0	0	0.5000	0.0595	0	1
(a)	(b)	(c)	(d)	(e)	(f)	(g)

Figure 2: Demonstration of Binary-coded DS algorithm using Tanh and Sigmoid functions with $\tau = 2.5$ and $\alpha = 0.5$.

random number generated between 0 and 1 in order to decide the generating elements of S_i as 0 or 1. The alteration of bits by Eq. (12) with transfer functions Tanh and Sigmoid are shown as light shading in Figure 2(f) and 2(g), respectively. From (f) and (g) in Figure 2, it can be observed that there are more alterations of bits by using Tanh than Sigmoid. It is also noticed that more alterations are involved by using Eq. (13) than by using Eq. (14). We will compare their performances on solving MKPs through extensive numerical experiments in Subsection 4.1.

3.4. Generate new feasible solutions by Brownian motion-like random walk process

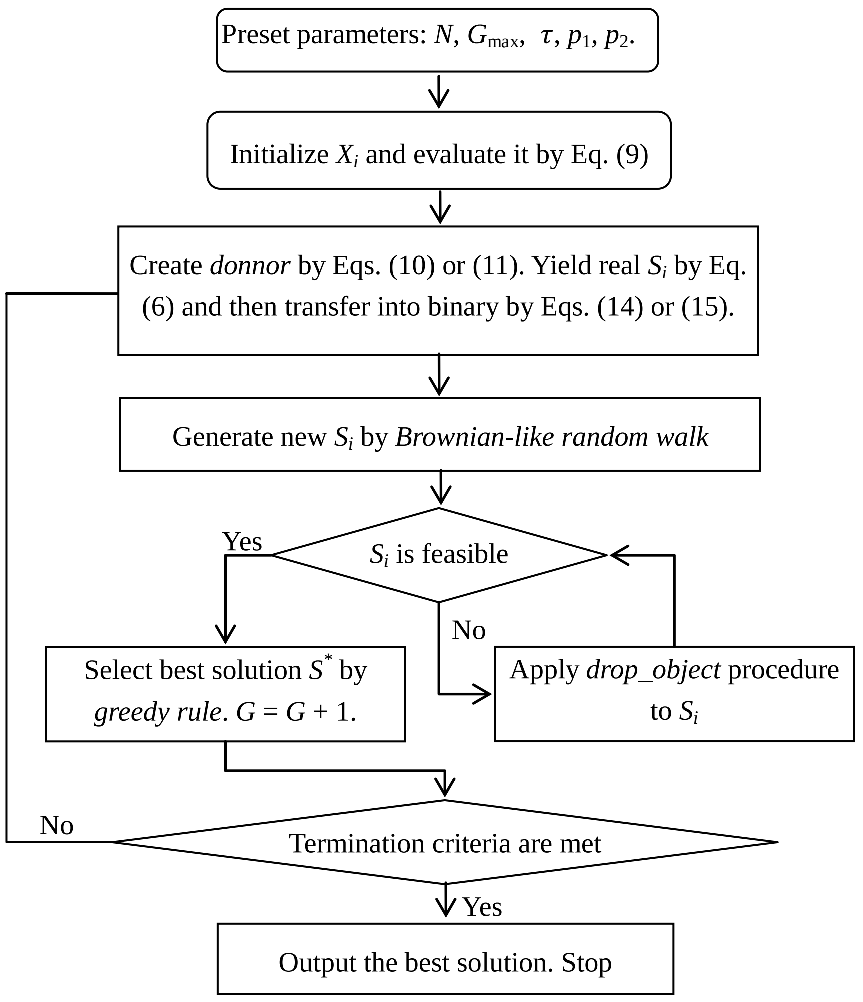
The Brownian motion-like random walk process searching for a stopover listed in Algorithm 1 is easily enforced for binary coded stopover position since the matrix R generated in the procedure of Algorithm 1 is a binary one valued. After *Brownian-like random walk*, the solution obtained may be infeasible. There are several standard approaches to handle the constraints in binary population-based methods [37, 38].

In BDS, a random heuristic procedure *drop_object* proposed in [19] is adopted to make solutions feasible. At first, we define a set $I = \{i_1, i_2, \cdots, i_D\}$ which composes of D randomly generated indices. Then the *drop_object* is performed on S_i using the set I to make the solution feasible. If a solution, which is such that the i_k-th $(k = 1, 2, \cdots, d)$ object is stored in the knapsack, does not satisfy the constraint, the i_k-th object is dropped (i.e., changing bit 1 to 0) each time from the knapsack according to the sequence of indices in the set I. This procedure is continued until a feasible solution is reached.

3.5. Termination criteria

For most swarm-based algorithms, the maximum number of iterations $Iter_{max}$ is usually taken as the termination criterion for each of these algorithms. Here, we choose another alternative criterion, maximum unimproved step (UI_{max}), along with the $Iter_{max}$. If it is not possible to improve the current best solution for a large number of iteration steps (UI_{max}) which is less than the maximum number of iterations, then the algorithm will terminate. The parameter UI_{max} is problem dependent. In general, the algorithm achieves good performance when UI_{max} is a constant satisfying $\frac{Iter_{max}}{20} \leq UI_{max} \leq \frac{Iter_{max}}{10}$.

The flowchart of BDS for MKPs is as shown in Figure 3. BDS is called Random Binary Differential Search (R-BDS) or Elitist Binary Differential Search (E-BDS) depending on whether *donor* is computed either through Eq. (10) or Eq. (11). If the binarization in R-BDS is enforced by Tanh function Eq. (13) (respectively, by Sigmoid function Eq.(14)), such an algorithm is called TR-BDS (respectively, SR-BDS). Similarly, TE-BDS (or SE-BDS) is the algorithm in which the binarization is enforced by Tanh function Eq. (13) (respectively, by Sigmoid function Eq.(14)).



Preset parameters: N, $G_{\max}$, τ, p_1, p_2.
Initialize X_i and evaluate it by Eq. (9)
Create *donnor* by Eqs. (10) or (11). Yield real S_i by Eq. (6) and then transfer into binary by Eqs. (14) or (15).
Generate new S_i by *Brownian-like random walk*
Yes
S_i is feasible
No
Select best solution S^* by *greedy rule*. $G = G + 1$.
Apply *drop_object* procedure to S_i
No
Termination criteria are met
Yes
Output the best solution. Stop

Figure 3: Flowchart of BDS.

4. Simulation and Evaluation

In order to test the performances of TR-BDS, TE-BDS, SR-BDS and SE-BDS, we will test them through solving three sets of well-known benchmark MKPs. Problem Set I includes 10 benchmark problems which are referred to as “sento” and “weing”. Problem Set II contains 30 benchmark problems which are referred to as “weish”. Problem Set I and Problem Set II are composed of small and medium sized knapsack problems, wherein the number of decision variables ranges from 20 to 105. Problem Set III are composed of high-dimensional knapsack problems, wherein the number of decision variables (n) ranges from 100 to 2500. All computational experiments are carried out within Matlab 7.5 environment on a PC equipped with Intel Pentium Dual-Core E5700 processor (3.00 GHz) with 2 GB of RAM under Windows System.

4.1. Parameters setting

It can be expected that the greater the values of the parameters N and G_{max}, the larger the probability is to obtain the optimal solution, and vice versa. However, it will take more computational time. In our implementations, we set $N = 30$ and $G_{max} = 10000$. In addition, the proposed BDS contains three key parameters: p_1, p_2 and τ in Eqs. (13) and (14). To investigate the influence of these three parameters on R-BDS and E-BDS, we test them on a medium-dimensional problem sento1 (number of objects $D = 60$, and number of knapsacks $m = 30$) for 100 independent trails by the method of the design of experiments (DOE) [39]. Considering five factor levels for each parameter, we list the orthogonal array L25(5^3) and the obtained average response variable (ARV, i.e., the average of total profit) values obtained by the BDS in Table 1. Meanwhile, we will investigate the effect of the tuning parameters to the numerical results. For this, we classify the numerical results in Table 1 into 5 different levels in which one of the parameters is fixed, while the other two parameters are varying. For example, the first response value 2896.2 in Table 2 means the average ARV of 5 problems with $\tau = 0.5$ in Table 1. All response values [39] are shown in Table 2 and depicted in Figure 4. Δ in Table 2 is the error between the largest ARV and the smallest ARV in the same column. It is clear that the larger Δ is, the more impact of the parameters to the algorithm. Thus, we can observe from Table 2 that all three parameters have significant impacts to SR-BDS and SE-BDS. However, for TR-BDS and TE-BDS, the parameter τ has more impact than the parameters p_1 and p_2.

Figure 4 clearly shows that the binary discretization scheme based on Eq. (13) achieves better performance than that based on Eq.(14) under the same parameters setting. Therefore, we only adopt TR-BDS and TE-BDS to solve MKP in the following discussions. From Table 2 and Figure 4, it can be observed that τ plays a key role for the numerical performance of the algorithm. The greater the value τ is, the more benefit it is to achieve. In the following discussions, we set $p_1 = 0.2, p_2 = 0.3$ and $\tau = 2.5$.

4.2. Analysis of numerical results by R-BDS and E-BDS

The comparison between TR-BDS and TE-BDS was carried out based on five performance measures: success rate (SR), mean value (Mean), standard deviation (Std) and average computational time (ACT). SR is the ratio between the number of runs in which a known optimal solution is captured and the number of executions. Mean is the average of the absolute differences between the simulation results and the known optimal solutions. Std is the standard deviation of final solutions over runs. ACT is the average time taken in each run of the algorithm. Table 3 shows the details of trial problems as well as the corresponding numerical results. More specifically, in the first three column, the name of the problem (Pro), number of knapsacks and items of the problem ($D \times m$) as well as the known optimum (Opt) are presented. Then, SR, Mean, Std and ACT corresponding to TR-BDS as well as TE-BDS are listed. The bar charts of success rate, Mean and Std are depicted in Fig. 5. From Table 3 and Fig. 5, we can observe that for the problems sento1 and sento2, TR-BDS has better success rate than TE-BDS but takes less average computational time. For the problems pb4 and weing4, both TR-BDS and TE-BDS capture the best known solution in the literature. However, for problems weing3 and weing7, the success rate of the two algorithms are near 0. For all the other problems, TR-BDS and TE-BDS have similar performance measures.

Table 4 displays the results for 30 problems of weish and the corresponding information is depicted in Fig. 6. All the results in Table 4 are based on 50 independent trials. For each problem, Table 4 not only reports the best known solutions from OR-library, but also reports the numerical results obtained by TR-BDS and TE-BDS in terms of SR, Mean, Std and ACT.

Table 1: Orthogonal array and average response variable (ARV) for BDS algorithms.

No. of Experiment	Parameter factors			ARV			
	τ	p_1	p_2	SR-BDS	SE-BDS	TR-BDS	TE-BDS
1	1 (0.5)	1 (0.3)	1 (0.3)	1508.0	4984.0	4660.0	4666.0
2	1 (0.5)	2 (0.4)	3 (0.5)	2170.0	2865.0	4430.0	4832.0
3	1 (0.5)	3 (0.5)	5 (0.7)	2625.0	4480.0	5390.0	4832.0
4	1 (0.5)	4 (0.6)	2 (0.4)	4846.0	2349.0	4610.0	4760.0
5	1 (0.5)	5 (0.7)	4 (0.6)	3332.0	1918.0	5132.0	4832.0
6	2 (1.0)	1 (0.3)	5 (0.7)	4096.0	3838.0	5939.0	6798.0
7	2 (1.0)	2 (0.4)	2 (0.4)	2134.0	5394.0	6213.0	6493.0
8	2 (1.0)	3 (0.5)	4 (0.6)	3095.0	4034.0	6615.0	6090.0
9	2 (1.0)	4 (0.6)	1 (0.3)	4691.0	3401.0	6485.0	6726.0
10	2 (1.0)	5 (0.7)	3 (0.5)	868.0	3661.0	6696.0	6003.0
11	3 (1.5)	1 (0.3)	4 (0.6)	4027.0	4741.0	6896.0	6818.0
12	3 (1.5)	2 (0.4)	1 (0.3)	2139.0	2273.0	7217.0	7077.0
13	3 (1.5)	3 (0.5)	3 (0.5)	4589.0	2724.0	7440.0	7067.0
14	3 (1.5)	4 (0.6)	5 (0.7)	3177.0	3156.0	7097.0	7271.0
15	3 (1.5)	5 (0.7)	2 (0.4)	3482.0	2743.0	7348.0	7332.0
16	4 (2.0)	1 (0.3)	3 (0.5)	3725.0	5533.0	7688.0	7294.0
17	4 (2.0)	2 (0.4)	5 (0.7)	2955.0	2456.0	7639.0	7641.0
18	4 (2.0)	3 (0.5)	2 (0.4)	2640.0	3500.0	7641.0	7502.0
19	4 (2.0)	4 (0.6)	4 (0.6)	3445.0	4825.0	7627.0	7685.0
20	4 (2.0)	5 (0.7)	1 (0.3)	5235.0	3056.0	7644.0	7355.0
21	5 (2.5)	1 (0.3)	2 (0.4)	2889.0	4459.0	7739.0	7719.0
22	5 (2.5)	2 (0.4)	4 (0.6)	4354.0	3492.0	7772.0	7263.0
23	5 (2.5)	3 (0.5)	1 (0.3)	5029.0	3452.0	7725.0	7728.0
24	5 (2.5)	4 (0.6)	3 (0.5)	2240.0	3800.0	7761.0	7709.0
25	5 (2.5)	5 (0.7)	5 (0.7)	2101.0	3286.0	7741.0	7675.0

Table 2: The response value for different BDS algorithms.

Level	SR-BDS			SE-BDS			TR-BDS			TE-BDS		
	τ	p_1	p_2	τ	p_1	p_2	τ	p_1	p_2	τ	p_1	p_2
level 1	2896.2	3249.0	3720.4	3319.2	4711.0	3433.2	4844.4	6584.4	6746.2	4784.4	6659.0	6710.4
level 2	2976.8	2750.4	3198.2	4065.6	3296.0	3689.0	6389.6	6654.2	6710.2	6422.0	6661.2	6761.2
level 3	3482.8	3595.6	2718.4	3127.4	3638.0	3716.6	7199.6	6962.2	6803.0	7113.0	6643.8	6581.0
level 4	3600.0	3679.8	3650.6	3874.0	3506.2	3802.0	7647.8	6716.0	6808.4	7495.4	6830.2	6537.6
level 5	3641.3	3003.6	2990.8	3590.8	2932.8	3443.2	7730.3	6912.2	6761.2	7574.8	6639.4	6843.4
Δ	745.1	929.4	1002.0	938.2	1778.2	368.8	2885.9	377.8	98.2	2790.4	190.8	305.8
Rank	3	2	1	2	1	3	1	2	3	1	3	2

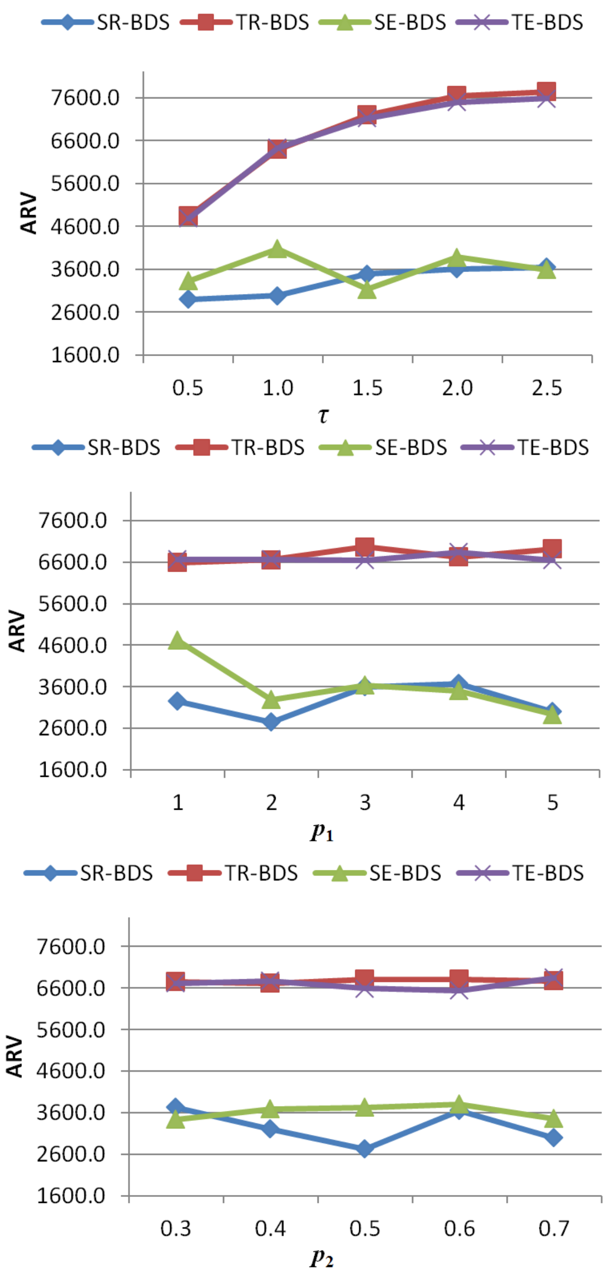


Figure 4: Fact level trend of BDS.

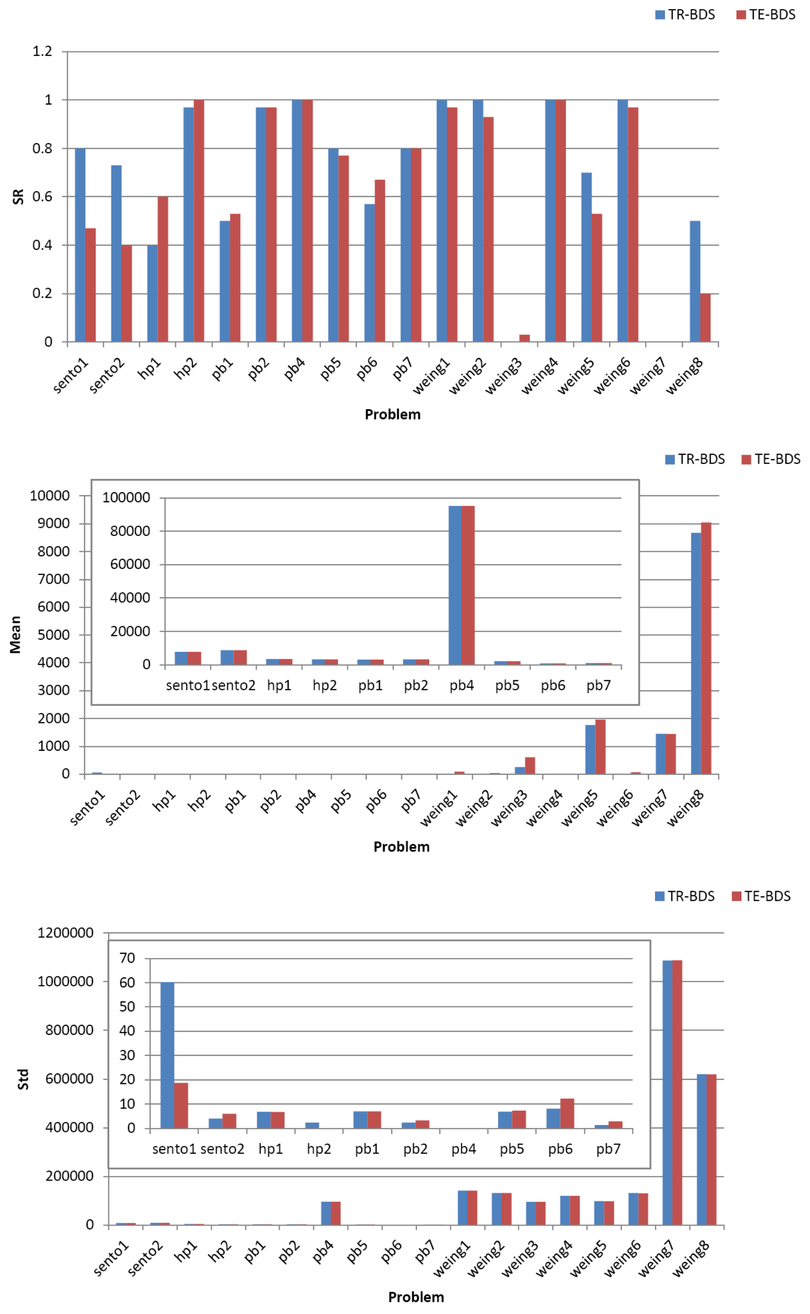


Figure 5: Success rate, mean value and standard derivation bar charts of TR-BDS and TE-BDS for Problem Set I.

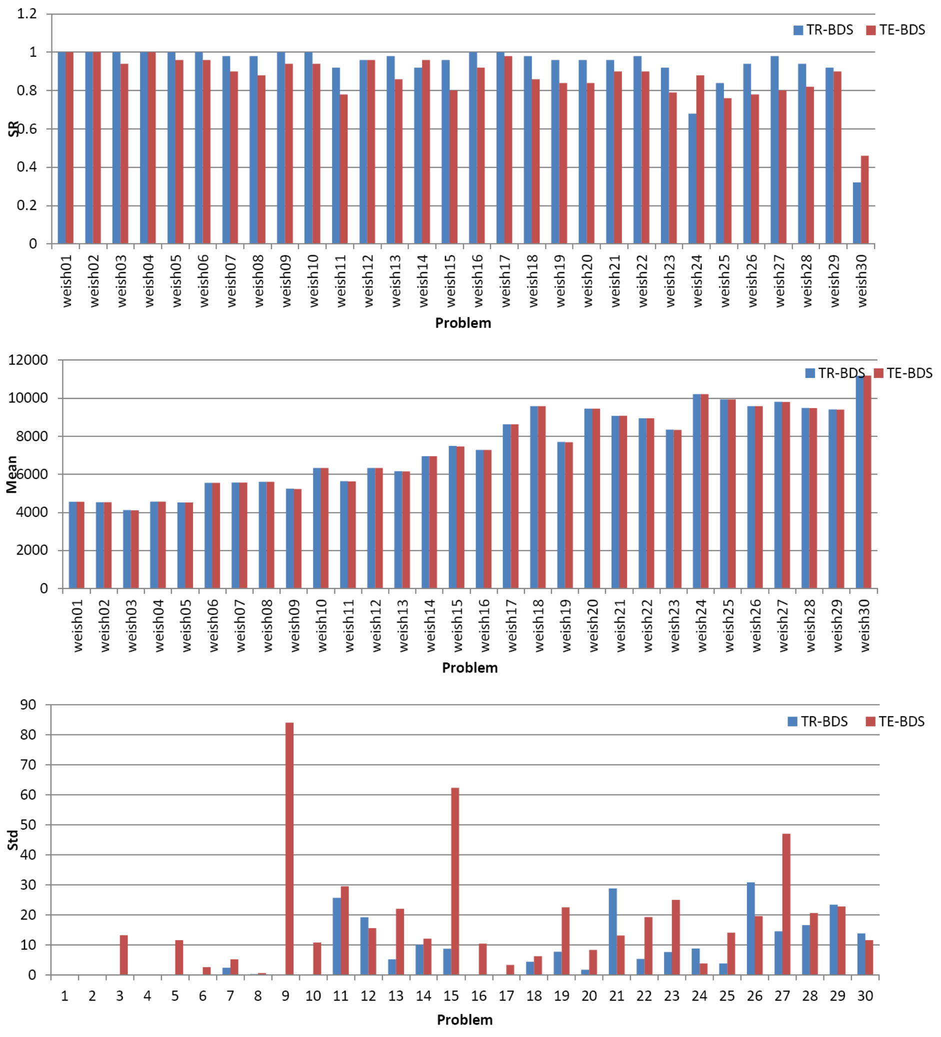


Figure 6: Success rate, mean value and standard derivation bar charts of TR-BDS and TE-BDS for Problem Set II.

Table 3: Results of both R-BDS and E-BDS on small scale MKP (Set I).

Prob.	$D \times m$	Opt	TR-BDS				TE-BDS			
			SR	Mean	Std	ACT (s)	SR	Mean	Std	ACT (s)
sento1	60×30	7772	0.80	7756.13	60.24	21.7131	0.47	7758.43	18.74	24.2188
sento2	60×30	8722	0.73	8719.93	4.03	21.6898	0.40	8717.43	6.07	24.8044
hp1	28×4	3418	0.40	3409.80	6.82	16.3319	0.60	3412.60	6.73	19.0435
hp2	35×4	3186	0.97	3185.57	2.37	16.5738	1.00	3186.00	0.00	19.2472
pb1	27×4	3090	0.50	3083.17	6.96	16.1599	0.53	3083.60	6.97	18.6705
pb2	34×4	3186	0.97	3185.57	2.37	16.5243	0.97	3185.40	3.29	19.0826
pb4	29×2	95168	1.00	95168.00	0.00	15.8367	1.00	95168.00	0.00	18.3799
pb5	20×10	2139	0.80	2135.60	6.92	16.4857	0.77	2135.03	7.31	18.8569
pb6	40×30	776	0.57	769.80	8.12	20.1615	0.67	768.87	12.28	22.8154
pb7	37×30	1035	0.80	1034.53	1.36	20.1421	0.80	1033.97	2.83	22.7094
weing1	28×2	141278	1.00	141278.00	0.00	15.6541	0.97	141261.33	91.29	17.7692
weing2	28×2	130883	1.00	130883.00	0.00	15.7355	0.93	130872.33	40.59	17.6515
weing3	28×2	95677	0.00	95003.87	248.59	15.6108	0.03	94837.67	611.37	17.6014
weing4	28×2	119337	1.00	119337.00	0.00	15.6454	1.00	119337.00	0.00	17.6944
weing5	28×2	98796	0.70	97662.60	1760.89	16.9703	0.53	97001.27	1955.54	17.6121
weing6	28×2	130623	1.00	130623.00	0.00	16.7528	0.97	130610.00	71.20	17.5885
weing7	105×2	1095445	0.00	1087354.87	1453.51	21.6950	0.00	1088624.43	1445.27	22.6013
weing8	105×2	624319	0.50	620258.83	8671.78	20.0473	0.20	619356.73	9046.16	21.2074

From Fig. 6, we can observe that TR-BDS and TE-BDS have similar performance measures. In particular, Mean obtained by TR-BDS are nearly the same as that obtained by TE-BDS from Fig. 6. Except for the problem weish30, the SRs of both algorithm are greater than 75%. The most encouraging result is that there are 10 problems in which success rates obtained by TR-BDS are 100%. Comparing TR-BDS and TE-BDS, there are only 3 problems in which the success rates obtained by TR-BDS are less than those obtained by TE-BDS. Furthermore, there are 26 problems from 30 problems in which Stds obtained by TR-BDS are better than those obtained by TE-BDS. In terms of computing time, for all problems in Table 4, nearly all of them can terminate and exit the computation within 20 seconds. Thus, we conclude that TR-BDS outperforms TE-BDS if the scale of MKP is not large.

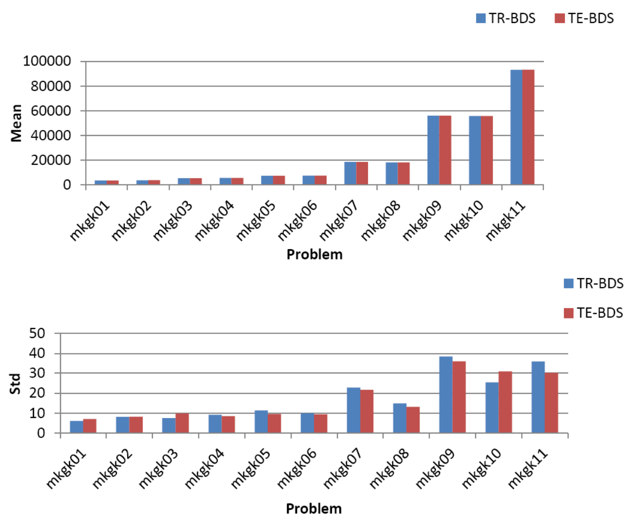


Figure 7: Mean value and standard derivation bar charts of TR-BDS and TE-BDS for Problem Set III.

Table 4: Comparisons of TR-BDS with TE-BDS on medium scale MKP (Set II).

Prob.	$D \times m$	Opt	TR-BDS				TE-BDS			
			SR	Mean	Std	ACT (s)	SR	Mean	Std	ACT (s)
weish01	30×5	4554	1.00	4554.00	0.00	15.2652	1.00	4554.00	0.00	16.7416
weish02	30×5	4536	1.00	4536.00	0.00	15.2703	1.00	4536.00	0.00	16.7622
weish03	30×5	4115	1.00	4115.00	0.00	15.2181	0.94	4111.78	13.28	16.7433
weish04	30×5	4561	1.00	4561.00	0.00	15.1311	1.00	4561.00	0.00	16.7153
weish05	30×5	4514	1.00	4514.00	0.00	15.1210	0.96	4511.66	11.62	16.6559
weish06	40×5	5557	1.00	5557.00	0.00	15.8183	0.96	5556.48	2.57	17.4351
weish07	40×5	5567	0.98	5566.66	2.40	15.8109	0.90	5565.28	5.21	17.3659
weish08	40×5	5605	0.98	5604.96	0.28	15.8611	0.88	5604.76	0.66	17.4458
weish09	40×5	5246	1.00	5246.00	0.00	15.7656	0.94	5232.54	84.01	17.2558
weish10	50×5	6339	1.00	6339.00	0.00	16.3044	0.94	6336.58	10.79	17.7758
weish11	50×5	5643	0.92	5636.40	25.67	16.1803	0.78	5628.28	29.50	17.7822
weish12	50×5	6339	0.96	6335.12	19.20	16.2363	0.96	6335.98	15.56	17.7879
weish13	50×5	6159	0.98	6158.26	5.23	16.1985	0.86	6151.14	22.02	17.7417
weish14	60×5	6954	0.92	6951.10	10.03	16.8444	0.96	6951.94	12.13	18.3319
weish15	60×5	7486	0.96	7484.24	8.71	16.8303	0.80	7471.56	62.36	18.3912
weish16	60×5	7289	1.00	7289.00	0.00	16.8837	0.92	7287.44	10.46	18.3757
weish17	60×5	8633	1.00	8633.00	0.00	17.1654	0.98	8632.52	3.39	18.9749
weish18	70×5	9580	0.98	9579.38	4.38	17.7140	0.86	9578.02	6.26	19.4470
weish19	70×5	7698	0.96	7696.64	7.75	17.4233	0.84	7689.00	22.56	19.1762
weish20	70×5	9450	0.96	9449.64	1.78	17.6414	0.84	9447.02	8.36	19.3375
weish21	70×5	9074	0.96	9069.04	28.85	17.5878	0.90	9069.76	13.17	19.2294
weish22	80×5	8947	0.98	8946.24	5.37	17.8769	0.90	8942.06	19.28	19.5396
weish23	80×5	8344	0.92	8342.74	7.64	17.8119	0.79	8332.24	24.95	19.4928
weish24	80×5	10220	0.68	10215.88	8.77	18.1588	0.88	10218.86	3.84	19.8913
weish25	80×5	9939	0.84	9937.74	3.87	18.0170	0.76	9932.98	14.11	19.6974
weish26	90×5	9584	0.94	9578.74	30.85	18.3924	0.78	9575.70	19.60	20.0068
weish27	90×5	9819	0.98	9816.94	14.57	18.2732	0.80	9802.18	47.03	19.9562
weish28	90×5	9492	0.94	9488.74	16.59	18.2810	0.82	9484.30	20.63	19.9651
weish29	90×5	9410	0.92	9403.62	23.36	18.3022	0.90	9402.84	22.79	19.9990
weish30	90×5	11191	0.32	11181.52	13.82	18.6734	0.46	11184.16	11.63	20.0000

Table 5: Comparisons of TR-BDS with TE-BDS on large-scale MKP (Set III).

Prob.	$D \times m$	TR-BDS			TE-BDS		
		Mean	Std	ACT (s)	Mean	Std	ACT (s)
mk_gk01	100×15	3688.26	6.02	21.1994	3720.86	7.01	22.5922
mk_gk02	100×25	3878.88	8.09	22.6084	3905.62	8.20	24.0208
mk_gk03	150×25	5512.06	7.55	25.6115	5542.22	9.87	26.9987
mk_gk04	150×50	5623.70	9.16	29.6552	5648.32	8.50	31.1158
mk_gk05	200×25	7349.14	11.41	28.4238	7376.84	9.56	29.9581
mk_gk06	200×50	7488.88	9.95	32.9163	7504.88	9.34	34.3660
mk_gk07	500×25	18588.20	22.74	42.4770	18600.36	21.70	44.1570
mk_gk08	500×50	18299.68	14.88	48.5358	18308.58	13.09	50.0867
mk_gk09	1500×25	56035.20	38.46	101.9255	56058.74	36.00	103.5136
mk_gk10	1500×50	55719.48	25.50	111.5450	55746.32	30.97	112.9763
mk_gk11	2500×100	93132.92	35.92	217.0807	93192.98	30.19	224.0877

To test the scalability of TR-BDS and TE-BDS, we will apply them to the MKPs in which the number of variables ranges from 100 to 2500. The corresponding information of problems as well as the numerical results obtained by TR-BDS and TE-BDS are presented in Table 5. The bar charts of Mean and Std are depicted in Figure 7. From Table 5 and Figure 7, we can observe that for all the problems, TE-BDS achieves better performance in terms of the Mean measure. Meanwhile, TR-BDS and TE-BDS seem to have the similar trend as for Std, i.e., if Std obtained by TR-BDS is large, Std obtained by TE-BDS will be large too, and vice versa. ACT shows that even for 2500-dimensional knapsack problem, each run for both of the two algorithms takes within 5 minutes. Thus, TR-BDS and TE-BDS are promising to solve large-scale MKPs. A standard convergence curves for three large-scale problems: mk gk09, mk gk10 and mk gk11 are shown in Figure 8.

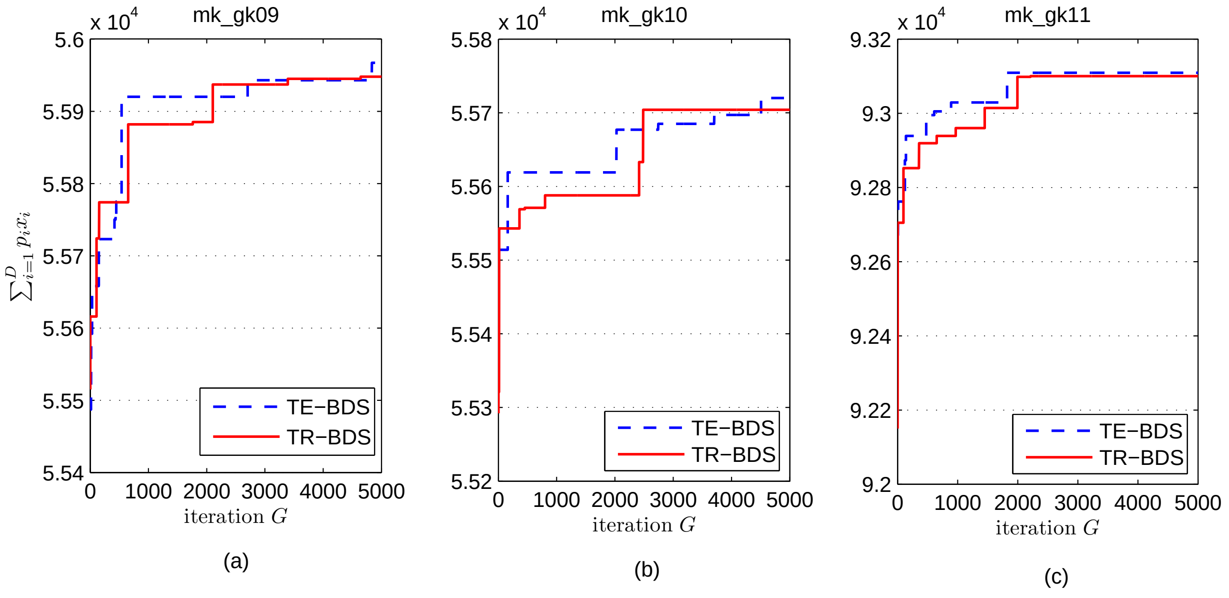


Figure 8: Convergence Curves for three large-scale problems, (a) for mk_gk09, (b) for mk_gk10 and (c) for mk_gk11.

4.3. Results comparison with other algorithms

This section will do numerical experiment on Problem Set I and Problem Set II mentioned above and 9 more complex benchmark problems from cbm.n presented by Chu and Beasley in [40]
.

4.3.1. Numerical results comparison on Problem Sets I and II

In this subsection, we will compare TR-BDS and TE-BDS with MPSO [20] and CBPSOCTVA [14]. To achieve this aim, we choose the benchmark problems from Problem Set I and Problem Set II for comparison since solving large-scale MKPs takes long time. The information of the problems as well as the numerical results obtained by TR-BDS, TE-BDS, MBPSO and CBPSOCTVA are listed in Table 6 and Table 7 in which only the success rates and standard deviations are presented.

We set the parameters in our algorithms as follow, $N = 30$, $G_{max} = 10000$, $p_1 = 0.2$, $p_2 = 0.3$ and $\tau = 2.5$.

Both Table 6 and Table 7 show that TR-BDS and TE-BDS outperform MBPSO in terms of SRs and Stds. Thus, TR-BDS and TE-BDS outperform MBPSO. Now the remaining is to compare TR-BDS and TE-BDS with CBPSOCTVA. Let us first analysis results in Table 6. In fact, for all the problems, except the problem weing3, SRs obtained by TR-BDS and TE-BDS, are much better than that obtained by CBPSOCTVA. In addition, for over 80% problems, Stds obtained by TR-BDS and TE-BDS are less than that obtained by CBPSOCTVA. This observation is also true in Table 7. The box plots of Problem Set I and Problem Set II are depicted in Figure 9 and Figure 10. These two figures are further evidenced that TR-BDS and TE-BDS are superior to the algorithms MBPSO and CBPSOCTVA.

Table 6: Comparisons of TR-BDS, TE-BDS with MBPSO, and CBPSOTVAC on small scale MKP (Set I).

Prob.	$D \times m$	Opt	TR-BDS		TE-BDS		MBPSO[20]		CBPSOCTVA[14]	
			SR	Std	SR	Std	SR	Std	SR	Std
sento1	60×30	7772	0.80	60.24	0.47	18.74	0.16	43.23	0.39	357.78
sento2	60×30	8722	0.73	4.03	0.40	6.07	0.03	18.80	0.20	101.03
hp1	28×4	3418	0.40	6.82	0.60	6.73	0.10	25.52	0.38	10.69
hp2	35×4	3186	0.97	2.37	1.00	0.00	0.11	39.15	0.59	21.35
pb1	27×4	3090	0.50	6.96	0.53	6.97	0.11	24.32	0.40	10.52
pb2	34×4	3186	0.97	2.37	0.97	3.29	0.16	39.31	0.51	18.73
pb4	29×2	95168	1.00	0.00	1.00	0.00	0.27	1803	0.84	875.1
pb5	20×10	2139	0.80	6.92	0.77	7.31	0.08	24.36	0.80	6.83
pb6	40×30	776	0.57	8.12	0.67	12.28	0.28	29.12	0.54	40.17
pb7	37×30	1035	0.80	1.36	0.80	2.83	0.05	16.29	0.40	24.25
weing1	28×2	141278	1.00	0.00	0.97	91.29	0.82	250.43	0.92	281.98
weing2	28×2	130883	1.00	0.00	0.93	40.59	0.65	314.08	0.88	545.50
weing3	28×2	95677	0.00	248.59	0.03	611.37	0.11	876.78	0.75	672.42
weing4	28×2	119337	1.00	0.00	1.00	0.00	0.76	1270.80	0.97	378.58
weing5	28×2	98796	0.70	1760.89	0.53	1955.54	0.52	1923.5	0.94	572.82
weing6	28×2	130623	1.00	0.00	0.97	71.20	0.36	322.40	0.97	343.45
weing7	105×2	1095445	0.00	1453.51	0.00	1445.27	0.02	1130.60	0.00	30020.00
weing8	105×2	624319	0.50	8671.78	0.20	9046.16	0.03	4704.30	0.20	75169.00

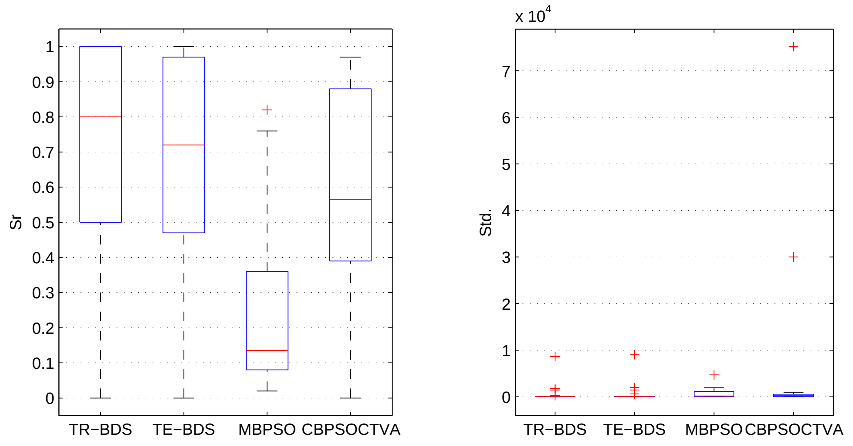


Figure 9: Boxplots of Problem set I of Multidimensional Knapsack Problem.

Table 7: Comparisons of TR-BDS, TE-BDS with MBPSO, and CBPSOTVAC on medium scale MKP (Set II).

Prob.	$D \times m$	Opt	TR-BDS		TE-BDS		MBPSO[20]		CBPSOCTVA[14]	
			SR	Std	SR	Std	SR	Std	SR	Std
weish01	30×5	4554	1.00	0.00	1.00	0.00	0.82	26.34	1.00	0.00
weish02	30×5	4536	1.00	0.00	1.00	0.00	0.55	18.01	0.66	23.12
weish03	30×5	4115	1.00	0.00	0.94	13.28	0.63	34.98	0.95	52.69
weish04	30×5	4561	1.00	0.00	1.00	0.00	0.96	8.99	0.99	85.90
weish05	30×5	4514	1.00	0.00	0.96	11.62	0.99	5.40	0.98	74.45
weish06	40×5	5557	1.00	0.00	0.96	2.57	0.32	14.39	0.53	79.28
weish07	40×5	5567	0.98	2.40	0.90	5.21	0.64	18.92	0.78	71.95
weish08	40×5	5605	0.98	0.28	0.88	0.66	0.44	13.07	0.68	42.81
weish09	40×5	5246	1.00	0.00	0.94	84.01	0.78	25.65	0.85	65.70
weish10	50×5	6339	1.00	0.00	0.94	10.79	0.56	22.17	0.67	188.63
weish11	50×5	5643	0.92	25.67	0.78	29.50	0.40	43.95	0.62	403.03
weish12	50×5	6339	0.96	19.20	0.96	15.56	0.65	35.68	0.71	304.43
weish13	50×5	6159	0.98	5.23	0.86	22.02	0.87	25.19	0.85	180.04
weish14	60×5	6954	0.92	10.03	0.96	12.13	0.66	25.95	0.79	364.66
weish15	60×5	7486	0.96	8.71	0.80	62.36	0.72	18.64	0.80	554.35
weish16	60×5	7289	1.00	0.00	0.92	10.46	0.44	17.49	0.43	367.29
weish17	60×5	8633	1.00	0.00	0.98	3.39	0.56	7.38	0.72	227.16
weish18	70×5	9580	0.98	4.38	0.86	6.26	0.38	18.40	0.53	275.53
weish19	70×5	7698	0.96	7.75	0.84	22.56	0.55	33.67	0.62	489.37
weish20	70×5	9450	0.96	1.78	0.84	8.36	0.53	15.99	0.69	410.74
weish21	70×5	9074	0.96	28.85	0.90	13.17	0.61	24.97	0.67	378.38
weish22	80×5	8947	0.98	5.37	0.90	19.28	0.33	31.55	0.17	486.71
weish23	80×5	8344	0.92	7.64	0.79	24.95	0.24	35.43	0.58	437.23
weish24	80×5	10220	0.68	8.77	0.88	3.84	0.27	18.09	0.55	295.79
weish25	80×5	9939	0.84	3.87	0.76	14.11	0.29	13.39	0.32	361.88
weish26	90×5	9584	0.94	30.85	0.78	19.60	0.31	24.27	0.28	710.77
weish27	90×5	9819	0.98	14.57	0.80	47.03	0.65	48.62	0.83	640.43
weish28	90×5	9492	0.94	16.59	0.82	20.63	0.64	26.72	0.62	887.33
weish29	90×5	9410	0.92	23.36	0.90	22.79	0.46	34.73	0.48	854.50
weish30	90×5	11191	0.32	13.82	0.46	11.63	0.38	14.48	0.63	491.81

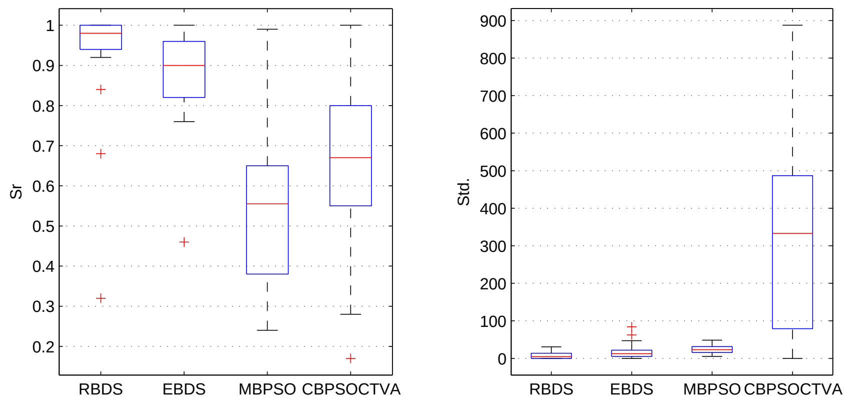


Figure 10: Boxplots of problem set II of Multidimensional Knapsack Problem.

Boxplots of TR-BDS, TE-BDS as well as MBPSO and CBPSOCTVA on Problem Set I and Problem Set II are shown in Figure 9 and Figure 10. From the two figures, we can clearly observe that TR-BDS and TE-BDS perform better than MBPSO and CBPSOCTVA in terms of SR and Std.

4.3.2. Numerical results comparison on MKP problems set proposed by Chu and Beasley

cb$m.n_r$ (where m, n and r is the number of constraints, variables and the instance, respectively) is another well-known set of MKP instances available on the OR-Library website [41]. We choose 9 and 3 problems from cb5.100 and cb10.500, respectively, to test our algorithms. We set the parameters in our algorithms as follow, $N = 100$, $G_{max} = 50000$, $p_1 = 0.2$, $p_2 = 0.3$ and $\tau = 2.5$. The comparison results on 9 instances from cb5.500 with BSA [42] and ABC [22] are presented in Tables 8. The comparison results on 12 instances from cb10.500 with other two algorithms [13, 7] are listed in Table 9. For clarity, these problems are renamed as ORmxn-t_r (where m, n and r is still the number of constraints, variables and the instance, respectively, and t is a tightness rate by which the coefficients in both sides of Eq. (2) are constrained, that is $C_j = t\sum_{i=1}^{D} w_{ij}, \quad \forall j = 1, 2, \cdots, m$) in the first collum of Tables 8 and 9.

Table 8: Comparisons of TR-BDS, TE-BDS with BSA and ABC on cb5.100.

Prob.	Best known	BSA [42]		ABC [22]		TR-BDS		TE-BDS	
		Best	Mean	Best	Mean	Best	Mean	Best	Mean
OR5x100-0.25_1	24381	24381	24381	24381	24381	24381	24381	24381	24381
OR5x100-0.25_2	24274	24274	24274	24274	24274	24274	24224.68	24274	24274
OR5x100-0.25_3	23551	23551	23551	23551	23547.1	23551	23535.96	23551	23551
OR5x100-0.50_1	23534	23534	23534	23534	23534	23534	23534	23534	23534
OR5x100-0.50_2	23991	23991	23991	23991	23988.5	23966	23944.64	23991	23991
OR5x100-0.50_3	24613	24613	24613	24613	24613	24613	24539.52	24613	24613
OR5x100-0.75_1	25591	25591	25591	25591	25591	25591	25523.88	25591	25591
OR5x100-0.75_2	23410	23410	23410	23410	23410	23410	23356.80	23410	23410
OR5x100-0.75_3	24216	24216	24216	24216	24216	24216	24198.24	24216	24216

Table 8 lists test results for 9 problems selected from benchmarks cb5.100. For each problem, the table reports the best and average solutions found by TR-BDS, TE-BDS as well as those obtained by M. Kong et al [42] and S. Sundar et al [22].

We can observe from Table 8 that BSA, ABC and TE-BDS find all the best solutions for the 9 tested problems. For TR-BDS, it achieves the best solutions for 8 problems, but it cannot for problem OR5x100-0.50_2. Both BSA and TE-BDS get better average solutions for all the 9 problems. Indeed, in each trial, both BSA and TE-BDS can capture the best solution for each tested problem. However, TR-BDS and ABC are not the case. From this point of view, BSA and TE-BDS are more robust than ABC and TR-BDS.

Table 9 shows the results for 12 problems of cb10.500. For each problem, the best solutions found by M. Vasquez et al [13] and S. Boussier et al [7], the best and average solution achieved by TR-BDS and TE-BDS, the average computing time spent by TR-BDS and TE-BDS are listed in this table. Among them, Alg. 2 [7] achieves the best solutions for all problems; TR-BDS achieves three best solutions and TE-BDS achieves seven best solutions for 12 tested problems. It is a pity that the average solutions obtained by TR-BDS and TE-BDS are not so stable as those reported in the previous tables. This may be caused by the complexity of this set of tested problems. The average time spent by our algorithms are less than 7500 seconds. Comparing TE-BDS with Alg.

Table 9: Comparisons of TR-BDS, TE-BDS with Algorithm 1 in [13] and Algorithm 2 in [7] for cb10.500, the computing time by TR-BDS and TE-BDS.

Prob.	Alg. 1 [13]	Alg. 2 [7]	TR-BDS			TE-BDS		
			Best	Mean	Time(s)	Best	Mean	Time(s)
OR10x500-0.25_1	117811	117821	114716	114425.40	6650	117811	117801.20	6725
OR10x500-0.25_2	119232	119249	119232	119223.00	6652	**119249**	118024.00	6795
OR10x500-0.25_3	119215	119215	117821	117625.60	6637	**119215**	117801.40	6795
OR10x500-0.25_4	118813	118829	118813	117625.80	6645	118813	117801.20	6825
OR10x500-0.50_1	217377	217377	209191	208710	7092	**217377**	212570	7094
OR10x500-0.50_2	219077	219077	219077	217277	7068	**219077**	218570	7140
OR10x500-0.50_3	217806	217847	210282	210172	7131	217377	212570	7543
OR10x500-0.50_4	216868	216868	209242	206178	7192	**216868**	216868	7640
OR10x500-0.75_1	304387	304387	304387	302658	7469	**304387**	304264	7552
OR10x500-0.75_2	302379	302379	302379	302658	6969	**302379**	302164	6998
OR10x500-0.70_3	302416	302417	290931	290859	7102	302416	302014	7610
OR10x500-0.70_4	300757	300784	290859	290021	7082	291295	291170	7640

2 [7], we can observe that the performance of TE-BDS is slightly worse than that of Alg. 2 [7] which is based on exact algorithm. However, the solution time obtained by our method is much less than that obtained by Alg. 2 [7]. it is better if you can list the computational time for a test problem and compare it. Thus, TE-BDS can be viewed as a complement of Alg. 2 [7] to trade off computational time and solution performance.

5. Conclusion

In this paper, a novel binary DS algorithm incorporating two different solution strategies, TR-BDS and TE-BDS, was introduced to solve 0-1 MKPs. Three sets of benchmark tests are solved so as to investigate the numerical performances of TR-BDS and TE-BDS. In addition, TR-BDS and TE-BDS are compared with similar algorithms developed recently in the literature. The numerical results show that TR-BDS and TE-BDS not only can be used to solve large-scale 0-1 MKPs, but also outperform the existing algorithms for Problem Sets I and II. For more complex benchmarks cbm.n, our algorithms perform as better as other meta-heuristic algorithms, but slightly worse than the exact algorithm in [7]. However, exact algorithms are suffered from expensive computation, but meta-heuristic algorithm are not. An interesting future research topic is to investigate the applications of TR-BDS and TE-BDS in other real applications, such as project scheduling, discrete-valued optimal control and resource allocation.

References

[1] A. Frieze, M. Clarke, Approximation algorithms for the m-dimensional 0–1 knapsack problem: Worst-case and probabilistic analyses, European Journal of Operational Research 15 (1) (1984) 100–109. 1

[2] C. Wilbaut, S. Hanafi, S. Salhi, A survey of effective heuristics and their application to a variety of knapsack problems, IMA Journal of Management Mathematics 19 (3) (2008) 227–244. 1

[3] J. Nawrocki, W. Complak, J. Błażewicz, S. Kopczyńska, M. Maćkowiaki, The knapsack-lightening problem and its application to scheduling hrt tasks, Bulletin of the Polish Academy of Sciences: Technical Sciences 57 (1) (2009) 71–77. 1

[4] C. Wu, X. Wang, J. Lin, Optimizations in project scheduling: A state-of-art survey, in: Optimization and Control Methods in Industrial Engineering and Construction, Springer, 2014, pp. 161–177. 1

[5] J. Puchinger, G. R. Raidl, U. Pferschy, The multidimensional knapsack problem: Structure and algorithms, INFORMS Journal on Computing 22 (2) (2010) 250–265. 1

[6] Y. Zhang, F. Zhang, M. Cai, Some new results on multi-dimension knapsack problem, Journal of Industrial and Management Optimization 1 (3) (2005) 315. 2

[7] S. Boussier, M. Vasquez, Y. Vimont, S. Hanafi, P. Michelon, A multi-level search strategy for the 0–1 multidimensional knapsack problem, Discrete Applied Mathematics 158 (2) (2010) 97–109. 2, 20, 21

[8] M. Varnamkhasti, Overview of the algorithms for solving the multidimensional knapsack problems, Advanced Studies in Biology 4 (1) (2012) 37–47. 2

[9] C. Wu, C. Li, Q. Long, A dc programming approach for sensor network localization with uncertainties in anchor positions, Journal of Industrial and Management Optimization 10 (3) (2014) 817–826. 2

[10] Q. Long, C. Wu, A hybrid method combining genetic algorithm and hooke-jeeves method for constrained global optimization, Journal of Industrial and Management Optimization 10 (4) (2014) 1279–1296. 2

[11] X. Sun, H. Sheng, D. Li, An exact algorithm for 0-1 polynomial knapsack problems, Journal of industrial and management optimization 3 (2) (2007) 223. 2

[12] J. Zhou, D. Chen, Z. Wang, W. Xing, A conic approximation method for the 0-1 quadratic knapsack problem, MANAGEMENT 9 (3) (2013) 531–547. 2

[13] M. Vasquez, Y. Vimont, Improved results on the 0–1 multidimensional knapsack problem, European Journal of Operational Research 165 (1) (2005) 70–81. 2, 20, 21

[14] M. Chih, C.-J. Lin, M.-S. Chern, T.-Y. Ou, Particle swarm optimization with time-varying acceleration coefficients for the multidimensional knapsack problem, Applied Mathematical Modelling 38 (4) (2014) 1338–1350. 2, 17, 18, 19

[15] S. Balev, N. Yanev, A. Fréville, R. Andonov, A dynamic programming based reduction procedure for the multidimensional 0–1 knapsack problem, European Journal of Operational Research 186 (1) (2008) 63–76. 2

[16] D. C. Vanderster, N. J. Dimopoulos, R. Parra-Hernandez, R. J. Sobie, Resource allocation on computational grids using a utility model and the knapsack problem, Future Generation computer systems 25 (1) (2009) 35–50. 2

[17] V. Boyer, D. El Baz, M. Elkihel, A dynamic programming method with lists for the knapsack sharing problem, Computers & Industrial Engineering 61 (2) (2011) 274–278. 2

[18] Q. Long, C. Wu, T. Huang, X. Wang, A genetic algorithm for unconstrained multi-objective optimization, Swarm and Evolutionary Computation 22 (2015) 1–14. 2

[19] M. A. K. Azad, A. M. A. Rocha, E. M. Fernandes, A simplified binary artificial fish swarm algorithm for 0–1 quadratic knapsack problems, Journal of Computational and Applied Mathematics 259 (2014) 897–904. 2, 8

[20] J. C. Bansal, K. Deep, A modified binary particle swarm optimization for knapsack problems, Applied Mathematics and Computation 218 (22) (2012) 11042–11061. 2, 17, 18, 19

[21] A. Baykasoğlu, F. B. Ozsoydan, An improved firefly algorithm for solving dynamic multidimensional knapsack problems, Expert Systems with Applications 41 (8) (2014) 3712–3725. 2

[22] S. Sundar, A. Singh, A. Rossi, An artificial bee colony algorithm for the 0–1 multidimensional knapsack problem, in: Contemporary Computing, Springer, 2010, pp. 141–151. 2, 20

[23] C. Changdar, G. Mahapatra, R. K. Pal, An ant colony optimization approach for binary knapsack problem under fuzziness, Applied Mathematics and Computation 223 (2013) 243–253. 2, 3

[24] L. Wang, X.-l. Zheng, S.-y. Wang, A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem, Knowledge-Based Systems 48 (2013) 17–23. 2

[25] Q. Gong, Y. Zhou, Q. Luo, Hybrid artificial glowworm swarm optimization algorithm for solving multi-dimensional knapsack problem, Procedia Engineering 15 (2011) 2880–2884. 2

[26] 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 (19) (2013) 9959–9970. 2

[27] 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. 2

[28] 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. 3

[29] P. Civicioglu, Transforming geocentric cartesian coordinates to geodetic coordinates by using differential search algorithm, Computers & Geosciences 46 (2012) 229–247. 3

[30] J. Liu, C. Wu, X. Wang, G. Wu, A novel differential search algorithm and application for structure design, Applied Mathematics and Computation`doi:10.1016/j.amc.2015.06.036`. 3

[31] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, Evolutionary Computation, IEEE Transactions on 1 (1) (1997) 67–82. 4

[32] X.-S. Yang, X. He, Firefly algorithm: recent advances and applications, International Journal of Swarm Intelligence 1 (1) (2013) 36–50. 4

[33] Z. I. Botev, D. P. Kroese, An efficient algorithm for rare-event probability estimation, combinatorial optimization, and counting, Methodology and Computing in Applied Probability 10 (4) (2008) 471–505. 6

[34] J. Liu, K. Teo, X. Wang, C. Wu, An exact penalty function-based differential search algorithm for constrained global optimization, Soft Computing (2015) 1–9doi:10.1007/s00500-015-1588-6. 6

[35] L. Wang, C. Singh, Unit commitment considering generator outages through a mixed-integer particle swarm optimization algorithm, Applied soft computing 9 (3) (2009) 947–953. 7

[36] K. Chandrasekaran, S. P. Simon, N. P. Padhy, Binary real coded firefly algorithm for solving unit commitment problem, Information Sciences 249 (2013) 67–84. 7

[37] M. A. K. Azad, A. M. A. Rocha, E. M. Fernandes, Improved binary artificial fish swarm algorithm for the 0–1 multidimensional knapsack problems, Swarm and Evolutionary Computation 14 (2014) 66–75. 8

[38] M. Sakawa, K. Kato, Genetic algorithms with double strings for 0–1 programming problems, European Journal of Operational Research 144 (3) (2003) 581–597. 8

[39] D. C. Montgomery, Design and analysis of experiments, John Wiley & Sons, 2008. 10

[40] P. Chu, J. Beasley, A genetic algorithm for the multidimensional knapsack problem, Journal of Heuristics 4 (1) (1998) 63–86. 17

[41] J. E. Beasley, Or-library: distributing test problems by electronic mail, Journal of the operational research society (1990) 1069–1072.
URL http://people.brunel.ac.uk/~mastjjb/jeb/info.html 20

[42] M. Kong, P. Tian, Y. Kao, A new ant colony optimization algorithm for the multidimensional knapsack problem, Computers & Operations Research 35 (8) (2008) 2672–2683. 20