A two-phase tabu-evolutionary algorithm for the 0–1 multidimensional knapsack problem

Xiangjing Lai [a], Jin-Kao Hao [b,c,*], Fred Glover [d], Zhipeng Lü [e]

[a] *Institute of Advanced Technology, Nanjing University of Posts and Telecommunications, Nanjing 210023, China*

[b] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[c] *Institut Universitaire de France, 1 Rue Descartes, 75231 Paris, France*

[d] *OptTek Systems, Inc., 2241 17th Street Boulder, Colorado 80302, USA*

[e] *SMART, School of Computer Science and Technology, Huazhong University of Science and Technology, 430074 Wuhan, P.R.China*

Information Sciences 436-437:282-301, 2018
https://doi.org/10.1016/j.ins.2018.01.026

Abstract

The 0–1 multidimensional knapsack problem is a well-known NP-hard combinatorial optimization problem with numerous applications. In this work, we present an effective two-phase tabu-evolutionary algorithm for solving this computationally challenging problem. The proposed algorithm integrates two solution-based tabu search methods into the evolutionary framework that applies a hyperplane-constrained crossover operator to generate offspring solutions, a dynamic method to determine search zones of interest, and a diversity-based population updating rule to maintain a healthy population. We show the competitiveness of the proposed algorithm by presenting computational results on the 281 benchmark instances commonly used in the literature. In particular, in a computational comparison with the best algorithms in the literature on multiple data sets, we show that our method on average matches more than twice the number of best known solutions to the harder problems than any other method and in addition yields improved best solutions (new lower bounds) for 4 difficult instances. We investigate two key ingredients of the algorithm to understand their impact on the performance of the algorithm.
Keywords: Combinatorial optimization; Multidimensional knapsack problem; Solution-based tabu search; Meta-heuristics.

---

* Corresponding author.
Email addresses: `laixiangjing@gmail.com` (Xiangjing Lai), `jin-kao.hao@univ-angers.fr` (Jin-Kao Hao), `glover@opttek.com` (Fred Glover),

1 Introduction

Given a set $V = \{1, 2, \ldots, n\}$ of n items and m resources with a capacity limit b_i for each resource i, each item j has a profit p_j and consumes a given quantity of each resource r_{ij}. Then the 0–1 multidimensional knapsack problem (MKP) is to select a subset of items such that the resource consumed by the selected items does not exceed the capacity limit for each resource (knapsack constraints), while maximizing the total profit of the selected items.

Formally, the MKP can be formulated as the following general 0–1 linear program with multiple constraints.

$$\text{Maximize} \quad f(s) = \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} r_{ij} x_j \leq b_i, \forall i \in \{1, 2, \ldots, m\} \tag{2}$$

$$x_j \in \{0, 1\}, \forall j \in \{1, 2, \ldots, n\} \tag{3}$$

where the decision variables x_j ($j \in \{1, 2, \ldots, n\}$) indicate whether the associated items are selected, i.e., $x_j = 1$ if the item j is selected, and $x_j = 0$ otherwise. Constraints (2) ensure that the m knapsack constraints are satisfied while equation (1) maximizes the total profit of the selected items.

The MKP is a well-known constrained combinatorial optimization problem that has numerous applications, including cutting stock [16], loading problem [38], resources allocation in distributed computing [14], among others. The MKP is known to be NP-hard [13] and thus computationally challenging.

Due to its practical importance and NP-hard character, much effort has been dedicated to the MKP, and in the past several decades, a large number of exact and heuristic algorithms have been proposed for solving it. In [12], a comprehensive review of the studies till 2004 is provided. Here, we focus mainly on some of the most representative work. The best known exact algorithms include several branch & bound algorithms [15,38,41], the hybrid exact algorithm that combines resolution search, branch & bound and depth first search [6], the CORAL algorithm that combines branch & bound and variable fixation [32]. At present, the most efficient exact algorithms are able to produce the optimal solutions for instances of small and moderate size, primarily with the number of variables and constraints limited to $n \leq 250$ and $m \leq 10$. However, they may fail on larger instances, e.g., when $n \geq 500$ and $m \geq 30$. Indeed, even

zhipeng.lui@gmail.com (Zhipeng Lü).

the best hybrid exact algorithms like those proposed in [6,32] have trouble to solve some instances with $n = 250$ and $m = 30$.

In addition to exact solution approaches, a variety of heuristic algorithms are available in the literature, which can mainly be divided into two categories, namely single-solution based local search algorithms and population-based optimization algorithms. Examples of successful local search algorithms include tabu search [23,25,40], simulated annealing [11] and kernel search [2]. Representative population-based methods include genetic and memetic algorithms [10,37], hybrid binary particle swarm optimization [5,9,24,30], ant colony optimization [1], tabu search-based PSO algorithm [28], and path relinking [3]. Other fashionable algorithms include harmony search [29,42,47], binary differential evolution algorithm [4], fruit fly algorithm [33] and binary artificial bee colony algorithm [34], etc. However, these latter methods only perform well on small instances with $n \leq 100$ and are rarely competitive on large instances, especially with $n \geq 500$.

Given the NP-hard feature of the MKP, there is a continuing need for more powerful and effective methods to better solve the problem. In this work, we introduce a two-phase tabu evolutionary algorithm (TPTEA) that particularly relies on two solution-based tabu search procedures to explore different search spaces. Solution-based tabu search [7,8,43,45] is an interesting search approach that nevertheless has received much less attention than the popular attribute-based tabu search approach [22]. According to some recent studies like [43], solution-based tabu search algorithms may be particularly efficient for solving some binary optimization problems like the minimum difference dispersion problem. In this work, we investigate the solution-based tabu search approach in combination with the popular hybrid evolutionary framework.

The main contributions of this work can be summarized as follows.

- We investigate for the first time solution-based tabu search for solving the MKP and design two dedicated solution-based tabu search procedures to explore different search spaces. We integrate these tabu search procedures within a population-based evolutionary framework to obtain a two-phase search algorithm that is able to ensure an effective intensification and diversification within the search space.
- We develop a self-adapting mechanism to locate interesting search regions (represented by specific hyperplanes) which are examined thoroughly by the algorithm.
- We provide computational results on the 281 commonly used benchmark instances and compare our outcomes with those of state-of-the-art MKP algorithms in the literature. In particular, we obtain improved best known results (new lower bounds) for 4 hard benchmark instances. To our knowledge, the last updates of lower bounds for the MKP instances occurred in

2012 [32].

The remainder of the paper is organized as follows. In the next section, we describe our proposed algorithm and its key components. Computational results and comparisons are presented in Section 3. In Section 4, two essential strategies of the algorithm are analyzed to shed light on how they affect the performance of the algorithm. Finally, we draw conclusions and provide perspectives for future studies.

2 Two-phase hybrid tabu-evolutionary algorithm for the MKP

To describe the two-phase tabu-evolutionary algorithm for the MKP (TPTEA), we first introduce the different search spaces explored by the algorithm and then explain the procedures for handling them.

2.1 Solution representation and search space

Given a MKP instance with a set $V = \{1, 2, \ldots, n\}$ of n items, any candidate solution s can be represented by a n-dimensional binary vector $(x_1, x_2, \ldots, x_n)$ such that $x_j = 1$ if the item j is selected, and $x_j = 0$ otherwise. In this work, we employ additionally two vectors S and NS to indicate respectively the sets of selected items and unselected items in a solution. Thus, any solution s in the search space can also be represented by $s =< S, NS >$. This representation is convenient for describing the *swap* operator used by the tabu search procedures of Section 2.4.

Let Ω be the set of all n-dimensional binary vectors, i.e.,

$$\Omega = \{x \ : \ x \in \{0,1\}^n\} \tag{4}$$

Clearly, Ω contains both feasible solutions and infeasible solutions, which defines the largest possible search space for the given instance.

The feasible search space $\Omega^F \subset \Omega$ can be written as

$$\Omega^F = \{x \in \{0,1\}^n \ : \ \sum_{j=1}^{n} r_{ij} x_j \leq b_i, 1 \leq i \leq m\} \tag{5}$$

Finally, given a candidate solution $s = (x_1, x_2, \ldots, x_n)$ in Ω, let $\sum_{j=1}^{n} x_j = k$ be a k-dimensional hyperplane constraint that restricts the components x_j of the solution to have exactly k variables taking the value of 1, we define $\Omega_{[k]}$ to be a subspace of Ω satisfying the hyperplane-constraint i.e.,

$$\Omega_{[k]} = \{x \in \{0,1\}^n \ : \ \sum_{j=1}^{n} x_j = k\} \tag{6}$$

$\Omega_{[k]}$ will be called the k-dimensional hyperplane space or simply a hyperplane.

Note that the space Ω can be decomposed into a series of hyperplanes $\Omega_{[k]}$ ($k = 1, 2, \ldots, n$), i.e.,

$$\Omega = \cup_{k=1}^{n} \Omega_{[k]} \tag{7}$$

As we explain in the following sections, the proposed two-phase algorithm explores the feasible space Ω^F during its first search phase and the limited subspaces of Ω identified by a small number of (promising) hyperplanes $\Omega_{[k]}$ during the second phase. Proposals to explore solution spaces by reference to such hyperplanes were also introduced in [18] where they constitute an instance of more general "exploiting inequalities".

2.2 General procedure

Our algorithm is composed of two search phases that explore different search spaces described in Section 2.1. The first phase examines only feasible solutions of Ω^F to determine promising search regions. For this purpose, it applies a first tabu search procedure (Section 2.4.1) to generate a population POP of np high-quality feasible solutions with possibly different hyperplane dimensions (np is set to 15 in this work). Among those solutions, the best solution s^* is identified and the number k^* of the selected items in s^* is used to define the most promising hyperplane $\Omega_{[k^*]}$ that serves as the starting region around which a thorough examination is performed during TPTEA's second phase.

The second phase explores subspaces of Ω identified by a limited number of hyperplanes $\Omega_{[k]}$ for $k \in [k^* - \Delta_k, k^* + \Delta_k]$ where k^* comes from the first phase and Δ_k is a small integer (set to 1 in this work). This phase is achieved by a combined strategy that uses a specific crossover operator (Section 2.5) to generate new solutions and a second tabu search procedure (Section 2.4.2) to improve each newly generated solution. During the search, the best solution s^* and its k^* value are updated, implying that the search will dynamically visit different hyperplanes of interest. To perform a detailed examination of each particular hyperplane $\Omega_{[k]}$, the second tabu search procedure considers both feasible and infeasible solutions in $\Omega_{[k]}$.

Algorithm 1: Main frame of the two-phase hybrid evolutionary algorithm for the MKP

```
1  Function TPTEA()
   Input: Instance I, time limit t_max
   Output: The best solution s* found
2  begin
      /* First phase: generate np high-quality solutions and
      identify a promising hyperplane k*. Only feasible solutions
      are considered                                              */
3     s* ← InitialSolution(I)                     /* Sections 2.3 */
4     POP ← ∅
5     for i ← 1 to np do
6        s_i ← InitialSolution(I)
7        (s_i, k) ← TabuSearch_1(s_i) /* Section 2.4.1             */
8        POP ← POP ⋃ {s_i}
9        if f(s_i) > f(s*) then
10          s* ← s_i
11          k* ← k
12       end
13    end
      /* Second phase: search around the hyperplane k* identified
      during the first phase (k* can be updated). Both feasible and
      infeasible solutions are considered                         */
14    while time() ≤ t_max do
15       Pick randomly two solutions s_i and s_j from the population POP
16       for k ← (k* − Δ_k) to (k* + Δ_k) do
17          s_off ← Crossover(s_i, s_j, k)  /* Section 2.5            */
18          s ← TabuSearch_2(s_off) /* Section 2.4.2                  */
19          PoolUpdating(s_off, POP) /* Section 2.6                   */
20          if f(s_off) > f(s*) then
21             s* ← s_off          /* Update the best solution found */
22             k* ← k
23          end
24       end
25    end
26    return s*
27 end
```

The general scheme of the proposed TPTEA algorithm is shown in Algorithm 1. The first phase (lines 3–13) of TPTEA uses the first tabu search procedure ($TabuSearch\_1$) to obtain a high-quality initial population. Each of these np (feasible) solutions is first generated by a randomized construction procedure (Section 2.3) and then improved by $TabuSearch\_1$ that only explores feasible solutions in the search space Ω. At the end of the first phase, the best solution s^* and the associated k^* are identified and passed to the second phase.

The second phase is defined by the "while" loop (lines 14–25) and explores a number of hyperplanes $\Omega_{[k]}$ ($k \in [k^* - \Delta_k, k^* + \Delta_k]$). At each "while" iteration, the algorithm first randomly selects two parent solutions s_i and

s_j from the population and then performs a series of operations for each $k \in [k^* - \Delta_k, k^* + \Delta_k]$. Specifically, for each k considered, the algorithm first applies a crossover operator to the parent solutions s_i and s_j to generate an offspring solution s_{off} on the hyperplane $\Omega_{[k]}$. Then the algorithm improves s_{off} by the second tabu search procedure ($TabuSearch\_2$) that limits its search to the given hyperplane $\Omega_{[k]}$ and examines both feasible and infeasible solutions having exactly k selected items. During the search, each time an improved best solution s^* is found, the updated k^* identifies a new promising hyperplane which is examined during the subsequent iterations of the second phase. The "while" loop is repeated until the timeout limit (t_{max}) is reached. Finally, the best solution s^* found is returned as the result of the algorithm.

2.3 Preprocessing procedure of instances and generation of initial solutions

For each instance of the MKP, the items are preprocessed and renumbered as follows. First, for each item j, we compute a surrogate constraint evaluation ratio σ_j [36] as follows.

$$\sigma_j = \frac{p_j}{\sum_{i=1}^{m} \frac{r_{ij}}{b_i}}, \forall j \in \{1, 2, \ldots, n\} \tag{8}$$

This ratio, following the form proposed in [17], utilizes the simple surrogate constraint normalization that divides each constraint through by its constant (right hand side) term. Then, all items are sorted and renumbered in a non-decreasing order according to their surrogate constraint ratios. Finally, the vectors $(p_1, p_2, \ldots, p_n)$, $(b_1, b_2, \ldots, b_m)$ and r_{ij} $(i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n)$ are adjusted according to the new order of the items. The new numbering of items will be used in the whole algorithm. (More advanced forms of surrogate constraints that can be used to form such ratios are introduced in [18] and [21].)

For the generation of initial solutions of the population, we use a randomized procedure to generate np feasible solutions to form the initial population. To obtain a feasible initial solution, the initialization procedure performs a number of iterations. At each iteration, an item is randomly picked from the set of unchecked items (at the beginning, all items are marked unchecked), then the item is considered as being selected if adding it to the solution does not violate any knapsack constraint. Otherwise, the item is dropped. The initialization procedure stops as soon as all items have been checked.

2.4 Solution-based tabu search methods

The proposed algorithm employs two solution-based tabu search procedures as its main optimization components. The first one works on the feasible search space Ω^F using the objective function as its evaluation function, while the second one works on a given subspace $\Omega_{[k]}$ using a penalty-based augmented objective function as its evaluation function. In the next two subsections, we present the two tabu search procedures.

2.4.1 Tabu search method exploring the feasible search space

Algorithm 2: Tabu search procedure exploring feasible search space

```
1  Function TabuSearch_1()
   Input: Initial solution s, objective function f, hash vectors H1, H2, H3 with
          a length of LH, hash functions h1, h2, h3, the maximum number of
          iterations IterMax
   Output: The best solution s* found
2  begin
       /* Initialization of hash vectors                                    */
3      for i ← 0 to LH − 1 do
4          H1[i] ← 0; H2[i] ← 0; H3[i] ← 0;
5      end
6      s* ← s
7      iter ← 0
       /* Main search procedure                                             */
8      while iter ≤ IterMax do
9          Find a best neighbor solution s' in terms of objective function f from
           the current neighborhood N1(s) ∪ N2(s) such that
           H1(h1(s')) ∧ H2(h2(s') ∧ H3(h3(s')) = 0      /* N1(s) and N2(s) are
           defined in Eqs. (9) and (10) */
10         s ← s'                          /* Update the incumbent solution */
11         if f(s) > f(s*) then
12             s* ← s
13         end
           /* Update the hash vectors (i.e., tabu lists) with s             */
14         H1[h1(s)] ← 1; H2[h2(s)] ← 1; H3[h3(s)] ← 1
15         iter ← iter + 1
16     end
17     return s*
18 end
```

The first tabu search procedure $TabuSearch\_1$ (Algorithm 2), which is used by TPTEA during its first search phase, employs the objective function (Eq. (1)) as its evaluation function, and only explores the feasible space Ω^F. This procedure starts from the initialization of the tabu lists (i.e., three hash vectors H_1, H_2, and H_3, see below) and a feasible initial solution, and then performs a number of iterations to improve the initial solution (lines 8–16). At each iteration, a best non-tabu neighbor solution is chosen from the neighborhood to become the new incumbent solution, followed by the updates of the tabu lists (line 14).

For the tabu search methods, the neighborhood structures and the tabu strategy are two most essential components which must be considered with care. Our tabu search method uses two basic neighborhoods: the restricted one-flip neighborhood $N_1(s)$ and the restricted swap neighborhood $N_2(s)$.

The restricted one-flip neighborhood N_1 is defined by the one-flip operator ($Flip$). Specifically, given a solution $s = (x_1, x_2, \dots, x_n)$, an one-flip move $Flip(q)$ changes the value of a variable x_q to its complementary value $1 - x_q$. Given a solution $s = (x_1, x_2, \dots, x_n)$, the neighborhood $N_1(s)$ is composed of all the feasible solutions that can be obtained by applying the one-flip operator to s. Formally, the $N_1(s)$ can be written as follows.

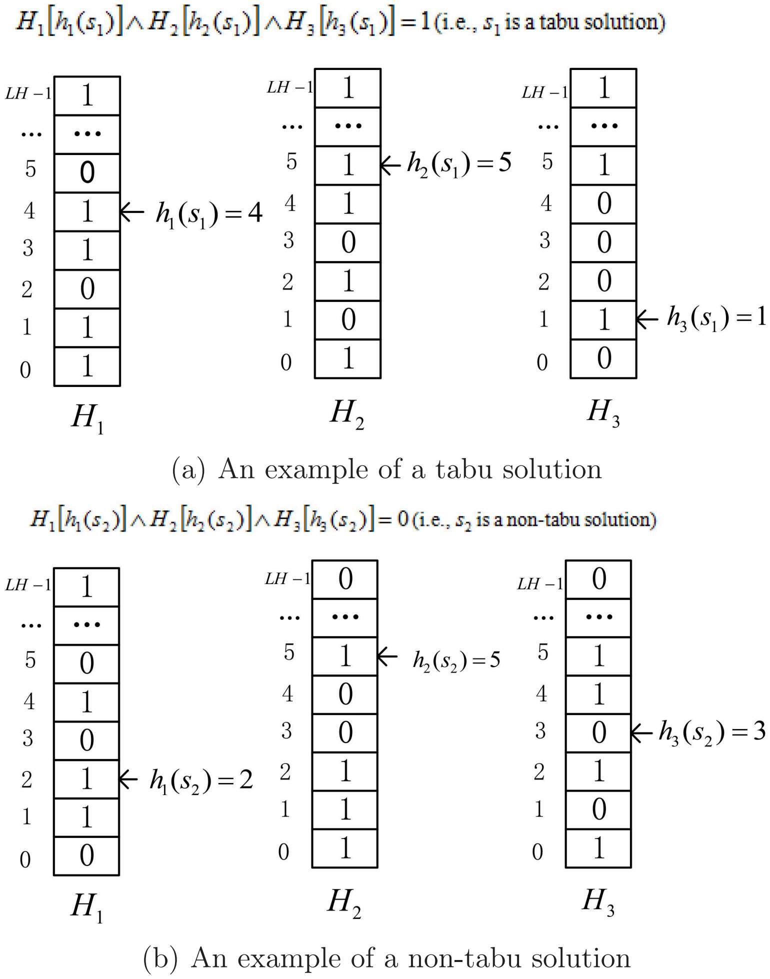
$$N_1(s) = \{s \bigoplus Flip(q) \ : \ \sum_{j=1}^{n} r_{ij}x_j + r_{iq}(1 - x_q) \leq b_i, 1 \leq q \leq n, 1 \leq i \leq m\} \tag{9}$$

In addition, as explained in Section 2.1, a solution $s = (x_1, x_2, \dots, x_n)$ can equivalently be represented by $< S, NS >$, where S represents the set of the selected items and NS represents the set of the unselected items. With this representation of solutions, the restricted swap neighborhood N_2 can be defined by the swap operator $Swap(v, u)$. Given a solution $< S, NS >$, the swap operator $Swap(v, u)$ exchanges the values of variables x_v and x_u to generate a neighboring solution, where $v \in S$ and $u \in NS$. Clearly, the swap operator will lead to a neighborhood whose size is bounded by $O(|S| \times |NS|)$ that is very large for large-scale instances. To speed up the tabu search method, we apply the successive filter candidate list strategy of [22] to two high-quality subsets $X \subset S$ and $Y \subset NS$. Specifically, to define the set X, the variables in S are first sorted in an ascending order according to their surrogate constraint ratios σ in Eq. (8) (see Section 2.3), and then the first $Min\{|S|, \theta \times n\}$ variables are selected to form X, where θ is a parameter that is used to control the size of sets X and Y. Similarly, to construct the set Y, the variables in NS are sorted in a descending order according to their surrogate constraint ratios, and the first $Min\{|NS|, \theta \times n\}$ variables are selected to form Y. Formally, the restricted swap neighborhood $N_2(s)$ can be written as follows.

$$N_2(s) = \{s \bigoplus Swap(v, u) \ : \ v \in X, u \in Y; \sum_{j=1}^{n} r_{ij}x_j + r_{iu} - r_{iv} \leq b_i, 1 \leq i \leq m\} \tag{10}$$

Clearly, the size of $N_2(s)$ is bounded by $\theta^2 n^2$.

As to the tabu strategy, unlike the popular attribute-based approaches, the proposed tabu search method adopts the solution-based tabu strategy that relies on the three hash vectors as well as the associated hash functions to rapidly determine the tabu status of neighbor solutions. To illustrate our tabu strategy, we give in Figure 1 an example of determining the tabu status of candidate solutions, where three hash vectors H_1, H_2, and H_3 with a length of LH are given, and each position of these vectors represents a binary variable that takes the value of 0 or 1. In addition, each hash vector H_t $(t = 1, 2, 3)$ is associated with a hash function h_t which maps a candidate solution in the search space Ω to an index of H_t (i.e., $h_t : x \in \Omega \rightarrow \{0, 1, 2, \ldots, LH - 1\}$).



(a) An example of a tabu solution

(b) An example of a non-tabu solution

Fig. 1. Two illustrative examples for determining the tabu status of the given candidate solution using three hash functions as well as the associated hash vectors.

Using these hash vectors and the associated hash functions, the tabu status of a candidate solution s can be rapidly determined by the following rule. If all $H_t[h_t(s)]$ $(t = 1, 2, 3)$ take 1, then s is determined as a tabu solution. Otherwise, s is determined as a non-tabu solution, as illustrated in Figure 1.

The choice of the hash functions is another important issue for hash-based tabu search methods. Generally, the hash functions should follow the principle that the hash values of candidate solutions can be easily calculated. Following previous studies [7,43,45], we use the following hash functions. Let $s = (x_1, x_2, \ldots, x_n)$ denote a candidate solution where $x_i \in \{0, 1\}$, the hash functions h_t $(t = 1, 2, 3)$ are defined as

$$h_t(s) = (\sum_{i=1}^{n} \lfloor i^{\gamma_t} \rfloor \times x_i) \ mod \ LH \tag{11}$$

where γ_t is a parameter that is used to define the hash function and takes different values for the different hash functions (see Table 1 for its setting), and LH is the length of the hash vectors that is set to 10^7 in this work.

$$h_t(s \oplus M) = \begin{cases} h_t(s) - \lfloor v^{\gamma_t} \rfloor, & \text{for } M = Flip(v) \wedge x_v = 1; \quad (12) \\ h_t(s) + \lfloor u^{\gamma_t} \rfloor, & \text{for } M = Flip(u) \wedge x_u = 0; \quad (13) \\ h_t(s) + (\lfloor u^{\gamma_t} \rfloor - \lfloor v^{\gamma_t} \rfloor), & \text{for } M = swap(v, u); \quad (14) \end{cases}$$

Moreover, for a given solution $s = (x_1, x_2, \ldots, x_n)$, the hash value of a neighbor solution $s \oplus M$ can be easily calculated in $O(1)$ according to Eq. (12–14), where M denotes a *Flip* or *Swap* move.

2.4.2 Tabu search method exploring a given subspace $\Omega_{[k]}$

The second tabu search procedure *TabuSearch_2* (Algorithm 3) works on a given hyperplane $\Omega_{[k]}$ that contains all feasible and infeasible solutions $s = (x_1, x_2, \ldots, x_n)$ with $\sum_{j=1}^{n} x_j = k$. Similar to the first tabu search procedure, *TabuSearch_2* starts from the initialization of the hash vectors (lines 3–5), and then performs a number of iterations to improve the starting solution (lines 8–16). At each iteration, the method first scans the current neighborhood and then selects a best eligible solution in terms of its evaluation function (see below) to replace the current solution. The best feasible solution found, s^*, is updated each time a better feasible solution s is encountered (lines 11–13). Subsequently, the hash vectors are updated accordingly using the new solution (line 14). The method stops when a maximum number of iterations is reached, and the best feasible solution found during the search process is returned as the results of the second tabu search procedure.

TabuSearch_2 uses the same tabu strategy as the first tabu search procedure, while the other components including the neighborhood and evaluation function are described as follows. To ensure an efficient examination of the

Algorithm 3: Tabu search procedure exploring a hyperplane $\Omega_{[k]}$

Function *TabuSearch_2()*
Input: Initial solution s, evaluation function $F(s)$, penalty function $V(s)$, hash vectors H_1, H_2, H_3 with a length of LH, hash functions h_1, h_2, h_3, the maximum number of iterations $IterMax$
Output: The best feasible solution s^* found
begin
  /* Initialization of hash vectors */
  for $i \leftarrow 0$ ***to*** $LH - 1$ **do**
    $H_1[i] \leftarrow 0$; $H_2[i] \leftarrow 0$; $H_3[i] \leftarrow 0$;
  end
  $s^* \leftarrow s$
  $iter \leftarrow 0$
  /* Main search procedure */
  while $iter \leq IterMax$ **do**
    Find a best neighbor solution $s^{'}$ in terms of the evaluation function F from the current neighborhood $N_3(s)$ such that $H_1(h_1(s^{'})) \wedge H_2(h_2(s^{'}) \wedge H_3(h_3(s^{'})) = 0$ /* $N_3(s)$ is defined in Eq. (15) */
    $s \leftarrow s^{'}$ /* Update the incumbent solution */
    if $F(s) > F(s^*) \wedge V(s) = 0$ **then**
      $s^* \leftarrow s$
    end
    /* Update the hash vectors (i.e., tabu lists) with s */
    $H_1[h_1(s)] \leftarrow 1$; $H_2[h_2(s)] \leftarrow 1$; $H_3[h_3(s)] \leftarrow 1$
    $iter \leftarrow iter + 1$
  end
  return s^*
end

solutions on the fixed hyperplane $\Omega_{[k]}$, $TabuSearch\_2$ uses a reduced swap neighborhood N_3 that is defined as

$$N_3(s) = \{s \bigoplus Swap(v, u) \ : \ v \in X, u \in Y; f(s \bigoplus Swap(v, u)) > f(s^*)\} \tag{15}$$

where s^* represents the best feasible solution found so far in the current tabu search process, and the subsets X and Y are the same as in Eq.(10). Note that the strategy of reducing the size of N_3 is similar to that used in [39], and this strategy allows the search process to focus on the improving solutions, and significantly decreases the number of neighbor solutions to be examined.

Since $TabuSearch\_2$ visits both feasible and infeasible solutions of $\Omega_{[k]}$, it employs an augmented evaluation function F that integrates a penalty component V to evaluate the candidate solutions. The penalty $V(s)$, which is used to assess the degree of constraint violation of a candidate solution $s = (x_1, x_2, \ldots, x_n)$, is defined as follows.

$$V(s) = \sum_{i=1}^{m} Max\{0, \sum_{j=1}^{n} r_{ij}x_j - b_i\} \tag{16}$$

As such, a smaller $V(s)$ value indicates less constraint violation and a solution with $V(s) = 0$ represents a feasible solution.

The augmented evaluation function $F(s)$ combines the objective function $f(s)$ in Eq.(1) and the penalty function $V(s)$ as follows.

$$F(s) = \sum_{j=1}^{n} p_j x_j + \lambda \sum_{i=1}^{m} Max\{0, \sum_{j=1}^{n} r_{ij}x_j - b_i\} \tag{17}$$

where λ is a scaling factor that is set to -10^2 in this work. Given two solutions s_1 and s_2 in $\Omega_{[k]}$, s_1 is considered to be better than s_2 if $F(s_1) > F(s_2)$.

2.5 Hyperplane-constrained crossover operator

In evolutionary algorithms, the crossover operator is another important ingredient [26]. In the TPTEA algorithm, we adopt a hyperplane-constrained crossover operator (Algorithm 4), which combines two solutions to produce a new solution in $\Omega_{[k]}$. Our crossover operator is adapted from the popular uniform crossover operator and ensures that the new solution belongs to $\Omega_{[k]}$ (i.e., containing exactly k selected items). Specifically, the crossover operator works as follows. Given two parent solutions $s_a = (x_1^a, x_2^a, \ldots, x_n^a)$ and $s_b = (x_1^b, x_2^b, \ldots, x_n^b)$ as well as a positive integer k, the component x_i^{off} $(i = 1, 2, \ldots, n)$ of the offspring solution s_{off} takes randomly the value of x_i^a or x_i^b with equal probability. Then we consider the following three situations. First, if $\sum_{j=1}^{n} x_j^{off} = k$, the offspring solution contains exactly k selected items (i.e., $s_{off} \in \Omega_{[k]}$) and we are done. Second, if $\sum_{j=1}^{n} x_j^{off} < k$, then we change the values of the last $k - \sum_{j=1}^{n} x_j^{off}$ variables taking the value of 0 to 1. Second, if $\sum_{j=1}^{n} x_j^{off} > k$, we change the values of the first $\sum_{j=1}^{n} x_j^{off} - k$ variables taking 1 to 0. It is worth noting that a variable with a small index has a weak surrogate constraint ratio, as shown in Section 2.3. We observe a possible variation of the foregoing procedure. As noted in [20], uniform crossover produces solution combinations that are a special instance of combinations provided earlier by the evolutionary scatter search approach [19] and scatter search also includes the possibility of other weightings which give rise to a probabilistic determination of values assigned to offspring. Similarly, our present approach can be generalized to utilize such probabilistic determinations of values assigned to variables before applying the surrogate constraint ratio indexing to compel exactly k variables to be 1.

Algorithm 4: The hyperplane crossover operator

Function *Crossover()*

Input: Two selected solutions $s_a = (x_1^a, x_2^a, \ldots, x_n^a)$ and $s_b = (x_1^b, x_2^b, \ldots, x_n^b)$, the number of variables taking 1 (k)

Output: A offspring solution $s_{off} = (x_1^{off}, x_2^{off}, \ldots, x_n^{off})$ in the $\Omega_{[k]}$

```
for i ← 1 to n do
    r ← rand(0,1)            /* rand(0,1) is a random number in (0,1) */
    if r < 0.5 then
        x_i^off ← x_i^a
    end
    else
        x_i^off ← x_i^b
    end
end
counter ← the number of variables taking 1 in s_off
if counter > k then
    for i ← 1 to n do
        if x_i^off = 1 then
            x_i^off ← 0
            counter ← counter − 1
            if counter = k then
                break
            end
        end
    end
end
if counter < k then
    for i ← n to 1 do
        if x_i^off = 0 then
            x_i^off ← 1
            counter ← counter + 1
            if counter = k then
                break
            end
        end
    end
end
return s_off = (x_1^off, x_2^off, ..., x_n^off)
```

2.6 Diversity-based population updating rule

To ensure a healthy diversity of the population POP, we employ in this work a diversity-based population updating rule [31,35,46] that takes into account the quality of solutions and the population diversity. For this purpose, we first introduce two definitions.

Definition 1 (Distance between a solution and its population). Given a solution s_i and the population $POP = \{s_1, s_2, \ldots, s_{np}\}$, the distance $D(s_i)$ be-

Algorithm 5: Pseudo-code of pool updating method

```
1 Function PoolUpdating()
  Input: Population POP = {s_1, s_2, ..., s_np}, offspring s_off
  Output: Updated population POP
2 begin
3   | POP ← POP ∪ {s_off}
4   | for i ← 1 to np + 1 do
5   |   | Calculate Score(s_i) of s_i according to Eq. (19)
6   | end
7   | s_worst ← argmin{Score(s_i)|i = 1, 2, ..., np + 1}
8   | POP ← POP \ {s_worst}
9 end
```

tween s_i and POP is defined as follows:

$$D(s_i) = Min\{distance(s_i, s_j) \ : \ s_j \in POP, s_j \neq s_i\} \tag{18}$$

where $distance(s_i, s_j)$ represents the Hamming distance between s_i and s_j.

Definition 2 (Goodness score of a solution in the population). The goodness score $Score(s_i)$ of a solution s_i is defined by its objective function value as well as its distance to the population as follows:

$$Score(s_i) = \beta \times \frac{f(s_i) - f_{min}}{f_{max} - f_{min}} + (1 - \beta) \times \frac{D(s_i) - D_{min}}{D_{max} - D_{min}} \tag{19}$$

where f_{max} and f_{min} denote respectively the maximum and minimum objective values of the solutions in the population, D_{max} and D_{min} are respectively the maximum and minimum distances between a solution to the population, and β is a parameter that is empirically set to 0.7 in this work.

The population updating rule works as follows (Algorithm 5). When an offspring solution s_{off} is generated by the crossover operator and improved by the tabu search method in Algorithm 3, s_{off} is first added into the population (line 3), and then the goodness score of each individual in the population is calculated according to Eq.(19) (line 5). Finally, the worst individual in terms of the goodness score is deleted from the population (lines 7–8).

3 Experimental results and comparisons

We now assess the proposed algorithm by performing extensive computational experiments on the benchmark instances commonly used in the literature and making comparisons with several state-of-the-art algorithms.

3.1 Benchmark instances

We tested the TPTEA algorithm on the 281 popular benchmark instances whose main characteristics are described as follows.

- OR-Library instances: These instances were generated by Chu and Beasley in [10] and are available at `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html`. For these instances, the number of variables n is set to 100, 250 and 500, and the number of constraints m is set to 5, 10, and 30. For each (n, m) combination, 30 instances were generated. Specifically, r_{ij} $(1 \le i \le m,\ 1 \le j \le n)$ are integers uniformly and randomly generated in $[0, 1000]$, $b_i = \alpha \times \sum_{j=1}^{n} r_{ij}$ $(1 \le i \le m)$, where α is called the tightness ratio and set to 0.25, 0.5, and 0.75 for the first 10 instances, the next 10 instances and the remaining 10 instances, respectively. The p_j values are set as follows: $p_j = \sum_{i=1}^{m} r_{ij}/m + 500q_j$ $(1 \le j \le n)$, where q_j is a real number generated uniformly and randomly in $[0, 1]$ [10]. It is worth noting that the optimum solution has been proven in previous work for most instances of this set [6,32,39,40].
- MK_GK instances: This set contains 11 instances with $m \in \{15, 25, 50, 100\}$ and $n \in \{100, 200, 500, 1000, 1500, 2500\}$, which were proposed by Glover and Kochenberger. We make these instances available at `http://www.info.univ-angers.fr/pub/hao/mkp.html`, since they are no more accessible on the initial website `http://hces.bus.olemiss.edu/tools.html` [39].

3.2 Parameter settings and experimental protocol

Table 1
Settings of important parameters

Parameters	Section	Description	Values
$IterMax$	2.4	maximum number of iterations for the tabu search methods	$\{5 \times 10^3, 5 \times 10^4\}$
θ	2.4.1	parameter used in constructing the reduced neighborhoods	$\{0.1 \times rand(0,1) + 0.15, 0.15, 0.35\}$
β	2.6	parameter used in the population updating rule	0.7
Δ_k	2.2	parameter used to determine the proper hyperplane	1
γ_1	2.4.1	parameter used in the hash function	1.3
γ_2	2.4.1	parameter used in the hash function	1.8
γ_3	2.4.1	parameter used in the hash function	2.0

The proposed TPTEA algorithm requires several parameters, whose values are empirically set (see Table 1). The parameter $IterMax$ that defines the maximum number of iterations of the tabu search methods is set according to the size of instances as well as the type of the tabu search methods. Specifically, for the first tabu search method $IterMax$ is to set to 5×10^3 for all the instances. For the second tabu search method and the instances with $n \ge 1000$, $IterMax$ is also set to 5×10^3, while it is set to 5×10^4 for the remaining instances. For the parameter θ that is used to control the size of neighborhoods, its values are respectively 0.35, $0.15+0.1\times rand(0,1)$ and 0.15 for the instances with $n \leq 250$, $n = 500$ and $n \geq 1000$, respectively. For the parameters Δ_k and β, their values are respectively set to 1 and 0.7 according to the sensitivity analysis shown in Section 4.

TPTEA was programmed in C and compiled using the g++ compiler with the -O3 option. The computational experiments were performed on a computer with an Intel E5-2670 processor (2.5 GHz and 2G RAM), running the Linux operating system. For the DIMACS machine benchmark procedure [1], the processor requires respectively 0.19, 1.17, and 4.54 seconds to solve the graphs r300.5, r400.5, r500.5. Following recent studies like [24] and due to the stochastic character of the proposed algorithm, our algorithm was run 30 times for each instance. The timeout limit t_{max} was set to 0.02 hours, 1 hour, 2 hours, and 3 hours for the instances with $n \leq 150$, $200 \leq n \leq 250$, $n = 500$, and $n \geq 1000$, respectively.

To evaluate the performance of the proposed algorithm, eight state-of-the-art heuristic algorithms in the literature are used as the main reference algorithms, including the popular genetic algorithm (GA) [10] (as a base reference), the filter-and-fan heuristic (F&F) [27], two self-adaptive check and repair operator-based particle swarm optimization algorithms (SACRO-BPSO) [9], the hybrid quantum particle swarm optimization algorithm (QPSO*) [24], the tabu search-based PSO algorithm (TEPSOq) [28], the critical event tabu search method (TS_GK) [23], and the hybrid method using linear programming and tabu search (LP+TS) [39]. These reference algorithms are among the best performing heuristic algorithms currently available in the literature.

3.3 Computational results and comparisons

Our results on the 281 benchmark instances according to the above experimental protocol are summarized in Tables 2–11. In Tables 2–10, the first two columns show, for each instance, the name and the best known result (or the optimal result when it is known). The results obtained by our TPTEA algorithm are reported in the last four columns, including the best objective value (f_{best}) over 30 independent runs, the average objective value (f_{avg}), the standard deviation ($Std.$) of objective values, and the average CPU time ($t_{avg}(s)$) in seconds to reach its final objective value. The other columns show the best known results (f_{best}) produced by the reference algorithms in the literature. The best objective values obtained by the compared algorithms are indicated in bold if they match or improve the best known results reported in the literature.

[1] dmclique, `ftp://dimacs.rutgers.edu/pub/dsj/clique`

Table 2
Computational results and comparisons on the small instances with $n = 100$ and $m = 5$.

Problem		GA	F&F	SACRO-BPSO(1)	SACRO-BPSO(2)	QPSO*	TPTEA (this work)			
Instance	Optimum	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	$Std.$	$t_{avg}(s)$
5.100.0	**24381**	**24381**	**24381**	24343	24343	**24381**	**24381**	24381.00	0.0	0.6
5.100.1	**24274**	**24274**	**24274**	**24274**	**24274**	**24274**	**24274**	24274.00	0.0	0.4
5.100.2	**23551**	**23551**	**23551**	23538	23538	**23551**	**23551**	23551.00	0.0	0.4
5.100.3	**23534**	**23534**	**23534**	23527	23527	**23534**	**23534**	23534.00	0.0	1.5
5.100.4	**23991**	**23991**	**23991**	**23991**	23966	**23991**	**23991**	23991.00	0.0	0.8
5.100.5	**24613**	**24613**	**24613**	24601	24601	**24613**	**24613**	24613.00	0.0	0.4
5.100.6	**25591**	**25591**	**25591**	**25591**	**25591**	**25591**	**25591**	25591.00	0.0	0.4
5.100.7	**23410**	**23410**	**23410**	**23410**	**23410**	**23410**	**23410**	23410.00	0.0	0.3
5.100.8	**24216**	**24216**	**24216**	24204	**24216**	**24216**	**24216**	24216.00	0.0	1.7
5.100.9	**24411**	**24411**	**24411**	24399	**24411**	**24411**	**24411**	24411.00	0.0	0.5
5.100.10	**42757**	**42757**	**42757**	42705	42705	**42757**	**42757**	42757.00	0.0	1.1
5.100.11	**42545**	**42545**	**42545**	42494	42471	**42545**	**42545**	42545.00	0.0	9.3
5.100.12	**41968**	**41968**	**41968**	41959	41959	**41968**	**41968**	41968.00	0.0	0.9
5.100.13	**45090**	**45090**	**45090**	**45090**	**45090**	**45090**	**45090**	45090.00	0.0	10.8
5.100.14	**42218**	**42218**	**42218**	**42218**	**42218**	**42218**	**42218**	42218.00	0.0	0.5
5.100.15	**42927**	**42927**	**42927**	**42927**	**42927**	**42927**	**42927**	42927.00	0.0	0.5
5.100.16	**42009**	**42009**	**42009**	**42009**	**42009**	**42009**	**42009**	42009.00	0.0	0.4
5.100.17	**45020**	**45020**	**45020**	45010	**45020**	**45020**	**45020**	45020.00	0.0	0.5
5.100.18	**43441**	**43441**	**43441**	**43441**	43381	**43441**	**43441**	43441.00	0.0	1.7
5.100.19	**44554**	**44554**	**44554**	**44554**	44529	**44554**	**44554**	44554.00	0.0	2.7
5.100.20	**59822**	**59822**	**59822**	**59822**	**59822**	**59822**	**59822**	59822.00	0.0	0.3
5.100.21	**62081**	**62081**	**62081**	**62081**	**62081**	**62081**	**62081**	62081.00	0.0	0.6
5.100.22	**59802**	**59802**	**59802**	**59802**	59754	**59802**	**59802**	59802.00	0.0	0.3
5.100.23	**60479**	**60479**	**60479**	60478	60478	**60479**	**60479**	60479.00	0.0	0.3
5.100.24	**61091**	**61091**	**61091**	61055	61079	**61091**	**61091**	61091.00	0.0	0.4
5.100.25	**58959**	**58959**	**58959**	**58959**	58937	**58959**	**58959**	58959.00	0.0	0.4
5.100.26	**61538**	**61538**	**61538**	**61538**	**61538**	**61538**	**61538**	61538.00	0.0	0.3
5.100.27	**61520**	**61520**	**61520**	61489	**61520**	**61520**	**61520**	61520.00	0.0	0.3
5.100.28	**59453**	**59453**	**59453**	**59453**	**59453**	**59453**	**59453**	59453.00	0.0	0.3
5.100.29	**59965**	**59965**	**59965**	59960	59960	**59965**	**59965**	59965.00	0.0	0.6
Avg.	42640.4	**42640.4**	**42640.4**	42630.7	42626.9	**42640.4**	**42640.4**	42640.4	0.0	1.3
#Best		30	30	16	16	30	30			
p-value	1.0	1.0	1.0	1.80e-4	1.10e-4	1.0				

Table 3
Computational results and comparisons on the small instances with $n = 100$ and $m = 10$.

Problem		GA	F&F	SACRO-BPSO(1)	SACRO-BPSO(2)	QPSO*	TPTEA (this work)			
Instance	Optimum	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	$Std.$	$t_{avg}(s)$
10.100.0	**23064**	**23064**	**23064**	**23064**	**23064**	**23064**	**23064**	23064.00	0.0	1.5
10.100.1	**22801**	**22801**	**22801**	22739	22750	**22801**	**22801**	22801.00	0.0	1.4
10.100.2	**22131**	**22131**	**22131**	**22131**	**22131**	**22131**	**22131**	22131.00	0.0	1.1
10.100.3	**22772**	**22772**	**22772**	**22772**	22717	**22772**	**22772**	22772.00	0.0	19.7
10.100.4	**22751**	**22751**	**22751**	**22751**	**22751**	**22751**	**22751**	22751.00	0.0	0.4
10.100.5	**22777**	**22777**	22739	22725	22716	**22777**	**22777**	22777.00	0.0	5.9
10.100.6	**21875**	**21875**	**21875**	**21875**	**21875**	**21875**	**21875**	21875.00	0.0	0.5
10.100.7	**22635**	**22635**	**22635**	22551	22542	**22635**	**22635**	22635.00	0.0	5.0
10.100.8	**22511**	**22511**	**22511**	**22511**	22438	**22511**	**22511**	22511.00	0.0	0.5
10.100.9	**22702**	**22702**	**22702**	**22702**	**22702**	**22702**	**22702**	22702.00	0.0	0.5
10.100.10	**41395**	**41395**	**41395**	**41395**	41388	**41395**	**41395**	41395.00	0.0	16.6
10.100.11	**42344**	**42344**	**42344**	**42344**	**42344**	**42344**	**42344**	42344.00	0.0	0.9
10.100.12	**42401**	**42401**	**42401**	42350	42350	**42401**	**42401**	42401.00	0.0	13.1
10.100.13	**45624**	**45624**	**45624**	45585	45511	**45624**	**45624**	45624.00	0.0	21.5
10.100.14	**41884**	**41884**	**41884**	41799	41833	**41884**	**41884**	41884.00	0.0	6.5
10.100.15	**42995**	**42995**	**42995**	**42995**	**42995**	**42995**	**42995**	42995.00	0.0	0.6
10.100.16	**43574**	43559	**43574**	43497	43517	43553	**43574**	43574.00	0.0	16.2
10.100.17	**42970**	**42970**	**42970**	**42970**	**42970**	**42970**	**42970**	42970.00	0.0	15.3
10.100.18	**42212**	**42212**	**42212**	**42212**	**42212**	**42212**	**42212**	42212.00	0.0	0.5
10.100.19	**41207**	**41207**	**41207**	41123	41134	**41207**	**41207**	41207.00	0.0	18.8
10.100.20	**57375**	**57375**	**57375**	**57375**	**57375**	**57375**	**57375**	57375.00	0.0	0.4
10.100.21	**58978**	**58978**	**58978**	58922	**58978**	**58978**	**58978**	58978.00	0.0	1.0
10.100.22	**58391**	**58391**	**58391**	**58391**	**58391**	**58391**	**58391**	58391.00	0.0	0.5
10.100.23	**61966**	**61966**	**61966**	**61966**	**61966**	**61966**	**61966**	61966.00	0.0	1.7
10.100.24	**60803**	**60803**	**60803**	**60803**	**60803**	**60803**	**60803**	60803.00	0.0	0.5
10.100.25	**61437**	**61437**	**61437**	61368	61368	**61437**	**61437**	61437.00	0.0	5.7
10.100.26	**56377**	**56377**	**56377**	**56377**	**56377**	**56377**	**56377**	56377.00	0.0	7.9
10.100.27	**59391**	**59391**	**59391**	59332	**59391**	**59391**	**59391**	59391.00	0.0	0.4
10.100.28	**60205**	**60205**	**60205**	**60205**	**60205**	**60205**	**60205**	60205.00	0.0	0.6
10.100.29	**60633**	**60633**	**60633**	60629	60629	**60633**	**60633**	60633.00	0.0	0.5
Avg.	41606.0	41605.5	41604.8	41582.0	41580.8	41605.3	**41606.0**	41606.0	0.0	5.5
#Best		29	29	18	17	29	30			
p-value	1.0	0.32	0.32	5.30e-4	3.10e-4	0.32				

Table 4
Computational results and comparisons on the small instances with $n = 100$ and $m = 30$.

Problem		GA	F&F	QPSO*	TPTEA (this work)			
Instance	Optimum	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	*Std.*	$t_{avg}(s)$
30.100.0	**21946**	**21946**	**21946**	**21946**	**21946**	21946.00	0.0	7.5
30.100.1	**21716**	**21716**	**21716**	**21716**	**21716**	21716.00	0.0	17.7
30.100.2	**20754**	**20754**	**20754**	**20754**	**20754**	20754.00	0.0	10.4
30.100.3	**21464**	**21464**	**21464**	**21464**	**21464**	21464.00	0.0	12.9
30.100.4	**21844**	21814	**21844**	**21844**	**21844**	21844.00	0.0	15.6
30.100.5	**22176**	**22176**	**22176**	**22176**	**22176**	22176.00	0.0	0.7
30.100.6	**21799**	**21799**	**21799**	21772	**21799**	21799.00	0.0	19.5
30.100.7	**21397**	**21397**	**21397**	**21397**	**21397**	21397.00	0.0	15.8
30.100.8	**22525**	22493	22493	**22525**	**22525**	22525.00	0.0	15.0
30.100.9	**20983**	**20983**	**20983**	**20983**	**20983**	20983.00	0.0	0.8
30.100.10	**40767**	**40767**	**40767**	**40767**	**40767**	40767.00	0.0	22.5
30.100.11	**41308**	41304	41304	**41308**	**41308**	41308.00	0.0	19.2
30.100.12	**41630**	41560	**41630**	**41630**	**41630**	41630.00	0.0	28.5
30.100.13	**41041**	**41041**	**41041**	**41041**	**41041**	41041.00	0.0	22.1
30.100.14	**40889**	40872	**40889**	40872	**40889**	40889.00	0.0	21.6
30.100.15	**41058**	**41058**	**41058**	**41058**	**41058**	41058.00	0.0	0.9
30.100.16	**41062**	**41062**	**41062**	**41062**	**41062**	41062.00	0.0	11.3
30.100.17	**42719**	**42719**	**42719**	**42719**	**42719**	42719.00	0.0	19.4
30.100.18	**42230**	**42230**	**42230**	**42230**	**42230**	42230.00	0.0	1.2
30.100.19	**41700**	**41700**	**41700**	**41700**	**41700**	41700.00	0.0	14.3
30.100.20	**57494**	**57494**	**57494**	**57494**	**57494**	57494.00	0.0	0.6
30.100.21	**60027**	**60027**	**60027**	**60027**	**60027**	60027.00	0.0	1.4
30.100.22	**58052**	58025	**58052**	**58052**	**58052**	58052.00	0.0	18.2
30.100.23	**60776**	**60776**	**60776**	**60776**	**60776**	60776.00	0.0	4.5
30.100.24	**58884**	**58884**	**58884**	**58884**	**58884**	58884.00	0.0	4.9
30.100.25	**60011**	**60011**	**60011**	**60011**	**60011**	60011.00	0.0	2.3
30.100.26	**58132**	**58132**	58104	**58132**	**58132**	58132.00	0.0	0.7
30.100.27	**59064**	**59064**	**59064**	**59064**	**59064**	59064.00	0.0	0.7
30.100.28	**58975**	**58975**	**58975**	**58975**	**58975**	58975.00	0.0	19.6
30.100.29	**60603**	**60603**	**60603**	60593	**60603**	60603.00	0.0	2.3
Avg.	40767.5	40761.5	40765.4	40765.7	**40767.5**	40767.5	0.0	11.1
#Best		25	27	27	30			
p-value	1.0	1.43e-2	8.0e-2	8.0e-2				

Table 5
Computational results and comparisons on the instances with $n = 250$ and $m = 5$.

Problem		GA	F&F	QPSO*	TPTEA (this work)			
Instance	Optimum	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	*Std.*	$t_{avg}(s)$
5.250.0	**59312**	**59312**	**59312**	**59312**	**59312**	59312.00	0.00	67
5.250.1	**61472**	**61472**	61468	**61472**	**61472**	61472.00	0.00	252
5.250.2	**62130**	**62130**	**62130**	**62130**	**62130**	62130.00	0.00	83
5.250.3	**59463**	59446	59436	59427	**59463**	59462.33	2.49	1017
5.250.4	**58951**	**58951**	**58951**	**58951**	**58951**	58951.00	0.00	99
5.250.5	**60077**	60056	60062	**60077**	**60077**	60069.50	7.50	742
5.250.6	**60414**	**60414**	**60414**	**60414**	**60414**	60414.00	0.00	97
5.250.7	**61472**	**61472**	61454	**61472**	**61472**	61472.00	0.00	153
5.250.8	**61885**	**61885**	**61885**	**61885**	**61885**	61885.00	0.00	85
5.250.9	**58959**	**58959**	**58959**	**58959**	**58959**	58959.00	0.00	79
5.250.10	**109109**	**109109**	**109109**	109066	**109109**	109109.00	0.00	289
5.250.11	**109841**	**109841**	**109841**	**109841**	**109841**	109841.00	0.00	118
5.250.12	**108508**	108489	**108508**	**108508**	**108508**	108508.00	0.00	107
5.250.13	**109383**	**109383**	**109383**	109356	**109383**	109383.00	0.00	145
5.250.14	**110720**	**110720**	**110720**	**110720**	**110720**	110720.00	0.00	611
5.250.15	**110256**	**110256**	**110256**	**110256**	**110256**	110256.00	0.00	258
5.250.16	**109040**	109016	**109040**	**109040**	**109040**	109040.00	0.00	115
5.250.17	**109042**	109037	109016	**109042**	**109042**	109042.00	0.00	103
5.250.18	**109971**	109957	109957	**109971**	**109971**	109971.00	0.00	212
5.250.19	**107058**	107038	**107058**	**107058**	**107058**	107058.00	0.00	211
5.250.20	**149665**	149659	149659	**149665**	**149665**	149665.00	0.00	250
5.250.21	**155944**	155940	**155944**	**155944**	**155944**	155943.87	0.72	61
5.250.22	**149334**	149316	**149334**	**149334**	**149334**	149334.00	0.00	119
5.250.23	**152130**	**152130**	**152130**	**152130**	**152130**	152130.00	0.00	56
5.250.24	**150353**	**150353**	**150353**	**150353**	**150353**	150353.00	0.00	58
5.250.25	**150045**	**150045**	**150045**	**150045**	**150045**	150045.00	0.00	42
5.250.26	**148607**	**148607**	**148607**	**148607**	**148607**	148607.00	0.00	36
5.250.27	**149782**	149772	**149782**	149772	**149782**	149782.00	0.00	52
5.250.28	**155075**	**155075**	**155075**	155057	**155075**	155075.00	0.00	42
5.250.29	**154668**	154662	**154668**	**154668**	**154668**	154668.00	0.00	118
Avg.	107088.9	107083.4	107085.2	107084.4	**107088.9**	107088.6	0.4	189.2
#Best		18	23	25	30			
p-value	1.0	5.32e-4	8.15e-3	2.53e-2				

Table 6
Computational results and comparisons on the instances with $n = 250$ and $m = 10$.

Problem		GA	F&F	TEPSOq	QPSO*	TPTEA (this work)			
Instance	Optimum	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	*Std.*	$t_{avg}(s)$
10.250.0	**59187**	**59187**	59164	**59187**	59182	**59187**	59187.00	0.00	195
10.250.1	**58781**	58662	58693	**58781**	**58781**	**58781**	58743.13	35.42	715
10.250.2	**58097**	58094	58094	**58097**	**58097**	**58097**	58097.00	0.00	190
10.250.3	**61000**	**61000**	60972	60662	**61000**	**61000**	60998.57	4.55	839
10.250.4	**58092**	**58092**	**58092**	**58092**	**58092**	**58092**	58090.57	5.36	822
10.250.5	**58824**	58803	**58824**	58549	**58824**	**58824**	58822.60	5.24	462
10.250.6	**58704**	58607	58632	58350	58606	**58704**	58704.00	0.00	385
10.250.7	**58936**	58917	58917	57902	58902	**58936**	58932.10	2.47	733
10.250.8	**59387**	59384	59381	**59387**	59372	**59387**	59387.00	0.00	102
10.250.9	**59208**	59193	**59208**	**59208**	**59208**	**59208**	59208.00	0.00	327
10.250.10	**110913**	110863	110889	**110913**	110857	**110913**	110913.00	0.00	371
10.250.11	**108717**	108659	108702	108713	108687	**108717**	108717.00	0.00	529
10.250.12	**108932**	**108932**	108922	108491	108891	**108932**	108932.00	0.00	77
10.250.13	**110086**	110037	110059	**110086**	**110086**	**110086**	110086.00	0.00	1071
10.250.14	**108485**	108423	**108485**	108225	**108485**	**108485**	108485.00	0.00	129
10.250.15	**110845**	110841	110841	110257	**110845**	**110845**	110843.67	1.89	1064
10.250.16	**106077**	106075	106075	**106077**	106047	**106077**	106075.73	0.96	239
10.250.17	**106686**	**106686**	106685	106455	**106686**	**106686**	106686.00	0.00	563
10.250.18	**109829**	109825	109822	109225	109788	**109829**	109827.40	1.96	845
10.250.19	**106723**	**106723**	**106723**	**106723**	**106723**	**106723**	106723.00	0.00	81
10.250.20	**151809**	151790	151790	151194	151779	**151809**	151809.00	0.00	177
10.250.21	**148772**	**148772**	**148772**	**148772**	**148772**	**148772**	148772.00	0.00	25
10.250.22	**151909**	151900	**151909**	151858	**151909**	**151909**	151909.00	0.00	86
10.250.23	**151324**	151275	151281	**151324**	151281	**151324**	151324.00	0.00	629
10.250.24	**151966**	151948	**151966**	151372	**151966**	**151966**	151961.80	7.61	414
10.250.25	**152109**	**152109**	**152109**	152007	**152109**	**152109**	152109.00	0.00	51
10.250.26	**153131**	**153131**	**153131**	153046	**153131**	**153131**	153131.00	0.00	36
10.250.27	**153578**	153520	153533	**153578**	153529	**153578**	153578.00	0.00	96
10.250.28	**149160**	149155	**149160**	**149160**	**149160**	**149160**	149160.00	0.00	59
10.250.29	**149704**	**149704**	149688	149637	149646	**149704**	149704.00	0.00	56
Avg.	106365.7	106343.6	106350.6	106177.6	106348.0	**106365.7**	106363.9	2.2	379.0
#Best		10	11	14	17	30			
p-value	1.0	7.74e-6	1.31e-5	6.33e-5	3.12e-4				

Table 7
Computational results and comparisons on the instances with $n = 250$ and $m = 30$.

Problem		GA	F&F	QPSO*	TPTEA (this work)			
Instance	Best Known	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	*Std.*	$t_{avg}(s)$
30.250.0	**56842**	56693	56796	56796	56824	56824.00	0.00	131
30.250.1	**58520**	58318	58333	58302	**58520**	58520.00	0.00	216
30.250.2	**56614**	56553	56553	**56614**	**56614**	56614.00	0.00	216
30.250.3	**56930**	56863	**56930**	**56930**	**56930**	56930.00	0.00	91
30.250.4	**56629**	**56629**	**56629**	**56629**	**56629**	56629.00	0.00	74
30.250.5	**57205**	57119	57149	57146	**57205**	57205.00	0.00	374
30.250.6	**56357**	56292	56263	56303	**56357**	56333.40	26.79	1155
30.250.7	**56457**	56403	**56457**	56392	**56457**	56457.00	0.00	103
30.250.8	**57447**	57442	57373	**57447**	<u>57474</u>	57458.90	15.21	971
30.250.9	**56447**	**56447**	**56447**	**56447**	**56447**	56447.00	0.00	99
30.250.10	**107770**	107689	107735	107703	**107770**	107763.10	8.60	1034
30.250.11	**108392**	108338	108338	108338	**108392**	108387.23	6.26	438
30.250.12	**106442**	106385	106415	**106442**	**106442**	106439.60	3.67	587
30.250.13	**106876**	106796	106832	106851	**106876**	106876.00	0.00	205
30.250.14	**107414**	107396	**107414**	107382	**107414**	107414.00	0.00	230
30.250.15	**107271**	107246	**107271**	**107271**	**107271**	107271.00	0.00	294
30.250.16	**106372**	106308	106277	106248	**106372**	106371.77	1.26	682
30.250.17	**104032**	103993	104003	103988	**104032**	104019.00	8.03	497
30.250.18	**106856**	106835	106835	**106856**	**106856**	106852.50	7.83	322
30.250.19	**105780**	105751	105742	105751	**105780**	105779.17	4.49	441
30.250.20	**150163**	150083	150138	150096	**150163**	150163.00	0.00	457
30.250.21	**149958**	149907	**149958**	149958	**149958**	149958.00	0.00	101
30.250.22	**153007**	152993	**153007**	**153007**	**153007**	153007.00	0.00	131
30.250.23	**153234**	153169	153182	**153234**	**153234**	153234.00	0.00	84
30.250.24	**150287**	**150287**	**150287**	**150287**	**150287**	150287.00	0.00	51
30.250.25	**148574**	148544	148549	148544	**148574**	148574.00	0.00	77
30.250.26	**147477**	147471	147455	147471	**147477**	147477.00	0.00	79
30.250.27	**152912**	152841	152841	152835	**152912**	152912.00	0.00	71
30.250.28	**149570**	149568	**149570**	**149570**	**149570**	149570.00	0.00	61
30.250.29	**149668**	149572	149587	**149668**	**149668**	149668.00	0.00	742
Avg.	104716.8	104664.4	104678.9	104683.5	**104717.1**	104714.7	2.7	333.8
#Best		3	9	12	29			
p-value	5.60e-1	1.03e-7	7.74e-6	3.73e-5				

Table 8
Computational results and comparisons on the large instances with $n = 500$ and $m = 5$. The entries marked by "*" imply that the corresponding results are not available.

Problem		GA	F&F	SACRO-BPSO(2)	LP+TS	QPSO*	TPTEA (this work)			
Instance	Optimum	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	*Std.*	$t_{avg}(s)$
5.500.0	**120148**	120130	120134	120009	120134	120130	**120148**	120126.90	9.53	3753
5.500.1	**117879**	117837	117864	117699	117864	117844	**117879**	117850.83	11.96	3876
5.500.2	**121131**	121109	**121131**	120923	121112	**121131**	**121131**	121112.23	6.74	3148
5.500.3	**120804**	120798	120794	120563	**120804**	120752	**120804**	120786.40	10.32	2918
5.500.4	**122319**	**122319**	**122319**	122054	**122319**	**122319**	**122319**	122319.00	0.00	1936
5.500.5	**122024**	122007	**122024**	121901	**122024**	**122024**	**122024**	122008.83	10.14	4421
5.500.6	**119127**	119113	119109	118846	**119127**	119094	**119127**	119120.50	4.79	3419
5.500.7	**120568**	**120568**	**120568**	120376	**120568**	120536	**120568**	120548.10	10.40	2739
5.500.8	**121586**	121575	121575	121185	121575	**121586**	121575	121559.17	9.00	2719
5.500.9	**120717**	120699	120707	120453	120707	120685	**120717**	120695.00	8.52	3353
5.500.10	**218428**	218422	**218428**	218269	**218428**	**218428**	**218428**	218411.27	11.61	4100
5.500.11	**221202**	221191	**221202**	221007	221191	**221202**	221191	221184.90	4.24	3560
5.500.12	**217542**	217534	217534	217398	217534	217528	**217542**	217525.90	6.62	3836
5.500.13	**223560**	223558	223558	223450	223558	**223560**	**223560**	223558.87	0.99	2139
5.500.14	**218966**	218962	**218966**	*	**218966**	218965	**218966**	218966.00	0.00	171
5.500.15	**220530**	220514	**220530**	220428	**220530**	220527	**220530**	220528.07	2.10	3183
5.500.16	**219989**	219987	**219989**	219734	**219989**	219943	**219989**	219985.90	4.58	2799
5.500.17	**218215**	218194	**218215**	218096	218194	**218215**	**218215**	218200.37	4.78	3700
5.500.18	**216976**	**216976**	**216976**	216851	**216976**	**216976**	**216976**	216976.00	0.00	608
5.500.19	**219719**	219693	**219719**	219549	219704	**219719**	**219719**	219715.60	3.61	2633
5.500.20	**295828**	**295828**	**295828**	295309	**295828**	**295828**	**295828**	295828.00	0.00	552
5.500.21	**308086**	308077	308079	307808	308083	**308086**	**308086**	308081.87	2.23	3225
5.500.22	**299796**	**299796**	**299796**	299393	**299796**	299788	**299796**	299796.00	0.00	638
5.500.23	**306480**	306476	306476	305992	306478	**306480**	**306480**	306478.47	1.69	2626
5.500.24	**300342**	**300342**	**300342**	299947	**300342**	**300342**	**300342**	300340.67	2.89	3454
5.500.25	**302571**	302560	**302571**	302156	302561	302560	**302571**	302565.40	4.29	2157
5.500.26	**301339**	301322	301329	300854	301329	301322	**301339**	301330.67	3.73	590
5.500.27	**306454**	306430	306430	306069	**306454**	306422	**306454**	306454.00	0.00	1707
5.500.28	**302828**	302814	302814	302447	302822	**302828**	**302828**	302820.70	8.05	2852
5.500.29	**299910**	299904	299904	299558	299904	**299910**	**299910**	299901.80	3.66	3492
Avg.	214168.8	214157.8	214163.7	*	214163.4	214157.7	**214168.1**	214159.2	4.9	2676.9
#Best		6	16	0	14	15	28			
p-value	1.57e-1	2.73e-6	1.34e-3	*	1.83e-4	2.70e-3				

In Tables 2–10, the row *Avg.* shows the average value of each quality indicator. The highest *Avg.* value of f_{best} among the compared algorithms is also highlighted in bold. The row *#Best* indicates the number of instances for which the associated algorithm obtains the best results in terms of f_{best} among the compared algorithms. Moreover, to check whether there exists a significant difference between the results of our TPTEA algorithm and those obtained by the reference algorithms in terms of f_{best}, we reported the *p-values* from the non-parametric Friedman test in the last row of tables where a *p-value* smaller than 0.05 implies a significant difference between the compared results.

It should be noted that the present comparative study mainly focuses on the solution quality rather than the computational time, since it is difficult to make a fair comparison of computing times due to the fact that the compared algorithms were run on different computing platforms and used different programming languages as well as compilers. Thus, the computational times are included in the tables just for indicative purposes.

Our results on the 90 small instances with $n = 100$ are reported in Tables 2–4 together with the results of five state-of-the-art MKP algorithms, including the genetic algorithm (GA) [10], the filter-and-fan heuristic (F&F) [27], two particle swarm optimization algorithms (SACRO-BPSO(1) and SACRO-

Table 9
Computational results and comparisons on the large instances with $n = 500$ and $m = 10$.

Problem		GA	F&F	TEPSOq	LP+TS	QPSO*	TPTEA (this work)			
Instance	Optimum	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	*Std.*	$t_{avg}(s)$
10.500.0	**117821**	117726	117734	117811	117779	117744	117801	117736.17	20.87	3665
10.500.1	**119249**	119139	119181	119232	119190	119177	119200	119137.47	26.48	3355
10.500.2	**119215**	119159	119194	118997	119194	**119215**	119159	119108.27	20.13	4470
10.500.3	**118829**	118802	118784	117999	118813	118775	**118829**	118793.93	16.73	3150
10.500.4	**116530**	116434	116471	115828	116462	116502	116456	116405.17	19.27	3582
10.500.5	**119504**	119454	119442	119410	**119504**	119402	119483	119441.80	20.50	3647
10.500.6	**119827**	119749	119764	119063	119782	**119827**	119775	119739.70	16.40	3790
10.500.7	**118344**	118288	118309	118329	118307	118309	118323	118258.27	23.63	4229
10.500.8	**117815**	117779	117781	117025	117781	117721	117801	117705.97	28.53	3169
10.500.9	**119251**	119125	119183	117815	119186	**119251**	119196	119161.90	18.67	3339
10.500.10	**217377**	217318	217318	**217377**	217343	217308	217351	217313.67	15.31	3984
10.500.11	**219077**	219022	219036	219068	219036	**219077**	219059	219022.70	15.56	3598
10.500.12	**217847**	217772	217797	217847	217797	217797	**217847**	217786.73	16.25	4010
10.500.13	**216868**	216802	216843	216257	216836	**216868**	**216868**	216836.33	18.66	3404
10.500.14	**213873**	213809	213811	213796	213859	213795	213814	213780.27	13.61	4095
10.500.15	**215086**	215013	215021	215086	215034	**215086**	**215086**	215049.57	14.45	3622
10.500.16	**217940**	217896	217880	217825	217903	217868	217926	217884.80	17.07	4281
10.500.17	**219990**	219949	219969	219825	219965	219949	219984	219947.37	18.07	3908
10.500.18	**214382**	214332	214346	214368	214341	**214382**	214363	214327.43	15.24	3999
10.500.19	**220899**	220833	220849	220168	220865	220827	220887	220864.43	16.89	3212
10.500.20	**304387**	304344	304344	304387	304351	304344	**304387**	304364.47	10.35	3025
10.500.21	**302379**	302332	302345	302196	302333	302341	**302379**	302364.47	10.05	2884
10.500.22	**302417**	302354	302408	302416	302408	**302417**	302416	302398.13	10.22	3498
10.500.23	**300784**	300743	300743	300645	300757	**300784**	**300784**	300758.80	6.73	967
10.500.24	**304374**	304344	304357	304001	304344	304340	**304374**	304361.13	6.21	3509
10.500.25	**301836**	301730	301742	299774	301754	**301836**	301796	301740.43	14.11	3809
10.500.26	**304952**	304949	304911	304841	304949	**304952**	**304952**	304952.00	0.00	3133
10.500.27	**296478**	296437	296447	295875	296441	296437	**296478**	296455.53	6.61	3139
10.500.28	**301359**	301313	301331	300964	301331	301293	**301359**	301349.27	11.82	3187
10.500.29	**307089**	307014	307078	306010	307078	307002	**307089**	307088.23	3.20	2352
Avg.	212859.3	212798.7	212814.0	212474.5	212824.1	212820.9	**212840.7**	212804.5	15.1	3467.0
#Best		0	0	1	1	11	12			
p-value	2.21e-5	7.24e-8	2.07e-6	6.04e-3	2.61e-4	4.99e-2				

Table 10
Computational results and comparisons on the large instances with $n = 500$ and $m = 30$.

Problem		GA	F&F	TEPSOq	LP+TS	QPSO*	TPTEA (this work)			
Instance	Best Known	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	*Std.*	$t_{avg}(s)$
30.500.0	**116056**	115868	115903	116055	115950	115991	115968	115897.20	23.80	3969
30.500.1	**114810**	114667	114718	**114810**	**114810**	114684	114769	114733.00	20.01	3549
30.500.2	**116741**	116661	116583	115998	116683	116712	116708	116619.10	29.95	4413
30.500.3	**115354**	115237	115198	115268	115301	**115354**	115313	115251.60	25.19	3499
30.500.4	**116525**	116353	116474	**116525**	116435	116435	116455	116364.80	32.83	3436
30.500.5	**115741**	115604	115734	115626	115694	115594	115734	115674.00	22.98	3848
30.500.6	**114181**	113952	113996	114122	114003	113987	114085	114037.10	38.04	4785
30.500.7	**114348**	114199	114266	114305	114213	114184	114278	114164.40	33.17	4110
30.500.8	**115419**	115247	**115419**	115287	115288	**115419**	115288	115221.43	26.06	4043
30.500.9	**117116**	116947	117011	117101	117055	116909	117112	116984.37	35.94	3779
30.500.10	**218104**	217995	218068	218073	218068	218068	**218104**	218069.60	9.28	3163
30.500.11	**214648**	214534	214626	214645	214562	214626	214645	214544.93	35.69	3796
30.500.12	**215978**	215854	215836	215918	215903	215839	215946	215898.80	17.46	4008
30.500.13	**217910**	217836	217862	217836	**217910**	217816	**217910**	217831.33	30.80	3259
30.500.14	**215689**	215596	215592	213625	215596	215544	**215689**	215602.07	24.60	3945
30.500.15	**215919**	215762	215784	215086	215842	215753	215840	215766.23	26.09	3558
30.500.16	**215907**	215772	215824	214999	215838	215789	**215907**	215857.23	19.46	3177
30.500.17	**216542**	216336	216418	216425	216419	216387	**216542**	216459.73	25.80	3643
30.500.18	**217340**	217290	217225	216368	217305	217217	**217340**	217304.30	11.59	3461
30.500.19	**214739**	214624	214663	214168	214671	**214739**	**214739**	214671.30	30.71	3418
30.500.20	**301675**	301627	301643	301601	301643	301643	**301675**	301641.63	11.43	2849
30.500.21	**300055**	299985	299982	300002	**300055**	299965	**300055**	300035.73	19.46	3863
30.500.22	**305087**	304995	305062	304416	305028	305038	**305087**	305080.47	7.80	3785
30.500.23	**302032**	301935	301982	301645	302004	301982	302015	301983.60	19.13	3200
30.500.24	**304462**	304404	304413	304001	304411	304346	**304462**	304427.53	11.95	3102
30.500.25	**297012**	296894	296918	296774	296961	296892	296999	296964.97	17.11	3743
30.500.26	**303364**	303233	303320	303329	303328	303287	**303364**	303335.60	13.11	2828
30.500.27	**307007**	306944	306908	306940	306999	306915	306999	306972.50	15.25	3922
30.500.28	**303199**	303057	303109	303158	303080	303169	**303199**	303168.53	13.57	3283
30.500.29	**300572**	300460	300471	300129	300532	300449	**300596**	300530.23	16.91	3937
Avg.	211451.1	211328.9	211366.9	211141.2	211386.2	211357.8	**211427.4**	211369.8	22.2	3645.7
#Best		0	1	2	3	3	14			
p-value	2.75e-4	4.32e-8	3.44e-6	4.18e-4	1.60e-5	9.64e-5				

Table 11
Computational results and comparisons on the 11 MK_GK instances. The current best known results are indicated in bold, and the improved results are underlined.

Problem			TS_GK	F&F	LP+TS	QPSO*	TPTEA (this work)			
Instance	n	m	f_{best}	f_{best}	f_{best}	f_{best}	f_{best}	f_{avg}	*Std.*	$t_{avg}(s)$
mk_gk01	100	15	**3766**	**3766**	**3766**	**3766**	**3766**	3766.00	0.00	12.39
mk_gk02	100	25	**3958**	**3958**	**3958**	**3958**	**3958**	3958.00	0.00	7.14
mk_gk03	150	25	5650	5650	**5656**	**5656**	**5656**	5652.43	1.94	98.84
mk_gk04	150	50	5764	5764	**5767**	**5767**	**5767**	5766.27	0.68	109.46
mk_gk05	200	25	7557	7557	7560	7560	<u>**7561**</u>	7560.00	0.82	820.41
mk_gk06	200	50	7672	7671	7677	7677	<u>**7680**</u>	7674.17	1.13	875.12
mk_gk07	500	25	19215	19217	**19220**	**19220**	19215	19213.73	0.68	2596.11
mk_gk08	500	50	18801	18802	**18806**	**18806**	18797	18794.07	1.29	2967.29
mk_gk09	1500	25	58085	58085	**58087**	**58087**	58082	58079.03	1.14	5294.44
mk_gk10	1500	50	57292	57292	**57295**	57292	57277	57272.97	1.89	5509.31
mk_gk11	2500	100	95231	95234	**95237**	95234	95181	95175.47	4.33	6978.26

BPSO(2)) [9], and the hybrid quantum particle swarm optimization algorithm (QPSO*) [24]. Note that the absence of a reference in these tables means that its results are not available.

The results show that our algorithm reaches, in 0.3 to 29 seconds, the optimum solutions with a success rate of 100% for all the instances without exception, showing a good robustness of the algorithm. In terms of $\#Best$, our algorithm outperforms the reference algorithms for the instances with $m = 10$ and 30. In addition, the *p-values* indicate that there does not exist a significant difference between our algorithm and GA, F&F, and QPSO* in terms of f_{best}, but our algorithm significantly outperforms SACRO-BPSO(1) and SACRO-BPSO(2).

The results on the 90 moderate size instances with $n = 250$ and $m = 5, 10, 30$ in Tables 5 – 7 show that the proposed algorithm performs very well. Specifically, for the 30 instances with $m = 5$ (Tables 5), our TPTEA algorithm consistently reaches the optimum solutions for all the instances with a success rate of 100% while the GA, F&F, QPSO* algorithms attain the optimum solutions only for 18, 23, and 25 instances, respectively. For the 30 instances with $m = 10$ (Table 6), our algorithm reaches the optimum solutions for all the instances with a small standard deviation, while the four reference algorithms report the optimum solutions only for 10, 11, 14, and 17 instances, respectively. For the 30 instances with many constraints (i.e., $m = 30$, Table 7), our algorithm matches the best known results for 28 instances, and misses the best known results only for one instance. Interestingly, our algorithm discovers an improved best known result for one hard instance (30.250.08). The reference algorithms GA, F&F, QPSO* match the best known results only for 3, 9, 12 instances. Finally, the small *p-values* (<0.05) in Tables 5 – 7 indicate that there exists a significant difference between our algorithm and the reference algorithms in terms of f_{best} for these moderate size instances with $n = 250$.

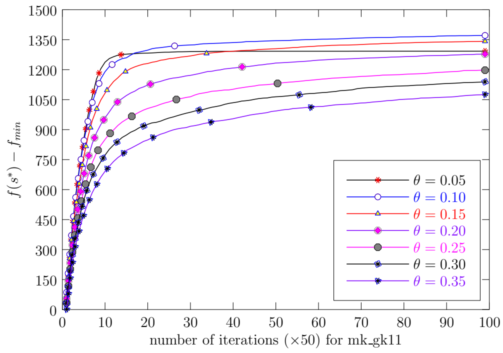
The results on the 90 large instances with $n = 500$, $m = 5, 10, 30$ are summarized in Tables 8 – 10. Table 8 shows that our algorithm is also very efficient for the large instances with a small number of constraints. Specifically, the TPTEA algorithm matches the optimum solutions for 28 out of 30 instances with a small standard deviation of the objective values, while the five reference algorithms obtain the optimum solutions only for 6, 16, 0, 14 and 15 instances, respectively. Moreover, the small *p-values* show that the differences between our results and those obtained by the reference algorithms are significant in terms of f_{best}. Tables 9 – 10 indicate that like the reference algorithms, the performance of our algorithm decreases as the number of constraints m increases. For the instances with $m = 10$ and 30, our algorithm obtains the best known objective values only for 12 and 14 instances, respectively. Still this performance is remarkable compared to the five reference algorithms. Indeed, for the 30 instances with $m = 10$, the five reference algorithm yield the optimum solutions only for 0, 0, 1, 1, and 11 instances, respectively. For the 30 instances with $m = 30$, the reference algorithms report the best known results only for 0, 1, 2, 3, and 3 instances, respectively. The small *p-values* also confirm the dominance of our algorithm in terms of f_{best}. Finally, for the instance 30.500.29, our algorithm improves the best known objective value, thus yielding a new lower bound for this instance.

Our results on 11 instances proposed by Glover and Kochenberger are reported in Table 11 together with the results of four reference algorithms. We observe that our algorithm matches the best known results for 4 instances and additionally improves the best known results for 2 other instances (new lower bounds). However, our algorithm fails to reach the best known result for 5 very large instances, indicating that there is room for improvement. Table 11 also shows that TPTEA is very time-consuming especially for the large instances in comparison with reference algorithms like QPSO* according to the results reported in [24]. Nevertheless, compared to the tabu search-based methods, our computational times are acceptable. For example, for the LP+TS algorithm [39], up to 3 days were needed to obtain the reported results for the instances with $n \geq 1000$.

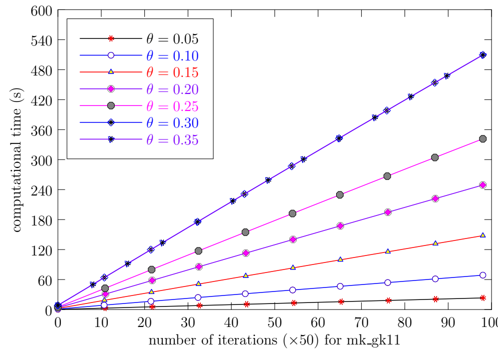
In summary, these results mean that our TPTEA algorithm is very competitive compared to the state-of-the-art MKP algorithms in the literature. For all test sets of medium to large size problems except one, which represent 180 out of 191 problems, our method obtains on average more than twice the number of best known results as the best of the other methods tested, and on the remaining 11 problems we obtain two new best solutions.

4 Analysis and discussions

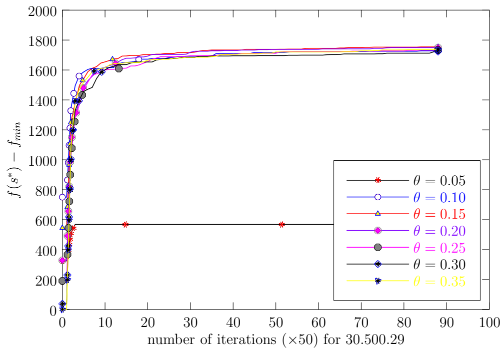
In this section, we analyze two key algorithmic components of the algorithm to understand their impacts on the performance of the algorithm, including the reduced swap neighborhood and the parameter Δ_k used to control the number of the hyperplanes to be searched.



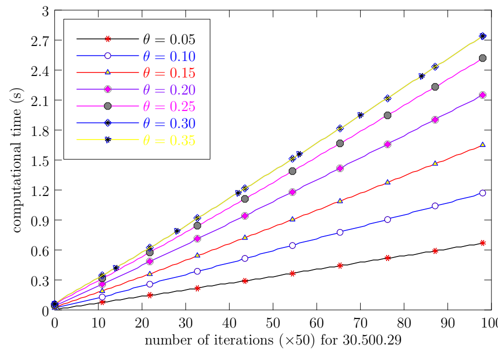
(a) Evolution of the best objective value



(b) Evolution of run time



(c) Evolution of the best objective value



(d) Evolution of run time

Fig. 2. Influence of the size of the reduced neighborhood for the tabu search method

The solution-based tabu search method (Section 3) that works on a fixed hyperplane $\Omega_{[k]}$ is an important ingredient of the proposed TPTEA algorithm, and its computational efficiency depends largely on the size of the neighborhood used. We show an analysis of the neighborhood size on the performance of the algorithm via an additional experiment. We conducted the experiment based on two selected instances, i.e., mk_gk11 and 30.500.29. According to the definitions of the neighborhoods N_2 and N_3 (see Section 2.4.1), their sizes are closely related to the parameter θ, and a larger value of θ leads generally to a larger neighborhood. Hence, in this experiment, for each selected instance and each value of θ in the range of $\{0.05, 0.10, 0.15, 0.20, 0.25, 0.30.0.35\}$, the tabu search procedure was independently run 10 times on the hyperplane $\Omega_{[k]}$, where the maximum number of iterations $IterMax$ is set to 5×10^3 and k is set according to the best known solution. The average results over 10 runs are recorded both in terms of run times and the gaps between $f(s^*)$ and the minimum objective value f_{min} among the initial solutions for the tested θ values. The evolutions of the run time and $(f(s^*) - f_{min})$ as a function of the iterations are plotted in Fig. 2.

From Fig. 2 ((a) and (b)), one observes that the tabu search method with a small value of θ performs generally better than those with a large value of θ. Nevertheless, a value that is too small can lead to a bad behavior of the algorithm. For example, for the instance 30.500.29, the algorithm with $\theta = 0.05$ performs the worst, as shown in Fig. 2(c). In the general case, the tabu search algorithm with $\theta = 0.15$ performs well for the large instances with $n \geq 500$. On the other hand, Fig.2 ((b) and (d)) show that for each tested value of θ the run time increases linearly with the increase of the number of iterations, and that the algorithm is more time-consuming using a large value of θ than a small θ value, which is consistent with the principle that a larger θ value corresponds to a larger neighborhood.

In summary, for the solution-based tabu search method working on the hyperplane $\Omega_{[k]}$, a small value of θ that corresponds to a small and high-quality swap neighborhood is very desirable for reaching a high performance of the algorithm. Nevertheless, it is important to avoid making θ too small since this will restrict the search region of the algorithm too much, causing the search to miss high-quality solutions. In general, a medium value of θ will lead to a good tradeoff between the computing speed and solution quality.

4.2 Sensitivity analysis of the parameter Δ_k

Table 12
Influence of the parameter Δ_k on the performance of algorithm. The best results are indicated in bold in terms of the best and average objective values.

	$\Delta_k = 0$		$\Delta_k = 1$		$\Delta_k = 2$		$\Delta_k = 3$	
Instance	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
5.500.0	120143	120110.90	**120148**	**120126.20**	120129	120120.50	120141	120119.40
5.500.1	**117879**	117847.10	117864	117851.20	117864	**117853.50**	117864	117845.20
5.500.2	121115	121105.80	**121131**	**121115.70**	**121131**	121113.70	121123	121111.30
5.500.3	**120804**	**120790.20**	**120804**	120788.50	**120804**	120785.30	**120804**	120782.10
5.500.4	**122319**	122299.40	**122319**	**122319.00**	**122319**	**122319.00**	**122319**	122317.50
10.500.0	117777	117739.00	**117781**	**117742.90**	117779	117740.00	117778	117739.80
10.500.1	119161	119111.40	**119194**	**119160.60**	119161	119140.00	119163	119142.40
10.500.2	**119168**	**119133.40**	119158	119124.50	119157	119117.70	119158	119112.90
10.500.3	118813	118801.00	**118814**	**118811.00**	118813	118806.80	118813	118800.00
10.500.4	**116470**	**116407.60**	116451	116402.10	116427	116395.70	116471	116406.10
30.500.0	115906	115831.80	**115952**	**115920.00**	115964	115918.70	115945	115892.30
30.500.1	114769	**114750.40**	**114780**	114734.10	114769	114728.80	114769	114710.00
30.500.2	**116716**	116618.50	**116716**	**116634.20**	116666	116613.50	116634	116602.10
30.500.3	115277	115226.50	**115301**	**115259.10**	115272	115254.00	115252	115233.90
30.500.4	116448	116362.80	116453	**116410.60**	**116480**	116397.30	116432	116375.90
Avg.	118184.33	118142.39	**118191.07**	**118159.98**	118182.33	118153.63	118177.73	118146.06

In the TPTEA algorithm, a self-adapting mechanism is used to determine the most promising hyperplane to be searched. Specifically, at each iteration of the algorithm, $2\Delta_k + 1$ offspring solutions that lie at different hyperplanes $\Omega_{[k]}$ ($k \in [k^* - \Delta_k, k^* + \Delta_k]$) are generated by the hyperplane-constrained crossover operator, where k^* is the number of variables equal to 1 in the best solution s^* found so far. Clearly, a larger value of Δ_k means that more hyperplanes will be searched at each iteration of the algorithm. To assess the impact of the parameter Δ_k and to find an appropriate value for it, we carried out another experiment based on 15 selected instances. In this experiment, for each instance and each value of Δ_k in $\{0, 1, 2, 3\}$, the algorithm was independently run 10 times according to the experimental protocol in section 3.2. The results are summarized in Table 12, where row 1 and column 1 indicate the setting of Δ_k and the instance names, and f_{best} and f_{avg} show respectively the best and average objective values over 10 runs. The last row of the table gives the average results over all tested instances.

Table 12 shows that the performance of the algorithm is sensitive to the value of Δ and $\Delta_k = 1$ leads to the best performance among all the tested settings. Specifically, the last row of the table shows that the setting of $\Delta_k = 1$ leads to the best results both in terms of the best and average objective values. Moreover, a larger value like $\Delta_k = 3$ will deteriorate the performance of the algorithm, since the algorithm with a larger Δ_k value requires more computational effort at each iteration.

5 Conclusions and future work

In this paper, we presented an effective hybrid evolutionary algorithm for solving the NP-hard 0–1 multidimensional knapsack problem. The proposed algorithm integrates a number of original features, including two solution-based tabu search methods exploring different search spaces, a reduced swap neighborhood, a hyperplane-constrained crossover operator, and a self-adapting mechanism to select the proper hyperplane to be examined by the optimization procedure.

The computational results on the 281 instances commonly used in literature showed that the proposed algorithm performs competitively in comparison with stat-of-the-art algorithms in the literature. Specifically, the algorithm reproduces the best known results for the vast majority of instances tested, and establishes new best known solutions (improved lower bounds) for 4 hard instances.

The impacts of two essential ingredients of the algorithm are analyzed, including the size of neighborhood of tabu search method and the self-adapting mechanism to determine the hyperplane. It was shown that both components play a key role for the performance of the algorithm.

There are several possible directions to further improve the present work.

First, compared to the state-of-the-art algorithms in the literature, the proposed algorithm is time-consuming for solving some large instances. To speed up the search process of its underlying tabu search procedures, the neighborhoods can be further refined by self-adaptively controlling the fitness values of candidate solutions, similarly to the construction of neighborhood N_3. Second, to better explore different hyperplanes, the present tabu search methods can be combined with other local search methods. Third, more advanced surrogate constraint ratios can be used to modify solutions where the crossover fails to yield offspring with k variables equal to 1, and our uniform crossover operator for combining solutions can be replaced with a scatter design that allows probabilistic choices instead of random choices for assigning values to variables. Fourth, employing an approach that extends the framework underlying scatter search, the path-relinking method can be employed to generate new solutions from existing solutions. Finally, given that the proposed solving framework is quite general, it would be reasonable to apply the approach to other related binary optimization problems (e.g., allocation problems, general assignment problem, balanced loading problems, maximum diversity problems).

Acknowledgments

We are grateful to the reviewers for their valuable comments which helped us to improve the paper. This work is partially supported by the National Natural Science Foundation of China (Grant No. 61703213), the Natural Science Foundation of Jiangsu Province of China (Grant No. BK20170904), and NUPTSF (Grant No. NY217154).

References

[1] Al-Shihabi S., Ólafsson S., 2010, A hybrid of nested partition, binary ant system, and linear programming for the multidimensional knapsack problem. *Computers & Operations Research*, 37(2), 247–255.

[2] Angelelli E., Mansini R., Speranza M.G., 2010, Kernel search: A general heuristic for the multi-dimensional knapsack problem. *Computers & Operations Research*, 37, 2017–2026.

[3] Arin A., Rabadi G., 2016, Local search versus path relinking in metaheuristics: redesigning meta-raps with application to the multidimensional knapsack problem. *Applied Soft Computing*, 46, 317–327.

[4] Banitalebi A., Aziz M.I.A., Aziz Z.A., 2016, A self-adaptive binary differential evolution algorithm for large scale binary optimization problems. *Information Sciences*, 367, 487–511.

[5] Beheshti Z., Shamsuddin S., Hasan S., 2015, Memetic binary particle swarm optimization for discrete optimization problems. *Information Sciences*, 299, 58–84.

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

[7] Carlton W.B., Barnes J.W., 1996, A note on hashing functions and tabu search algorithms. *European Journal of Operational Research*, 95(1), 237–239.

[8] Carlton W.B., Barnes J.W., 1996, Solving the traveling salesman problem with time windows using tabu search. *IIE Transactions*, 28, 617–629.

[9] Chih M., 2015, Self-adaptive check and repair operator-based particle swarm optimization for the multidimensional knapsack problem. *Applied Soft Computing*, 26, 378–389.

[10] Chu P.C., Beasley J.E., 1998, A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4, 63–86.

[11] Drexl A., 1988, A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing*, 40, 1–8.

[12] Fréville A., 2004, The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155, 1–21.

[13] Garey, M. R., Johnson, D. S., 1979, Computers and Intractability: A Guide to the Theory of NP-Completeness. A Series of Books in the Mathematical Sciences. San Francisco, Calif.: W. H. Freeman and Co.

[14] Gavish B., Pirkul H., Allocation of databases and processors in a distributed data processing, in: J. Akola (Ed.), Management of Distributed Data Processing, North-Holland, Amsterdam, 1982, pp. 215–231.

[15] Gavish B., Pirkul H., 1985, Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical Programming*, 31,78–105.

[16] Gilmore P.C., Gomory R.E., 1966, The theory and computation of knapsack functions. *Operations Research*, 14(6), 1045–1075.

[17] Glover F., 1965, A multiphase-dual algorithm for the zero-one integer programming problem. *Operations Research*, 13(6), 879–919.

[18] Glover F., 1968, Surrogate constraints. *Operations Research*, 16(4), 741–749.

[19] Glover F., 1977, Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1), 156–166.

[20] Glover F., 1994, Genetic algorithms and ccatter search: unsuspected potentials. *Statistics and Computing*, 4, 131–140.

[21] Glover F., 2003, Tutorial on surrogate constraint approaches for optimization in graphs. *Journal of Heuristics*, 9(3), 175–228.

[22] Glover F., Laguna M., 1997, Tabu search. *Kluwer Academic Publishers*, Boston.

[23] Glover F., Kochenberger G.A., 1996, Critical event tabu search for multidimensional knapsake problems. In: Meta-hueristics, Springer, pp.407–427.

[24] Haddar B., Khemakhem M., Hanafi S., Wilbaut C., 2016, A hybrid quantum particle swarm optimization for the multidimensional knapsack problem. *Engineering Applications of Artificial Intelligence*, 55, 1–13.

[25] Hanafi S., A. Fréville A., 1998, An efficient tabu search approach for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 106, 659–675.

[26] Hao J.K., 2012, Memetic algorithms in discrete optimization. In F. Neri, C. Cotta, P. Moscato (Eds.) Handbook of Memetic Algorithms. Studies in Computational Intelligence 379, Chapter 6, pages 73–94.

[27] Khemakhem M., Haddar B., Chebil K., Hanafi S., 2012, A filter-and-fan metaheuristic for the 0–1 multidimensional knapsack problem. *International Journal of Applied Metaheuristic Computing*, 3(4), 43–63.

[28] Ktari R., Chabchoub H., 2013, Essential particle swarm optimization queen with tabu search for MKP resolution. *Computing*, 95, 897–921.

[29] Kong X., Gao L., Ouyang H., Li S., 2015, Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm. *Computers & Operations Research*, 63, 7–22.

[30] López L., Blas N., Albert A.A., 2017, Multidimensional knapsack problem optimization using a binary particle swarm model with genetic operations. *Soft Computing*, DOI: 10.1007/s00500-017-2511-0.

[31] Lü Z., Hao J.K., 2010, A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1), 241–250.

[32] Mansini R., Speranza M.G., 2012, CORAL: an exact algorithm for the multidimensional knapsack problem. *INFORMS Journal on Computing*, 24(3), 399–415.

[33] Meng T., Pan Q.K., 2017, An improved fruit fly optimization algorithm for solving the multidimensional knapsack problem, *Applied Soft Computing*, 50,79–93.

[34] Ozturk C., Hancer E., Karaboga D., 2015, A novel binary artificial bee colony algorithm based on genetic operators. *Information Sciences*, 297, 154–170.

[35] Porumbel D., Hao J.K., Kuntz P., 2010, An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Computers and Operations Research*, 37(10), 1822–1832.

[36] Puchinger J., Raidl G.R., Pferschy U., 2009, The multidimensional knapsack problem: structure and algorithms. *INFORMS Journal on Computing*, 22(2), 250–265.

[37] Rezoug A., Bader-El-Den M., Boughaci D., 2017, Guided genetic algorithm for the multidimensional knapsack problem. *Memetic Computing*, DOI: 10.1007/s12293-017-0232-7.

[38] Shih W., 1979, A branch & bound method for the multiconstraint zero-one knapsack problem. *Journal of the Operational Research Society*, 30, 369–378.

[39] Vasquez M., Hao J.K., A hybrid approach for the 0–1 multidimensional knapsack problem. Proc. of the 17th Intl. Joint Conference on Artificial Intelligence (IJCAI-01), pages 328-333, Seattle, Washington, USA, August 2001. Morgan Kaufmann Publishers.

[40] Vasquez M., Vimont Y., 2005, Improved results on the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 165, 70–81.

[41] Vimont Y., Boussier S., Vasquez M., 2008, Reduced costs propagation in an efficient implicit enumeration for the 0–1 multidimensional knapsack problem. *Journal of Combinatorial Optimization*, 15, 165–178.

[42] Wang L., Yang R., Xu Y., Niu Q., Pardalos P.M., Fei M., 2013, An improved adaptive binary harmony search algorithm. *Information Sciences*, 232, 58–87.

[43] Wang Y., Wu Q., Glover F., 2017, Effective metaheuristic algorithms for the minimum differential dispersion problem. *European Journal of Operational Research*, 258, 829–843.

[44] Weingartner H.M., Ness D.N., 1967, Methods for the solution of the multidimensional 0–1 knapsack problem. *Operations Research*, 15(1), 83–103.

[45] Woodruff D.L., Zemel E., 1993, Hashing vectors for tabu search, *Annals of Operations Research*, 41(2), 123–137.

[46] Wu Q., Hao J.K., 2013, A hybrid metaheuristic method for the maximum diversity problem, *European Journal of Operational Research*, 231(2), 452–464.

[47] Zhang B., Pan Q.K., Zhang X.L., Duan P.Y., 2015, An effective hybrid harmony search-based algorithm for solving multidimensional knapsack problems, *Applied Soft Computing*, 29, 288–297.