European Journal of Operational Research 274 (2019) 35–48

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem

Xiangjing Lai[a], Jin-Kao Hao[b,c,*], Dong Yue[a]

[a] Institute of Advanced Technology, Nanjing University of Posts and Telecommunications, Nanjing 210023, China
[b] LERIA, Université d'Angers, 2 Boulevard Lavoisier, Angers 49045, France
[c] Institut Universitaire de France, 1 Rue Descartes, Paris 75231, France

A R T I C L E I N F O

Article history:
Received 7 December 2017
Accepted 2 October 2018
Available online 9 October 2018

Keywords:
Metaheuristics
Multidemand multidimensional knapsack problem
Two-stage optimization
Solution-based tabu search
Combinatorial optimization

A B S T R A C T

The multidemand multidimensional knapsack problem (MDMKP) is a significant generalization of the popular multidimensional knapsack problem with relevant applications. In this work we investigate for the first time how solution-based tabu search can be used to solve this computationally challenging problem. For this purpose, we propose a two-stage search algorithm, where the first stage aims to locate a promising hyperplane within the whole search space and the second stage tries to find improved solutions by exploring the reduced subspace defined by the hyperplane. Computational experiments on 156 benchmark instances commonly used in the literature show that the proposed algorithm competes favorably with the state-of-the-art results. We analyze several key components of the algorithm to highlight their impacts on the performance of the algorithm.

1. Introduction

Given a set $V = \{1, 2, \ldots, n\}$ of n items, a set $R = \{r_1, r_2, \ldots, r_m\}$ of m resources with a capacity upper limit b_i for resource r_i $(1 \le i \le m)$, where each item j of V is associated with a profit c_j and consumes a given quantity a_{ij} for each resource r_i $(i \in \{1, 2, \ldots, m\})$, the popular NP-hard 0–1 multidimensional knapsack problem (MKP) involves selecting a subset of items from V such that the resource consummation of the selected items does not exceed the given capacity upper limit for each resource in R (knapsack constraints), while maximizing the total profit of the selected items. Formally, the MKP can be written as follows:

$$(MKP) \qquad \text{Maximize} \quad z = \sum_{j=1}^{n} c_j x_j \tag{1}$$

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

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

where $c_j \ge 0$, $a_{ij} > 0$, $b_i > 0$, $\forall i \in \{1, 2, \ldots, m\}, \forall j \in \{1, 2, \ldots, n\}$ and Eq. (3) indicates that the binary decision variable x_j $(1 \le j \le n)$ takes the value of 1 if the item j is selected, 0 otherwise.

The multidemand multidimensional knapsack problem (MDMKP) studied in this work is an important extension of the MKP, where q greater-than-or-equal-to constraints are imposed, in addition to m less-than-or-equal-to constraints (Eq. (2)). Moreover, unlike the MKP, the profit c_j in the MDMKP can take a positive, negative or zero value for each item j $(j \in V)$. Formally, the MDMKP can be formulated as follows (Arntzen, Hvattum, & Lokketangen, 2006; Cappanera & Trubian, 2005):

$$(MDMKP) \qquad \text{Maximize} \quad z = \sum_{j=1}^{n} c_j x_j \tag{4}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i, \forall i \in \{1, 2, \ldots, m\} \tag{5}$$

$$\sum_{j=1}^{n} a_{ij} x_j \ge b_i, \forall i \in \{m+1, m+2, \ldots, m+q\} \tag{6}$$

$$x_j \in \{0, 1\}, \forall j \in \{1, 2, \ldots, n\} \tag{7}$$

where the following conditions are assumed:

$$b_i > 0, a_{ij} \ge 0 \quad \forall i \in \{1, 2, \ldots, m+q\}, \forall j \in \{1, 2, \ldots, n\} \tag{8}$$

$$\sum_{j=1}^{n} a_{ij} > b_i \quad \forall i \in \{1, 2, \ldots, m+q\} \tag{9}$$

$$max_j\{a_{ij}\} \le b_i \quad \forall i \in \{1, 2, \ldots, m\} \tag{10}$$

$$min_j\{a_{ij}\} < b_i \quad \forall i \in \{m+1, 2, \ldots, m+q\} \tag{11}$$

* Corresponding author at: LERIA, Universitéd'Angers, 2 Boulevard Lavoisier, Angers 49045, France.
E-mail addresses: laixiangjing@gmail.com (X. Lai), jin-kao.hao@univ-angers.fr (J.-K. Hao), medongy@vip.163.com (D. Yue).

https://doi.org/10.1016/j.ejor.2018.10.001
0377-2217/

In above formulas, the inequalities in Eq. (5) are called the knapsack constraints, and those in Eq. (6) are called the demand constraints.

Clearly, the classic MKP is a special case of the MDMKP when q equals 0 and the profit c_j of item j takes a nonnegative value (i.e., $c_j \geq 0$, $\forall j \in V$).

Like the MKP, the MDMKP has a number of practical applications (Cappanera & Trubian, 2005) like obnoxious and semiobnoxious facility location (Cappanera, Gallo, & Maffioli, 2004; Plastria, 2001; Romero-Moraleses, Carrizosa, & Conde, 1997), capital-budgeting, and portfolio-selection (Beaujon, Marin, & McDonald, 2001), among others. On the other hand, the MDMKP is computationally challenging, given that it generalizes the NP-hard multidimensional knapsack problem. Consequently, there is no polynomial-time algorithm for the MDMKP, unless $P = NP$.

Unlike the MKP that has been subject of intensive studies in the past decades (see e.g., Chu & Beasley, 1998; Fréville, 2004; Glover & Kochenberger, 1996; Hanafi & Fréville, 1998; Lai, Hao, Glover, & Lü, 2018a; Mansini & Speranza, 2012; Puchinger, Raidl, & Pferschy, 2009; Shih, 1979; Vasquez & Hao, 2001; Vasquez & Vimont, 2005; Vimont, Boussier, & Vasquez, 2008), the MDMKP receives much less attention until now. Still, there exist several exact and heuristic approaches in the literature. For example, general mixed integer programming solvers like CPLEX can be used to solve instances with $n \leq 100$ to optimality within an acceptable time. However, it is usually difficult for the existing exact approaches to find an optimal solution for larger instances. As a result, several heuristic algorithms have been proposed to solve large instances approximately.

Specifically, in 2005, Cappanera and Trubian (2005) presented a nested-tabu-search heuristic, which combines a standard attribute-based tabu search with an oscillation method presented in Glover and Kochenberger (1996). In 2006, Arntzen et al. (2006) proposed an adaptive memory search method called Almha, which uses a dynamical tabu search mechanism and a weighting scheme to handle infeasible solutions. Their computational results show the Almha algorithm outperforms the previous best MDMKP methods and can be viewed as one of the best performing MDMKP algorithms in the literature. In 2007, Hvattum and Løkketangen (2007) investigated the behavior of scatter search on the MDMKP. In 2009, Gortázar, Duarte, Laguna, and Martí (2010) introduced a black box scatter search method for general classes of binary optimization problems, and assessed their method on the MDMKP and some other binary problems. In particular, their method uses a static penalty approach proposed in Yeniay (2005) to handle the constraints of the MDMKP. In 2010, Hvattum, Arntzen, Løkketangen, and Glover (2010) proposed an alternating control tree (ACT) search framework for the MDMKP, which can lead to an exact algorithm or heuristic algorithm by choosing the routine of solving subproblems. Their computational results show that the associated ACT algorithms have a high performance compared to a previous tabu search algorithm and scatter search algorithm. At the same year, Balachandar (2010) proposed a dominance principle based heuristic for the MDMKP.

In addition to these studies, there exist some theoretical investigations dedicated to the MDMKP in the literature. For example, Delissa (2014) investigated the existence and usefulness of equality cuts for the MDMKP, while Wishon and Villalobos (2016) studied robust efficiency measures for the MDMKP.

To enrich the solution arsenal for the MDMKP, we present in this work the first study of using solution-based tabu search (Carlton & Barnes, 1996a; 1996b; Woodruff & Zemel, 1993) to effectively solve the MDMKP. Actually, unlike the popular attribute-based tabu search approach (Glover & Laguna, 1997), solution-based tabu search began to attract attention only very recently. Interestingly, this approach already showed excellent performances on several binary optimization problems as reported in Lai, Yue, Hao, and Glover (2018b), Lai et al. (2018a), and Wang, Wu, and Glover (2017). This work aims thus to investigate the interest of the solution-based tabu search approach for the MDMKP. Compared to attribute-based tabu search, solution-based tabu search has at least two appealing features. First, this approach ensures a stronger intensification ability, which is crucial for locating good local optima. Second, this approach makes the notion of tabu tenure irrelevant, thus simplifying the design of the algorithm and reducing the number of required parameters.

We summarize the contributions of this work as follows. First, based on the solution-based tabu search approach, we introduce an effective two-stage search algorithm for the MDMKP. The first search stage aims to identify a promising hyperplane within the whole search space while the second search stage tries to find improved solutions by examining both feasible and infeasible solutions on the identified hyperplane. For both stages, the solution-based tabu search strategy is employed, which relies on a one-flip and swap neighborhoods and a hash-based mechanism to efficiently determine the tabu status of neighbor solutions. Second, we assess the performance of the proposed algorithm based on 96 benchmark instances commonly used in the literature ($n = 100, 250$, $m = 5, 10$ and $q = 2, 5, 10$). The computational results show that the algorithm improves and matches the best known solutions for 17 and 71 instances respectively. Moreover, we report detailed computational results of the proposed algorithm for 60 additional instances with a large number of constraints (with $n = 100, 500$, $m = q = 30$). Third, given that the ideas of the two-stage search framework and solution-based tabu search developed in this work are quite general, they could be applied to solve other related binary optimization problems.

The remaining parts of the paper are structured as follows. In the next section, the proposed two-stage tabu search algorithm is described. In Section 3, we present the computational assessment of the proposed algorithm and report experimental results on the well-known benchmark instances. In Section 4, several essential ingredients of the algorithm are investigated to show how they affect the performance of the algorithm. In the last section, we summarize the present work and provide research perspectives.

2. Two-stage tabu search algorithm for the MDMKP

Our two-stage solution-based tabu search (TSTS) algorithm combines two search procedures working on two different search spaces. The first stage of the algorithm performs an exploratory search within the whole search space to find a feasible solution as good as possible. Starting from this solution, the second stage carries out a focused exploitation within the reduced space composed of candidate solutions with exactly k selected items (k being identified by the final solution of the first search stage). To explore both search spaces, TSTS relies on two solution-based tabu search procedures guided by a penalty-based evaluation function. One notices that the two-stage search strategy has been used with success to solve other knapsack problems like the Quadratic Knapsack Problem (Chen & Hao, 2017) and the classic MKP (Vasquez & Hao, 2001).

2.1. General procedure

The proposed TSTS algorithm is thus composed of two optimization stages, where the first stage identifies a suitable hyperplane $\Omega_{[k]}$ (see Section 2.4.2 for the definition of hyperplane) that is exploited intensively during the second optimization stage to locate improved solutions (see Algorithm 1).

Specifically, the TSTS algorithm first generates randomly an initial solution by its initialization procedure (Section 2.2). Then

Algorithm 1: General procedure of two-stage tabu search algorithm for the MDMKP.

```
1 Function TSTS()
  Input: Instance I, time limit t_max
  Output: The best solution s* found
2 begin
     /* Initialization of solution                     */
3    s ← InitialSolution(I)              /* Sections 2.2 */
     /* Optimization of the first stage                */
4    {s, t} ← TabuSearch_1(s)            /* Sections 2.3 */
     /* Optimization of the second stage               */
5    s ← TabuSearch_2(s, t, t_max)       /* Sections 2.4 */
6    return s
7 end
```

the algorithm enters the first search stage where the initial solution is improved by the solution-based tabu search procedure presented in Sections 2.3 (line 4). During this search stage, the algorithm explores both feasible and infeasible solutions within the whole search space to find a high-quality feasible solution. At the end of its search stage, the best (feasible) solution found and the consumed time t are returned. At this point, the second search stage is triggered, which starts from the solution returned by the first stage and uses another solution-based tabu search procedure (Section 2.4) to seek improved solutions (line 5). During this stage, the search is limited to the hyperplane $\Omega_{[k]}$ (k being the number of the selected items in the returned solution of the first stage, see Section 2.4.1). Finally, the whole algorithm terminates when a given time limit t_{max} is met, and the best solution found during the search process is returned as the final result of the algorithm (line 6).

2.2. Initial solution

The initial solution of the TSTS algorithm is generated by a randomized procedure whose pseudo-code is given in Algorithm 2. Specifically, given an instance with n items, the initialization

Algorithm 2: Procedure of generating initial solution.

```
1 Function InitialSolution()
  Input: An instance I, the size of instance (n)
  Output: An initial solution s = (x_1, x_2, ..., x_n)
2 begin
3    for i ← 1 to n do
4       x_j ← rand() mod 2       /* rand() denotes a random
        integer */
5    end
6    s ← (x_1, x_2, ..., x_n)
7    return s
8 end
```

procedure assigns randomly a value from the set {0, 1} to each component x_i $(i = 1, 2, \dots, n)$ to obtain an initial solution $s = (x_1, x_2, \dots, x_n)$. This random initialization has the advantage of being simple and fast. However, an initial solution generated in this way can be infeasible. If this is the case, its feasibility will be established during the first search stage described below.

2.3. Tabu search method of the first search stage

The first search stage is ensured by a solution-based tabu search algorithm (denoted by $TabuSearch_1()$), whose pseudo-code is given in Algorithm 3. After initiating its tabu lists (lines 3–5),

Algorithm 3: Tabu search method used in the first search stage.

```
1  Function TabuSearch_1()
   Input: Initial solution s, extended evaluation function F, hash
          vectors H_1, H_2, H_3 of length L, hash functions h_1, h_2,
          h_3, depth of tabu search α, time limit t_max
   Output: The best solution s* found, the time t consumed by
           the search
2  begin
      /* Initialization of hash vectors (i.e., tabu
      lists)                                               */
3     for i ← 0 to L − 1 do
4        H_1[i] ← 0; H_2[i] ← 0; H_3[i] ← 0;
5     end
6     s* ← s
7     NoImprove ← 0
      /* Main search procedure                             */
8     while (time() < t_max) ∧ ((NoImprove < α) ∨ (s* is
      infeasible)) do
9        Find in N_1(s) ⋃ N_2(s) a best non-tabu solution s′ in
         terms of the extended evaluation function F in Eq. (17)
         /* N_1(s) and N_2(s) are defined in Eqs. (13) and
         (14), and the tabu rule is given in Section 2.3.2
                                                           */
10       s ← s′             /* Update the current solution */
11       if (F(s) > F(s*)) then
12          s* ← s
13          NoImprove ← 0
14       end
         /* Update the hash vectors (i.e., tabu lists)
         with s                                            */
15       H_1[h_1(s)] ← 1; H_2[h_2(s)] ← 1; H_3[h_3(s)] ← 1
16       NoImprove ← NoImprove + 1
17    end
18    t ← time()
19    return {s*,t}
20 end
```

$TabuSearch_1()$ performs a number of iterations to improve the current solution until 1) a feasible solution is found and no improvement can be observed during α consecutive iterations, where α is a parameter called the depth of tabu search, or 2) the allowed maximum time limit t_{max} is reached (lines 8–17). At each iteration, according to the tabu rule and the penalty-based evaluation function defined in Sections 2.3.2 and 2.3.3, a best non-tabu neighbor solution is selected to replace the current solution, and then the tabu lists are accordingly updated. Finally, the best feasible solution found s^* during this search stage and the computation time elapsed t are returned as the results of the first search stage. As our experiments in Section 3 show, the first search stage always ends up with a feasible solution for all tested instances, including those with 30 demand constraints and 30 knapsack constraints. In other words, the first solution-based tabu search algorithm is typically able to locate a promising valid hyperplane for the second search stage.

The ingredients of this tabu search algorithm, including the search space, the neighborhood structures and the tabu strategy, are respectively described in the following subsections.

2.3.1. Search space and neighborhood

The search space Ω explored by the $TabuSearch_1()$ procedure is composed of all feasible and infeasible solutions of the given problem instance, i.e.,

$$\Omega = \{(x_1, x_2, \ldots, x_n) | x_i \in \{0, 1\}, 1 \le i \le n\} \tag{12}$$

The neighborhood used by *TabuSearch*$_1$() is a combined neighborhood composed of two basic neighborhoods, namely the one-flip neighborhood N_1 and the swap neighborhood N_2. The one-flip neighborhood N_1 is defined by the one-flip operator (denoted by *Flip*(·)). Given a solution $s = (x_1, x_2, \ldots, x_n)$, a one-flip move *Flip*(q) consists of changing the value of a variable x_q to its complementary value $1 - x_q$. As such, the neighborhood $N_1(s)$ of solution s includes all possible solutions that can be obtained by applying the one-flip operator to s. Formally, the $N_1(s)$ can be written as follows:

$$N_1(s) = \{s \oplus Flip(q) : 1 \le q \le n\} \tag{13}$$

The neighborhood N_2 is defined by the swap operator (denoted by $Swap(\cdot, \cdot)$). Given a solution $s = (x_1, x_2, \ldots, x_n)$, let $I^1 = \{q : x_q = 1 \text{ in } s\}$ and $I^0 = \{q : x_q = 0 \text{ in } s\}$, the swap neighborhood $N_2(s)$ can be written as follows:

$$N_2(s) = \{s \oplus Swap(i, j) : i \in I^1, j \in I^0\} \tag{14}$$

The first tabu search algorithm explores the union of these two neighborhoods, i.e., $N(s) = N_1(s) \cup N_2(s)$, whose size equals to $n + |I^1| \times |I^0|$. At each iteration of the algorithm, a best non-tabu solution from $N(s)$ according to the extended evaluation function defined by Eq. (17) in Section 2.3.3 and the tabu strategy explained in Section 2.3.2 is selected to replace the current solution s.

2.3.2. Tabu strategy

In the present tabu search method, we adopt the solution-based tabu strategy to determine the tabu status of neighbor solutions. Specifically, the tabu lists are based on three hash vectors H_1, H_2, and H_3 of length of L, where each position of them represents a binary variable, and each hash vector H_t is associated with a hash function h_t. In particular, the effect of hash functions is to map a candidate solution of the search space Ω to an index of H_t, i.e., h_t: $\Omega \rightarrow \{0, 1, 2, \ldots, L - 1\}$.

Based on these hash vectors and the associated hash functions, we determine the tabu status of candidate solutions by the following rule. Given a candidate solution s, the hash vectors H_t ($t = 1, 2, 3$) and the associated hash functions h_t, s is identified as a tabu solution if $H_1(h_1(s)) \wedge H_2(h_2(s)) \wedge H_3(h_3(s)) = 1$. Otherwise, s is determined as a non-tabu solution.

Following previous studies (Carlton & Barnes, 1996a; Wang et al., 2017; Woodruff & Zemel, 1993), we define our hash functions as follows. Let $s = (x_1, x_2, \ldots, x_n)$ be a candidate solution, our hash functions h_t ($t = 1, 2, 3$) are given by:

$$h_t(s) = \left(\sum_{i=1}^{n} \lfloor i^{\gamma_t} \rfloor \times x_i \right) mod\ L \tag{15}$$

where γ_t is a parameter that is used to define each hash function and L is the length of hash vectors that is empirically set to 10^8 in this work.

Given a solution s and its hash value $h(s)$, the hash value of its neighbor solutions can be determined in $O(1)$ according to Eq. (15). Thus, the time complexity of determining the tabu status of a neighbor solution is $O(1)$.

2.3.3. Extended evaluation function

Since the search space explored by the first tabu search algorithm contains both feasible and infeasible solutions, we devise an extended evaluation function F which uses a penalty function P to assess constraint violations.

Let s be a candidate solution in Ω, the penalty value $P(s)$ is defined as the summation of all constraint violations, i.e.,

$$P(s) = \sum_{i=1}^{m} Max\left\{0, \sum_{j=1}^{n} a_{ij}x_j - b_i\right\} + \sum_{i=q+1}^{q+m} Max\left\{0, b_i - \sum_{j=1}^{n} a_{ij}x_j\right\} \tag{16}$$

Thus, a small (large) function value $P(s)$ means a weak (strong) constraint violation in s. In particular, $P(s) = 0$ means that s is a feasible solution.

Given this penalty function, the extended evaluation function $F(s)$ is defined as a linear combination of the objective function $f(s)$ in Eq. (4) and $P(s)$:

$$F(s) = \sum_{j=1}^{n} c_j x_j - \lambda \times P(s) \tag{17}$$

where λ is a weighting factor that is empirically set to 10^2 in this work.

For any two solutions s' and s'' in $\Omega_{[k]}$, s' is considered to be better than s'' if $F(s') > F(s'')$.

As shown in our experimental results (Sections 3, 4.2 and the Appendix), the first search stage equipped with the extended evaluation function always ends up with a feasible solution for all tested instances, including those with 30 demand constraints and 30 knapsack constraints. In other words, the first solution-based tabu search algorithm is typically able to locate a promising valid hyperplane that is further explored by the second search stage.

2.4. Tabu search method of the second search stage

The second optimization stage of the TSTS algorithm uses another tabu search algorithm (denoted by *TabuSearch*$_2$(), Algorithm 4) to examine candidate solutions of a given hyperplane $\Omega_{[k]}$ (see below). Unlike the first tabu search algorithm, *TabuSearch*$_2$() explores only solutions that contain exactly k selected items. *TabuSearch*$_2$() first initializes the hash vectors (lines 3–5), and then performs a number of iterations to improve the current solution (lines 7–14). At each iteration, the algorithm replaces the current solution by a best non-tabu neighbor solution s' in terms of the evaluation function in Eq. (17). During the search, the best feasible solution encountered s^* is updated each time a better feasible solution is found, and the hash vectors are accordingly updated by the new solution (line 13). Finally, the algorithm terminates if the time limit t_{max} is reached, and then returns the best feasible solution s^* found during the search process.

2.4.1. Search space, tabu strategy and evaluation function

The search space $\Omega_{[k]}$ explored by *TabuSearch*$_2$() is composed of both feasible and infeasible solutions with a fixed number of k selected items. In other words, $\Omega_{[k]}$ contains all n-dimensional 0–1 vectors with $\sum_{i=1}^{n} x_i = k$, i.e., $\Omega_{[k]} = \{x \in \{0, 1\}^n | \sum_{i=1}^{n} x_i = k\}$. $\Omega_{[k]}$ is also called a hyperplane of the search space Ω defined in Section 2.3.1, i.e., $\Omega = \cup_{k=1}^{n} \Omega_{[k]}$.

Additionally, like the first tabu search algorithm, *TabuSearch*$_2$() uses the solution-based tabu strategy described in Section 2.3.2 to determine the tabu status of neighbor solutions, and employs the extended evaluation function in Eq. (17) to evaluate the solutions in the search space $\Omega_{[k]}$.

2.4.2. Neighborhood structure

To search effectively the hyperplane $\Omega_{[k]}$, *TabuSearch*$_2$() uses a constrained swap neighborhood $N_3(s)$. Formally, given a solution $s = (x_1, x_2, \ldots, x_n)$, the neighborhood $N_3(s)$ is given by:

$$N_3(s) = \{s \oplus Swap(i, j) : i \in I^1, j \in I^0; f(s \oplus Swap(i, j)) > f(s^*)\} \tag{18}$$

Algorithm 4: The tabu search method used in the second search stage.

Function *TabuSearch*$_2$()
Input: Initial solution s, extended evaluation function F, penalty function P, hash vectors H_1, H_2, H_3 of length L, hash functions h_1, h_2, h_3, time limit t_{max}, and time (t) consumed in the first search stage.
Output: The best solution s^* found
begin
  /* Initialization of hash vectors (i.e., tabu lists) */
  for $i \leftarrow 0$ ***to*** $L-1$ **do**
    $H_1[i] \leftarrow 0$; $H_2[i] \leftarrow 0$; $H_3[i] \leftarrow 0$
  end
  $s^* \leftarrow s$
  /* Main search procedure */
  while $time() < t_{max} - t$ **do**
    Find in $N_3(s)$ a best non-tabu solution s' in terms of the extended evaluation function F in Eq. (17)
    /* $N_3(s)$ is defined in Eq. (18), and the tabu rule is given in Section 2.4.1. */
    $s \leftarrow s'$ /* Update the current solution */
    if $F(s) > F(s^*) \wedge P(s) = 0$ **then**
      $s^* \leftarrow s$
    end
    /* Update the hash vectors (i.e., tabu lists) with s */
    $H_1[h_1(s)] \leftarrow 1$; $H_2[h_2(s)] \leftarrow 1$; $H_3[h_3(s)] \leftarrow 1$
  end
  return s^*
end

where $f(\cdot)$ is the objective function of the MDMKP in Eq. (4), s^* is the best feasible solution found so far in the current tabu search run, I^1 and I^0 denote respectively the sets of indices having the value of 1 (selected items) and 0 (non selected items) in s. Clearly, the size of this neighborhood is bounded by $O(|I^1| \times |I^0|)$. It is worth noting that we constraint the neighborhood by $f(s \oplus Swap(i, j)) > f(s^*)$ to eliminate non-promising neighbor solutions. Similar idea was previously investigated for the related MKP in Vasquez and Hao (2001).

2.5. Space and time complexities of the algorithm

At each stage of our TSTS algorithm, in addition to three hash vectors (i.e., H_1, H_2 and H_3) with a length of L, we maintain three solutions (i.e., s, s', and s^*) to follow the search process, where each solution is stored by two vectors (i.e., I^1 and I^0) with a maximum length of n and a vector $W = (w_1, w_2, \ldots, w_{m+q})$ where $w_i = \sum_{j=1}^{n} a_{ij} x_j$ $(j \in V)$ holds. Thus, the space complexity of our TSTS algorithm is bounded by $O(n+m+q+L)$.

In addition, for each neighbor solution in the search space, the time complexity of evaluating its quality is bounded by $O(m+q)$, since there are $m+q$ constraints needed to be checked. Hence, for each iteration, the time complexities of the first and second tabu search stages are respectively bounded by $O((m+q) \times (n + |I^1| \times |I^0|))$ and $O((m+q) \times (|I^1| \times |I^0|))$ according to the size of neighborhoods explored by the algorithm (see Sections 2.3.1 and 2.4.2).

3. Experimental results and comparisons

We evaluate the proposed TSTS algorithm by conducting extensive computational experiments based on four sets of benchmark instances commonly used in the literature and by making a comparison between our results and the state-of-the-art results in the literature.

Table 1
Settings of parameters.

Parameters	Section	Description	Values
α	2.3	Tabu depth of *TabuSearch*$_1$()	10^4
γ_1	2.3	Parameter used in the hash function	1.9
γ_2	2.3	Parameter used in the hash function	2.1
γ_3	2.3	Parameter used in the hash function	2.3

3.1. Benchmark instances

In this study, we employed four sets of benchmark instances to assess the performance of our TSTS algorithm, where the first two sets of benchmark instances are available at http://www.optsicom.es/binaryss, and the third and fourth sets of benchmark instances are available in OR-Library[1]. The first set contains 48 small instances with $n = 100$, and the second set contains 48 larger instances with $n = 250$. In addition, for the instances in the first two sets, the number of knapsack constrains m varies from 5 to 10, and the number of demand constraints q belongs to {2, 5, 10}. The third set includes 30 instances with $n = 100$, where both the number of knapsack constrains m and the number of demand constraints q equal to 30. The fourth set is composed of 30 large instances with $n = 500$, $m = 30$ and $q = 30$.

3.2. Parameter settings and experimental protocol

Our TSTS algorithm employs four parameters, including γ_1, γ_2 and γ_3 that are used to define the hash functions, and the depth α of tabu search used in the first search stage. The parameters γ_1, γ_2 and γ_3 are set as in Table 1 according to the analysis shown in Section 4.3 while α is empirically set to 10^4.

In addition, our algorithm was implemented in C++ and compiled by g++ compiler with -O3 option[2]. All computational experiments were carried out on a computer with an Intel E5-2670 processor (2.5 gigahertz and 2 gigabytes RAM), running the Linux operating system. Moreover, when running the DIMACS machine benchmark procedure dmclique[3], our processor requires 0.19, 1.17, and 4.54 seconds to solve the graphs r300.5, r400.5, and r500.5, respectively. Finally, due to the stochastic nature of the algorithm, we independently ran the algorithm 30 times to solve each instance, where the time limit t_{max} for each run was set to 60 seconds for instances with $n \le 250$ according to Gortázar et al. (2010). For the large instances with $n = 500$, the time limit t_{max} was set to n seconds, where n is the size of instances.

3.3. Computational results and comparison

In this section, we report the computational results of our TSTS algorithm on the first two sets of benchmark instances. We provide in the Appendix the computational results of TSTS on the third and fourth sets of benchmark instances for which no detailed results are available for existing algorithms in the literature.

The computational results of our TSTS algorithm on the first set of benchmark instances with $n = 100$ are summarized in Table 2, together with the results of the Almha algorithm implemented in Gortázar et al. (2010). Column 1 gives the names of instances. Columns 2 and 3 indicate respectively the best known objective

[1] http://www.people.brunel.ac.uk/~mastjjb/jeb/info.html
[2] The source code of our TSTS algorithm will be available at our website: http://www.info.univ-angers.fr/pub/hao/mdmkp.html
[3] ftp://www.dimacs.rutgers.edu/pub/dsj/clique

Table 2
Computational results and comparison on the 48 instances with $n = 100$. In terms of f_{best}, the improved results are indicated in bold compared to the best known objective values (*BKV*).

Instance	BKV	Almha	TSTS (this work)			
			f_{best}	f_{avg}	f_{worst}	σ_f
100-5-2-0-0	28384	28384	28384	28374.63	28103	50.44
100-5-2-0-1	26386	26386	26386	26386.00	26386	0.00
100-5-2-0-2	23484	23424	23484	23450.83	23285	74.16
100-5-2-0-3	27374	27374	27374	27365.33	27290	20.45
100-5-2-0-4	30632	30632	30632	30632.00	30632	0.00
100-5-2-0-5	44674	44614	44674	44650.93	44518	51.70
100-5-2-1-0	10379	10307	10379	10359.13	10276	22.53
100-5-2-1-1	11114	11074	11114	11114.00	11114	0.00
100-5-2-1-2	10124	10022	10124	10108.53	10066	25.65
100-5-2-1-3	10567	10559	10567	10567.00	10567	0.00
100-5-2-1-4	10658	10658	10658	10566.00	10451	49.10
100-5-2-1-5	17550	17550	17550	17540.30	17494	14.72
100-5-5-0-0	21892	21892	21892	21831.20	21740	74.46
100-5-5-0-1	26280	26280	26280	26280.00	26280	0.00
100-5-5-0-2	20628	20628	20628	20628.00	20628	0.00
100-5-5-0-3	21547	21547	21547	21547.00	21547	0.00
100-5-5-0-4	25074	25067	25074	25074.00	25074	0.00
100-5-5-0-5	40327	40327	40327	40320.47	40272	16.80
100-5-5-1-0	10263	10263	10263	10248.87	10210	23.44
100-5-5-1-1	10625	10625	10625	10625.00	10625	0.00
100-5-5-1-2	10198	10126	10198	10149.27	10124	34.47
100-5-5-1-3	10030	9959	10030	10019.60	9874	38.91
100-5-5-1-4	9964	9838	9964	9926.20	9775	64.25
100-5-5-1-5	15603	15591	15603	15603.00	15603	0.00
100-10-5-0-0	21852	21843	21852	21852.00	21852	0.00
100-10-5-0-1	20645	20586	20645	20593.10	20514	45.39
100-10-5-0-2	19517	19517	19517	19507.37	19228	51.88
100-10-5-0-3	20596	20556	20596	20514.20	20454	69.20
100-10-5-0-4	19423	19278	19423	19248.37	19218	41.68
100-10-5-0-5	35933	35903	35933	35856.00	35743	92.42
100-10-5-1-0	10018	10000	10018	10018.00	10018	0.00
100-10-5-1-1	9839	9839	9839	9837.83	9804	6.28
100-10-5-1-2	10000	10000	10000	9989.60	9688	56.01
100-10-5-1-3	10544	10544	10544	10535.33	10479	22.10
100-10-5-1-4	10011	9878	10011	9961.57	9908	46.28
100-10-5-1-5	16230	16210	16230	16220.33	16095	33.69
100-10-10-0-0	22054	22054	22054	22054.00	22054	0.00
100-10-10-0-1	20103	20103	20103	20103.00	20103	0.00
100-10-10-0-2	19381	19312	19381	19371.80	19312	23.46
100-10-10-0-3	17434	17434	17434	17434.00	17434	0.00
100-10-10-0-4	18792	18792	**18833**	18794.73	18792	10.23
100-10-10-0-5	33837	33833	33837	33832.23	33702	24.20
100-10-10-1-0	8560	8560	8560	8513.17	8475	26.22
100-10-10-1-1	8493	8493	8493	8489.80	8397	17.23
100-10-10-1-2	9266	9227	9266	9266.00	9266	0.00
100-10-10-1-3	9823	9823	9823	9819.13	9707	20.82
100-10-10-1-4	8929	8929	8929	8914.00	8839	33.54
100-10-10-1-5	14152	14151	14152	14144.33	14106	17.14
Avg.	18108.10	18083.17	18108.96	18088.27	18023.38	24.98
#Better			1			
#Equal			47			
#Worse			0			
p-value			3.2e−1	8.7e−1		

values (*BKV*) and the results of the Almha algorithm, which were reported in Gortázar et al. (2010) and available at http://www.optsicom.es/binaryss. The results of our TSTS algorithm are reported in columns 4–7, including the best objective value obtained over 30 runs (f_{best}), the average objective value (f_{avg}), the worst objective value (f_{worst}), and the standard deviation (σ_f) of objective values. The row *Avg.* shows the average result over all instances tested for each column. The rows *#Better*, *#Equal*, and *#Worse* respectively show the number of instances for which our best result f_{best} is better than, equal to, worse than the BKV. Moreover, in terms of f_{best}, our improved results (new lower bounds) are indicated in bold and our worse results are indicated in italic compared to the *BKV*. Finally, to verify whether there exists a significant difference between our best results (f_{best}) and the BKV, we provide in the last row the p-values ($\in [0, 1]$) from the non-parametric Friedman test, where a p-value less than 0.05 means a significant difference between the compared results. This test was also performed to compare the results of the Almha algorithm and our average results (f_{avg})[4].

Table 2 shows that our TSTS algorithm matches the best known results for 47 out of 48 instances, and improves the best known result for the remaining one instance, leading to an improved *Avg.* value compared to the averaged *BKV* (18108.96 vs. 18108.10). When comparing the average objective values f_{avg} of our TSTS algorithm with the results of the Almba algorithm, one can find that both algorithms have a very similar performance, which is confirmed

[4] Since the results of Almha are based on only one run, we mainly use our average results for this comparison.

Table 3
Computational results and comparison on the 48 instances with $n = 250$. In terms of f_{best}, the improved results are indicated in bold and the worse results are indicated in italic compared to the best known objective value (*BKV*).

Instance	BKV	Almha	TSTS (this work)			
			f_{best}	f_{avg}	f_{worst}	σ_f
250-5-2-0-0	78486	78413	78486	78289.00	77644	146.76
250-5-2-0-1	75132	75086	75132	74833.33	73702	269.45
250-5-2-0-2	71003	70895	*70898*	70674.93	69762	285.03
250-5-2-0-3	80311	80227	80311	80206.57	80065	51.12
250-5-2-0-4	70935	70918	70935	70834.30	70583	70.55
250-5-2-0-5	130981	130863	*130191*	129271.40	127061	780.97
250-5-2-1-0	26666	26529	26666	26573.83	26457	54.46
250-5-2-1-1	26864	26778	26864	26806.77	26690	46.08
250-5-2-1-2	27280	27158	27280	27235.83	27109	47.32
250-5-2-1-3	26269	26160	*26250*	26173.90	26098	37.88
250-5-2-1-4	27293	27149	*27287*	27204.13	27131	25.01
250-5-2-1-5	44419	44216	*44395*	44302.57	44163	58.29
250-5-5-0-0	68026	68000	68026	68017.03	67978	15.64
250-5-5-0-1	60795	60727	*60766*	60627.90	60258	141.78
250-5-5-0-2	62093	62093	62093	62072.57	61960	28.50
250-5-5-0-3	66567	66513	66567	66519.80	66384	42.28
250-5-5-0-4	61929	61929	61929	61925.90	61878	11.66
250-5-5-0-5	127934	127890	*127922*	127708.10	127211	181.12
250-5-5-1-0	26966	26898	**26973**	26918.43	26853	31.01
250-5-5-1-1	26665	26520	26665	26576.10	26462	54.58
250-5-5-1-2	26648	26468	26648	26556.97	26403	49.58
250-5-5-1-3	25923	25701	*25885*	25784.30	25695	38.65
250-5-5-1-4	26021	25931	**26060**	25992.03	25882	41.71
250-5-5-1-5	41372	41131	*41338*	41237.67	41104	49.44
250-10-5-0-0	56306	56142	*56260*	55900.43	55344	274.20
250-10-5-0-1	59564	59504	**59619**	59551.47	59330	64.43
250-10-5-0-2	54898	54817	*54890*	54657.33	54367	109.54
250-10-5-0-3	52399	51987	*52249*	52105.73	51588	155.34
250-10-5-0-4	58234	57970	*58119*	57750.73	57113	291.28
250-10-5-0-5	99682	99452	*99512*	99201.73	98604	213.06
250-10-5-1-0	26867	26845	**26961**	26866.77	26716	59.78
250-10-5-1-1	26585	26441	**26658**	26538.87	26390	62.11
250-10-5-1-2	25737	25543	25737	25598.13	25322	83.93
250-10-5-1-3	27162	26982	*27159*	27089.60	26952	60.98
250-10-5-1-4	26816	26774	*26815*	26729.87	26635	46.01
250-10-5-1-5	46244	46087	46244	46145.40	46112	18.18
250-10-10-0-0	52441	52343	*52407*	52326.03	52045	113.59
250-10-10-0-1	53720	53603	**53745**	53663.80	53493	48.05
250-10-10-0-2	46927	46703	46927	46770.97	46487	81.50
250-10-10-0-3	54782	54779	**54831**	54745.93	54441	84.37
250-10-10-0-4	49675	49562	*49660*	49575.43	49327	72.88
250-10-10-0-5	92959	92792	**92975**	92821.53	92473	98.62
250-10-10-1-0	26696	26651	26696	26667.67	26564	40.85
250-10-10-1-1	25757	25692	**25876**	25786.43	25721	28.17
250-10-10-1-2	26356	26438	**26517**	26470.70	26418	32.17
250-10-10-1-3	26684	26443	26684	26614.77	26518	50.58
250-10-10-1-4	26554	26428	**26676**	26617.33	26511	28.13
250-10-10-1-5	42528	42284	**42629**	42464.80	42376	64.06
Avg.	49836.48	49717.81	49821.10	49687.60	49403.75	98.76
#Better			12			
#Equal			18			
#Worse			18			
p-value			2.7e−1	4.0e−2		

by a large p-value of 0.87. In addition, regarding the average results over all instances (*Avg.*), the value of f_{avg} of our TSTS algorithm is 18088.27, which is slightly better than that of Almba (i.e., 18083.17). These outcomes show that our TSTS algorithm performs similarly on these small instances $n = 100$ compared to the state-of-the-art Almba algorithm.

The second experiment aims to evaluate our TSTS algorithm on the set of 48 larger instances with $n = 250$, and the computational results are summarized in Table 3, where we report the same statistics as in Table 2. We observe from Table 3 that our algorithm matches the best known results for 18 out of 48 instances, improves the best known results for 12 instances, and misses the best known results for the remaining 18 instances. In terms of *Avg.*, the value of f_{best} of TSTS is slightly worse than the value of *BKV* (i.e., 49821.10 vs. 49836.48), and slightly better than the result of Almha (i.e., 49821.10 vs. 49717.81), which is however slightly better than the value of f_{avg} of TSTS (i.e., 49687.60 vs. 49717.81). This experiment indicates that under the short time limit $t_{max} = 60$ seconds, TSTS performs globally well on these larger instance with $n = 250$ especially by finding 12 improved best solutions. Additionally, we observe from Table 3 that TSTS performs particularly well for instances with a large number of constraints and achieves a better result than Almha for most of the 12 instances with 10 demand constraints and 10 knapsack constraints. Finally, we mention that the results of TSTS can be further improved by increasing the time limit (see the detailed results in the Appendix), implying that the current time limit ($t_{max} = 60$) is too short for the TSTS algorithm on these instances.

Table 4
Statistical results over 30 runs in terms of the number k of items in the obtained solutions.

$n = 100$				$n = 250$			
Instance	k_{best}	k_{avg}	σ_k	Instance	k_{best}	k_{avg}	σ_k
100-5-2-0-0	30	29.97	0.30	250-5-2-0-0	79	78.47	0.85
100-5-2-0-1	31	31.00	0.34	250-5-2-0-1	78	76.00	0.89
100-5-2-0-2	31	30.83	0.34	250-5-2-0-2	78	77.03	1.08
100-5-2-0-3	32	31.83	0.42	250-5-2-0-3	79	79.07	0.44
100-5-2-0-4	31	31.00	0.00	250-5-2-0-4	79	78.33	0.54
100-5-2-0-5	56	55.83	0.40	250-5-2-0-5	137	134.87	1.50
100-5-2-1-0	28	27.00	0.30	250-5-2-1-0	71	70.03	0.66
100-5-2-1-1	29	29.00	0.00	250-5-2-1-1	69	69.23	0.67
100-5-2-1-2	27	27.27	0.34	250-5-2-1-2	72	72.07	0.63
100-5-2-1-3	29	29.00	0.00	250-5-2-1-3	69	68.57	0.72
100-5-2-1-4	29	28.17	0.44	250-5-2-1-4	70	70.67	0.83
100-5-2-1-5	54	53.00	0.31	250-5-2-1-5	129	127.70	1.07
100-5-5-0-0	28	28.40	0.46	250-5-5-0-0	76	75.50	0.50
100-5-5-0-1	30	30.00	0.00	250-5-5-0-1	74	71.87	1.02
100-5-5-0-2	30	30.00	0.00	250-5-5-0-2	74	74.37	0.55
100-5-5-0-3	30	30.00	0.00	250-5-5-0-3	76	75.40	0.66
100-5-5-0-4	29	29.00	0.00	250-5-5-0-4	76	76.00	0.26
100-5-5-0-5	54	54.00	0.18	250-5-5-0-5	136	134.60	0.80
100-5-5-1-0	27	26.73	0.48	250-5-5-1-0	68	66.90	0.65
100-5-5-1-1	28	28.00	0.00	250-5-5-1-1	66	65.40	0.55
100-5-5-1-2	27	27.37	0.47	250-5-5-1-2	65	65.53	0.62
100-5-5-1-3	28	27.93	0.34	250-5-5-1-3	65	65.80	0.65
100-5-5-1-4	27	27.07	0.18	250-5-5-1-4	66	66.90	0.60
100-5-5-1-5	52	52.00	0.00	250-5-5-1-5	126	126.23	0.80
100-10-5-0-0	28	28.00	0.00	250-10-5-0-0	69	67.20	1.05
100-10-5-0-1	28	27.40	0.50	250-10-5-0-1	71	71.07	0.44
100-10-5-0-2	27	26.97	0.00	250-10-5-0-2	70	69.23	0.56
100-10-5-0-3	28	27.43	0.42	250-10-5-0-3	68	67.33	0.94
100-10-5-0-4	28	27.07	0.25	250-10-5-0-4	69	67.53	0.96
100-10-5-0-5	53	52.60	0.47	250-10-5-0-5	130	128.93	0.81
100-10-5-1-0	26	26.00	0.18	250-10-5-1-0	65	65.13	0.62
100-10-5-1-1	26	26.03	0.00	250-10-5-1-1	67	66.73	0.63
100-10-5-1-2	26	26.03	0.25	250-10-5-1-2	65	63.97	0.71
100-10-5-1-3	26	26.13	0.30	250-10-5-1-3	65	65.03	0.60
100-10-5-1-4	27	26.47	0.50	250-10-5-1-4	66	65.43	0.50
100-10-5-1-5	51	50.93	0.18	250-10-5-1-5	128	127.20	0.40
100-10-10-0-0	28	28.00	0.00	250-10-10-0-0	68	66.83	0.58
100-10-10-0-1	27	27.00	0.00	250-10-10-0-1	68	68.47	0.56
100-10-10-0-2	27	27.00	0.00	250-10-10-0-2	67	66.70	0.64
100-10-10-0-3	27	27.00	0.26	250-10-10-0-3	68	67.30	0.59
100-10-10-0-4	28	27.30	0.45	250-10-10-0-4	67	66.13	0.56
100-10-10-0-5	53	52.97	0.00	250-10-10-0-5	130	129.53	0.56
100-10-10-1-0	24	24.77	0.42	250-10-10-1-0	64	63.70	0.46
100-10-10-1-1	25	25.03	0.00	250-10-10-1-1	63	63.03	0.18
100-10-10-1-2	26	26.00	0.00	250-10-10-1-2	64	63.63	0.48
100-10-10-1-3	25	25.03	0.37	250-10-10-1-3	63	63.40	0.49
100-10-10-1-4	25	25.17	0.30	250-10-10-1-4	65	64.00	0.37
100-10-10-1-5	50	49.83	0.30	250-10-10-1-5	123	123.70	0.46
Avg.			0.22				0.66

4. Analysis and discussions

To shed light on the functioning of the proposed algorithm, we now analyze and discuss several essential components of the TSTS algorithm.

4.1. Effectiveness and robustness analysis for two-stage strategy

To study the effectiveness of the underlying two-stage search strategy, we summarize in Table 4 the computational results about the k values returned by the TSTS algorithm, where the results are based on the experiments in Section 3.3 and the value of k represents the number of selected items in the solution found. It is worth noting that the purpose of the first search stage is just to discover a promising hyperplane $\Omega_{[k]}$ that contains high quality solutions, and the second search stage aims to locate improved solutions in the given hyperplane. Hence, the two-stage search strategy can be considered to be relevant and robust if the first search stage is able to reach very stably the identical or close hyperplane to the best known solution (i.e., $\Omega_{[k_{best}]}$).

The results of small instances with $n = 100$ are reported in the first 4 columns of Table 4, including the name of instances, the k value of the best solution obtained over 30 runs (k_{best}), the average k value of solutions obtained, and the standard deviation of k values obtained (σ_k). The results of larger instances with $n = 250$ are reported in columns 5–8, with the same statistics as in columns 1–4. In addition, the row *Avg.* shows the average results of standard deviations σ_k of k values over all tested instances of each test set.

Table 4 shows that the value of k_{avg} is very close to that of k_{best} for most tested instances, which means that the first search stage of the TSTS algorithm is able to find a hyperplane that is very close to the best hyperplane containing the current best known solution. On the other hand, we observe that the standard deviations σ_k of k values obtained are very small for most instances. In particular, the average standard deviations of k values are respectively 0.22 and 0.66 for the set of small instances with $n = 100$ and the set of larger instances with $n = 250$. Hence, this experiment confirms

Table 5
Comparison between the two search stages on the instances with $n = 250$.

Instance	f_1	t_1(s)	f_2	t_2 (s)	$f_2 - f_1$	ρ
250-5-2-0-0	78029.83	9.60	78289.00	31.53	259.17	0.33
250-5-2-0-1	74599.40	11.44	74833.33	25.52	233.93	0.31
250-5-2-0-2	70402.30	13.79	70674.93	34.09	272.63	0.39
250-5-2-0-3	79927.80	4.89	80206.57	32.33	278.77	0.35
250-5-2-0-4	70555.47	2.91	70834.30	24.41	278.83	0.40
250-5-2-0-5	128985.57	11.40	129271.40	28.83	285.83	0.22
250-5-2-1-0	26322.20	3.98	26573.83	37.52	251.63	0.96
250-5-2-1-1	26522.33	4.43	26806.77	30.21	284.43	1.07
250-5-2-1-2	26945.23	4.75	27235.83	31.91	290.60	1.08
250-5-2-1-3	25892.27	4.91	26173.90	38.03	281.63	1.09
250-5-2-1-4	26997.30	5.39	27204.13	39.99	206.83	0.77
250-5-2-1-5	44068.70	5.99	44302.57	28.11	233.87	0.53
250-5-5-0-0	67902.80	7.89	68017.03	10.63	114.23	0.17
250-5-5-0-1	60341.40	7.58	60627.90	37.09	286.50	0.47
250-5-5-0-2	61948.87	5.17	62072.57	16.03	123.70	0.20
250-5-5-0-3	66363.73	7.16	66519.80	16.84	156.07	0.24
250-5-5-0-4	61803.67	3.57	61925.90	4.69	122.23	0.20
250-5-5-0-5	127473.87	10.57	127708.10	26.23	234.23	0.18
250-5-5-1-0	26574.67	2.38	26918.43	35.36	343.77	1.29
250-5-5-1-1	26197.30	2.56	26576.10	26.45	378.80	1.45
250-5-5-1-2	26129.73	3.26	26556.97	34.95	427.23	1.64
250-5-5-1-3	25386.40	2.84	25784.30	34.02	397.90	1.57
250-5-5-1-4	25586.80	2.34	25992.03	31.64	405.23	1.58
250-5-5-1-5	40721.77	4.86	41237.67	37.49	515.90	1.27
250-10-5-0-0	55376.00	10.35	55900.43	31.43	524.43	0.95
250-10-5-0-1	59234.87	7.83	59551.47	37.65	316.60	0.53
250-10-5-0-2	54262.60	8.68	54657.33	35.68	394.73	0.73
250-10-5-0-3	51626.67	9.97	52105.73	34.57	479.07	0.93
250-10-5-0-4	57155.03	8.66	57750.73	36.08	595.70	1.04
250-10-5-0-5	98688.90	13.45	99201.73	34.25	512.83	0.52
250-10-5-1-0	26410.57	5.71	26866.77	34.05	456.20	1.73
250-10-5-1-1	26189.03	4.29	26538.87	33.99	349.83	1.34
250-10-5-1-2	25149.17	5.15	25598.13	35.00	448.97	1.79
250-10-5-1-3	26671.27	3.54	27089.60	38.49	418.33	1.57
250-10-5-1-4	26317.50	5.12	26729.87	34.43	412.37	1.57
250-10-5-1-5	45787.90	6.45	46145.40	31.66	357.50	0.78
250-10-10-0-0	51902.63	8.52	52326.03	36.24	423.40	0.82
250-10-10-0-1	53349.40	4.96	53663.80	33.53	314.40	0.59
250-10-10-0-2	46352.07	7.58	46770.97	38.03	418.90	0.90
250-10-10-0-3	54397.33	9.80	54745.93	25.69	348.60	0.64
250-10-10-0-4	49214.60	5.47	49575.43	44.01	360.83	0.73
250-10-10-0-5	92424.20	9.18	92821.53	35.22	397.33	0.43
250-10-10-1-0	26297.53	4.05	26667.67	28.92	370.13	1.41
250-10-10-1-1	25386.40	4.20	25786.43	35.09	400.03	1.58
250-10-10-1-2	25963.63	5.58	26470.70	21.09	507.07	1.95
250-10-10-1-3	25996.40	5.25	26614.77	27.03	618.37	2.38
250-10-10-1-4	26132.07	2.62	26617.33	39.61	485.27	1.86
250-10-10-1-5	41890.40	9.74	42464.80	32.11	574.40	1.37
Avg.	49330.32	6.45	49687.60	31.41	357.28	0.96

to some extend the effectiveness and robustness of the two-stage search strategy employed by the TSTS algorithm.

4.2. Effects of two stages on the performance of algorithm

To investigate the respective role of the two stages of our algorithm, we carried out an experiment based on the instances with $n = 250$. We ran our TSTS algorithm 30 times to solve each instance according to the experimental protocol of Section 3.2. The average results from the first stage and the second stage over 30 independent runs are summarized in Table 5. The first column gives the name of instances, columns 2 and 3 show the objective value (f_1) obtained by the first stage and computation time (t_1) in seconds needed to reach f_1. Columns 4 and 5 show the objective value (f_2) obtained by the second stage and the computation time (t_2) needed to reach f_2 from f_1. The last two columns indicate the gap between f_2 and f_1 and the improvement ratio (ρ) of f_2 over f_1, which is calculated as $\rho = 100 \times (f_2 - f_1)/f_1$.

We observe from Table 5 that the first search stage of the TSTS algorithm is able to obtain a high-quality feasible solution for each instance and the solutions obtained in the first stage can be further improved during the second search stage. Furthermore, the improvements of f_2 over f_1 are significant with an average improvement ratio ρ close to 1%. On the other hand, regarding the computation times needed by the two search stages, we observe that most computational efforts are required by the second search stage and that t_2 is about five times larger than t_1. Of course, this proportion depends also on the setting of the parameters α and t_{max}. These outcomes indicate that both search stages of the TSTS algorithm are indispensable for the high performance of the algorithm. The first search stage is able to generate high-quality feasible solutions while the second search stage is able to further improve the solutions by performing an intensified search in the reached hyperplane.

4.3. Sensitivity analysis of hash functions

Now, we investigate the impacts of hash functions on the performance of the algorithm and discuss the sensitivity of the associated parameters. For this purpose, we carried out an additional experiment based on 30 representative instances in terms of the numbers of knapsack and demand constraints. We ran our TSTS algorithm 30 times for each instance and each parameter combination of (γ_1, γ_2, γ_3), where γ_i ($i = 1, 2, 3$) are the parameters used to define the hash functions h_i (see Section 2.3.2 for details). Specifically, we tested 10 different settings, i.e., (γ_1, γ_2, γ_3) ∈ {(1.1,1.5,1.8), (1.3,1.5,1.8), (1.3,1.5,2.0), (1.5,2.0,2.5), (1.6,1.8,2.0), (1.6,1.8,2.5), (1.8,2.0,2.2), (1.8,2.0,2.5), (1.9,2.1,2.3), (2.0,2.2,2.5)}.

The experimental results are summarized in Table 6, where the first column gives the name of instances, the second row shows the settings of parameters, and the average objective values (f_{avg}) obtained over 30 runs are reported in columns 2–11 for each parameter combination and each instance, respectively. In addition, the rows *#Best* and *Avg.* of the table indicate respectively the number of instances for which the associated setting of parameters yielded the best results and the average results over all instances tested.

We observe from Table 6 that the algorithm is sensitive to the setting of the parameters γ_1, γ_2, and γ_3. For the parameter combinations containing two small parameter values, the TSTS algorithm performed badly. For example, the algorithm with the combination (1.1,1.5,1.8) yielded the worst results in terms of *Avg.* However, when all parameters in (γ_1, γ_2, γ_3) take a relatively large value, the algorithm obtained a much better performance. Taking (1.9,2.1,2.3) as an example, the algorithm achieved the best results on 9 instances. In summary, the parameters γ_1, γ_2, and γ_3 have an important impact on the performance of the algorithm, and parameter combinations containing at least two large parameter values lead usually to a good performance of the algorithm.

4.4. Effectiveness of solution-based tabu strategy

The solution-based tabu strategy is an essential ingredient of our TSTS algorithm. To show its effectiveness with respect to the popular attribute-based tabu strategy, we created a variant A-TSTS of the TSTS algorithm by replacing the solution-based tabu strategy with the popular attribute-based tabu strategy, while keeping the other TSTS components unchanged. In A-TSTS, the adopted tabu strategy can be simply described as follows. Given an incumbent solution $s = (x_1, x_2, \ldots, x_n)$, once a 0–1 variable x_i ($1 \le i \le n$) is flipped as $x_i \leftarrow 1 - x_i$, x_i is forbidden to be flipped again for the next *tt* iterations (*tt* is the tabu tenure) and the associated neighbor solutions are excluded for consideration during the period identified by the tabu tenure. We empirically set $tt = C + rand[0, 2]$ where C is a parameter that takes the value of 20 and $rand[0, 2]$ is a random integer in [0,2]. Finally, the tabu status of a variable is disabled if flipping the variable leads to a solution better than

Table 6
Influence of the hash functions on the performance of algorithm. Each instance was independently solved 30 times for each parameter combination, and the average objective values (f_{avg}) over 30 runs are reported.

Instance/(γ_1, γ_2, γ_3)	f_{avg}									
	(1.1,1.5,1.8)	(1.3,1.5,1.8)	(1.3,1.5,2.0)	(1.5,2.0,2.5)	(1.6,1.8,2.0)	(1.6,1.8,2.5)	(1.8,2.0,2.2)	(1.8,2.0,2.5)	(1.9,2.1,2.3)	(2.0,2.2,2.5)
250-5-2-0-0	78221.23	78257.70	78240.20	78306.50	78222.97	78259.90	**78330.10**	78249.13	78271.70	78314.23
250-5-2-0-1	74711.67	74823.20	74810.57	74820.77	74749.43	74719.00	74796.47	74871.30	**74881.40**	74763.97
250-5-2-0-2	70668.73	70612.37	70589.13	70717.43	70522.20	**70733.50**	70683.43	70687.33	70615.77	70682.90
250-5-2-0-3	80152.93	80150.67	80179.63	80177.47	80192.50	**80205.67**	80200.03	80196.33	80182.67	80184.60
250-5-2-0-4	70805.20	70802.13	70828.47	70808.83	70827.30	70825.47	70836.80	70838.90	**70854.17**	70823.17
250-5-2-0-5	129152.17	129156.73	129101.97	128934.40	129141.57	129022.13	128910.60	129028.70	129205.90	**129245.57**
250-5-2-1-0	26544.90	26531.37	26547.13	26577.33	26561.60	26566.57	**26592.30**	26579.20	26576.87	26557.97
250-5-2-1-1	26744.33	26750.03	26764.23	26802.10	26789.80	26803.20	26794.60	**26817.83**	26812.87	26817.20
250-5-2-1-3	26125.63	26122.33	26143.10	26161.43	26154.03	26170.10	**26185.47**	26183.40	26185.40	26165.03
250-5-2-1-4	27164.03	27166.27	27185.90	27202.93	27206.27	27204.73	**27212.97**	27205.23	27208.03	27195.53
250-5-2-1-5	44252.07	44267.73	44284.83	44291.53	44277.13	44283.07	44300.17	44280.53	**44306.17**	44285.67
250-5-5-0-4	61911.17	61910.40	61914.80	61923.10	61918.37	61914.87	**61926.20**	61923.53	61920.00	61917.20
250-5-5-0-5	127545.60	127622.33	127673.70	**127740.23**	127643.83	127602.00	127718.67	127526.43	127618.57	127649.40
250-5-5-1-0	26870.87	26871.23	26878.20	26908.63	26898.83	26900.03	**26910.93**	26902.07	26908.87	26907.33
250-5-5-1-1	26527.90	26560.83	26570.03	26586.57	26575.47	26564.93	**26598.27**	26576.53	26593.87	26586.77
250-10-5-0-0	55855.80	55907.33	55882.50	**55930.33**	55907.27	55898.90	55888.23	55913.90	55843.83	55904.90
250-10-5-0-1	59484.63	59488.40	59496.33	59513.67	59533.07	59527.07	59534.97	59534.17	**59558.07**	59500.83
250-10-5-0-2	54558.87	54542.57	54553.77	54639.63	54603.23	54642.97	**54670.97**	54545.70	54586.17	54602.53
250-10-5-0-3	52070.43	52009.47	52082.17	52112.63	52065.67	52043.67	**52113.03**	52089.00	52082.63	52111.47
250-10-5-1-2	25530.00	25515.17	25562.50	25589.87	25567.83	25591.30	**25614.03**	25586.77	25594.57	25579.17
250-10-5-1-3	27024.40	26999.23	27057.57	**27082.40**	27068.90	27042.27	27078.20	27081.83	27080.10	27066.00
250-10-5-1-4	26675.00	26694.23	26718.73	26740.40	26717.37	26722.43	26725.93	26741.33	26715.90	**26758.10**
250-10-5-1-5	46090.07	46094.93	46122.93	46145.23	46123.77	46137.63	46129.57	46132.30	**46154.63**	46138.30
250-10-10-0-4	49488.50	49538.87	49524.60	49565.67	49525.07	49564.53	49548.50	49551.30	**49577.33**	49555.60
250-10-10-0-5	92757.03	92789.73	92795.13	92798.97	92813.90	92800.37	92831.43	92757.33	**92833.17**	92812.40
250-10-10-1-0	26610.33	26624.83	26645.80	26656.13	26656.17	26645.97	26654.03	26666.30	**26668.73**	26642.87
250-10-10-1-1	25727.50	25736.73	25758.33	25772.73	25754.33	25773.77	25772.23	25765.07	**25781.70**	25776.60
250-10-10-1-2	26439.20	26427.07	26461.90	26469.03	26463.20	26469.17	26463.10	26470.77	26464.57	**26473.93**
250-10-10-1-4	26546.37	26563.37	26572.87	26592.53	26590.93	**26602.63**	26598.90	26589.47	26600.03	26600.33
250-10-10-1-5	42384.30	42377.87	42418.47	42450.73	42431.67	42462.73	42438.90	**42472.50**	42455.53	42441.17
Avg.	51154.70	51163.84	51178.85	51200.64	51183.46	51190.02	51201.97	51192.14	**51204.64**	51202.02
#Best	0	0	0	3	0	3	10	2	**9**	3

all previously visited solutions (this is the so-called aspiration criterion).

To compare TSTS and A-TSTS, we carried out an experiment based on the set of 48 large instances with $n = 250$, where both algorithms were run 30 times according to the experimental protocol given in Section 3.2. The computational results are summarized in Table 7 where we show for each algorithm the best results (f_{best}) obtained over 30 runs, the average results (f_{avg}), and the worst results obtained (f_{worst}). In addition, the row 'Avg.' shows the average results for each performance indicator. Finally, to check whether there exists a significant difference between the two algorithms in terms of f_{best}, f_{avg} and f_{worst}, we provide the p-values from the non-parametric Friedman test in the last row, where a p-value smaller than 0.05 implies a significant difference between the compared results.

We observe from Table 7 that the solution-based algorithm TSTS dominates the attribute-based algorithm A-TSTS in terms of all indicators, by reporting better results in terms of f_{best}, f_{avg} and f_{worst} on all the instances. Furthermore, the small p-values indicate that the performance differences between the compared results are statistically significant. Therefore, this experiment confirms that under the two-stage framework of this work, the solution-based tabu strategy is much more suitable than the attribute-based tabu strategy for solving the MDMKP.

4.5. Discussion about the solution-based and attribute-based tabu search approaches

Attribute-based tabu search is a popular approach, whose key idea is to prevent one attribute or a combination of several attributes of a solution from being changed during a number of iterations since their last changes. For such a method, a tabu attribute or a combination of several attributes is usually associated with a number of tabu solutions. However, unlike attribute-based tabu search, solution-based tabu search tries to record all the visited solutions and prevent them from being revisited during the following search process. Thus, solution-based tabu search ensures a stronger intensification search ability.

Recent studies on several binary optimization cases including two dispersion problems (minimum difference dispersion Wang et al., 2017 and maximum min-sum dispersion Lai et al., 2018b) and the classic multidimensional knapsack problem (Lai et al., 2018a) demonstrate that solution-based tabu search is more suitable than attribute-based tabu search. On the other hand, our experience on another dispersion problem (max-mean dispersion Lai & Hao, 2016) does not confirm the advantage of the solution-based tabu search approach over the attribute-based tabu search approach. So one interesting question concerning the solution-based and attribute-based tabu search approaches is under what circumstances one approach will be more suitable than the other. Given that solution-based tabu search has been investigated only very recently, little knowledge is currently available, which makes it difficult to provide a meaningful guidance on the choice between these two approaches. To fully characterize these approaches and understand the relationships between these approaches and the optimization problem under consideration, more studies are clearly needed, which constitutes an interesting research perspective.

Finally, the ideas of the present solution-based tabu search algorithm being quite general, they could conveniently be tested on other binary optimization problems, by adjusting the γ parameters of the hash functions (Eq. (15), Section 2.3.2) or by increasing the number of the hash vectors and the associated hash functions.

Table 7
Comparison between the solution-based and attribute-based tabu strategies on the instances with $n = 250$. The better results between the two algorithms are indicated in bold in terms of f_{best}, f_{avg} and f_{worst}.

Instance	f_{best}		f_{avg}		f_{worst}	
	A-TSTS	TSTS	A-TSTS	TSTS	A-TSTS	TSTS
250-5-2-0-0	72278	**78486**	69459.47	**78289.00**	66794	**77644**
250-5-2-0-1	68945	**75132**	65578.57	**74833.33**	62546	**73702**
250-5-2-0-2	65744	**70898**	62624.13	**70674.93**	60403	**69762**
250-5-2-0-3	75654	**80311**	71754.33	**80206.57**	69139	**80065**
250-5-2-0-4	66935	**70935**	63119.17	**70834.30**	60155	**70583**
250-5-2-0-5	125571	**130191**	122346.53	**129271.40**	118507	**127061**
250-5-2-1-0	25126	**26666**	23250.10	**26573.83**	20310	**26457**
250-5-2-1-1	25352	**26864**	23872.30	**26806.77**	22806	**26690**
250-5-2-1-2	25204	**27280**	23651.70	**27235.83**	21274	**27109**
250-5-2-1-3	24796	**26250**	23342.57	**26173.90**	21765	**26098**
250-5-2-1-4	25474	**27287**	24349.20	**27204.13**	22944	**27131**
250-5-2-1-5	42388	**44395**	40612.43	**44302.57**	37348	**44163**
250-5-5-0-0	64755	**68026**	62158.80	**68017.03**	58973	**67978**
250-5-5-0-1	58390	**60766**	56174.20	**60627.90**	53842	**60258**
250-5-5-0-2	59571	**62093**	57389.37	**62072.57**	53856	**61960**
250-5-5-0-3	64279	**66567**	61548.97	**66519.80**	59335	**66384**
250-5-5-0-4	59003	**61929**	56954.03	**61925.90**	54735	**61878**
250-5-5-0-5	123840	**127922**	121678.73	**127708.10**	116272	**127211**
250-5-5-1-0	25188	**26973**	23643.70	**26918.43**	21024	**26853**
250-5-5-1-1	24472	**26665**	23106.03	**26576.10**	21433	**26462**
250-5-5-1-2	24316	**26648**	23054.23	**26556.97**	20703	**26403**
250-5-5-1-3	24165	**25885**	22694.20	**25784.30**	20186	**25695**
250-5-5-1-4	23813	**26060**	22712.63	**25992.03**	21016	**25882**
250-5-5-1-5	38977	**41338**	37637.83	**41237.67**	34688	**41104**
250-10-5-0-0	53124	**56260**	50822.33	**55900.43**	47977	**55344**
250-10-5-0-1	56060	**59619**	53640.83	**59551.47**	50706	**59330**
250-10-5-0-2	51944	**54890**	49772.90	**54657.33**	47349	**54367**
250-10-5-0-3	49409	**52249**	47341.53	**52105.73**	44605	**51588**
250-10-5-0-4	54681	**58119**	52606.43	**57750.73**	49637	**57113**
250-10-5-0-5	96521	**99512**	94044.77	**99201.73**	89840	**98604**
250-10-5-1-0	25431	**26961**	23844.97	**26866.77**	21589	**26716**
250-10-5-1-1	25399	**26658**	23638.70	**26538.87**	21920	**26390**
250-10-5-1-2	23836	**25737**	21624.83	**25598.13**	15478	**25322**
250-10-5-1-3	24940	**27159**	23633.13	**27089.60**	21621	**26952**
250-10-5-1-4	24819	**26815**	23390.37	**26729.87**	20190	**26635**
250-10-5-1-5	43814	**46244**	42276.70	**46145.40**	40483	**46112**
250-10-10-0-0	49931	**52407**	48520.97	**52326.03**	45896	**52045**
250-10-10-0-1	52102	**53745**	50398.53	**53663.80**	47944	**53493**
250-10-10-0-2	44797	**46927**	43186.07	**46770.97**	40408	**46487**
250-10-10-0-3	52271	**54831**	50908.37	**54745.93**	48603	**54441**
250-10-10-0-4	47965	**49660**	46648.33	**49575.43**	44642	**49327**
250-10-10-0-5	90501	**92975**	88609.13	**92821.53**	83712	**92473**
250-10-10-1-0	24671	**26696**	22845.10	**26667.67**	20131	**26564**
250-10-10-1-1	24350	**25876**	22838.50	**25786.43**	19149	**25721**
250-10-10-1-2	24341	**26517**	22613.70	**26470.70**	20116	**26418**
250-10-10-1-3	24577	**26684**	23209.00	**26614.77**	21611	**26518**
250-10-10-1-4	24679	**26676**	23352.07	**26617.33**	21524	**26511**
250-10-10-1-5	39576	**42629**	37427.70	**42464.80**	34366	**42376**
Avg.	47166.15	**49821.10**	45206.42	**49687.60**	42490.65	**49403.75**
p-value		4.26e−12		4.26e−12		4.26e−12

5. Conclusions

In this work, we investigated the NP-hard multidemand multidimensional knapsack problem, by proposing a two-stage tabu search (TSTS) algorithm that combines two solution-based tabu search procedures and a penalty-based evaluation function to explore different search spaces. Computational results on 156 benchmark instances showed that the proposed algorithm is competitive compared to the state-of-art results in the literature, especially for those instances with a large number of knapsack and demand constraints.

The experimental analysis showed the usefulness of the two-stage search strategy and the respective impacts of two stages on the performance of the algorithm. The first search stage is able to reach a promising hyperplane containing high-quality solutions, and the second search stage is able to find elite solutions by an intensified examination of the given hyperplane. We also showed that the hash functions used by the tabu search algorithms are a key component that significantly influences the performance of the algorithm.

This work enriches the existing tools for effectively solving the MDMKP and invites more research and attention on solution-based tabu search for solving binary optimization problems in the future. Specifically, given that the ideas of the two-stage strategy and the solution-based tabu search procedures developed in this work are quite general, it would be interesting to investigate their effectiveness on other problems, especially those related to subset selection problems for which the search space can be divided into a series of hyperplanes. It is also interesting to study search strategies mixing solution-based and attribute-based approaches. As a more fundamental research perspective, studies on a characterization of both solution-based and attribute-based search approaches are needed, which could ease the choice of one or the other approach to solve additional binary optimization problems.

Table A.1
Computational results of the TSTS algorithm on the instances with a large number of constraints under the condition presented in Section 3.2. For each of these instances, there are 30 knapsack constraints and 30 demand constraints.

Instance	f_{best}	f_{avg}	f_{worst}	σ_f	k_{best}	k_{avg}	σ_k
100-30-30-0-2-1	11312	11300.00	11252	24.00	25	25.20	0.40
100-30-30-0-2-2	9945	9945.00	9945	0.00	25	25.00	0.00
100-30-30-0-2-3	11195	11130.33	10225	241.96	25	25.07	0.25
100-30-30-0-2-4	11324	11290.40	11198	55.72	25	25.00	0.00
100-30-30-0-2-5	9704	9704.00	9704	0.00	25	25.00	0.00
100-30-30-0-2-6	23296	23296.00	23296	0.00	50	50.00	0.00
100-30-30-0-2-7	22442	22224.40	22126	125.49	51	50.27	0.44
100-30-30-0-2-8	23452	23312.90	23182	45.57	51	50.13	0.34
100-30-30-0-2-9	22756	22756.00	22756	0.00	50	50.00	0.00
100-30-30-0-2-10	24371	24323.60	24287	28.16	50	50.00	0.00
100-30-30-0-2-11	33472	33472.00	33472	0.00	75	75.00	0.00
100-30-30-0-2-12	32670	32670.00	32670	0.00	75	75.00	0.00
100-30-30-0-2-13	32942	32942.00	32942	0.00	75	75.00	0.00
100-30-30-0-2-14	35106	35106.00	35106	0.00	75	75.00	0.00
100-30-30-0-2-15	30930	30788.73	30767	55.41	75	75.00	0.00
100-30-30-1-5-1	5340	5340.00	5340	0.00	26	26.00	0.00
100-30-30-1-5-2	4390	4390.00	4390	0.00	25	25.00	0.00
100-30-30-1-5-3	4227	4227.00	4227	0.00	25	25.00	0.00
100-30-30-1-5-4	4706	4433.40	4424	50.62	25	25.97	0.18
100-30-30-1-5-5	2597	2591.90	2546	15.30	25	25.00	0.00
100-30-30-1-5-6	10808	10647.60	10462	92.10	50	50.00	0.00
100-30-30-1-5-7	9807	9789.37	9766	5.37	51	50.03	0.18
100-30-30-1-5-8	10882	10865.47	10753	42.27	50	50.10	0.30
100-30-30-1-5-9	10595	10595.00	10595	0.00	50	50.00	0.00
100-30-30-1-5-10	10297	10285.87	9963	59.95	50	50.03	0.18
100-30-30-1-5-11	11029	11029.00	11029	0.00	75	75.00	0.00
100-30-30-1-5-12	11884	11823.27	11747	45.74	75	75.00	0.00
100-30-30-1-5-13	10751	10751.00	10751	0.00	75	75.00	0.00
100-30-30-1-5-14	11567	11567.00	11567	0.00	75	75.00	0.00
100-30-30-1-5-15	10671	10368.47	10351	59.75	75	75.00	0.00
500-30-30-0-2-1	85188	84991.27	84418	185.69	128	127.57	0.62
500-30-30-0-2-2	82073	81856.17	81504	141.84	129	128.07	0.44
500-30-30-0-2-3	77393	76995.10	76516	205.93	129	127.40	0.80
500-30-30-0-2-4	82304	82049.27	81628	158.83	128	127.37	0.75
500-30-30-0-2-5	83525	83300.07	82779	170.66	129	127.87	0.72
500-30-30-0-2-6	145967	145705.17	145474	88.16	253	252.73	0.68
500-30-30-0-2-7	152246	152019.70	151665	132.72	253	252.67	0.79
500-30-30-0-2-8	157687	157487.13	157087	129.19	254	253.30	0.82
500-30-30-0-2-9	153751	153548.53	153365	87.25	254	252.33	0.75
500-30-30-0-2-10	142173	141943.90	141721	110.37	253	252.67	0.75
500-30-30-0-2-11	185226	184986.67	184781	91.72	377	376.97	0.55
500-30-30-0-2-12	194614	194444.30	194275	72.88	376	376.87	0.67
500-30-30-0-2-13	208246	208129.67	207904	76.15	378	377.53	0.72
500-30-30-0-2-14	215849	215693.30	215381	90.85	378	377.67	0.65
500-30-30-0-2-15	194224	194037.03	193788	91.74	376	376.57	0.62
500-30-30-1-5-1	51666	51574.93	51359	65.87	126	126.50	0.56
500-30-30-1-5-2	50101	49871.50	49566	123.11	126	126.33	0.54
500-30-30-1-5-3	51226	50979.60	50842	88.53	126	126.13	0.62
500-30-30-1-5-4	51637	51483.20	51261	91.69	127	126.77	0.42
500-30-30-1-5-5	52078	51860.97	51655	103.52	128	127.00	0.63
500-30-30-1-5-6	84052	83834.80	83548	104.57	251	251.03	0.41
500-30-30-1-5-7	82850	82637.00	82342	116.19	250	250.87	0.56
500-30-30-1-5-8	82722	82576.90	82419	74.14	250	250.70	0.46
500-30-30-1-5-9	82825	82451.43	81944	146.82	250	250.23	0.62
500-30-30-1-5-10	82845	82559.93	82098	143.15	249	249.67	0.60
500-30-30-1-5-11	88887	88756.90	88556	75.33	374	374.23	0.42
500-30-30-1-5-12	87254	87136.80	86815	82.34	374	374.20	0.65
500-30-30-1-5-13	87315	87167.80	87028	61.20	375	374.80	0.54
500-30-30-1-5-14	87583	87486.57	87261	89.89	374	373.90	0.65
500-30-30-1-5-15	87956	87814.97	87616	72.05	374	374.10	0.40
Avg.	62265.52	62139.10	61957.25	70.33	150.88	150.78	0.34

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).

Appendix A

This Appendix presents the detailed computational results of the proposed TSTS algorithm for the third and fourth sets of benchmark instances for which no detailed results are available in the literature. These instances have 100 or 500 items, 30 knapsack constraints and 30 demand constraints, making them more difficult to solve. For each of these instances, the TSTS algorithm

Table A.2
Computational results of the TSTS algorithm on the instances with $n = 250$ under a time limit of $t_{max} = 300$ seconds. In terms of f_{best}, the improved results are indicated in bold and the worse results are indicated in italic compared to the best known objective value (*BKV*) reported in the literature.

Instance	*BKV*	f_{best}	f_{avg}	f_{worst}	σ_f	k_{best}	k_{avg}	σ_k
250-5-2-0-0	78486	78486	78282.30	77986	173.14	79	78.17	0.73
250-5-2-0-1	75132	75132	74849.07	74152	232.22	78	76.10	0.94
250-5-2-0-2	71003	*70898*	70676.77	70137	217.86	78	76.87	0.92
250-5-2-0-3	80311	80311	80250.07	80131	53.66	79	79.10	0.54
250-5-2-0-4	70935	70935	70856.43	70805	63.06	79	78.40	0.49
250-5-2-0-5	130981	*130448*	129283.31	127720	686.96	138	134.23	1.36
250-5-2-1-0	26666	26666	26603.30	26504	47.41	71	70.23	0.62
250-5-2-1-1	26864	26864	26841.70	26727	35.40	69	68.97	0.55
250-5-2-1-2	27280	27280	27232.97	27052	54.88	73	72.30	0.86
250-5-2-1-3	26269	26269	26217.17	26148	37.72	69	68.63	0.84
250-5-2-1-4	27293	27293	27233.30	27184	29.95	70	70.40	0.84
250-5-2-1-5	44419	*44386*	44318.67	44022	90.92	129	127.97	1.40
250-5-5-0-0	68026	68026	68023.43	68019	3.37	76	75.63	0.48
250-5-5-0-1	60795	*60785*	60675.60	60456	82.68	73	72.27	0.68
250-5-5-0-2	62093	62093	62072.10	61960	35.00	74	74.23	0.56
250-5-5-0-3	66567	66567	66522.10	66400	41.43	76	75.27	0.57
250-5-5-0-4	61929	61929	61920.00	61878	18.10	76	75.93	0.44
250-5-5-0-5	127934	*127922*	127769.60	127240	187.71	136	134.67	0.70
250-5-5-1-0	26966	**26973**	26925.60	26869	31.43	68	66.93	0.77
250-5-5-1-1	26665	26665	26616.73	26520	59.80	66	65.73	0.57
250-5-5-1-2	26648	26648	26586.10	26536	22.47	65	65.63	0.48
250-5-5-1-3	25923	25923	25813.13	25684	64.51	66	65.83	0.78
250-5-5-1-4	26021	**26064**	26008.77	25823	44.13	67	66.93	0.63
250-5-5-1-5	41372	41372	41270.67	41129	53.23	126	126.30	0.64
250-10-5-0-0	56306	*56221*	56008.07	55430	158.01	69	67.67	0.65
250-10-5-0-1	59564	**59619**	59575.30	59447	52.03	71	71.00	0.52
250-10-5-0-2	54898	**54912**	54700.40	54367	179.52	70	69.17	0.78
250-10-5-0-3	52399	*52388*	52209.00	51975	98.42	68	67.57	0.67
250-10-5-0-4	58234	58234	57833.13	57156	222.30	69	67.60	0.71
250-10-5-0-5	99682	*99646*	99359.67	99023	176.86	130	129.07	0.73
250-10-5-1-0	26867	**26976**	26918.33	26766	55.74	66	65.03	0.48
250-10-5-1-1	26585	**26658**	26562.03	26486	35.34	67	66.73	0.68
250-10-5-1-2	25737	**25749**	25661.60	25515	59.69	64	64.20	0.60
250-10-5-1-3	27162	**27181**	27138.93	26971	46.61	65	64.93	0.36
250-10-5-1-4	26816	**26856**	26776.10	26706	55.74	66	65.47	0.50
250-10-5-1-5	46244	46244	46170.97	46137	23.64	128	127.10	0.30
250-10-10-0-0	52441	52441	52363.33	52171	78.20	68	67.07	0.57
250-10-10-0-1	53720	**53745**	53689.97	53607	33.25	68	68.60	0.49
250-10-10-0-2	46927	46927	46830.67	46546	87.89	67	66.47	0.62
250-10-10-0-3	54782	**54856**	54780.47	54507	60.51	68	67.50	0.56
250-10-10-0-4	49675	49675	49578.80	49342	104.58	67	66.03	0.71
250-10-10-0-5	92959	**92989**	92823.07	92465	141.18	130	129.27	0.73
250-10-10-1-0	26696	26696	26678.47	26606	28.58	64	63.63	0.48
250-10-10-1-1	25757	**25893**	25822.07	25771	29.03	62	63.07	0.44
250-10-10-1-2	26356	**26517**	26490.50	26438	35.14	64	63.70	0.46
250-10-10-1-3	26684	26684	26641.83	26598	37.16	63	63.47	0.50
250-10-10-1-4	26554	**26676**	26630.10	26603	18.02	65	64.10	0.30
250-10-10-1-5	42528	**42629**	42520.90	42306	70.96	123	123.80	0.48
Avg.	49836.48	49840.56	49721.10	49500.44	88.66	79.65	79.15	0.64
#Better		16						
#Equal		24						
#Worse		8						

was run 30 times under the condition presented in Section 3.2 and the computational results are summarized in Table A.1, where the symbols have the same meanings as in the previous tables.In addition, we report in Table A.2 the computational results of our TSTS algorithm on instances with $n = 250$ under a long time limit of $t_{max} = 300$ (instead of $t_{max} = 60$ used in Section 3). The results reported in this Appendix can serve as references for future comparative studies of new MDMKP algorithms.

We observe from Table A.1 that TSTS is able to reach the best result (f_{best}) with a success rate of 100% for 15 out of 30 small instances with $n = 100$, which means a good robustness of our TSTS algorithm on these instances. Note that for some small instances with $n = 100$, $m = 30$, and $q = 30$, it is difficult for some state-of-the-art algorithms in the literature (Hvattum et al., 2010) to obtain a feasible solution. However, it is very easy for our TSTS algorithm to obtain a feasible solution for all these instances. For other instances, the standard deviation (σ_f) of objective values obtained by the TSTS algorithm is relatively small, which indicates a good robustness of the algorithm. Regarding the value of k which represents the number of items selected, it can be seen that the gap between k_{best} and k_{avg} and the standard deviation (σ_k) of k values obtained are very small, implying that the two-stage strategy of the algorithm is very robust and effective.

Table A.2 shows that our TSTS algorithm improves the best known results for 16 out of 48 instances, matches the best known results for 24 instances, and misses the best known results for only 8 instances. These results indicate that the performance of our TSTS algorithm can be further improved when a longer computation time is available.

References

Arntzen, H., Hvattum, L. M., & Lokketangen, A. (2006). Adaptive memory search for multidemand multidimensional knapsack problems. *Computers and Operations Research, 33*, 2508–2525.

Balachandar, S. R. (2010). On efficient techniques for NP-hard problems. Ph.D. Thesis, Chapter 5, Sastra University, http://www.hdl.handle.net/10603/17553.

Beaujon, G. J., Marin, S. P., & McDonald, G. C. (2001). Balancing and optimizing a portfolio of r&d projects. *Naval Research Logistics, 48*(1), 18–40.

Cappanera, P., Gallo, G., & Maffioli, F. (2004). Discrete facility location and routing of obnoxious activities. *Discrete Applied Mathematics, 133*, 3–28.

Cappanera, P., & Trubian, M. (2005). A local search based heuristic for the demand constrained multidimensional knapsack problem. *INFORMS Journal on Computing, 17*(1), 82–98.

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

Carlton, W. B., & Barnes, J. W. (1996b). Solving the traveling salesman problem with time windows using tabu search. *IIE Transactions, 28*, 617–629.

Chen, Y., & Hao, J. K. (2017). An iterated "hyperplane exploration" approach for the quadratic knapsack problem. *Computers & Operations Research, 77*, 226–239.

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

Delissa, L. (2014). The existence and usefulness of equality cuts in the multidemand multidensional knapsack problem. Thesis, Kansas State University. http://www.krex.k-state.edu/dspace/handle/2097/17399.

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

Glover, F., & Kochenberger, G. A. (1996). Critical event tabu search for multidimensional knapsack problems. In I. H. Osman, & J. P. Kelly (Eds.), *Metaheuristics: theory and applications* (pp. 407–427). Boston, MA: Kluwer Academic Publishers.

Glover, F., & Laguna, M. (1997). *Tabu search*. Boston: Kluwer Academic Publishers.

Gortázar, F., Duarte, A., Laguna, M., & Martí, R. (2010). Black box scatter search for general classes of binary optimization problems. *Computers & Operations Research, 37*, 1977–1986.

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

Hvattum, L. M., Arntzen, H., Løkketangen, A., & Glover, F. (2010). Alternating control tree search for knapsack/covering problems. *Journal of Heuristics, 16*(3), 239–258.

Hvattum, L. M., & Løkketangen, A. (2007). Experiments using scatter search for the multidemand multidimensional knapsack problem. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. Gutjahr, R. F. Hartl, & M. Reimann (Eds.), *Metaheuristics. operations research/computer science interfaces series: Vol. 39*. Boston, MA: Springer.

Lai, X. J., & Hao, J. K. (2016). A tabu search based memetic search algorithm for the max-mean dispersion problem. *Computers & Operations Research, 72*, 118–127.

Lai, X. J., Hao, J. K., Glover, F., & Lü, Z. P. (2018a). A two-phase hybrid evolutionary algorithm for the 0–1 multidimensional knapsack problem. *Information Sciences, 436*, 282–301.

Lai, X. J., Yue, D., Hao, J. K., & Glover, F. (2018b). Solution-based tabu search for the maximum min-sum dispersion problem. *Information Sciences, 441*, 79–94.

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

Plastria, F. (2001). Static competitive facility location: An overview of optimisation approaches. *European Journal of Operational Research, 129*, 461–470.

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

Romero-Moraleses, D., Carrizosa, E., & Conde, E. (1997). Semi-obnoxious location models: A global optimization approach. *European Journal of Operational Research, 102*, 295–301.

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

Vasquez, M., & Hao, J. K. (2001). A hybrid approach for the 0–1 multidimensional knapsack problem. *Proceedings of the 17th international joint conference on artificial intelligence (IJCAI-01)* (pp. 328–333). Seattle, Washington, USA: Morgan Kaufmann Publishers. August

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

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.

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

Wishon, C., & Villalobos, J. R. (2016). Robust efficiency measures for linear knapsack problem variants. *European Journal of Operational Research, 254*(2), 398–409.

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

Yeniay, O. (2005). Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and Computational Applications, 10*(1), 45–56.