A threshold search based memetic algorithm for the disjunctively constrained knapsack problem

Zequn Wei and Jin-Kao Hao *

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France

Computers & Operations Research, June 2021

Abstract

The disjunctively constrained knapsack problem consists in packing a subset of pairwisely compatible items in a capacity-constrained knapsack such that the total profit of the selected items is maximized while satisfying the knapsack capacity. DCKP has numerous applications and is however computationally challenging (NP-hard). In this work, we present a threshold search based memetic algorithm for solving the DCKP that combines the memetic framework with threshold search to find high quality solutions. Extensive computational assessments on two sets of 6340 benchmark instances in the literature demonstrate that the proposed algorithm is highly competitive compared to the state-of-the-art methods. In particular, we report 24 and 354 improved best-known results (new lower bounds) for Set I (100 instances) and for Set II (6240 instances), respectively. We additionally apply the approach to solve a real-life daily photograph scheduling problem of an earth observation satellite. We analyze the key algorithmic components and shed lights on their roles for the performance of the algorithm.

Keywords: Knapsack problems; Disjunctive constraint; Threshold search; Heuristics.

1 Introduction

As a generalization of the conventional 0-1 knapsack problem (KP) [25], the disjunctively constrained knapsack problem (DCKP) is defined as follows. Let

* Corresponding author.
Email addresses: zequn.wei@gmail.com (Zequn Wei),
jin-kao.hao@univ-angers.fr (Jin-Kao Hao).

$V = \{1, \ldots, n\}$ be a set of n items, where each item $i \in \{1, \ldots, n\}$ has a profit $p_i > 0$ and a weight $w_i > 0$. Let $G = (V, E)$ be a conflict graph, where V is the set of n items and an edge $\{i, j\} \in E$ defines the incompatibility of items i and j. Let $C > 0$ be the capacity of a given knapsack. Then the DCKP involves finding a subset S of pairwisely compatible items of V to maximize the total profit of S while ensuring that the total weight of S does not surpass the knapsack capacity C. Let x_i be a binary variable such that $x_i = 1$ if item i is selected, $x_i = 0$ otherwise. Formally, the DCKP can be stated as follows.

$$(DCKP) \qquad \text{Maximize} \quad f(S) = \sum_{i=1}^{n} p_i x_i \tag{1}$$

$$\text{subject to} \quad W(S) = \sum_{i=1}^{n} w_i x_i \leq C, \; S \subseteq V, \tag{2}$$

$$x_i + x_j \leq 1, \forall (i, j) \in E, \tag{3}$$

$$x_i \in \{0, 1\}, i = 1, \ldots, n. \tag{4}$$

Objective function (1) commits to maximize the total profit of the selected item set S. Constraint (2) ensures that the knapsack capacity constraint is satisfied. Constraints (3), called disjunctive constraints, guarantee that two incompatible items are never selected simultaneously. Constraints (4) force that each item is selected at most once.

It is easy to observe that the DCKP reduces to the NP-hard KP when G is an empty graph. The DCKP is equivalent to the NP-hard maximum weighted independent set problem [24] when the knapsack capacity is unbounded. Moreover, the DCKP is closely related to other combinatorial optimization problems, such as multiple-choice knapsack [5,25], quadratic knapsack [7], and the bin packing with conflicts [14,23]. In addition to its theoretical significance, the DCKP is a useful model for practical applications, where the resources with conflicts cannot be used simultaneously while a given budget envelope must be respected.

Given the importance of the DCKP, a number of solution methods have been developed including exact, approximation and heuristic algorithms. As the literature review shown in Section 2, considerable progresses have been continually made since the introduction of the problem. Meanwhile, given the NP-hard nature of the problem, more powerful algorithms are still needed to push the limits of existing methods.

In this work, we investigate for the first time the population-based memetic framework [28] for solving the DCKP and design an effective algorithm mixing threshold based local optimization and crossover based solution recombination. The threshold search procedure ensures the main role of search intensification by finding high quality local optimal solutions. The specialized backbone crossover generates promising offspring solutions for search diversification. The algorithm uses also a distance-and-quality strategy for population management. The algorithm has the advantage of avoiding the difficult task of parameter tuning.

From a perspective of performance assessment, we apply the proposed algorithm to solve the two sets of DCKP benchmark instances in the literature. The results show that for the 100 instances of Set I (optimality still unknown) which were commonly tested by heuristic algorithms, our algorithm discovers 24 new best-known results (new lower bounds) and matches almost all other best-known results. For the 6240 instances of Set II which were tested by exact algorithms, our algorithm finds 354 improved best lower bounds on the difficult instances whose optimal values are unknown and attains the known optimal results on most of the remaining instances. To demonstrate its practical usefulness, we additionally apply the approach to solve a real-life daily photograph scheduling problem of an earth observation satellite (SPOT5).

The rest of the paper is organized as follows. Section 2 provides a literature review on the DCKP. Section 3 presents the proposed algorithm. Section 4 shows computational results of our algorithm and provides comparisons with the state-of-the-art algorithms. Section 5 shows how we use the proposed approach to solve the daily photograph scheduling application. Section 6 analyzes essential components of the algorithm. Finally, Section 7 summarizes the work and provides perspectives for future research.

2 Related work

The DCKP has attracted considerable attentions in the past two decades. In this section, we review related literature for solving the DCKP. Existing solution methods can be roughly classified into two categories as follows.

(1) *Exact and approximation algorithms*: These algorithms are able to guarantee the quality of the solutions they find. In 2002, Yamada et al. [42] introduced the DCKP and proposed the first implicit enumeration algorithm, where the disjunctive constraints are relaxed. In 2007, Hifi and Michrafy [21] introduced three versions of an exact algorithm based on a local reduction strategy. In 2009, Pferschy and Schauer [29] proposed a pseudo-polynomial time and space algorithm for solving three special cases of the DCKP and proved the DCKP is strongly NP-hard on perfect graphs. In 2016, Salem et al. [34] developed a branch-and-cut algorithm that combines a greedy clique generation procedure with a separation procedure. In 2017, Bettinelli et al. [3]

presented a branch-and-bound algorithm by combining a upper bounding procedure that considers both the capacity constraint and the disjunctive constraints with a branching procedure that employs a dynamic programming to presolve the 0-1 KP. They generated 4800 DCKP instances with conflict graph densities between 0.001 and 0.9 (see Section 4.1). Also in 2017, Pferschy and Schauer [30] applied the approximation methods of modular decompositions and clique separators to the DCKP, and showed complexity results on special graph classes. In 2019, Gurski and Rehs [17] designed a dynamic programming algorithm and achieved pseudo-polynomial solutions for the DCKP. In 2020, Coniglio et al. [8] presented another branch-and-bound algorithm based on an n-ary branching scheme and solved the integer linear programming formulations of the DCKP by the CPLEX solver. They introduced 1440 new and challenging DCKP instances (see Section 4.1).

(2) *Heuristic algorithms*: These algorithms aim to find good near-optimal solutions with a given time. In 2002, Yamada et al. [42] proposed a greedy algorithm to generate an initial solution and a 2-opt neighborhood search algorithm to improve the obtained solution. In 2006, Hifi and Michrafy [20] reported a local search algorithm, which combines a complementary constructive procedure to improve the initial solution and a degrading procedure to diversify the search. They generated a set of 50 DCKP instances with 500 and 1000 items (see Section 4.1), which was widely tested in later studies. In 2012, Hifi and Otmani [22] studied two scatter search algorithms. In 2014, Hifi [19] devised an iterative rounding search-based algorithm that uses a rounding strategy to perform a linear relaxation of the fractional variables. In 2017, Salem et al. [33] designed a probabilistic tabu search algorithm (PTS) that explores multiple neighborhoods in a probabilistic way. In the same year, Quan and Wu investigated two parallel algorithms: the parallel neighborhood search algorithm (PNS) [32] and the cooperative parallel adaptive neighborhood search algorithm (CPANS) [31]. They also designed a new set of 50 DCKP large instances with 1500 and 2000 items (see Section 4.1).

Existing studies have significantly contributed to better solving the DCKP. According to the computational results reported in the literature, the parallel neighborhood search algorithm [32], the cooperative parallel adaptive neighborhood search algorithm [31], and the probabilistic tabu search algorithm [33] can be regarded as the state-of-the-art methods for the instances of Set I. For the instances of Set II, the branch-and-bound algorithms presented in [3,8] and the integer linear programming formulations solved by the CPLEX solver [8] showed the best performance.

In this work, we aim to advance the state-of-the-art of solving the problem by proposing the first threshold search based memetic approach, which proves to be effective on the two sets of DCKP instances tested in the literature.

3 Threshold search based memetic algorithm for the DCKP

Our threshold search based memetic algorithm (TSBMA) for the DCKP is a population-based algorithm combining evolutionary search and local optimization. In this section, we first present the general procedure of the algorithm and then describe its components.

3.1 General procedure

The TSBMA algorithm relies on the general memetic algorithm framework [28] and follows the design principles recommended in [18]. The flowchart of TSBMA and its pseudo-code are shown in Figure 1 and Algorithm 1, respectively.

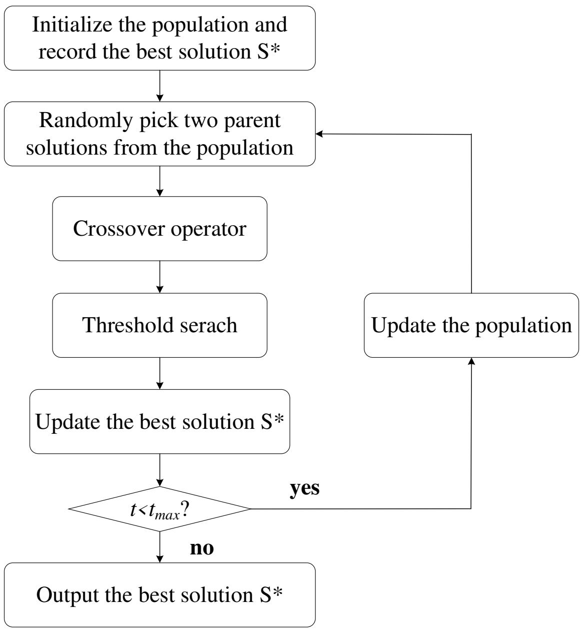

Fig. 1. Flowchart of the proposed TSBMA algorithm.

The algorithm starts from a set of feasible solutions of good quality that are generated by the population initialization procedure (line 4, Alg. 1, and Section 3.3). The best solution is identified and recorded as the overall best solution S^* (line 5, Alg. 1). Then the algorithm enters the main "while" loop (lines 6-15, Alg. 1) to perform a number of generations. At each generation, two solutions are randomly picked and used by the crossover operator to create an offspring solution (line 7-8, Alg. 1, and Section 3.5). Afterwards, the threshold search procedure is triggered to perform local optimization with three neighborhoods N_1, N_2 and N_3 (line 9, Alg. 1, and Section 3.4). After conditionally updating the overall best solution S^* (lines 11-13, Alg. 1), the diversity-based pool updating procedure is applied to decide whether the best solution S_b found during the threshold search should be inserted into the population (line 14, Alg. 1, and Section 3.6). Finally, when the given time limit t_{max} is reached, the algorithm returns the overall best solution S^* found during the search and terminates.

Algorithm 1 Main framework of threshold search based memetic algorithm for the DCKP

1: **Input**: Instance I, cut-off time t_{max}, population P, the maximum number of iterations $IterMax$, neighborhoods N_1, N_2, N_3.
2: **Output**: The overall best solution S^* found.
3: $S^* \leftarrow \emptyset$ /* Initialize S^* (i.e., $f(S^*) = 0$)*/
4: $POP = \{S^1, \ldots, S^{|P|}\} \leftarrow Population\_Initialization(I)$ /* Section 3.3 */
5: $S^* \leftarrow argmax\{f(S^k)|k = 1, \ldots, p\}$
6: **while** $Time \leq t_{max}$ **do**
7: Randomly pick two solutions S^i and S^j from the population POP
8: $S^o \leftarrow Crossover\_Operator(S^i, S^j)$ /* Section 3.5 */
9: $S_b \leftarrow Threshold\_Search(S^o, N_{1-3}, IterMax)$ /* Section 3.4 */
10: /* Record the best solution S_b found during threshold search */
11: **if** $f(S_b) > f(S^*)$ **then**
12: $S^* \leftarrow S_b$ /* Update the overall best solution S^* found so far */
13: **end if**
14: $POP \leftarrow Pool\_Updating(S_b, POP)$ /* Section 3.6 */
15: **end while**
16: **return** S^*

3.2 Solution representation, search space, and evaluation function

The DCKP is a subset selection problem. Thus, a candidate solution for a set $V = \{1, \ldots, n\}$ of n items can be conveniently represented by a binary vector $S = (x_1, \ldots, x_n)$, such that $x_i = 1$ if item i is selected, and $x_i = 0$ otherwise. Equivalently, S can also be represented by $S =< A, \bar{A} >$ such that $A = \{q : x_q = 1 \text{ in } S\}$ and $\bar{A} = \{p : x_p = 0 \text{ in } S\}$.

Let $G = (V, E)$ be the given conflict graph and C be the knapsack capacity. Our TSBMA algorithm explores the following feasible search space Ω^F satisfying both the disjunctive constraints and the knapsack constraint.

$$\Omega^F = \{x \in \{0,1\}^n : \sum_{i=1}^{n} w_i x_i \leq C; x_i + x_j \leq 1, \forall \{i,j\} \in E, 1 \leq i,j \leq n, i \neq j\} \tag{5}$$

The quality of a solution S in Ω^F is determined by the objective value $f(S)$ of the DCKP (Equation 1).

3.3 Population initialization

As shown in Algorithm 2, the TSBMA algorithm builds each of the $|P|$ initial solutions of the population P in two steps. First, it randomly adds one by one non-selected items into an individual solution S^i ($i = 1, \ldots, |P|$) until the capacity of the knapsack is reached, while keeping the disjunctive constraints satisfied (line 5, Alg. 2). Second, to obtain an initial population of reasonable quality, TSBMA improves the solution S^i by a short run of the threshold search procedure presented in Section 3.4 (line 5, Alg. 2) by setting the maximum consecutive iterations $IterMax = 2n$, where n is the number of items in the instance. The population initialization procedure terminates when $|P|$ initial solutions are generated and added into the population P.

Algorithm 2 Population initialization procedure

1: **Input**: Instance I, population size $|P|$, maximum number of iterations $IterMax$, neighborhoods N_1, N_2, N_3.
2: **Output**: Initial population P.
3: $0 \leftarrow i$
4: **while** $i \leq |P|$ **do**
5: $S^i \leftarrow Random\_Initial(I)$ /* S_i is the initial solution */
6: $S^i \leftarrow Threshold\_Search(S^i, N_{1-3}, IterMax)$ /* Improve the solution S_i */
7: Add the improved solution S^i into the population P
8: $i \leftarrow i + 1$
9: **end while**
10: **return** P

It is worth mentioning that the population size $|P|$ is determined according to the number of candidate items n of the given instance, i.e., $|P| = n/100 + 5$. This strategy is based on two considerations. First, since the TSBMA algorithm is powerful enough to solve the instances of small size, a smaller population size can help to reduce the initialization time. Second, the instances of large size are more challenging, a larger population size helps to diversify the search.

3.4 Local optimization using threshold search

The local optimization procedure of the TSBMA algorithm relies on the threshold accepting method [12]. To explore a given neighborhood, the method accepts both improving and deteriorating neighbor solutions so long as the solution satisfies a quality threshold. One notices that this method has been successfully applied to solve several knapsack problems (e.g., quadratic multiple knapsack problem [6], multi-constraint knapsack problem [13] and multiple-choice knapsack problem [44]) and other combinatorial optimization problems (e.g., [4,35]). In this work, we adopt for the first time this method for solving the DCKP and devise a multiple neighborhood threshold search procedure reinforced by an operation-prohibiting mechanism.

3.4.1 Main scheme of the threshold search procedure

As shown in Algorithm 3, the threshold search procedure (TSP) starts its process from an input solution and three empty hash vectors (used for the operation-prohibiting mechanism, lines 3-5, Alg. 3). It then performs a number of iterations to explore three neighborhoods (Section 3.4.2) to improve the current solution S. Specifically, for each "while" iteration (lines 9-25, Alg. 3), the TSP procedure explores the neighborhoods N_1, N_2 and N_3 in a deterministic way as explained in the next section. Any sampled non-prohibited neighbor solution S' (i.e., $H_1[h_1(S')] \wedge H_2[h_2(S')] \wedge H_3[h_3(S')]$ $= 0$, see Section 3.4.3) is accepted immediately if the quality threshold T is satisfied (i.e., $f(S') \geq T$). Then the hash vectors are updated for solution prohibition and the best solution found during the TSP procedure is recorded in S_b (lines 18-20, Alg. 3). The main search ("while" loop) terminates when 1) no admissible neighbor solution (i.e., non-prohibited and satisfying the quality threshold) exists in the neighborhoods N_1, N_2 and N_3, or 2) the best solution S_b cannot be further improved during $IterMax$ consecutive iterations. Specifically, the quality threshold T is determined adaptively by $f(S_b) - n/10$ (n is the number of items of each instance) while $IterMax$ is set to $(n/500 + 5) \times 10000$.

3.4.2 Neighborhoods and their exploration

The TSP procedure examines candidate solutions by exploring three neighborhoods induced by the popular move operators: *add*, *swap* and *drop*. Let S be the current solution and mv is one of these operators. We use $S' = S \oplus mv$ to denote a feasible neighbor solution obtained by applying mv to S and N_x ($x = 1, 2, 3$) to represent the resulting neighborhoods. To avoid the examination of unpromising neighbor solutions, TSP employs the

Algorithm 3 Threshold search procedure

Input: Input solution S^o, threshold T, the maximum number of iterations $IterMax$, hash vectors H_1, H_2, H_3, hash functions h_1, h_2, h_3, length of hash vectors L, neighborhoods N_1, N_2, N_3.
Output: The best feasible solution S_b found by threshold search procedure.
for $i \leftarrow 0$ **to** $L-1$ **do**
  $H_1[i] \leftarrow 0$; $H_2[i] \leftarrow 0$; $H_3[i] \leftarrow 0$; /* Initialization of hash vectors */
end for
$S_b \leftarrow S^o$ /* S_b record the best solution found */
$S \leftarrow S^o$ /* S record the current solution */
$iter \leftarrow 0$
while $iter \leq IterMax$ **do**
  /* Examine the neighborhoods $N_1(S)$, $N_2(S)$, $N_3(S)$ in a token-ring way; Section 3.4.2 */
  for Each non-prohibited S' of $N_1(S)$ or $N_2(S)$ or $N_3(S)$ **do**
    if $f(S') \geq T$ **then**
      $S \leftarrow S'$
      /* Update the hash vectors with S, Section 3.4.3 */
      $H_1[h_1(S)] \leftarrow 1$; $H_2[h_2(S)] \leftarrow 1$; $H_3[h_3(S)] \leftarrow 1$
      break;
    end if
  end for
  if $f(S) > f(S_b)$ **then**
    $S_b \leftarrow S$ /* Update the best solution S_b found during threshold search */
    $iter \leftarrow 0$
  else
    $iter \leftarrow iter + 1$
  end if
end while
return S_b

following dynamic neighborhood filtering strategy inspired by [27,39]. Let S' be a neighbor solution in the neighborhood currently under examination, and S_c be the best neighbor solution encountered during the current neighborhood examination. Then S' is excluded for consideration if it is no better than S_c (i.e., $f(S') \leq f(S_c)$). By eliminating the unpromising neighbor solutions, TSP increases the efficiency of its neighborhood search.

Specifically, the associated neighborhoods induced by *add*, *swap* and *drop* are defined as follows.

- $add(p)$: This move operator expands the selected item set A by one non-selected item p from the set $\bar{A}$ such that the resulting neighbor solution is feasible. This operator induces the neighborhood N_1.

$$N_1(S) = \{S' : S' = S \oplus add(p), p \in \bar{A}\} \tag{6}$$

- *swap*(q, p): This move operator exchanges a pair of items (q, p), where item q belongs to the selected item set A and p belongs to the non-selected item set $\bar{A}$ such that the resulting neighbor solution is feasible. This operator induces the neighborhood N_2.

$$N_2(S) = \{S' : S' = S \oplus swap(q, p), q \in A, p \in \bar{A}, f(S') > f(S_c)\} \quad (7)$$

- *drop*(q): This operator displaces one selected item q from the set A to the non-selected item set $\bar{A}$ and induces the neighborhood N_3.

$$N_3(S) = \{S' : S' = S \oplus drop(q), q \in A, f(S') > f(S_c)\} \quad (8)$$

One notices that the *add* operator always leads to a better current solution with an additional eligible item, and thus the neighborhood filtering is not needed for N_1. The *drop* operator always deteriorates the quality of the current solution, and the feasibility of a neighbor solution is always ensured. The *swap* operator may either increase or decrease the objective value and the feasibility of a neighbor solution needs to be verified. For N_2 and N_3, neighborhood filtering excludes uninteresting solutions that can in no way be accepted during the TSP process.

The TSP procedure examines the neighborhoods N_1, N_2, and N_3 in a sequential way to explore different local optimal solutions. For N_1, as long as there exists a non-prohibited neighbor solution, TSP selects such a neighbor solution to replace the current solution (ties are broken randomly). Once N_1 becomes empty, TSP moves to N_2, if there exists a non-prohibited neighbor solution S' satisfying $f(S') \geq T$, TSP selects S' to become the current solution and immediately returns to the neighborhood N_1. When N_2 becomes empty, TSP continues its search with N_3 and explores N_3 exactly like with N_2. When N_3 becomes empty, TSP terminates its search and returns the best solution found S_b. TSP may also terminate if its best solution remains unchanged during $IterMax$ consecutive iterations.

3.4.3 Operation-prohibiting mechanism

During the TSP procedure, it is important to prevent the search from revisiting a previously encountered solution. For this purpose, TSP utilizes an operation-prohibiting (OP) mechanism that is based on the tabu list strategy [16]. To implement the operation-prohibiting (OP) mechanism, we adopt the solution-based tabu search technique [41]. Specifically, we employ three hash vectors H_v $(v = 1, 2, 3)$ of length L $(|L| = 10^8)$ to record previously visited solutions.

Given a solution $S = (x_1, \ldots, x_n)$ $(x_i \in \{0, 1\})$, we pre-compute the weights $\mathcal{W}_i^v$ $(v = 1, 2, 3)$ for each item i by $\mathcal{W}_i^1 = i^{1.2}, \mathcal{W}_i^2 = i^{1.6}$, and $\mathcal{W}_i^3 = i^{2.0}$. Then the hash values of solution S are given by the following hash functions h_v

$(v = 1, 2, 3)$.

$$h_1(S) = (\sum_{i=1}^{n} \lfloor \mathcal{W}_i^1 \times x_i \rfloor) \bmod |L| \tag{9}$$

$$h_2(S) = (\sum_{i=1}^{n} \lfloor \mathcal{W}_i^2 \times x_i \rfloor) \bmod |L| \tag{10}$$

$$h_3(S) = (\sum_{i=1}^{n} \lfloor \mathcal{W}_i^3 \times x_i \rfloor) \bmod |L| \tag{11}$$

The hash values of a neighbor solution S' from the $add(p)$, $swap(q, p)$ or $drop(q)$ operator (see Section 3.4.2) can be efficiently computed as follows.

$$h_v(S') = \begin{cases} h_v(S) + \mathcal{W}_p^v, & \text{for } add(p) \\ h_v(S) - \mathcal{W}_q^v + \mathcal{W}_p, & \text{for } swap(q, p) \\ h_v(S) - \mathcal{W}_q^v, & \text{for } drop(q) \end{cases} \tag{12}$$

where v is equal to $1, 2, 3$, $\mathcal{W}_q^v$ and $\mathcal{W}_p^v$ are the pre-computed weights of items p and q involved in the move operations.

Starting with the hash vectors set to 0, the corresponding positions in the three hash vectors H_v $(v = 1, 2, 3)$ are updated by 1 whenever a new neighbor solution S' is accepted to replace the current solution S (lines 12-16, Alg. 3). For each candidate neighbor solution S', its hash values $h_v(S')$ $(v = 1, 2, 3)$ are calculated with Equation (12) in $O(1)$. Then, this neighbor solution S' is previously visited if $H_1[h_1(S')] \wedge H_2[h_2(S')] \wedge H_3[h_3(S')] = 1$ and is prohibited from consideration by the TSP procedure.

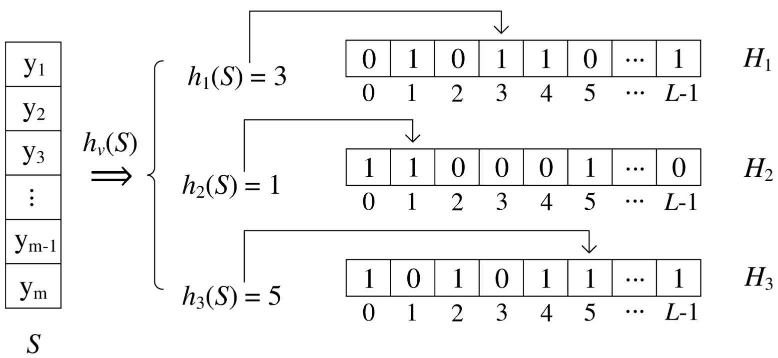

Fig. 2. An example of the operation-prohibiting mechanism [40].

Fig. 2 from [40] illustrates the hash based operation-prohibiting mechanism. In this example, applying the three hash functions to the given solution S leads to hash values $h_1(S) = 3$, $h_2(S) = 1$ and $h_3(S) = 5$. Checking the three hash vectors with these hash values indicates that S is a prohibited solution since $H_1[3] \wedge H_2[1] \wedge H_3[5] = 1$.

Note that a hashing-memory list was adopted in [20] to distinguish solutions with the same objective value. This method is clearly different from our operation-prohibiting mechanism because we use the hash vectors to record all the solutions encountered during the TSP procedure, rather than just the solutions with the same objective value. Moreover, unlike [20] that requires another move-memory list to prevent the search from revisiting previously encountered solutions, our approach does not need such an additional structure.

3.5 Crossover operator

The crossover operator generally creates new solutions by recombining two existing solutions. For the DCKP, we adopt the idea of the double backbone-based crossover (DBC) operator [43] and adapt it to the problem.

Given two solutions S^i and S^j, we use them to divide the set of n items into three subsets: the common items set $X_1 = S^i \cap S^j$, the unique items set $X_2 = (S^i \cup S^j) \setminus (S^i \cap S^j)$ and the unrelated set $X_3 = V \setminus (S^i \cup S^j)$. The basic idea of the DBC operator is to generate an offspring solution S^o by selecting all items in set X_1 (the first backbone) and some items in set X_2 (the second backbone), while excluding items in set X_3.

As shown in Algorithm 4, from two randomly selected parent solutions S^i and S^j, the DBC operator generates S^o in three steps. First, we initialize S^o by setting all the variables x^o_a ($a = 1, \ldots, n$) to 0 (line 3, Alg. 4). Second, we identify the common items set X_1 and the unique items set X_2 (line 4-10, Alg. 4). Third, we add all items belonging to X_1 into S^o and randomly add items from X_2 into S^o until the knapsack constraint is reached (line 11-17, Alg. 4). Note that the knapsack and disjunctive constraints are always satisfied during the crossover process.

Since the DCKP is a constrained problem, the DBC operator adopted for TSBMA has several special features to handle the constraints, which is different from the DBC operator introduced in [43]. First, we iteratively add an item into S^o by selecting one item from the unique items set X_2 randomly until the knapsack constraint is reached, while each item in X_2 is considered with a probability p_0 ($0 < p_0 < 1$) in [43]. Second, unlike [43], where a repair operation is used to achieve a feasible offspring solution, our DBC operator ensures the satisfaction of the problem constraints during the offspring generation process.

3.6 Population updating

Once a new offspring solution is obtained by the DBC operator in the last section, it is further improved by the threshold search procedure presented in Section 3.4. Then we adopt a diversity-based population updating strategy [26] to decide whether the improved offspring solution should replace an existing solution in the population.

Algorithm 4 The double backbone-based crossover operator

Input: Two parent solutions $S^i = (x_1^i, x_2^i, \dots, x_n^i)$ and $S^j = (x_1^j, x_2^j, \dots, x_n^j)$.
Output: An offspring solution $S^o = (x_1^o, x_2^o, \dots, x_n^o)$.
$S^o \leftarrow \emptyset$ /* Initialize S^o (i.e., $f(S^o) = 0$)*/
for $a \leftarrow 1$ **to** n **do**
    if $x_a^i = 1$ and $x_a^j = 1$ **then**
        $X_1 \leftarrow a$ /* X_1 is the common items set */
    else if $x_a^i = 1$ or $x_a^j = 1$ **then**
        $X_2 \leftarrow a$ /* X_2 is the unique items set */
    end if
end for
$S^o \leftarrow X_1$ /* Add all items belonging to X_1 into S^o */
Randomly shuffle all items in X_2;
for each $a \in X_2$ **do**
    if $S^o \cup (x_a^o = 1)$ is a feasible solution **then**
        $x_a^o \leftarrow 1$ /* The second backbone */
    end if
end for
return S^o

This strategy is beneficial to balance the quality of the offspring solution and its distance from the population.

To accomplish this task, we temporarily insert the improved offspring solution into the population and compute the distance (Hamming distance) between any two solutions in the population. Then we obtain the goodness score of each solution in the same way as proposed in [26]. Finally, the worst solution in the population is identified according to the goodness score and deleted from the population.

3.7 Time complexity of TSBMA

As shown in Section 3.3, the population initialization procedure includes two steps. Given a DCKP instance with n items, the first step of random selection takes time $O(n)$. Given an input solution $S =< A, \bar{A} >$ (see Section 3.2), the complexity of one iteration of the TSP procedure is $O((n + |A| \times |\bar{A}|))$. Then the second step of the initialization procedure can be realized in $O([(n + |A| \times |\bar{A}|)] \times IterMax)$, where $IterMax$ is set to $2n$ in the initialization procedure. The complexity of the population initialization procedure is $O(n^3)$.

Now we consider the four procedures in the main loop of the TSBMA algorithm: parent selection, crossover operator, the TSP procedure and population updating. The parent selection procedure is realized in $O(1)$. The crossover operator takes time $O(n)$. The complexity of the TSP procedure is $O([(n+|A| \times |\bar{A}|)] \times IterMax)$, where $IterMax$ is determined in Section 3.4.1. The population updating procedure can be achieved in $O(n|P|)$, where $|P|$ is the population size. Then, the complexity of one iteration of the main loop of the TSBMA algorithm is $O(n^2 \times IterMax)$.

3.8 Discussions

The proposed algorithm is based on the general memetic search and threshold search methods, and integrates a number of carefully designed problem-specific features. In what follows, we highlight the novelties and contributions of the presented work.

First, compared to the existing DCKP algorithms reviewed in Section 2, the primary novelty of our approach concerns the design of the threshold search procedure detailed in Section 3.4. This is the first local optimization procedure that adapts the general threshold accepting method to the DCKP. In particular, it employs an original neighborhood exploration strategy that relies on 1) a neighborhood filtering to eliminate non-promising neighboring solutions, 2) a hash function based prohibiting technique to avoid revisiting already encountered solutions, and 3) a token-ring policy to examine the three neighborhoods.

Second, the proposed algorithm reinforces its search capacity by adopting a specifically designed crossover operator (see Section 3.5), which is able to cope with the disjunctive constraints of the DCKP. It additionally adopts a distance-and-quality based population management method to maintain a healthy population.

Third, as the extensive computational results on two sets of 6340 benchmark instances indicate shown in Section 4, the algorithm integrating the above features reaches a high performance that no existing algorithm can compete. In particular, it reports a number of new lower bounds that are valuable for future research on the DCKP.

Fourth, we demonstrate the practical usefulness of our approach for solving real-life problems. For this, we present in Section 5 a real application, i.e., the daily photograph scheduling problem (DPSP) of an earth observation satellite (SPOT5). This application can be formulated as a logically-constrained knapsack problem whose key model corresponds to the DCKP and can thus be solved with our approach. The computational results on the set of 21 DPSP benchmark instances indicate that our approach can find optimal solutions or solutions close to the best-known results obtained by specific algorithms specially designed for this application.

Finally, although a number of DCKP algorithms exist in the literature, none of these algorithms has published the underlying code, making it difficult to apply them in practice. The publicly available code of our algorithm can help advance the research on the DCKP and better solve related problems as well.

4 Computational results and comparisons

In this section, we assess the proposed TSBMA algorithm by performing extensive experiments and making comparisons with state-of-the-art DCKP algorithms. We report computational results on two sets of 6340 benchmark instances.

4.1 Benchmark instances

The benchmark instances of the DCKP tested in our experiments were widely used in the literature, which can be divided into two sets [1] (see Tables 1 and 2 for the main characteristics of these instances).

Set I (100 instances): These instances are grouped into 20 classes (each with 5 instances) and named by xIy ($x = \{1, \ldots, 20\}$ and $y = \{1, \ldots, 5\}$). The first 50 instances ($1Iy$ to $10Iy$) were introduced in 2006 [20] and have the following features: number of items $n = 500$ or 1000, capacity $C = 1800$ or 2000, and density η going from 0.05 to 0.40. Note that the density is given by $2m/n(n-1)$, where m is the number of disjunctive constraints (i.e., the number of edges of the conflict graph). These instances have an item weight w_i uniformly distributed in $[1, 100]$ and a profit $p_i = w_i + 10$. For the instance classes $11Iy$ to $20Iy$ introduced in 2017 [31], the number of items n is set to 1500 or 2000, the capacity C is set to 4000, and the density η ranges from 0.04 to 0.20. These instances have an item weight w_i uniformly distributed in $[1, 400]$ and a profit p_i equaling $w_i + 10$.

Set II (6240 instances): This set of instances was introduced in 2017 [3] and expanded in 2020 [8]. For the four correlated instance classes $C1$ to $C15$ (denoted by CC) and four random classes $R1$ to $R15$ (denoted by CR), the number of items n is from 60 to 1000, the capacity C is from 150 to 15000, and the density η is from 0.10 to 0.90. Each of these eight classes contains 720 instances. For the correlated instance class SC and the random instance class SR of the sparse graphs, the number of items n is from 500 to 1000, the capacity C is from 1000 to 2000, and the density η is from 0.001 to 0.05. Each of these two classes contains 240 DCKP instances. More details about this set of instances can be found in [8].

4.2 Experimental settings

Reference algorithms. For the 100 DCKP instances of Set I that were widely tested by heuristic algorithms, we adopt as our reference methods three state-of-the-art heuristic algorithms: parallel neighborhood search algorithm (PNS) [32], cooperative parallel adaptive neighborhood search algorithm (CPANS)

[1] The benchmark instances are available from the Mendeley Data repository at: `http://dx.doi.org/10.17632/gb5hhjkygd.1`

Table 1
Summary of main characteristics of the 100 DCKP instances of Set I.

Class	Total	n	C	η	Class	Total	n	C	η
$1Iy$	5	500	1800	0.10	$11Iy$	5	1500	4000	0.04
$2Iy$	5	500	1800	0.20	$12Iy$	5	1500	4000	0.08
$3Iy$	5	500	1800	0.30	$13Iy$	5	1500	4000	0.12
$4Iy$	5	500	1800	0.40	$14Iy$	5	1500	4000	0.16
$5Iy$	5	1000	1800	0.05	$15Iy$	5	1500	4000	0.20
$6Iy$	5	1000	2000	0.06	$16Iy$	5	2000	4000	0.04
$7Iy$	5	1000	2000	0.07	$17Iy$	5	2000	4000	0.08
$8Iy$	5	1000	2000	0.08	$18Iy$	5	2000	4000	0.12
$9Iy$	5	1000	2000	0.09	$19Iy$	5	2000	4000	0.16
$10Iy$	5	1000	2000	0.10	$20Iy$	5	2000	4000	0.20

Table 2
Summary of main characteristics of the 6240 DCKP instances of Set II.

Class	Total	n		C		η	
		Min	Max	Min	Max	Min	Max
$C1$	720	60	1000	150	1000	0.10	0.90
$C3$	720	60	1000	450	3000	0.10	0.90
$C10$	720	60	1000	1500	10000	0.10	0.90
$C15$	720	60	1000	15000	15000	0.10	0.90
$R1$	720	60	1000	150	1000	0.10	0.90
$R3$	720	60	1000	450	3000	0.10	0.90
$R10$	720	60	1000	1500	10000	0.10	0.90
$R15$	720	60	1000	15000	15000	0.10	0.90
SC	240	500	1000	1000	2000	0.001	0.05
SR	240	500	1000	1000	2000	0.001	0.05

[31], and probabilistic tabu search algorithm (PTS) [33]. Note that PTS only reported results of the 50 instances $1Iy$ to $10Iy$, since the other 50 instances of $11Iy$ to $20Iy$ were designed later. For the 6240 DCKP instances of Set II that were only tested by exact algorithms until now, we cite the results of three best performing methods: branch-and-bound algorithms BCM [3] and CFS [8]) as well as the integer linear programming formulations solved by the CPLEX solver (ILP) [8].

Computing platform. The proposed TSBMA algorithm was written in C++ [2] and compiled using the g++ compiler with the -O3 option. All experiments were carried out on an Intel Xeon E5-2670 processor (2.5 GHz CPU and 2 GB RAM) under the Linux operating system. The results of the main reference algorithms have been obtained on computing platforms with

[2] The code of our TSBMA algorithm will be available at: `http://www.info.univ-angers.fr/pub/hao/DCKP_TSBMA.html`

the following features: an Intel Xeon processor with 2×3.06 GHz for CPANS and PNS, an Intel Pentium i5-6500 processor with 3.2 GHz and 4 GB RAM for PTS, and an Intel Xeon E5-2695 processor with 3.00GHz for CFS. Note that the parallel algorithms PNS and CPANS used 10 to 400 processors to obtain the results.

Parameter setting. The TSBMA algorithm has three parameters whose values are self-tuned according to the test instance or the best objective value reached during the search. Table 3 summarizes the parameter setting, where n is the number of items of the test instance, $MinP$ is the minimum profit in the instance, and $f(S_b)$ is the objective value of the best solution found during the TSP procedure. These parameter settings can be considered as the default settings and are consistently used in our experiments.

Table 3
Summarized parameter settings of the TSBMA algorithm.

Parameter	Section	Setting	Description	Comment
$\|P\|$	3.3	$n/100 + 5$	Population size	-
$IterMax$	3.3	$2n$	Maximum consecutive iterations	For population initialization
	3.4.1	$(n/500 + 5) \times 10000$	of the TSP procedure	For local optimization
T	3.4.1	$f(S_b) - n/10$	Threshold	For instances of Set I
		$MinP + rand(20)$		For instances of Set II

Stopping condition. For the 100 DCKP instances of Set I, the TSBMA algorithm adopted the same cut-off time as the reference algorithms (PNS, CPANS and PTS), i.e., 1000 seconds. Note that for the instances $11Iy$ to $20Iy$, PNS used a much longer limit of 2000 seconds. Given its stochastic nature, TSBMA was performed 20 times independently with different random seeds to solve each instance. For the 6240 instances of Set II, the cut-off time was set to 600 seconds as in the CFS algorithm and the number of repeated runs was set to 10.

4.3 Computational results and comparisons

In this section, we first present summarized comparisons of the proposed TSBMA algorithm against each reference algorithm on the 100 instances of Set I, and then show the comparative results on the 6240 DCKP instances of Set II. The detailed computational results of our algorithm and the reference algorithms on the instances of Set I are shown in the Appendix, while our solution certificates for these 100 instances are available at the webpage indicated in footnote 2. For the 6240 instances of Set II, we report their objective values at the same website.

4.3.1 Comparative results on the 100 benchmark instances of Set I

The comparative results of the TSBMA algorithm and each reference algorithm are summarized in Table 4. Column 1 indicates the pairs of compared algorithms and column 2 gives the names of instance class. Column 3 shows the quality indicators: the best objective value (f_{best}) and the average objective value (f_{avg}) (when the average results are available in the literature). To analyze the performance of our algorithm, we carried out the Wilcoxon signed-rank test to verify the statistical significance of the compared results between TSBMA and each compared algorithm in terms of the f_{best} and f_{avg} values (when the average results are available in the literature). Columns 4 and 5 give the additional sum of ranks for the results, where TSBMA performs better (R^+) or worse (R^-) in terms of the performance indicators. The outcomes of the Wilcoxon *p-values* are shown in last column, where NA means that the two sets of compared results are exactly the same.

From Table 4, one observes that the TSBMA algorithm competes very favorably with all the reference algorithms by reporting improved or equal results on all the instances. Compared to the probabilistic tabu search algorithm (PTS) [33] which reported results only on the first 50 instances of classes $1Iy$ to $10Iy$, TSBMA finds 8 (45) better f_{best} (f_{avg}) values, while matching the remaining results. Compared to the two parallel algorithms (PNS) [32] and (CPANS) [31] that reported only the f_{best} values, TSBMA obtained 35 and 29 better f_{best} results, respectively. The small *p-values* (< 0.05) from the Wilcoxon tests between TSBMA and its competitors indicate that the performance differences are statistically significant. Finally, it is remarkable that our TSBMA algorithm discovered 24 new lower bounds on the instances $11Iy$ to $20Iy$ (see the detailed results shown in the Appendix).

Table 4
Summarized comparisons of the TSBMA algorithm against each reference algorithm with the *p-values* of the Wilcoxon signed-rank test (significance level 0.05) on the 100 DCKP instances of Set I.

Algorithm pair	Instance	Indicator	R^+	R^-	*p-value*
TSBMA vs. PTS [33]	$1Iy - 10Iy$ (50)	f_{best}	8	0	1.40e-2
		f_{avg}	45	0	5.34e-9
TSBMA vs. PNS [32]	$1Iy - 10Iy$ (50)	f_{best}	9	0	8.91e-3
	$11Iy - 20Iy$ (50)	f_{best}	26	0	8.25e-6
TSBMA vs. CPANS [31]	$1Iy - 10Iy$ (50)	f_{best}	0	0	NA
	$11Iy - 20Iy$ (50)	f_{best}	29	0	2.59e-6

To complete the assessment, we provide the performance profiles [11] of the four compared algorithms on the 100 instances of Set I. Basically, the performance profile of an algorithm shows the cumulative distribution for a given performance metric, which reveals the overall performance of the algorithm on a set of instances. In our case, the plots concern the best objective values (f_{best}) of the compared algorithms since the average results of some reference algorithms are not available in the literature. Given a set of algorithms (solvers) $\mathcal{S}$ and an instance set $\mathcal{P}$, the performance ratio is given by $r_{p,s} = \frac{f_{p,s}}{min\{f_{p,s}: s \in \mathcal{S}\}}$, where $f_{p,s}$ is the f_{best} value of instance p of $\mathcal{P}$ obtained by algorithm s of $\mathcal{S}$. The performance profiles are shown in Figure 3, where the performance ratio and the percentage of instances solved by each compared algorithm are displayed on the X-$axis$ and Y-$axis$, respectively. When the value of X-$axis$ is 1, the corresponding value of Y-$axis$ indicates the fraction of instances for which algorithm s can reach the best f_{best} value of the set $\mathcal{S}$ of the compared algorithms.

From Figure 3, we observe that our TSBMA algorithm has a very good performance on the 100 benchmark instances of Set I compared to the reference algorithms. For the 50 instances $1Iy$ to $10Iy$, TSBMA and CPANS are able to reach 100% best f_{best} values on these 50 instances, while PTS and PNS fail on around 15% of the instances. When considering the 50 instances $11Iy$ to $20Iy$, the plot of TSBMA strictly runs above the plots of PNS and CPANS, revealing that our algorithm dominates the reference algorithms on these 50 instances. These outcomes again confirm the high performance of our TSBMA algorithm.

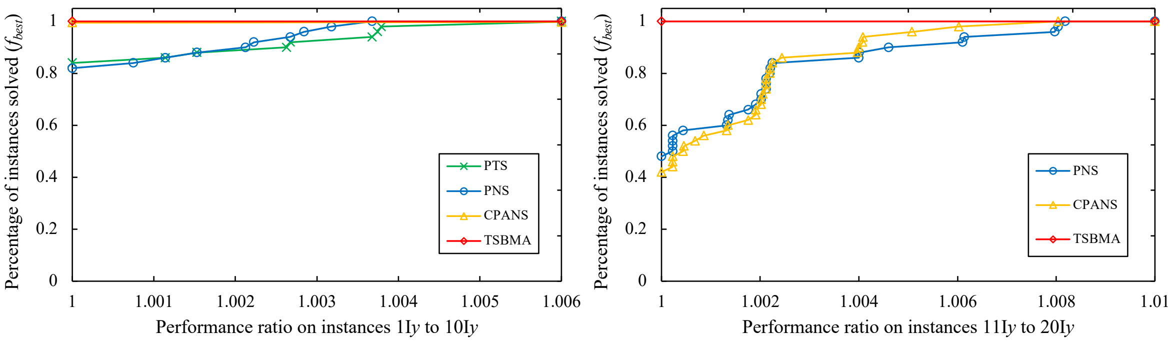

Fig. 3. Performance profiles of the compared algorithms on the 100 DCKP instances of Set I.

4.3.2 Comparative results on the 6240 benchmark instances of Set II

Table 5 summarizes the comparative results of our TSBMA algorithm on the 6240 instances of Set II, together with the three reference algorithms mentioned in 4.2. Note that three ILP formulations were studied in [8], we extracted the best results of these formulations in Table 5, i.e., the results on instances CC and CR (conflict graph density from 0.10 to 0.90) with ILP_2 and the results on very sparse instances SC and SR (conflict graph density from 0.001 to 0.05) with ILP_1. Columns 1 and 2 of Table 5 identify each instance class and the total number of instances of the class. Columns 3 to 5 indicate the number of instances solved to optimality by the three reference algorithms. Column 6 shows the number of instances for which our TSBMA algorithm reaches the optimal solution proved by exact algorithms. The number of new lower bounds (denoted by NEW LB in Table 5) found by TSBMA is provided in column 7. The best results of the compared algorithms are highlighted in bold. In order to further evaluate the performance of our algorithm, we summarize the available comparative results between MSBTS and the main reference algorithm CFS in columns 8 to 10. The columns #Wins, #Ties and #Losses present the number of instances for which TSBMA achieves a better, equal and worse result on the corresponding instance class. The last three rows provide an additional summary of the results for each column.

From Table 5, we observe that TSBMA performs globally very well on the instances of Set II. For the 5760 CC and CR instances, TSBMA reaches most of the proved optimal solutions (5381 out of 5389) and discovers new lower bounds for 323 difficult instances whose optima are still unknown. For the 240 very sparse SC instances, TSBMA matches 195 out of 200 proved optimal solutions and finds 24 new lower bounds for the remaining instances. Although TSBMA successfully solves only 9 out of the 229 solved very sparse SR instances, it discovers 7 new lower bounds. The high performance of TSBMA is further evidenced with the comparison with the best exact algorithm CFS (last three columns).

Notice that the performance of CPLEX with ILP_1 is better than TSBMA as well as the two reference algorithms BCM and CFS on the two classes of very sparse instances (SC and SR). As analyzed in [8], one of the main reasons is that the LP relaxation of ILP_1 provides a very strong upper bound, which makes the ILP_1 formulation very suitable for solving very sparse instances. The disjunctive constraints become very weak when the conflict graph is very sparse. For these two classes of instances, the pure branch-and-bound CFS algorithm is more effective on extremely sparse instances with densities up to 0.005. On the contrary, our TSBMA algorithm is more suitable for solving sparse instances with densities between 0.01 and 0.05. In fact, the new lower bounds found by TSBMA all concern instances with a density of 0.05. Finally, the TSBMA algorithm remains competitive on the 240 correlated sparse instances SC, even if the density is the smallest (0.001), which means that only the random sparse instance class SR is challenging for TSBMA.

In summary, our TSBMA algorithm is computational efficient on a majority of the 6240 benchmark instances of Set II and is able to discover new lower bounds on 354 difficult DCKP instances, whose optimal solutions are still unknown.

Table 5
Summarized comparisons of the TSBMA algorithm against each reference algorithm on the 6240 DCKP instances of Set II.

Class	Total	$ILP_{1,2}$ [8]	BCM [3]	CFS [8]	TSBMA (this work)		TSBMA vs. CFS		
		Solved	Solved	Solved	Solved	New LB	#Wins	#Ties	#Losses
$C1$	720	**720**	**720**	**720**	**720**	0	0	720	0
$C3$	720	584	**720**	**720**	716	0	0	716	4
$C10$	720	446	552	**617**	**617**	**91**	91	629	0
$C15$	720	428	550	**600**	**600**	**117**	117	603	0
$R1$	720	**720**	**720**	**720**	717	0	0	717	3
$R3$	720	680	**720**	**720**	**720**	0	0	720	0
$R10$	720	508	630	**670**	669	**37**	37	681	2
$R15$	720	483	590	**622**	**622**	**78**	78	641	1
SC	240	**200**	109	156	195	**24**	70	165	5
SR	240	229	154	176	9	**7**	43	8	189
Total on CC and CR	5760	4569	5201	5389	5381	323	323	5427	10
Total on SC and SR	480	429	263	332	204	31	113	173	194
Grand total	6240	4998	5424	5721	5585	354	436	5600	204

5 A real-world application

To demonstrate the practical usefulness of the DCKP model and the proposed TSBMA algorithm, this section shows how our approach can be applied to solve a real-world daily photograph scheduling problem (DPSP) of the earth observation satellite SPOT5.

5.1 SPOT5 daily photograph scheduling and its knapsack formulation

SPOT5 is the fifth earth optical observation satellite developed by the CNES (French National Space Agency), which was launched in May 2002. Informally, the SPOT5 daily photograph scheduling problem is to select a subset of photographs among the candidate photographs that will be taken by SOPT5, such that the total profit of the selected photographs is maximized while a knapsack-type constraint (i.e., a capacitated recording memory) and a large number of physical constraints (such as non-overlapping trials, minimal transition times between trials and bounded instantaneous data flow) are satisfied [1].

Let $P = \{p_1, \ldots, p_n\}$ be the set of n candidate photographs including mono and stereo photographs, where each photograph $\rho_i \in P$ has a profit $p_i > 0$ and a weight $w_i > 0$ (memory consumption). Each mono photograph can be taken by any of the three cameras of the satellite (front-camera1, middle-camera2 and rear-camera3), while each stereo photograph can only be obtained by the front and rear cameras simultaneously. A legal schedule must satisfy the following constraints.

- Knapsack constraint (C1): This constraint indicates that the photographs that are taken and recorded on board cannot excess the recording memory $Max\_Capacity$ of the satellite.
- Binary constraints (C2): These constraints forbid the simultaneous selection of two pairs (photo, camera) and express the non overlapping of two trials and the minimal transition time between two successive trials of a camera, as well as some constraints involving limitations on instantaneous data flow relating two pairs (photo, camera).
- Ternary constraints (C3): These constraints forbid the simultaneous presence of three pairs (photo, camera) and concern limitations on instantaneous data flow that cannot be expressed in the form of binary constraints (C3_1). Additionally, for a mono photograph, a ternary constraint is defined to ensure that the photograph can be scheduled to at most one camera (C3_2).

In [36], the DPSP was formulated as a "logic-constrained" 0/1 knapsack problem, which is highly related to the DCKP, where a binary variable is used to represent a pair (photo, camera). Let $P = P_1 \cup P_2$, where P_1 and P_2 are the set of mono and stereo photographs, respectively. Let $\rho \in P$ be a candidate photograph. If $\rho \in P_1$, three pairs (ρ, camera1), (ρ, camera2) and (ρ, camera3) are associated to three binary variables to enumerate the three possibilities of shooting the mono photograph. If $\rho \in P_2$, one binary variable is used to indicate the only shooting possibility (ρ, camera13) for the stereo photograph ρ. Then a photograph schedule can be represented by a binary l-vector $x = (x_1, x_2, \ldots, x_l)$ ($l = 3 * |P_1| + |P_2|$), where $x_i = 1$ if the corresponding pair (photo, camera) is selected, and $x_i = 0$ otherwise.

The DPSP corresponds to the following "logic-constrained" 0/1 knapsack problem [36].

$$(SPOT5) \qquad \text{Maximize} \quad \sum_{i=1}^{l} p_i x_i \tag{13}$$

$$\text{subject to} \quad \sum_{i=1}^{l} w_i x_i \leq Max\_Capacity \tag{C1}$$

$$x_i + x_j \leq 1, \forall i, j \in \{1, \ldots, l\}, i \neq j \tag{C2}$$

$$x_i + x_j + x_k \leq 2, \forall i, j, k \in \{1, \ldots, l\}, i \neq j \neq k \tag{C3\_1}$$

$$x_i + x_j + x_k \leq 1, \forall i, j, k \in \{1, \ldots, l\}, i \neq j \neq k \tag{C3\_2}$$

$$x_i \in \{0, 1\}, i = 1, \ldots, l \tag{14}$$

5.2 Solving DPSP as the DCKP model

It is easy to observe that the "logic-constrained" 0/1 knapsack model for the DPSP without the ternary constraints C3_1 and C3_2 is strictly equivalent to the DCKP. Moreover, one notes that each C3_2 ternary constraint $x_i+x_j+x_k \leq 1$ can be converted to three binary constraints, i.e., $x_i + x_j \leq 1$, $x_i + x_k \leq 1$, $x_j + x_k \leq 1$, and thus integrated into the DCKP model.

Thus, to solve the DPSP via the DCKP model, we adopt the following strategy. For each DPSP instance, we apply our TSBMA algorithm to solve the corresponding DCKP instance integrating constraints C1, C2 and C3_2 and temporarily ignore the ternary constraint C3_1. Let s be the final result returned by TSBMA. If solution s doesn't violate any C3_1 ternary constraint, s is a feasible solution to the DPSP. Otherwise, we employ a very simple two-step repairing procedure to satisfy the C3_1 constraints.

Suppose that the C3_1 constraint $x_i + x_j + x_k \leq 2$ is violated. This implies necessarily $x_i + x_j + x_k = 3$. Thus, to satisfy this violated C3_1 constraint, the first step identifies among x_i, x_j, x_k the variable with the smallest profit value and changes its value to zero, i.e., drops the corresponding (photo, camera) pair (break ties randomly). Since this step releases knapsack capacity, the second step uses the liberated capacity to accommodate additional photographs. For this, we add photographs with the largest profit value without violating any C1–C3 constraints until no more photograph can be added or the knapsack capacity is reached.

5.3 Computational results on the DPSP benchmark instances

We used the 21 real-life DPSP benchmark instances [3] provided by the CNES (see [1] for more details), which have been used to test a number of exact and heuristic algorithms [2,36,37,38]. For the experiment, we ran our TSBMA algorithm 10 times on each DPSP instance under the stopping condition of one hour per run. The computational results are shown in Table 6. The first column presents the name of each instance, while the asterisk (*) indicates that the optimal result is known [1]. The second column gives the number of photographs in each instance. Columns 3 to 5 show the number of C2, C3_1 and C3_2 constraints in each instance. Column BKV gives the best-known values from [1], which are compiled from the literature [2,36,38]. The remaining columns present the best f_{best} and average f_{avg} objective values obtained by our algorithm as well as the gap $gap(\%)$ between our best result

[3] The benchmark instances are available at the Mendeley Data repository at: `http://dx.doi.org/10.17632/2kbzg9nw3b.1`

and BKV calculated as gap = (f_{best} - BKV) / BKV). The best values (f_{best}) of our algorithm reaching the optimal results are highlighted in bold.

Table 6
Computational results of the TSBMA algorithm and comparison with the best-known values (BKV) on the 21 real-life instances of the SPOT5 daily photograph scheduling problem.

Instance	Photographs	Number of constraints			BKV	TSBMA (this work)		
		C2	C3_1	C3_2		f_{best}	f_{avg}	gap(%)
8*	8	17	0	4	10	**10**	10	0.000
54*	67	389	23	29	70	**70**	69.6	0.000
29*	82	610	0	19	12032	**12032**	12031.4	0.000
42*	190	1762	64	57	108067	**108067**	108067.0	0.000
28*	230	6302	590	58	56053	**56053**	56053.0	0.000
5*	309	13982	367	250	115	111	107.4	-3.478
404*	100	919	18	29	49	**49**	47.8	0.000
408*	200	2560	389	64	3082	3075	3074.2	-0.227
412*	300	6585	389	122	16102	16094	16092.4	-0.050
11*	364	9456	4719	164	22120	22111	22109.1	-0.041
503*	143	705	86	58	9096	**9096**	8994.6	0.000
505*	240	2666	526	104	13100	13096	12995.4	-0.031
507*	311	5545	2293	131	15137	15132	15127.3	-0.033
509*	348	7968	3927	152	19215	19113	19110.6	-0.531
1401	488	11893	2913	213	176056	170056	167960.5	-3.408
1403	665	14997	3874	326	176140	170146	167848.8	-3.403
1405	855	24366	4700	480	176179	168185	167882.4	-4.537
1021	1057	30058	5875	649	176246	170247	168049.7	-3.404
1502*	209	296	29	102	61158	**61158**	61158.0	0.000
1504	605	5106	882	324	124243	124238	124135.5	-0.004
1506	940	19033	4775	560	168247	164241	161639.3	-2.381

Table 6 shows that the TSBMA algorithm is able to match eight optimal results. For seven other instances, its results are very close to the best-known results with a very small gap(%) value (-0.004% to -0.531%). One notices that TSBMA reports a gap value ranging from -2.381% to -4.537% on most instances with more C3_1 constraints. This is because our approach first relaxes the C3_1 constraints and then only employs a simple procedure to repair these constraints. These outcomes can be considered to be remarkable because we just applied the TSBMA algorithm designed for the general DCKP model to this real-life problem, unlike previous algorithms for the DPSP that are specially designed for the problem.

To sum up, this real-world application shows the practical significance of the DCKP model and the proposed TSBMA algorithm.

6 Analysis and discussions

In this section, we analyze two essential components of the TSBMA algorithm: the importance of the threshold search and the contribution of the operation-prohibiting mechanism. The studies in this section are based on the 50 benchmark instances $11Iy$ to $20Iy$ of Set I.

6.1 Importance of the threshold search

The threshold search procedure of the TSBMA algorithm is the first adaptation of the threshold accepting method to the DCKP. To assess the importance of this component, we compare TSBMA with two TSBMA variants by replacing the TSP procedure with the *first-improvement* descent procedure and *best-improvement* descent procedure. In other words, these variants (named as MA1 and MA2) use, in each iteration, the first and the best improving solution S' in the neighborhood to replace the current solution, respectively. We carried out an experiment by running the two variants to solve the 50 instances $11Iy$ to $20Iy$ with the same experimental settings of Section 4.2. The performance profiles of TSBMA and these TSBMA variants are shown in Figure 4 based on the best objective values (left sub-figure) and the average objective values (right sub-figure).

From Figure 4, we can clearly observe that TSBMA dominates MA1 and MA2 according to the cumulative probability obtained by the f_{best} and f_{avg} values. The plots of TSBMA strictly run above the plots of MA1 and MA2, indicating TSBMA performs always better than the two variants. This experiment implies that the adopted threshold search procedure of TSBMA is relevant for its performance.

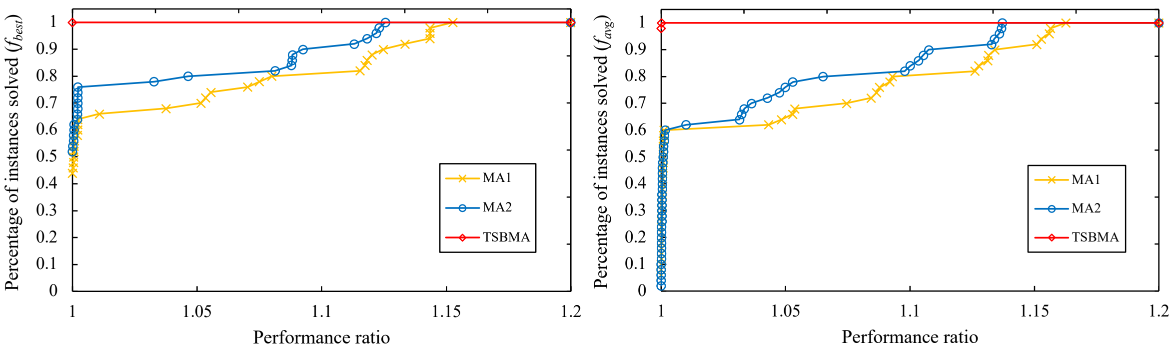

Fig. 4. Performance profiles of the compared algorithms on the 50 DCKP instances $11Iy$ to $20Iy$.

6.2 Contribution of the operation-prohibiting mechanism

TSBMA avoids revisiting previously encountered solutions with the OP mechanism introduced in Section 3.4.3. To assess the usefulness of the OP mechanism, we created a TSBMA variant (denoted by TSBMA$^-$) by disabling the OP component and keeping the other components unchanged. We ran TSBMA$^-$ to solve the 50 $11Iy$ to $20Iy$ instances according to experimental settings given in Section 4.2 and reported the results in Table 7. The first column gives the name of each instance and the remaining columns show the best objective values (f_{best}), the average objective values (f_{avg}) and the standard deviations (*std*). Row #Avg presents the average value of each column and row #Best indicates the number of instances for which an algorithm obtains the best values between the two sets of results. The last row shows the *p-values* from the Wilcoxon signed-rank test. The best results of the compared algorithms are highlighted in bold.

From Table 7, we observe that TSBMA$^-$ performs worse than TSMBA. TSBMA$^-$ obtains worse f_{best} values for 35 out of the 50 instances and worse f_{avg} values for 48 instances. Considering the *std* values, TSBMA$^-$ shows a much less stable performance than TSMBA. Moreover, the small *p-values* (< 0.05) from the Wilcoxon tests confirm the statistically significant difference between the results of TSMBA and TSBMA$^-$. This experiment demonstrates the effectiveness and robustness of the operation-prohibiting mechanism employed by the TSMBA algorithm.

7 Conclusions

The disjunctively constrained knapsack problem is a well-known NP-hard model. Given its practical significance and intrinsic difficulty, a variety of exact and heuristic algorithms have been designed for solving the problem. We proposed the threshold search based memetic algorithm that combines for the first time threshold search with the memetic framework. The primary novelty of our approach concerns the design of the threshold search procedure that relies on three complementary neighborhoods and an original neighborhood exploration strategy. This intensification oriented component is reinforced by the specially designed crossover operator and the distance-and-quality based population update strategy.

Extensive evaluations on a large number of benchmark instances in the literature (6340 instances in total) showed that the algorithm performs competitively with respect to the state-of-the-art algorithms. Our approach is able to discover 24 new lower bounds out of the 100 instances of Set I and

Table 7
Comparison between TSBMA$^-$ (without the OP mechanism) and TSBMA (with the OP mechanism) on the instances $11Iy$ to $20Iy$.

Instance	TSBMA$^-$			TSBMA		
	f_{best}	f_{avg}	*std*	f_{best}	f_{avg}	*std*
11I1	4960	4960	0.00	4960	4960	0.00
11I2	4940	4940	0.00	4940	4940	0.00
11I3	4950	4949.45	2.18	4950	**4950**	0.00
11I4	4930	4924	4.42	4930	**4930**	0.00
11I5	4920	4916.35	4.68	4920	**4920**	0.00
12I1	4685	4676.95	4.99	**4690**	**4687.65**	2.22
12I2	4670	4668.70	3.10	**4680**	**4680**	0.00
12I3	4690	4685.45	4.20	4690	**4690**	0.00
12I4	4680	4669.80	6.36	4680	**4679.50**	2.18
12I5	4670	4664.50	4.57	4670	**4670**	0.00
13I1	4525	4511.20	8.55	**4539**	**4534.80**	3.60
13I2	4521	4509.25	7.29	**4530**	**4528**	4.00
13I3	4520	4515.40	4.55	**4540**	**4531**	3.00
13I4	4520	4507.10	6.94	**4530**	**4529.15**	2.29
13I5	4530	4513.65	6.51	**4537**	**4534.20**	3.43
14I1	4429	4413.55	7.41	**4440**	**4440**	0.00
14I2	4420	4413.55	4.47	**4440**	**4439.40**	0.49
14I3	4420	4415.20	4.70	**4439**	**4439**	0.00
14I4	4420	4412.40	4.57	**4435**	**4431.50**	2.06
14I5	4420	4413.85	4.27	**4440**	**4440**	0.00
15I1	4359	4346.15	5.06	**4370**	**4369.95**	0.22
15I2	4359	4344.10	6.22	**4370**	**4370**	0.00
15I3	4359	4341.85	6.54	**4370**	**4369.25**	1.84
15I4	4350	4341.05	7.78	**4370**	**4369.85**	0.36
15I5	4360	4346.10	5.47	**4379**	**4373.15**	4.29
16I1	5020	5013.75	4.93	5020	**5020**	0.00
16I2	5010	5003.30	5.60	5010	**5010**	0.00
16I3	5020	5010.65	5.33	5020	**5020**	0.00
16I4	5020	5008.95	8.24	5020	**5020**	0.00
16I5	5060	5052.85	8.37	5060	**5060**	0.00
17I1	4730	4707.50	7.51	4730	**4729.70**	0.64
17I2	4716	4704.50	6.27	**4720**	**4719.50**	2.18
17I3	4720	4705.10	6.68	**4729**	**4723.60**	4.41
17I4	4722	4701.20	9.68	**4730**	**4730**	0.00
17I5	4720	4706.20	8.37	**4730**	**4726.85**	4.50
18I1	4555	4539.75	6.31	**4568**	**4565.80**	3.40
18I2	4540	4532.20	4.64	**4560**	**4551.40**	3.01
18I3	4570	4545.20	8.58	4570	**4569.40**	2.20
18I4	4550	4539.30	6.75	**4568**	**4565.20**	3.12
18I5	4550	4542.50	5.32	**4570**	**4567.95**	3.46
19I1	4432	4424.65	4.71	**4460**	**4456.65**	3.48
19I2	4443	4430.85	6.06	**4460**	**4453.25**	4.17
19I3	4440	4428.15	6.01	**4469**	**4462.05**	4.04
19I4	4450	4431.25	5.63	**4460**	**4453.20**	3.89
19I5	4449	4435.65	5.42	**4466**	**4460.75**	1.61
20I1	4364	4358.95	2.80	**4390**	**4383.20**	3.36
20I2	4360	4356.85	4.25	**4390**	**4381.80**	3.78
20I3	4370	4360.45	5.11	**4389**	**4387.90**	2.77
20I4	4370	4359.75	5.78	**4389**	**4380.40**	1.98
20I5	4366	4357.45	4.78	**4390**	**4386.40**	4.05
#Avg	4603.08	4593.13	5.56	**4614.14**	**4611.83**	1.80
#Best	15/50	2/50	-	50/50	50/50	-
p-values	2.51e-7	1.68e-9	-	-	-	-

354 new lower bounds out of the 6240 instances of Set II. These new lower bounds are useful for future studies on the DCKP. The algorithm also attains the best-known or known optimal results on most of the remaining instances. We carried out additional experiments to investigate the two essential ingredients of the algorithm (the threshold search technique and the operation-prohibiting mechanism). The disjunctively constrained knapsack problem is a useful model to formulate a number of practical applications. The algorithm and its code (that we will make available) can contribute to solving these problems. In this regards, we presented an application of our approach to deal with the real-life daily photograph scheduling problem of the earth observation satellite SPOT5.

There are at least two possible directions for future work. First, TSBMA performed badly on random sparse instances of SR. It would be interesting to improve the algorithm to better handle such instances. Second, given the good performance of the adopted approach, it is worth investigating its underlying ideas to solve related problems mentioned in the introduction as well as other knapsack problems such as multiple knapsack [10], knapsack with setups [15] and multiple non-linear separable knapsack [9].

Declaration of competing interest

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

Acknowledgments

We are grateful to the reviewers for their useful comments and suggestions which helped us to significantly improve the paper. We would like to thank Dr. Zhe Quan, Dr. Lei Wu, Dr. Pablo San Segundo, and their co-authors for sharing the DCKP instances and the detailed results of their algorithms reported in [8,31,32]. Special thanks go to Dr. Michel Vasquez for his assistance on the SPOT5 instances tested in [36] and his valuable helps on the daily photograph scheduling application. Support from the China Scholarship Council (Grant 201706290016) for the first author is also acknowledged.

References

[1] E. Bensana, M. Lemaitre, G. Verfaillie, Earth observation satellite management, Constraints 4 (3) (1999) 293–299.

[2] E. Bensana, G. Verfaillie, J.-C. Agnèse, N. Bataille, D. Blumstein, Exact and approximate methods for the daily management of an earth observation satellite, in: T. D. Guyenne (ed.), 4th Intl. Symposium on Space Mission Operations and Ground Data Systems (SpaceOps-96), Proceedings, Munich, Germany. 1996, pages 507–514, ESA Special Publication, 1996.

[3] A. Bettinelli, V. Cacchiani, E. Malaguti, A branch-and-bound algorithm for the knapsack problem with conflict graph, INFORMS Journal on Computing 29 (3) (2017) 457–473.

[4] D. Castelino, N. Stephens, Tabu thresholding for the frequency assignment problem, in: Meta-Heuristics, Springer, 1996, pp. 343–359.

[5] Y. Chen, J.-K. Hao, A "reduce and solve" approach for the multiple-choice multidimensional knapsack problem, European Journal of Operational Research 239 (2) (2014) 313–322.

[6] Y. Chen, J.-K. Hao, Iterated responsive threshold search for the quadratic multiple knapsack problem, Annals of Operations Research 226 (1) (2015) 101–131.

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

[8] S. Coniglio, F. Furini, P. San Segundo, A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts, European Journal of Operational Research 289 (2) (2021) 435–455.

[9] C. D'Ambrosio, S. Martello, L. Mencarelli, Relaxations and heuristics for the multiple non-linear separable knapsack problem, Computers & Operations Research 93 (2018) 79–89.

[10] P. Detti, A new upper bound for the multiple knapsack problem, Computers & Operations Research 129 (2021) 105210.

[11] E. D. Dolan, J. J. Moré, Benchmarking optimization software with performance profiles, Mathematical Programming 91 (2) (2002) 201–213.

[12] G. Dueck, T. Scheuer, Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing, Journal of Computational Physics 90 (1) (1990) 161–175.

[13] G. Dueck, J. Wirsching, Threshold accepting algorithms for 0–1 knapsack problems, in: Proceedings of the Fourth European Conference on Mathematics in Industry, pages 255–262, Springer, 1991.

[14] A. Ekici, Bin packing problem with conflicts and item fragmentation, Computers & Operations Research 126 (2021) 105113.

[15] F. Furini, M. Monaci, E. Traversi, Exact approaches for the knapsack problem with setups, Computers & Operations Research 90 (2018) 208–220.

[16] F. Glover, M. Laguna, Tabu search, Springer Science+Business Media New York, 1997.

[17] F. Gurski, C. Rehs, Solutions for the knapsack problem with conflict and forcing graphs of bounded clique-width, Mathematical Methods of Operations Research 89 (3) (2019) 411–432.

[18] J.-K. Hao, Memetic algorithms in discrete optimization, in: Handbook of Memetic Algorithms, Springer, 2012, pp. 73–94.

[19] M. Hifi, An iterative rounding search-based algorithm for the disjunctively constrained knapsack problem, Engineering Optimization 46 (8) (2014) 1109–1122.

[20] M. Hifi, M. Michrafy, A reactive local search-based algorithm for the disjunctively constrained knapsack problem, Journal of the Operational Research Society 57 (6) (2006) 718–726.

[21] M. Hifi, M. Michrafy, Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem, Computers & Operations Research 34 (9) (2007) 2657–2673.

[22] M. Hifi, N. Otmani, An algorithm for the disjunctively constrained knapsack problem, International Journal of Operational Research 13 (1) (2012) 22–43.

[23] K. Jansen, An approximation scheme for bin packing with conflicts, Journal of Combinatorial Optimization 3 (4) (1999) 363–377.

[24] D. S. Johnson, M. R. Garey, Computers and intractability: A guide to the theory of NP-completeness, WH Freeman, 1979.

[25] H. Kellerer, U. Pfersch, D. Pisinger, Knapsack problems, Spinger, 2004.

[26] X. Lai, J.-K. Hao, F. Glover, Z. Lü, A two-phase tabu-evolutionary algorithm for the 0–1 multidimensional knapsack problem, Information Sciences 436 (2018) 282–301.

[27] X. Lai, J.-K. Hao, D. Yue, Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem, European Journal of Operational Research 274 (1) (2019) 35–48.

[28] P. Moscato, Memetic algorithms: A short introduction, New Ideas in Optimization (1999) 219–234.

[29] U. Pferschy, J. Schauer, The knapsack problem with conflict graphs, Journal Graph Algorithms and Applications 13 (2) (2009) 233–249.

[30] U. Pferschy, J. Schauer, Approximation of knapsack problems with conflict and forcing graphs, Journal of Combinatorial Optimization 33 (4) (2017) 1300–1323.

[31] Z. Quan, L. Wu, Cooperative parallel adaptive neighbourhood search for the disjunctively constrained knapsack problem, Engineering Optimization 49 (9) (2017) 1541–1557.

[32] Z. Quan, L. Wu, Design and evaluation of a parallel neighbor algorithm for the disjunctively constrained knapsack problem, Concurrency and Computation: Practice and Experience 29 (20) (2017) e3848.

[33] M. B. Salem, S. Hanafi, R. Taktak, H. B. Abdallah, Probabilistic tabu search with multiple neighborhoods for the disjunctively constrained knapsack problem, RAIRO-Operations Research 51 (3) (2017) 627–637.

[34] M. B. Salem, R. Taktak, A. R. Mahjoub, H. Ben-Abdallah, Optimization algorithms for the disjunctively constrained knapsack problem, Soft Computing 22 (6) (2018) 2025–2043.

[35] C. D. Tarantilis, C. T. Kiranoudis, V. S. Vassiliadis, A threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem, European Journal of Operational Research 152 (1) (2004) 148–158.

[36] M. Vasquez, J.-K. Hao, A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite, Computational Optimization and Applications 20 (2) (2001) 137–157.

[37] M. Vasquez, J.-K. Hao, Upper bounds for the spot 5 daily photograph scheduling problem, Journal of Combinatorial Optimization 7 (1) (2003) 87–103.

[38] G. Verfaillie, M. Lemaitre, T. Schiex, Russian doll search for solving constraint optimization problems, in: AAAI/IAAI, Vol. 1, pages 181–187, 1996.

[39] Z. Wei, J.-K. Hao, Iterated two-phase local search for the set-union knapsack problem, Future Generation Computer Systems 101 (2019) 1005–1017.

[40] Z. Wei, J.-K. Hao, Multistart solution-based tabu search for the set-union knapsack problem, Applied Soft Computing 105 (2021) 107260.

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

[42] T. Yamada, S. Kataoka, K. Watanabe, Heuristic and exact algorithms for the disjunctively constrained knapsack problem, Journal of Information Processing Society of Japan 43 (9) (2002) 2864–2870.

[43] Y. Zhou, J.-K. Hao, F. Glover, Memetic search for identifying critical nodes in sparse graphs, IEEE Transactions on Cybernetics 49 (10) (2018) 3699–3712.

[44] Y. Zhou, V. Naroditskiy, Algorithm for stochastic multiple-choice knapsack problem and application to keywords bidding, in: Proceedings of the 17th International Conference on World Wide Web, pages 1175–1176, 2008.

A Computational results on the 100 DCKP instances of Set I

Tables A.1 and A.2 report the detailed computational results of the TSBMA algorithm and the reference algorithms (PNS [32], CPANS [31] and PTS [33]) on the 100 DCKP instances of Set I.

The first two columns of the tables give the name of each instance and the best-known objective values (BKV) ever reported in the literature. We employ the following four performance indicators to present our results: best objective value (f_{best}), average objective value over 20 runs (f_{avg}), standard deviations over 20 runs (std), and average run time t_{avg} in seconds to reach the best objective value. However, some of the performance indicators of the reference algorithms are not available in the literature (i.e., f_{avg}, t_{avg} and std). Note that for [32] (PNS) and [31] (CPANS), the authors reported several groups of results obtained by using different numbers of processors (range from 10 to 400). To make a fair comparison, we take the best f_{best} value of each instance in these groups of results as the final result. We use the average of the t_{avg} values in these groups as the final average run time. The last row #Avg indicates the average value of each column. The 24 new lower bounds discovered by our TSBMA algorithm are highlighted in bold.

Table A.1
Computational results of the TSBMA algorithm with the reference algorithms on the 50 DCKP instances of Set I ($1Iy$ to $10Iy$).

Instance	BKV	PNS [32]	CPANS [31]		PTS [33]		TSBMA (this work)			
		f_{best}	f_{best}	$t_{avg}(s)$	f_{best}	f_{avg}	f_{best}	f_{avg}	std	$t_{avg}(s)$
$1I1$	2567	2567	2567	17.133	2567	2567	2567	2567	0.00	163.577
$1I2$	2594	2594	2594	12.623	2594	2594	2594	2594	0.00	19.322
$1I3$	2320	2320	2320	14.897	2320	2320	2320	2320	0.00	6.060
$1I4$	2310	2310	2310	13.063	2310	2310	2310	2310	0.00	10.969
$1I5$	2330	2330	2330	20.757	2330	2321	2330	2330	0.00	63.663
$2I1$	2118	2118	2118	21.710	2118	2115.2	2118	2117.70	0.46	330.797
$2I2$	2118	2112	2118	129.390	2110	2110	2118	2111.60	3.20	705.755
$2I3$	2132	2132	2132	23.820	2119	2112.4	2132	2132	0.00	210.108
$2I4$	2109	2109	2109	31.377	2109	2105.6	2109	2109	0.00	14.182
$2I5$	2114	2114	2114	20.040	2114	2110.4	2114	2114	0.00	99.133
$3I1$	1845	1845	1845	34.683	1845	1760.3	1845	1845	0.00	3.780
$3I2$	1795	1795	1795	107.993	1795	1767.5	1795	1795	0.00	3.029
$3I3$	1774	1774	1774	22.490	1774	1757	1774	1774	0.00	3.585
$3I4$	1792	1792	1792	27.953	1792	1767.4	1792	1792	0.00	3.275
$3I5$	1794	1794	1794	34.820	1794	1755.5	1794	1794	0.00	9.159
$4I1$	1330	1330	1330	37.307	1330	1329.1	1330	1330	0.00	1.967
$4I2$	1378	1378	1378	40.827	1378	1370.5	1378	1378	0.00	3.926
$4I3$	1374	1374	1374	100.183	1374	1370	1374	1374	0.00	2.431
$4I4$	1353	1353	1353	26.930	1353	1337.6	1353	1353	0.00	4.167
$4I5$	1354	1354	1354	81.113	1354	1333.2	1354	1354	0.00	6.196
$5I1$	2700	2694	2700	122.637	2700	2697.9	2700	2700	0.00	78.215
$5I2$	2700	2700	2700	111.160	2700	2699	2700	2700	0.00	57.300
$5I3$	2690	2690	2690	73.640	2690	2689	2690	2690	0.00	18.566
$5I4$	2700	2700	2700	130.913	2700	2699	2700	2700	0.00	52.807
$5I5$	2689	2689	2689	279.377	2689	2682.7	2689	2687.65	3.21	289.966
$6I1$	2850	2850	2850	104.623	2850	2843	2850	2850	0.00	57.997
$6I2$	2830	2830	2830	93.887	2830	2829	2830	2830	0.00	76.883
$6I3$	2830	2830	2830	203.677	2830	2830	2830	2830	0.00	157.597
$6I4$	2830	2824	2830	160.587	2830	2824.7	2830	2830	0.00	328.817
$6I5$	2840	2831	2840	112.947	2840	2825	2840	2833.10	4.22	378.393
$7I1$	2780	2780	2780	186.970	2780	2771	2780	2779.40	1.43	483.465
$7I2$	2780	2780	2780	161.117	2780	2769.8	2780	2775.50	4.97	372.935
$7I3$	2770	2770	2770	136.310	2770	2762	2770	2768.50	3.57	393.018
$7I4$	2800	2800	2800	123.957	2800	2791.9	2800	2795.50	4.97	162.060
$7I5$	2770	2770	2770	149.933	2770	2763.6	2770	2770	0.00	290.591
$8I1$	2730	2720	2730	472.153	2720	2718.9	2730	2724	4.90	484.264
$8I2$	2720	2720	2720	109.373	2720	2713.6	2720	2720	0.00	214.760
$8I3$	2740	2740	2740	112.847	2740	2731.5	2740	2739.55	1.96	207.311
$8I4$	2720	2720	2720	253.230	2720	2712	2720	2715.35	4.85	518.579
$8I5$	2710	2710	2710	115.777	2710	2705	2710	2710	0.00	67.003
$9I1$	2680	2678	2680	134.023	2670	2666.9	2680	2679.70	0.71	316.210
$9I2$	2670	2670	2670	158.397	2670	2661.7	2670	2669.90	0.44	238.149
$9I3$	2670	2670	2670	123.280	2670	2666.5	2670	2670	0.00	161.176
$9I4$	2670	2670	2670	137.690	2663	2657.3	2670	2668.90	2.49	522.294
$9I5$	2670	2670	2670	131.247	2670	2662	2670	2670	0.00	98.124
$10I1$	2624	2620	2624	244.020	2620	2613.7	2624	2621.45	1.72	348.617
$10I2$	2642	2630	2630	144.867	2630	2620.8	2630	2630	0.00	182.474
$10I3$	2627	2620	2627	198.050	2620	2614.5	2627	2621.40	2.80	326.099
$10I4$	2621	2620	2620	148.997	2620	2609.7	2620	2620	0.00	105.609
$10I5$	2630	2627	2630	170.620	2627	2617.6	2630	2629.50	2.18	307.851
#Avg	2403.68	2402.36	2403.42	112.508	2402.18	2393.26	2403.42	2402.47	0.96	179.244

Table A.2
Computational results and comparison of the TSBMA algorithm with the reference algorithms on the 50 DCKP instances of Set I ($11Iy$ to $20Iy$).

Instance	BKV	PNS [32]	CPANS [31]		TSBMA (this work)			
		f_{best}	f_{best}	$t_{avg}(s)$	f_{best}	f_{avg}	std	$t_{avg}(s)$
11I1	4950	4950	4950	333.435	**4960**	4960	0.00	4.594
11I2	4940	4940	4928	579.460	4940	4940	0.00	14.305
11I3	4925	4920	4925	178.400	**4950**	4950	0.00	69.236
11I4	4910	4890	4910	320.067	**4930**	4930	0.00	139.197
11I5	4900	4890	4900	222.053	**4920**	4920	0.00	100.178
12I1	4690	4690	4690	230.563	4690	4687.65	2.22	416.088
12I2	4680	4680	4680	502.600	4680	4680	0.00	224.000
12I3	4690	4690	4690	229.116	4690	4690	0.00	215.103
12I4	4680	4680	4676	367.330	4680	4679.50	2.18	256.300
12I5	4670	4670	4670	487.563	4670	4670	0.00	79.190
13I1	4533	4533	4533	395.985	**4539**	4534.80	3.60	415.880
13I2	4530	4530	4530	573.718	4530	4528	4.00	361.229
13I3	4540	4530	4540	901.620	4540	4531	3.00	498.622
13I4	4530	4530	4530	315.076	4530	4529.15	2.29	366.951
13I5	4537	4537	4537	343.240	4537	4534.20	3.43	425.064
14I1	4440	4440	4440	483.156	4440	4440	0.00	205.733
14I2	4440	4440	4440	735.505	4440	4439.40	0.49	438.190
14I3	4439	4439	4439	614.733	4439	4439	0.00	146.119
14I4	4435	4435	4434	533.908	4435	4431.50	2.06	106.389
14I5	4440	4440	4440	473.448	4440	4440	0.00	160.900
15I1	4370	4370	4370	797.125	4370	4369.95	0.22	321.296
15I2	4370	4370	4370	676.703	4370	4370	0.00	181.021
15I3	4370	4370	4370	612.792	4370	4369.25	1.84	315.575
15I4	4370	4370	4370	649.398	4370	4369.85	0.36	424.873
15I5	4379	4379	4379	678.354	4379	4373.15	4.29	359.003
16I1	4980	4980	4980	286.130	**5020**	5020	0.00	205.964
16I2	4990	4990	4980	232.825	**5010**	5010	0.00	342.824
16I3	5009	5000	5009	199.880	**5020**	5020	0.00	155.070
16I4	5000	4997	5000	831.750	**5020**	5020	0.00	86.324
16I5	5040	5020	5040	982.970	**5060**	5060	0.00	32.837
17I1	4730	4730	4721	422.640	4730	4729.70	0.64	388.541
17I2	4710	4710	4710	248.770	**4720**	4719.50	2.18	300.275
17I3	4720	4720	4720	454.317	**4729**	4723.60	4.41	343.016
17I4	4720	4720	4720	432.900	**4730**	4730	0.00	288.961
17I5	4720	4720	4720	102.468	**4730**	4726.85	4.50	366.752
18I1	4566	4566	4566	225.010	**4568**	4565.80	3.40	269.545
18I2	4550	4550	4550	288.862	**4560**	4551.40	3.01	13.884
18I3	4570	4570	4570	328.555	4570	4569.40	2.20	466.748
18I4	4560	4560	4560	511.527	**4568**	4565.20	3.12	264.931
18I5	4570	4570	4570	651.887	4570	4567.95	3.46	572.589
19I1	4460	4460	4460	506.945	4460	4456.65	3.48	459.570
19I2	4459	4459	4459	666.900	**4460**	4453.25	4.17	307.224
19I3	4460	4460	4460	608.913	**4469**	4462.05	4.04	485.550
19I4	4450	4450	4450	476.755	**4460**	4453.20	3.89	430.824
19I5	4460	4460	4460	508.730	**4466**	4460.75	1.61	40.752
20I1	4389	4389	4388	957.410	**4390**	4383.20	3.36	929.372
20I2	4390	4390	4387	756.908	4390	4381.80	3.78	299.673
20I3	4389	4383	4389	966.010	4389	4387.90	2.77	568.988
20I4	4388	4388	4380	993.630	**4389**	4380.40	1.98	657.694
20I5	4389	4389	4389	772.495	**4390**	4386.40	4.05	646.570
#Avg	4608.54	4606.88	4607.58	513.011	4614.14	4611.83	1.80	303.390