algorithms

Article

Efficient 0/1-Multiple-Knapsack Problem Solving by Hybrid DP Transformation and Robust Unbiased Filtering

Patcharin Buayen * and Jeerapon Werapun

Department of Computer Science, Faculty of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand
* Correspondence: 56605013@kmitl.ac.th

Abstract: The multiple knapsack problem (0/1-mKP) is a valuable NP-hard problem involved in many science-and-engineering applications. In current research, there exist two main approaches: 1. the exact algorithms for the optimal solutions (i.e., branch-and-bound, dynamic programming (DP), etc.) and 2. the approximate algorithms in polynomial time (i.e., Genetic algorithm, swarm optimization, etc.). In the past, the exact-DP could find the optimal solutions of the 0/1-KP (one knapsack, n objects) in O(nC). For large n and massive C, the unbiased filtering was incorporated with the exact-DP to solve the 0/1-KP in O($n + C'$) with 95% optimal solutions. For the complex 0/1-mKP (m knapsacks) in this study, we propose a novel research track with hybrid integration of DP-transformation (DPT), exact-fit (best) knapsack order (m!-to-m^2 reduction), and robust unbiased filtering. First, the efficient DPT algorithm is proposed to find the optimal solutions for each knapsack in O([n^2,nC]). Next, all knapsacks are fulfilled by the exact-fit (best) knapsack order in O(m^2[n^2,nC]) over O(m![n^2,nC]) while retaining at least 99% optimal solutions as m! orders. Finally, robust unbiased filtering is incorporated to solve the 0/1-mKP in O(m^2n). In experiments, our efficient 0/1-mKP reduction confirmed 99% optimal solutions on random and benchmark datasets (n δ 10,000, m δ 100).

Keywords: multiple 0/1-knapsack problem (0/1-mKP); efficient NP-hard problem solving; exact-DP transformation; exact-fit (best) knapsack order; robust unbiased filtering

Citation: Buayen, P.; Werapun, J. Efficient 0/1-Multiple-Knapsack Problem Solving by Hybrid DP Transformation and Robust Unbiased Filtering. *Algorithms* **2022**, *15*, 366. https://doi.org/10.3390/a15100366

Academic Editor: Jesper Jansson

Received: 16 August 2022
Accepted: 27 September 2022
Published: 30 September 2022

1. Introduction

Presently, a variety of NP-hard problems are involved in many real-world applications and AI computing. Solving specific NP-hard problems (with high performance in efficient time) for those applications is challenging. Some of the interesting NP-hard problems are the 0/1-KP (knapsack problem), the 0/1-mKP (multiple m knapsacks), etc.

Formally, the 0/1-KP is defined as follows: Consider a set of n objects and a knapsack capacity C, where each object j (=0, 1, . . . , $n - 1$) has profit p_j and weight w_j.

The objective of the 0/1-KP is to select some objects for the maximum total profit kept in the knapsack that cannot exceed the knapsack capacity (C), defined in Equations (1)–(3). Recently (2018), unbiased filtering [1] was proposed (for the 0/1-KP) to select outstanding objects (from n objects) before applying the exact DP (dynamic programming) algorithm on small remaining n' ($\leq$200) in efficient time for most optimal solutions (at least 95%) of regular and irregular datasets.

$$\text{Maximize } \sum_{j=0}^{n-1} p_j x_j \tag{1}$$

$$\text{Subject to } \sum_{j=0}^{n-1} w_j x_j \leq C \tag{2}$$

$$\text{and } x_j \in \{0, 1\};\ j = 0, 1, 2, \ldots, n-1 \tag{3}$$

For the complex 0/1-mKP (m knapsacks), the objective is to select some objects for multiple knapsacks (each selected object j (x_{ij} = 1) in a proper knapsack i) that cannot exceed each of the knapsack capacities (C_i; i = 1, 2, 3, . . . , m), see Figure 1 (one and m knapsacks), for the maximized total profit, defined in Equations(4)–(6).

$$\text{Maximize } \sum_{i=1}^{m} \sum_{j=0}^{n-1} p_j x_{ij} \quad (4)$$

$$\text{Subject to } \sum_{j=0}^{n-1} w_j x_{ij} \le C_i\ ;\ i = 1,\ 2, \ldots,\ m;\ j = 0,\ 1,\ 2, \ldots,\ n-1 \quad (5)$$

$$\text{and } \sum_{i=1}^{m} x_{ij} \le 1,\ xij\ \{0,\ 1\};\ i,\ j \quad (6)$$

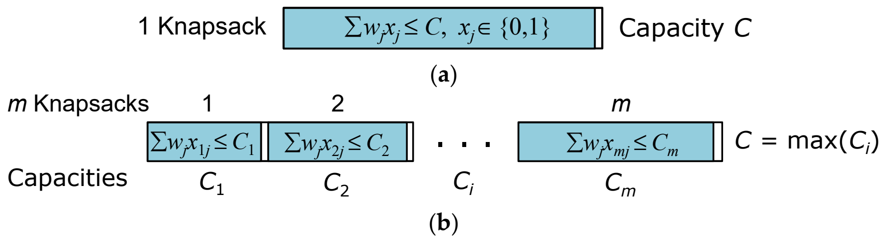


Figure 1. Two different 0/1-knapsack problems (n objects): (**a**) the popular 0/1-KP (one knapsack with capacity C) and (**b**) the complex 0/1-mKP (m knapsacks with capacity C_i, i = 1, 2, . . . , m).

The 0/1-KP and 0/1-mKP are constructive for science and engineering applications, such as resource allocation [2,3], capital budgeting [4], production planning [5], multicontainer packing [6], risk balancing and assortment optimization [7], other applications in network systems [8–10], etc. However, finding the optimal solution of the 0/1-mKP is much harder than that of the 0/1-KP since each selected object (x_{ij} = 1) must specify a proper knapsack (K_i with capacity C_i; $i \in \{1, \ldots, m\}$) from the available knapsacks.

In popular 0/1-KP research, two approaches are extensively studied: 1. the exact approach for optimal solutions (but in exponential time) and 2. the fast approximate approach (but the optimal solution may not be found). In theory, the optimal solution of the 0/1-KP can be computed by DP (dynamic programming) algorithms in O(nC) [11–15], or BnB (branch-and-bound) [16] and backtracking [17] algorithms in exponential time (O(2^n)). In practice, approximate methods (i.e., greedy methods [18,19], kernel search [20,21], genetic algorithms [22,23], swarm optimization [24–28], hybrid methods [29–32], hyper-heuristic method [33], etc.) can find the good solutions in polynomial time. Recently, the time-space reduction algorithm [1] was proposed to solve the 0/1-KP in O(n + C') for large n by unbiased filtering to preselect the outstanding objects (from n objects) before applying the exact DP algorithm on remaining n' and C' ($n' \le 200$, $C' << C$, and massive C may not be a polynomial bound of n), which could find most optimal solutions (at least 95%) in experiments.

In current KP-researches, a variety of 0/1-KPs have been studied, including the multiple KP (0/1-mKP) [34–36], no shared x_{ij} in m knapsacks ($\sum_{j=0}^{n-1} w_j x_{ij} \le C_i$), the multidimensional KP (0/1-MKP) [37–41], ($\sum_{i=1}^{m} \sum_{j=0}^{n-1} w_{ij} x_j \le C_i$ shared x_j in m knapsacks), and the multidimensional multiple-choice KP (0/1-MMKP) [42]. However, the exact solutions of those complex KPs could not be easily found on large n. Recently, the mathematical HyMKP [34] was proposed in O(mnC) for the 0/1-mKP with most optimal solutions (in τ s) on $n \le 500$. For large n and massive C, the existing meta-heuristic algorithms for the 0/1-KP can be applied to solve the 0/1-mKP in polynomial time (but requiring the proper knapsack orders). For high performance, the exact DP could find the optimal solution of the 0/1-KP in O(nC) and O($m!nC$) for the 0/1-mKP (with $m!$ orders, C = max(C_i)) for at least 99% optimal solutions (but for small m, n, and C). In this study, we are interested to solve the 0/1-mKP for large m, n, C with the proper orders in efficient time. Our hypothesis is "For each of m knapsacks, apply unbiased filtering before using the exact DP on remaining n' and C_i' can find most optimal solutions in efficient time".

In this research, we introduce a novel research track (a hybrid approach of time-space reduction) for solving the 0/1-mKP in efficient time with expected 99% optimal solutions. In our hybrid approach, we propose the integration of DP transformation (reducing C to C'), exact-fit (best) knapsack-order (reducing $m!$ to m^2), and 3. robust unbiased filtering (for polynomial time). First, we propose the DP transformation (DPT) algorithm to find the optimal solutions of the 0/1-KP (for each of m knapsacks) in O([n^2, nC]), or O(n^2) in the best case and O(nC) in the worst case, before being applied for m knapsacks. Second, for the 0/1-mKP (m knapsacks), we propose the exact-fit (best) knapsack order (in our multi-DPT) in O(m^2[n^2, nC]) for achieving the good solutions as $m!$ orders (at least 99% optimal solutions). Third, robust unbiased filtering is incorporated to solve the 0/1-mKP in polynomial time (O(m^2n)) while retaining 99% optimal solutions. The correctness and complexity of the DPT and multi-DPT algorithms are analyzed. In experiments, the original multi-DPT and the multi-DPT + robust unbiased filtering are evaluated on random and benchmark datasets ($n \leq$ 10,000, $m \leq 100$).

The main parts of this paper are organized as follows: Section 2 reviews the related work. Section 3 presents the DPT algorithm to find the optimal solutions of the 0/1-KP in O([n^2, nC]). Section 4 proposes the multi-DPT algorithm with the exact-fit (best) knapsack order to solve the 0/1-mKP in O(m^2[n^2, nC]) and reduced to O(m^2n) by our robust unbiased filtering. Section 5 provides the algorithm analysis (correctness and complexity). Section 6 performs the experiments to evaluate the performance of our multi-DPT algorithm and robust unbiased filtering. Section 7 concludes this study.

2. Related Work

For 0/1-mKP research, finding most optimal solutions (≥99% optimal performance) in an efficient time is challenging. First, Section 2.1 reviews the exact DP algorithms to find the optimal solutions of the 0/1-KP. Section 2.2 explores the time-space reduction algorithm to solve the 0/1-KP in polynomial time. Section 2.3 reviews the recent QDGWO (quantum-inspired differential evolution with adaptive grey wolf optimizer) for the 0/1-KP. Section 2.4 explores the efficient mathematical HyMKP model for the 0/1-mKP.

2.1. 0/1-KP Solving by Dynamic Programming Algorithm

For 0/1-KP solving, let $tp[C]$ be an array of total profits, *soltp* be a maximum total-profit, *soltw* be a total-weight (≤C), and $solx[n]$ be an array of solution X (x_j = 0/1). Algorithm 1 presents the basic DP [11] with two functions (preprocessing and X-tracking on a 2D array or a matrix ($n \times C$)) to find the optimal solution (*soltp*, *soltw*, $solx[n]$) in O(nC). For example, consider $n = 6$, $C = 18$, P = {42, 39, 38, 37, 35, 38}, and W = {10, 11, 11, 8, 10, 9}, Figure 2 displays the result of preprocessing (*soltp* = 79) and X-tracking (*solx* = {1, 4} from *soltp* = 79) by applying Algorithm 1. However, using the 2D array ($n \times C$) reserves a huge space, which is not practical if $C > n^2$. Thus, the fast DP (Algorithm 2) [11] uses a 1D array (of C elements) to find *soltp* in O(nC) but without $solx[n]$ since the 1D array of the last *tp*-result cannot support the X-tracking (for all selected objects). Finally, the complete DP (Algorithm 3) [11] (p. 24) uses a 1D array for the full optimal solution (*soltp*, *soltw*, $solx[n]$) by repeating the fast DP (for k selected objects) in O(knC).

j \ d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	42	42	42	42	42	42	42	42	42
2	0	0	0	0	0	0	0	0	0	0	42	42	42	42	42	42	42	42	42
3	0	0	0	0	0	0	0	0	0	0	42	42	42	42	42	42	42	42	42
4	0	0	0	0	0	0	0	0	37	37	42	42	42	42	42	42	42	42	79
5	0	0	0	0	0	0	0	0	37	37	42	42	42	42	42	42	42	42	79
6	0	0	0	0	0	0	0	0	37	38	42	42	42	42	42	42	42	75	79

Figure 2. An example of 0/1-KP solving (n = 6, C = 18) of the basic DP (Algorithm 1): *tp*-results and *soltp* = 79 (optimal) in an nxC-matrix and X-tracking for *solx* = {1, 4}.

Algorithm 1: Basic DP with 2D array (for *soltp*, *soltw*, *solx*[*n*]) in O(*nC*).

```
for (d = 0 to C) tp [0,d] = 0;
for (j = 1 to n) do                              // preprocessing for soltp
   for (d = 0 to w_{j−1}) tp[j,d] = tp[j − 1,d];
   for (d = w_j to C) do
      if (tp[j − 1,d-w_j] + p_j > tp[j − 1,d]) then tp[j,d] = tp[j − 1,d-w_j] + p_j;
      else tp[j,d] = tp[j − 1,d];
   end for d;
end for j;
soltp = tp[n,C];
d = C; j = n; solx = ∅;                          // X-tracking for solx
do// X-tracking for solx
   while (tp[j,d] = tp[j − 1,d]) j = j − 1;               // move up
   solx = solx U {j}; pp = tp[j,d]-p_j; j = j − 1;
   while (tp[j,d − 1] ≥ pp and pp > 0) d = d − 1;        // move left
while (pp > 0 and j ≥ 1).
```

Algorithm 2: Fast DP with 1D array (for *soltp*, *soltw*) in O(*nC*).

```
for (d = 0 to C) tp[d] = 0;
for (j = 1 to n)   do        // preprocessing for soltp
      for (d = C down to w_j) do
         if (tp[d-w_j] + p_j > tp[d]) then tp[d] = tp[d-w_j] + p_j;
   end for d;
end for j; soltp = tp[C].
```

Algorithm 3: Full DP with 1D array (for *soltp*, *soltw*, *solx*[*n*]) in O(*knC*).

```
solx = ∅; C′ = C; n′ = n;
do
   for (d = 0 to C′) do tp[d] = 0;
   for (j = 1 to n′) do              // preprocessing for soltp
      for (d = C′ down to w_j) do
         if (tp[d-w_j] + p_j > tp[d]) then
            x[d] = j; tp[d] = tp[d-w_j] + p_j;
      end for d;
   end for j;
   r = x[C′];                  // find solx (a selected object)
   solx = solx U {r}; k = k + 1;
   n′ = r − 1; C′ = C′- w_r;
while (C′> 0);                 // repeat for k selected objects
soltp = tp[C].
```

2.2. 0/1-KP Solving by Time-Space Reduction Algorithm

In 2018, time-space reduction ($TS_{Reduction}$) algorithm [1] (p. 198) was proposed to solve the 0/1-KP in O(n + C') for large n by focusing on unbiased filtering. That reduction method (Algorithm 4) consists of three main steps: 1. find the best three initial solutions; 2. perform object classification and unbiased filtering; and 3. apply the full DP on the remaining objects ($n' \leq 200$); see Equations (7)–(9). The advantages of the $TS_{Reduction}$ algorithm are the efficient time complexity O(n + C') and the good performance (95% optimal solutions). See an example (n = 20, C = 100) of the 0/1-KP solved by the $TS_{Reduction}$ algorithm in Figure 3 (*soltp* = 656 (optimal)).

Step 1 Initial solutions (by GH) and improved solutions (by GH⁺)

[P/W] X	8	16	1	9	6	0	4	18	2	17	12	11	14	10	19	3	5	13	7	15		
W	3	4	5	3	4	6	8	8	8	7	7	8	11	8	9	10	11	10	11	10		
P	45	54	55	31	32	43	55	50	45	37	36	40	54	37	38	42	39	35	38	34	**tw**	**tp**
GH1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	99	652
GH1+															0	1	0	0	0	0	100	656
[P] X	1	4	16	14	18	8	2	0	3	11	5	19	7	17	10	12	13	15	6	9		
W	5	8	4	11	8	3	8	6	10	8	11	9	11	7	8	7	10	10	4	3		
P	55	55	54	54	50	45	45	43	42	40	39	38	38	37	37	36	35	34	32	31	**tw**	**tp**
GH2	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	91	560
GH2+												0	0	1	1	0	0	0	0	1	100	627
[W] X	8	9	16	6	1	0	17	12	4	18	2	11	10	19	3	13	15	14	5	7		
W	3	3	4	4	5	6	7	7	8	8	8	8	8	9	10	10	10	11	11	11		
P	45	31	54	32	55	43	37	36	55	50	45	40	37	38	42	35	34	54	39	38	**tw**	**tp**
GH3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	98	640
GH3+															0	0	0	1	0	0	99	652

Step 2 Object classification and unbiased filtering

X	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
GH1⁺	5	5	5	5	5	0	5	0	5	5	5	5	5	0	5	0	5	5	5	0
GH2⁺	3	3	3	3	3	3	0	0	3	3	3	3	0	0	3	0	3	3	3	0
GH3⁺	1	1	1	0	1	0	1	0	1	1	1	1	1	0	1	0	1	1	1	1
dw	9	9	9	8	9	3	6	0	9	9	9	9	6	0	9	0	9	9	9	1

Filtering-in (13)={0,1,2,4,6,8,9,10,11,14,16,17,18}, Filtering-out(1)={15}

Step 3 DP on n'(6)= {3,5,7,12,13,19}, C'=17

X	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	**tw**	**tp**
solx	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	0	1	1	1	0	100	656

Figure 3. An example of 0/1-KP solving (n = 20, C = 100) by $TS_{Reduction}$ (Algorithm 4), *soltp* = 656 (optimal).

Algorithm 4: Time-space reduction for 0/1-KP in O(n + C').

Step 1: Apply the GH (greedy heuristic) algorithm by sorting 3 features (P/W, P, W) for top 3 initial solutions in O(n). Note: Sorting (in each GH) relies on the major-minor keys.

- For P/W-decreasing sort, a major key is P/W and 2 minor keys are P & W.
- For P-decreasing sort, a major key is P and a minor key is W.
- For W-increasing sort, a major key is W and a minor key is P.

Step 2: Object classification and unbiased filtering in O(n).

2.1 Improve 3 initial solutions by the GH^+ algorithm.

2.2 Compute dynamic weight (dw) by integrating 3 solutions of $GH_1{}^+$ to $GH_3{}^+$ (to support unbiased selection), where $dw = wx_1 + wx_2 + wx_3$; $wx_1 = 5$, $wx_2 = 3$, and $wx_3 = 1$ if ($x_j = 1$); otherwise $wx = 0$ (when $x_j = 0$).

2.3 Classify objects and perform unbiased filtering (Group 1 (dw = 9), Group 2 (dw = 8, 6, 5), Group 3 (dw = 4, 3, 1), and Group 4 (dw = 0)).

- Filtering in/out (to reduce $n' \leq 200$): Select worth objects ($x_i = 1$) with dw = 9 (selected by all 3-GH policies).
- Select other objects ($x_i = 1$) with dw = 8, 6, 5, except uncertain α (in Group 2) and do not select worst objects ($x_i = 0$) with dw = 0, except β (in Group 4).

Step 3: Apply the DP in O(C') on remaining $n' = \alpha + |\text{Group3}| + \beta$.

$$n' = \alpha + |\text{Group3}| + \beta \leq 200 \tag{7}$$

$$\alpha = min\,(0.7 \times |\text{Group2}|, 20) \tag{8}$$

$$\beta = min\,(0.7 \times |\text{Group4}|, 200 - |\text{Group3}| - \alpha) \tag{9}$$

2.3. 0/1-KP Solving by Quantim-Inspired Differential Evolution Algorithm

Recently, the QDGWO (Quantim-inspired differential evolution with adaptive grey wolf optimizer) algorothm (Figure 4) [43] was proposed in 2021 for solving the 0/1-KP by adopting the quantum computing principles plus mutation and crossover operations. In that study, the adaptive mutation operations, the crossover operations, and the quantum observation are combined to generate new solutions as trial individuals in the solution space. In experiments, the fast convergent of QDGWO (on n = 50 to n = 3000 objects) was compared with the existing quantum-based methods (QEA, AQDE, and QSE) using maximum 1000 iterations. The QDGWO results outperformed those of existing Q-based algorithms. However, the QDGWO algorithm could not guarantee the optimal solutions, especially on the irregular datasets.

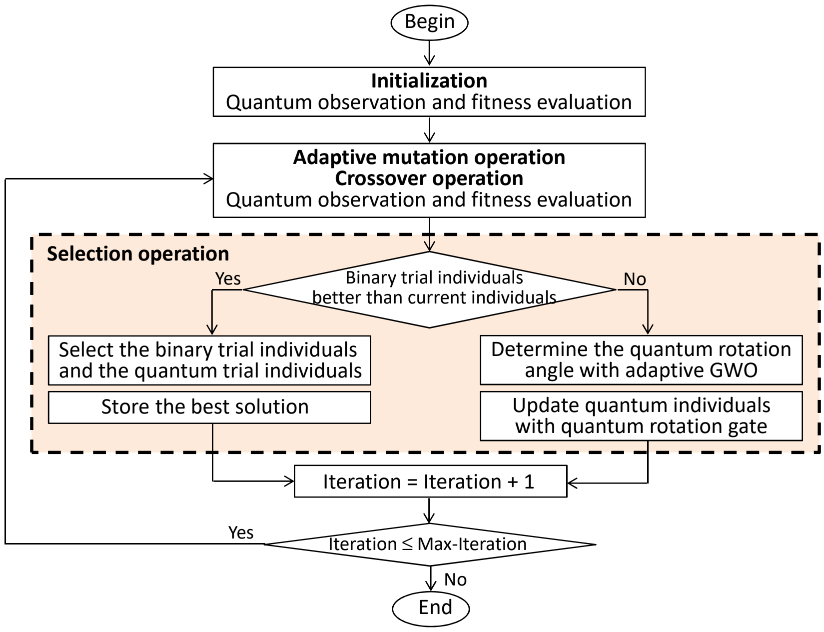


Figure 4. Framework of QDGWO (Quantum-inspired Differential (QD) evolution with adaptive Grey Wolf Optimizer (GWO)).

2.4. 0/1-mKP Solving by Mathematical HyMKP Algorithm

For solving the 0/1-mKP (m knapsacks, n objects), the mathematical HyMKP model (Algorithm 5) [34] (p. 893) was proposed. That hybrid model includes the MULKNAP (well-known partial BnB) program and the create-reflect-multigraph-MKP (Algorithm 6) [34] (p. 891) and two decomposition methods. Algorithm 6 was modified from the arc-flow model and the reflect model in O(mnC), $C = \max(C_i)$, $i = 1, 2, \ldots, m$, by starting with decreasing weights (of n items/objects) for the good initial-solution (in m knapsacks). Then, that solution was improved by the knapsack-based decomposition with v iterations (in τ secs.). In the past, such mathematical models were used to solve the classical stock problem (CSP: 1999-2017) and the cutting and packing problems (1970–1977). In experiments (on OR-benchmark datasets), the mathematical HyMKP algorithm (with time complexity O(mnC)) yielded 99.9% optimal solutions (in τ secs.) for small $n \leq 500$.

Algorithm 5: Mathematical HyMKP model.

Step 1: perform the existing preprocessing: *Instance reduction*, *Capacity lifting*, and *Item dominance*.
Step 2: call MULKNAP *branch-and-bound* (BnB) for τ secs.
if (the solution is optimal) then return.
Step 3: call ***Create-reflect-multigraph-MKP*** (Algorithm 6).
Step 4: for (i = 1 to v) do
execute *the knapsack-based decomposition*.
if (an optimal solution has been obtained) then return.
else add the resulting no-good-cut.
end for i
Step 5: if (the instance is not solved) then
execute *the reflect-based decomposition* and return.

Algorithm 6: Create-reflect-multigraph-MKP in O(mnC).

1. $N = \emptyset$; $A_s = \emptyset$; $A_r = \emptyset$; $A_c = \emptyset$; $A_l = \emptyset$; $A = \emptyset$; $s = 0$; $C = \max(C_i)_{i = 1,2,\cdots,m}$
2. for (l = 1 to $C/2$) do $T[l] = 0$;
3. $T[s] = 1$; $N = N \cup \{s\}$;
4. for (j = 1 to n) do
5.   for ($l = C/2 - 1$ down to 0) do
6.   if ($T[l] = 1$) then
7.   if ($l + w_j \leq C/2$) then
8. $A_s = A_s \cup \{l, l + w_j, j, 0)\}$; $T[l + w_j] = 1$; $N = N \cup \{(l + w_j)\}$;
9.   for (i = 1 to m) do
10.   if ($l + w_j > C/2$ and $l \leq C_i\text{-}(l + w_j)$) then
11. $A_r = A_r \cup \{l, C_i\text{-}(l + w_j), j, i)\}$; $N = N \cup \{(C_i\text{-}(l + w_j))\}$;
12.   end for i;
13.   end if;
14.   end for l;
15. end for j;
16. for (i = 1 to m) do $N = N \cup \{C_i/2\}$;
17. for (i = 1 to m) do $A_c = A_c \cup \{(C_i/2, C_i/2, 0, i\}$;
18. for ($l \in N$) do $A_l = A_l \cup \{(l, l', 0, 0)$: $l' = \min(e \in N$: $e > l)\}$;
19. $A = A_s \cup A_r \cup A_c \cup A_l$;
20. return N, A;

In particular, the arc-flow model was modified to fill a knapsack as a path in a graph, where arcs were items/objects. The looping conditions of Algorithm 6 (Lines 4–15) are similar to the basic DP (Algorithm 1) for each of m knapsacks with time complexity in O(mnC). Let (d, e, j, i) denote an arc in a set A from nodes d to e (Lines 8, 11 and 17–19).

$A_s = \{(d, d + w_j, j, 0), 1 \leq j \leq n\}$ is the set of standard item arcs.

$A_r = \{(d, C_i\text{-}(d + w_j), j, i), 1 \leq i \leq m, 1 \leq j \leq n\}$ is the set of reflected item arcs (satisfying $d + w_j > C_i/2$ and $d \leq C_i\text{-}w_j$).

$A_c = \{(C_i/2, C_i/2, 0, i)\}$ is the set of reflected connection arcs.

$A_l = \{(d, e, 0, 0), d < e\}$ is the set of loss arcs.

For solving the 0/1-mKP (in theory), the existing researches were focused on the exact algorithms, such as the branch-and-bound algorithm [35] and the mathematical HyMKP [34], where their performance results were compared to the (known) optimal solutions but those algorithms could execute in reasonable time on small $n \leq 500$ only.

For large n (in practice), each of the efficient meta-heuristic algorithms [26,30,32,43] proposed for the 0/1-KP (i.e., GA, swarm optimization, quantum computing, hybrid method, etc.) can be applied to solve the 0/1-mKP (for each of m knapsacks) with proper orders. For example, we can apply the recent QDGWO [43]) for the 0/1-KP (one knapsack) to the 0/1-mKP (m knapsacks). However, that meta-heuristic algorithm cannot guarantee the optimal solution for each knapsack, especially on the irregular datasets, and hence the total profit from many knapsacks ($m > 10$) may be near-optimal only.

Therefore, to solve the 0/1-mKP ($m > 10$) for large n, we are interested to find the high optimal performance in efficient time by our hybrid approach. In this study, our proposed algorithms (in the hybrid approach) are

1. Exact DP transformation (DPT) algorithm (in Section 3) to find the optimal solutions of the 0/1-KP (in each knapsack).
2. $m!$-to-m^2 knapsack-order reduction (in Section 4.1.2) to define the exact-fit (best) knapsack order for the 0/1-mKP (m knapsacks).
3. Robust unbiased filtering (in Section 4.2) to improve/reduce time complexity to polynomial time while retaining the high optimal performance.

In this 0/1-mKP research, we propose "a novel research track (a hybrid approach of the exact DPT + robust unbiased filtering) to solve the 0/1-mKP in efficient time with expected at least 99% optimal solutions". We start with our exact-DP transformation for 0/1-KP in $O([n^2, nC])$ over $O(nC)$ for each knapsack (in Section 3) before being applied to 0/1-mKP (m knapsacks) with the exact-fit (best) knapsack order in $O(m^2[n^2, nC])$ over $O(m![n^2, nC])$ in Section 4 and reduce it to $O(m^2 n)$ by our efficient unbiased filtering.

3. 0/1-KP Solving by DP Transformation to List-Based Time-Space Reduction

First, in our new research track (for solving the 0/1-mKP), we propose the DP transformation to list-based time-space reduction (DPT-List$_{TSR}$) algorithm to find the optimal solutions of the 0/1-KP in $O(n^2)$ in the best case and $O(nC)$ in the worst case. Our DPT-List$_{TSR}$ algorithm was renovated from the basic DP (Algorithm 1). That original DP can find the optimal solution of the 0/1-KP in $O(nC)$ by using the 2D-array ($n \times C$) in all (best, average, and worst) cases. In this study, the DPT-List$_{TSR}$ algorithm can find the optimal solution of the 0/1-KP by introducing the lists of effective nodes (e-nodes) in efficient $O([n^2, nC])$. The contribution of our DPT-List$_{TSR}$ is the forward reduction (F-reduction) in the preprocessing (for the (original) e-nodes) and the backward reduction (B-reduction) in the X-tracking (for the tight bound of the original e-nodes).

Next, to simplify our DPT-List$_{TSR}$ process (the preprocessing in Section 3.1 and the X-tracking in Section 3.2), the following data structures and proper functions are predefined. See a corresponding example of our DP transformation in Figure 5 ($n = 5$, $C = 18$).

The e-node is an effective node with improved *tp* (total profit) by object j at capacity d that is more than *tp* at previous capacity $d - 1$.

The original e-node is the original improved e-node by object j at capacity d (not by object $j + 1$ at the same d).

The F-list (forward list) is a list of e-nodes (used in object $j - 1$ and object j).

The B-list (backward list) is a list of original e-nodes (for X-tracking).

F-reduction is a function used to reduce e-nodes to the original e-nodes.

B-reduction is another function applied after finding $x_j = 1$ in X-tracking (to simplify the remaining X-tracking process).

3.1. Preprocessing of the DPT-List$_{TSR}$ Algorithm

In our preprocessing (Algorithm 7), two (temporary) F-lists of e-nodes (the previous F-list $j - 1$ and the current F-list j) must be built first; see Figure 5 ($j = 0$ to $n - 1$). Since the previous F-list $j - 1$ can inherit all e-nodes of object 0 to object $j - 1$, the current F-list j can be constructed from F-list $j - 1$ (to inherit the previous e-nodes and fulfill the current F-list j with new e-nodes before keeping only the original e-nodes in another B-list L_j). Our preprocessing can reduce the computing time and the using space on a variety of datasets, such as $O(n^2)$ in the best case and $O(nC)$ in the worst case, as proven in Section 5.2. The current F-list j of object j (w_j, p_j) can be created in two main steps. In Step 1 (Algorithm 7: Line 3), some e-nodes ($cn = (tp, d)$) from the head of F-list $j - 1$ are copied to F-list j (while ($cn.tp < p_j$ and $cn.d < w_j$)). In Step 2 (Algorithm 7: Lines 4–10 (to fulfill F-list j with new e-nodes)), from each e-node (en) of F-list $j - 1$ (head to tail) Step 2.0 checks to inherit each remaining e-node (from step 1) to F-list j (in a proper location) if it is worth (see detail in Section 5.1), Step 2.1 computes ($d = en.d + w_j$, $tp = en.tp + p_j$), and Step 2.2 adds new e-node to tails of F-list j and B-list j if ($d \leq$ C and $tp >$ TP). In this step, the desired TP (at $d = en.d + w_j$) can be decoded (from d and n_1 in F-list $j - 1$).

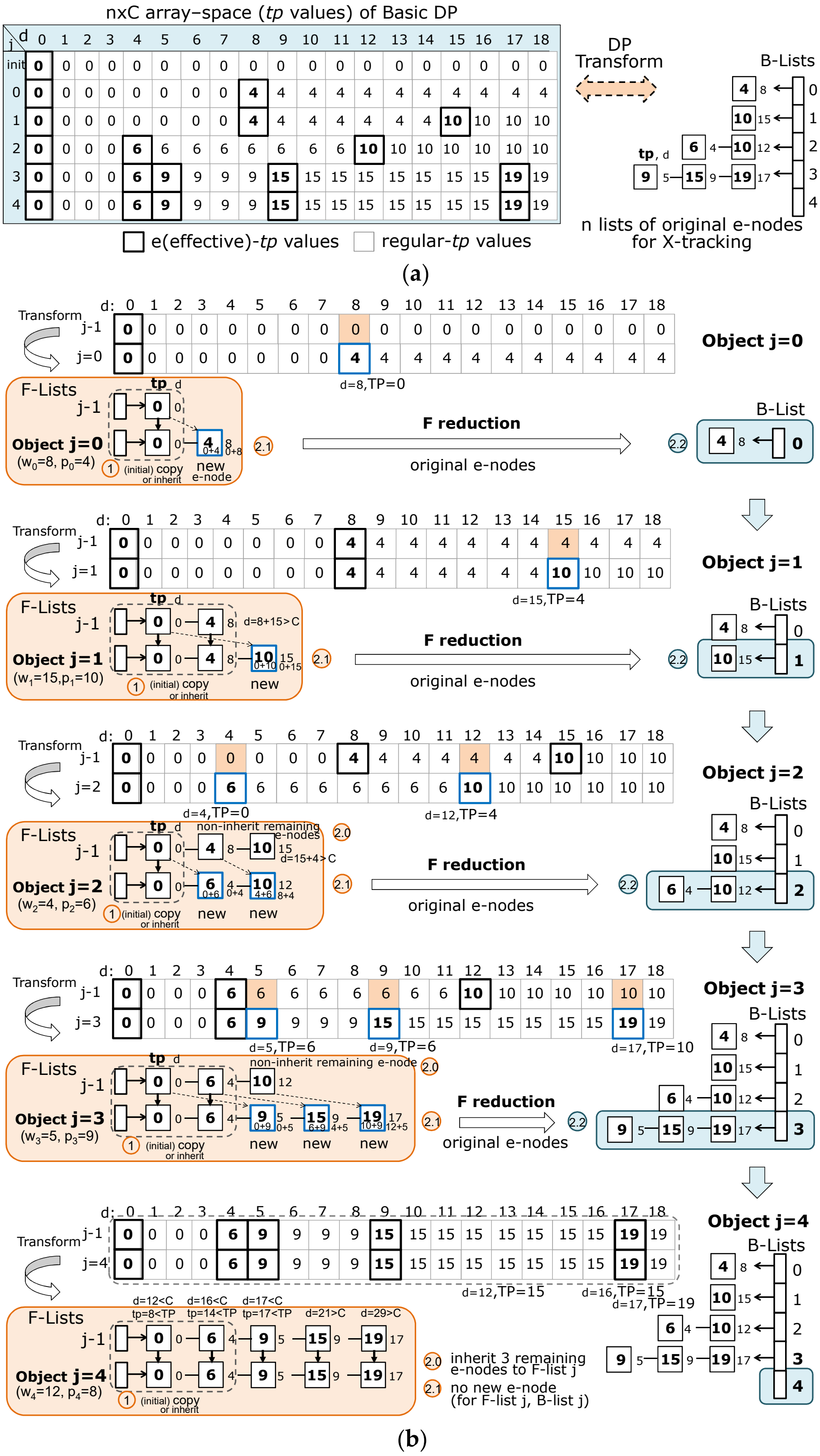


Figure 5. An example of preprocessing (n = 5, C = 18): (**a**) nxC array (tp) by the basic DP and (**b**) two F-lists and n B-lists by DPT-List$_{TSR}$ for objects j = 0–4.

Algorithm 7: Preprocessing of the DPT-List$_{TSR}$ algorithm.

create initial F-list $j-1$ (for j = 0) with one e-node (tp = 0,d = 0);
for (j = 0 to $n-1$) do // to fulfill F-list j of object j
initial copy e-nodes (cn) of F-list $j-1$ to F-list j (while ($cn.tp < p_j$ & $cn.d < w_j$)); // Step 1
e-node en = head(F-list $j-1$);
for (each e-node (en) in F-list $j-1$) do (from head to tail) // Step 2
inherit remaining e-node (if it is worth) to F-list j (in a proper location); // Step 2.0
compute $d = en.d + w_j$; $tp = en.tp + p_j$; $n_1 = en$; // Step 2.1
n_1 = decode(n_1,d); $n_2 = n_1$.next; TP = $n_1.tp$ (if $n_1.d \le d < n_2.d$);
if ($d \le C$ & tp > TP) add new e-node to tails of F-list j and B-list j; // Step 2.2
end for (F-list $j-1$);
end for j;
$soltp$ = max ($ori\text{-}en.tp$) of original e-node in B-list j;

For example (n = 5, C = 18), P = {4, 10, 6, 9, 8} and W = {8, 15, 4, 5, 12}, Figure 5a shows the B-list reduction (right) of seven original e-nodes (see detail in Figure 5b), compared to total profits = $n \times C$ = 90 elements (left) of the basic DP. Figure 5b displays the preprocessing of DPT-List$_{TSR}$ from j = 0 to 4.

- For object j = 0 (w_0 = 8, p_0 = 4), Step 1 copies the first e-node cn = (tp, d) = (0, 0) of F-list $j-1$ to F-list j (since $cn.tp < p_0$ and $cn.d < w_0$). In Step 2 (from head of F-list $j-1$), at e-node en = (0, 0), Step 2.1 computes $d = en.d + w_0 = 8$, $tp = en.tp + p_0 = 4$, and decode TP = 0 [($n_1.d$ = 0, $n_1.tp$ = 0), ($n_2.d = C$ = 18, $n_2.tp$ = 0)]. Since $d < C$ and tp = 4 > TP = 0, Step 2.2 adds new e-node = (4, 8) in F-list j = 0 and B-list L_0.
- For object j = 1 (w_1 = 15, p_1 = 10), Step 1 copies (0, 0), (4, 8) of F-list $j-1$ to F-list j (while $cn.tp < p_1$ and $cn.d < w_1$). In Step 2, at e-node en = (0, 0) of F-list $j-1$, compute d = 15, tp = 10, TP = 4 (($n_1.d$ = 8, $n_1.tp$ = 4), ($n_2.d = C$ = 18, $n_2.tp$ = 4)). Since tp = 10 > TP = 4, add new e-node (10, 15) in F-list j = 1 and B-list L_1. At en = (4, 8), d = 8 + 15 = 23 > C (no new e-node added).
- For object j = 2 (w_2 = 4, p_2 = 6), Step 1 copies (0, 0) of F-list $j-1$ to F-list j. In Step 2, at en = (0, 0) of F-list $j-1$, d = 4, tp = 6 > TP = 0, add new e-node (6, 4) in F-list j = 2 and B-list L_2. At en = (4, 8) of F-list $j-1$, d = 8 + 4 = 12, tp = 4 + 6 = 10 > TP = 4, add new e-node (10, 12). At en = (10, 15), d = 15 + 4 = 19 > C (no new e-node added).
- For object j = 3 (w_3 = 5, p_3 = 9), Step 1 copies (0, 0), (6, 4) of F-list $j-1$ to F-list j. In Step 2, at en = (0, 0) of F-list $j-1$, d = 5, tp = 9 > TP = 6 (add new e-node (9, 5)). At en = (6, 4) of F-list $j-1$, d = 4 + 5 = 9, tp = 6 + 9 = 15 > TP = 6 (add new e-node (15, 9)). At en = (10, 12), d = 17, tp = 19 > TP = 10 (add new e-node (19, 17)).
- For object j = 4 (w_4 = 12, p_4 = 8), Step 1 copies (0, 0), (6, 4) of F-list $j-1$ to F-list j. In Step 2, at en = (0, 0) of F-list $j-1$, compute d = 12, tp = 8 < TP = 15 (no new e-node added). At en = (6, 4), compute d = 4 + 12 = 16, tp = 6 + 8 = 14 < TP = 15 (no new e-node added). At en = (9, 5), add this remaining (worth) e-node to F-list j and compute d = 5 + 12 = 17, tp = 9 + 8 = 17 < TP = 19 (no new e-node added). At en = (15, 9), add this remaining e-node to F-list j and compute d = 9 + 12 = 21 > C (no new e-node added). At en = (19, 17), add this remaining e-node to F-list j and compute d = 17 + 12 = 29 > C (no new e-node added).

Figure 6 shows another example (n = 15, C = 40), P = {17, 14, 14, 15, 12, 16, 13, 15, 16, 18, 22, 24, 21, 13, 11} and W = {11, 14, 7, 5, 10, 12, 5, 8, 5, 11, 9, 10, 8, 5, 6}. The 223 e-nodes (black + gray) and 103 original e-nodes (black) are reduced over $nC = 15 \times 40$ = 600 cells.

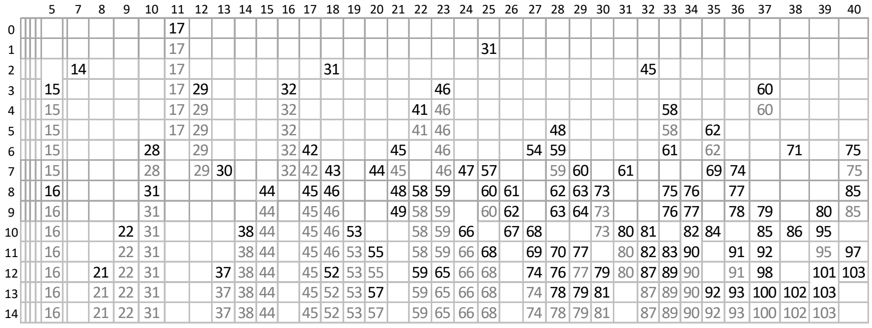


Figure 6. An example (n = 15, C = 40) of initial reduction (15 × 40-array (=600) to e-nodes (=223)) and original e-nodes (=103) after F-reduction.

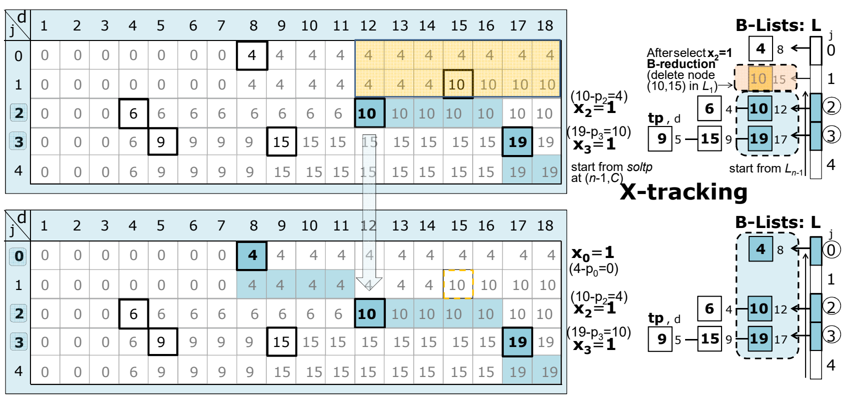
3.2. X-Tracking of the DPT-List$_{TSR}$ Algorithm

Our X-tracking for *solx*[n] (Algorithm 8) works on the B-lists (of the original e-nodes), similar to the effective-*tps* in the 2D array of the basic DP (Algorithm 1: Lines 9–15). On B-list X-tracking, moving left/up (from the B-list L_{n-1}) is processed by the back pointer in each B-list L_j. Moreover, to simplify the remaining of the X-tracking (after selecting any x_j = 1), the B-reduction (Algorithm 8: Line 4) is used to delete some of the original e-nodes of B-lists 0 to $j-1$ (if *e-node.d* $\geq$ *node-j.d*) after selecting x_j (of *node-j*(tp, d)).

Algorithm 8: X-tracking (for *solx*) on B-lists of the DPT-List$_{TSR}$ algorithm.

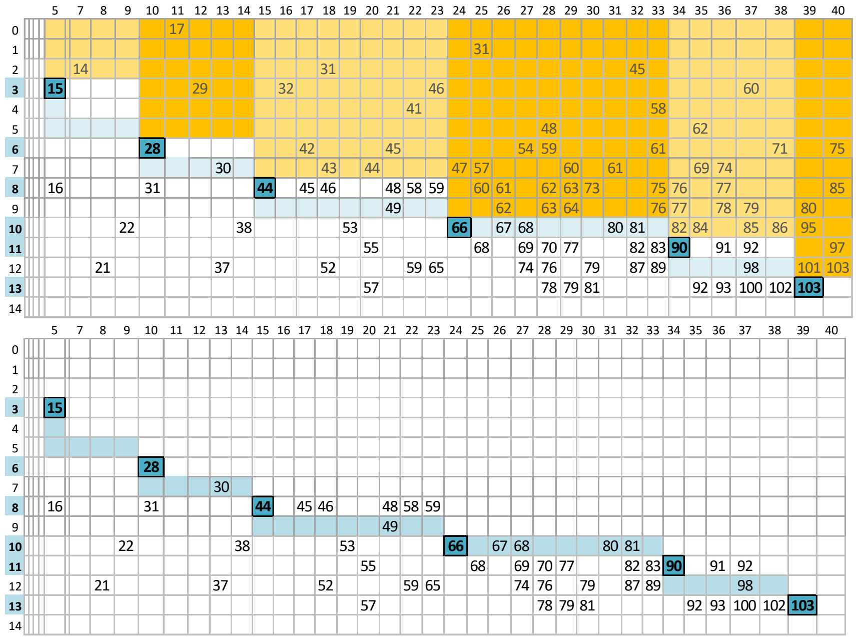
1. start from B-list L_{n-1} up to L_j(tp = *soltp*); tp = $L_j.tp$; tw = $L_j.tw$;
2. while (tp > 0 & $j \geq 0$) do
3. *solx* = *solx*∪{j}; // union a select object j (x_j = 1) with *node-j*(tp, d);
4. call B-reduction (delete *e-node*(tp, d) of L_0 to L_{j-1} if *e-node.d* $\geq$ *node-j.d*);
5. tp = tp-p_j; tw = tw-w_j; // update remaining (tp, tw);
6. j = moveLEFT-UP(n, C, j, tp, tw, L); // move to next *e-node*(tp, d = tw);
7. end while.

Figure 7a displays an example of X-tracking (n = 5) on B-lists (Figure 5). From list L_{n-1}, moving starts from $n-1$ = 4 with tp = 19 (tw = 17) to select x_3 = 1. Next, with tp = 10 (tw = 12) after selecting x_2 = 1, B-reduction deletes an unused e-node (tp = 10, d = 15) in list L_1 (since *e-node.d* (= 15) > *node-j.d* (= 12)) and finally selects x_0 = 1. Figure 7b shows a complex example of X-tracking and B-reduction (n = 15) on B-lists (dark color in Figure 6), i.e., after selecting x_{13} = 1, seven original e-nodes (at j = 0–12, d = 39–40) in B-lists are deleted, after selecting x_{11} = 1, 14 original e-nodes (at j = 0–10, d = 34–38) in B-lists are deleted, after selecting x_{10} = 1, 21 original e-nodes (at j = 0–9, d = 24–33) in B-lists are deleted, after selecting x_8 = 1, 8 original e-nodes (at j = 0–7, d = 15–23) in B-lists are deleted, etc.



(**a**)

Figure 7. *Cont.*

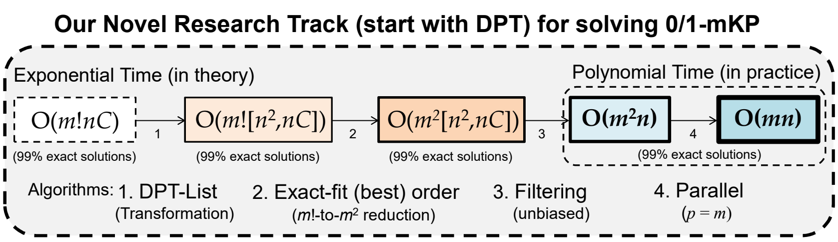


(**b**)

Figure 7. X-tracking and B-reduction: (**a**) an example (n = 5) of X-tracking for selecting j = 3, 2, 0 and (**b**) an example(n = 15) of X-tracking and B-reduction for selecting j = 13, 11, 10, 8, 6, 3.

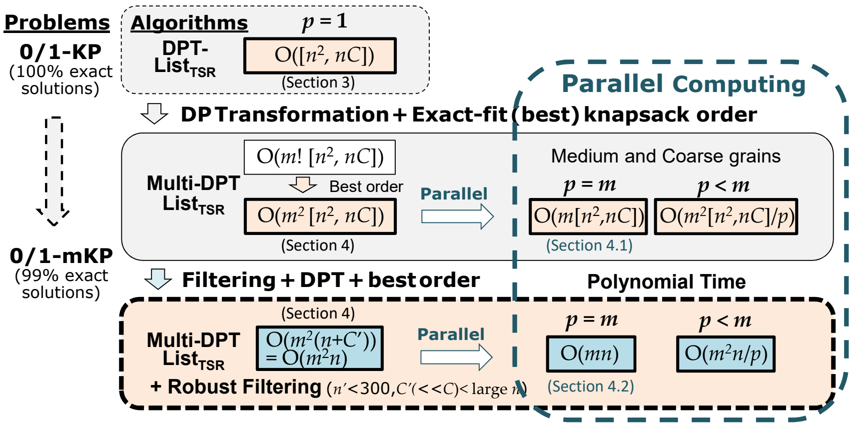
4. 0/1-mKP Solving by Multi-DPT-List$_{TSR}$ Plus Unbiased Filtering in Efficient Time

To solve the 0/1 mKP, we propose an efficient novel research track (Figure 8a) by starting with the exact DP in exponential time O($m!nC$) and ending with polynomial time O(m^2n) by our efficient unbiased filtering, while retaining 99% optimal solutions. To solve this complex NP-hard problem (0/1-mKP), we propose three effective algorithms: 1. the multi-DPT-List$_{TSR}$ algorithm (for m knapsacks) by applying the exact DPT-List$_{TSR}$ algorithm (in Section 3), 2. the exact-fit (best) knapsack order (with m!-to-m^2 reduction by applying the DPT-List$_{TSR}$) to achieve the good results as m! orders, and 3. robust unbiased filtering (for polynomial time). Moreover, Figure 8b presents a variety of our parallel reduction models based on medium and coarse grains ($p \leq m$ processors). First, in Section 4.1, we propose the multi-DPT-List$_{TSR}$ algorithm to find 99% optimal solutions (of the 0/1-mKP) in O($m^2[n^2, nC]$) and hence O($m[n^2, nC]$) in parallel ($p = m$ processors). Second, in Section 4.2, robust unbiased filtering is incorporated with our multi-DPT-List$_{TSR}$ in O($m^2(n + C')$) or O(m^2n) with C' (<<C) < large n and O(mn) in parallel ($p = m$ processors).



(**a**)

Figure 8. *Cont.*



(**b**)

Figure 8. (**a**) Our novel research track for solving 0/1-mKP in polynomial time with 99% optimal solutions and (**b**) our multi-DPT-List$_{TSR}$ algorithm and efficient parallel models.

4.1. Efficient Multi-DPT-List$_{TSR}$ Algorithm for Solving 0/1-mKP

The 0/1-mKP is one of the hardest KPs since it is difficult to find the optimal solutions in $O((m+1)^n)$ by the BnB algorithm, except on small n. For large n, we study the DPT-List$_{TSR}$ algorithm (in Section 3) first for the 0/1-KP since its optimal solution can be computed in $O([n^2, nC])$ in each knapsack (or the internal effect for the 0/1-mKP (m knapsacks)). Next, we can use $m!$ orders (m knapsacks) directly (for the external effect) in our multi-DPT-List$_{TSR}$ algorithm for at least 99% optimal solutions in $O(m!\ [n^2, nC])$ since the more orders there are, the higher the optimal precision. However, that exponential complexity cannot support large m, n, and C. Thus, we propose two efficient order reductions: 1. the top nine (knapsack) orders in $O(m[n^2, nC])$ for the regular datasets and 2. the exact-fit (best) knapsack order in $O(m^2[n^2, nC])$ for the irregular datasets.

4.1.1. Top Nine Knapsack Orders for Regular Datasets

Initially, the top nine (knapsack) orders are introduced in our multi-DPT-List$_{TSR}$ algorithm, which are good enough to solve the 0/1-mKP with 99% optimal solutions for the regular datasets. Each of the top nine orders is obtained by sorting m capacities (C_i, $i = 1, 2, 3, \ldots, m$). For example ($m = 5$), the forward order (F) of capacities $C = (66, 26, 80, 96, 70)$ is (1, 2, 3, 4, 5), and the top three orders are increasing (inc) = (2, 1, 5, 3, 4), decreasing (dec) = (4, 3, 5, 1, 2), and combined inc-dec = (2, 4, 1, 3, 5). In this study, the top nine effective orders include increasing (inc), decreasing (dec), combining inc-dec, combining dec-inc, forward (F), backward (B), odd-even (of F), odd-even (of inc), and odd-even (of dec); see a corresponding example in Figure 9. Moreover, each result of the top nine orders can be improved by the Latin square (LS) of m permutations to achieve at least 99% optimal solutions (for the regular datasets). In practice, the partial LS (first nine permutations) of the top nine orders are used to preserve the complexity in $O(m[n^2, nC])$ for $m > 9$.

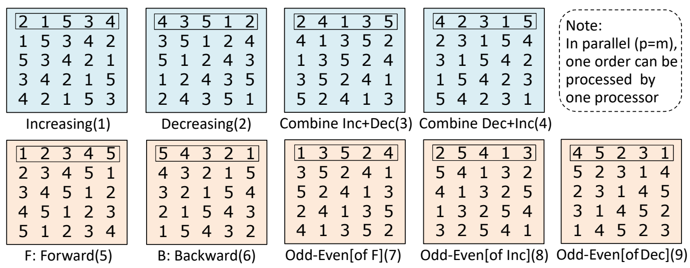


Figure 9. An example of the top nine (knapsack) orders and their Latin squares (of m permutations) for $m = 5$ knapsacks and $C = (66, 26, 80, 96, 70)$.

Algorithm 9 (multi-DPT-List$_{TSR}$) is proposed to solve the 0/1-mKP for each order (of top nine orders/Latin squares of nine orders) in $O(m[n^2, nC])$. Moreover, in some cases, there are different Xs (in X-tracking from many *soltws* of max *soltp*), called the nonunique solution Xs, in each knapsack. For the 0/1-KP, X-tracking can start at (*soltp*, min *soltw*) or (*soltp*, max *soltw*) for different Xs. For the 0/1-mKP (Algorithm 9: Lines 5–6), knapsack i ($\leq m-1$) should start at (*soltp*, max *soltw*) to allow the better result for the remaining knapsacks, while the last one ($i = m$) can start at (*soltp*, min *soltw*). For example, given a dataset (n = 25, m = 4, C = (20, 30, 40, 50), P = {17, 10, 14, 18, 14, 15, 27, 11, 12, 16, 24, 13, 22, 26, 15, 16, 18, 22, 19, 24, 21, 13, 14, 11, 28}, and W = {11, 4, 14, 3, 7, 5, 4, 4, 10, 12, 6, 5, 7, 6, 8, 5, 11, 9, 5, 10, 8, 5, 3, 6, 8}). In knapsacks $K_1 - K_2$, there are unique X-tracking results, but nonunique X-results occur in knapsack K_3 (C_3 = 40, n^* = 15, j = {0, 2, 4, 5, 8, 9, 11, 14, 15, 16, 17, 19, 20, 21, 23}). Figure 10a shows the result (393) when starting X-tracking at (103, 39), min *soltw* = 39 in K_3. Figure 10b shows the optimal result (398) when starting X-tracking at (103, 40), max *soltw* = 40 in K_3, leading to the better result in knapsack K_4 (select $j = 0$ (w_0 = 11, p_0 = 17) instead of $j = 8$ (w_8 = 10, p_8 = 12) in Figure 10a). Note: In parallel ($p = m$), we can assign one order per processor for at most m permutation orders (for independent computing for p solutions (at the same time) before selecting the best result).

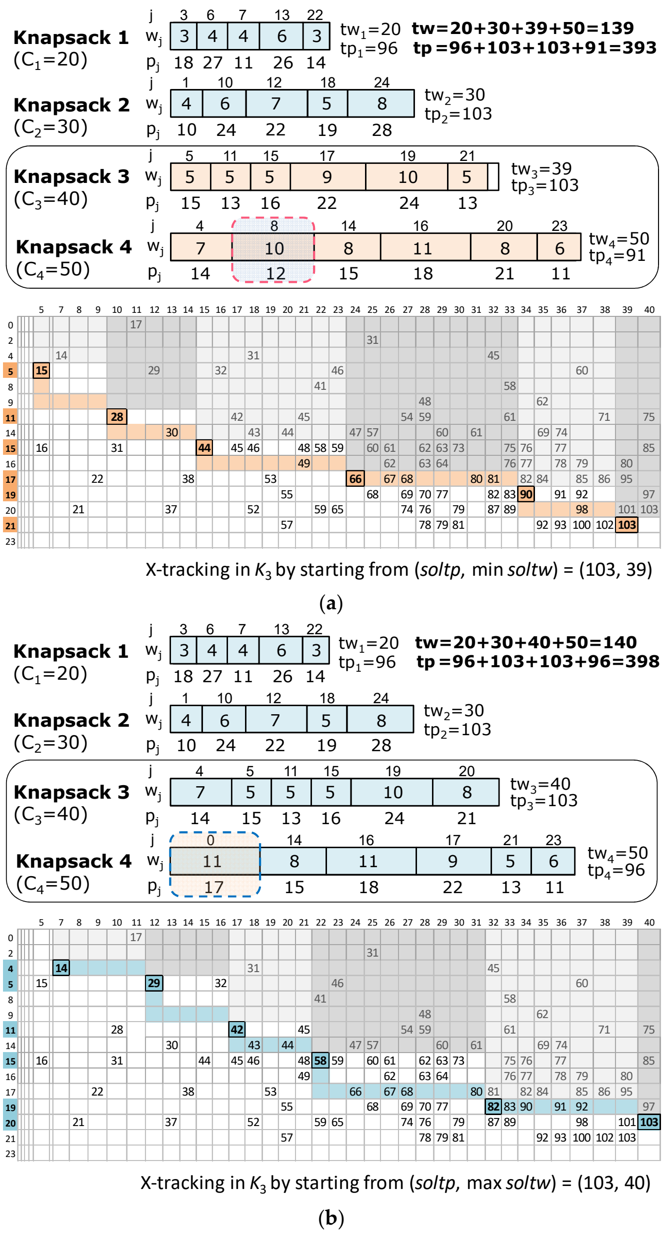


Figure 10. An example (n = 25, m = 4, C = (20, 30, 40, 50)) of two X-tracking with nonunique solution Xs in K_3: (**a**) start tracking from (*soltp*, min *soltw*) and (**b**) start tracking from (*soltp*, max *soltw*).

Algorithm 9: Multi-DPT-List$_{TSR}$ for one proper order: O(m[n^2,nC]).

1. $n^* = n$;
2. for (i = 1 to m) do
3. apply DPT-List$_{TSR}$ (n^* objects) on knapsack i (K_i);
4. call Algorithm 7; // preprocessing (of DPT-List$_{TSR}$)
5. if ($i < m$) start X-tracking at (max tp, max tw);
6. else ($i = m$) start X-tracking at (max tp, min tw);
7. call Algorithm 8; // X-tracking (of DPT-List$_{TSR}$) for max tp
8. Total profit = Total profit + max tp;
9. update n^* (exclude k_i selected objects of knapsack i);
10. end for i;
11. return Total profit.

4.1.2. The Exact-Fit (Best) Knapsack Order for Regular and Irregular Datasets

For the irregular datasets, we may use all possible m! orders to find at least 99% optimal solutions in O(m! [n, nC]) but m! orders work on small m only. Thus, for large m, to achieve the optimal precision as m! orders, we propose the exact-fit (best) knapsack order (Algorithm 10) in O(m^2[n^2, nC]), where both internal and external effects must be solved by the exact DPT-List$_{TSR}$ algorithm. For the external effect (among m knapsacks), the DPT-List$_{TSR}$ algorithm is used for computing the exact (TP_i, TW_i) in each of available knapsacks before selecting K_i with the best exact-fit$_i$ = $min(dFit_i)$, where different Fit_i ($dFit_i$) = $C_i - TW_i$, $\forall i \leq m$. For instance, Figure 11 shows the exact-fit (best) order for m = 5, C = (66, 26, 80, 96, 70), n = 33, P = {18, 44, 7, 21, 22, 29, 42, 24, 36, 17, 13, 23, 12, 25, 15, 41, 15, 19, 33, 5, 8, 18, 28, 25, 12, 30, 19, 14, 48, 25, 16, 23, 25}, and W = {6, 12, 16, 12, 14, 14, 5, 12, 12, 15, 10, 17, 14, 9, 19, 5, 7, 12, 8, 14, 14, 15, 14, 12, 7, 6, 13, 15, 10, 14, 8, 12, 10}. In Figure 11b, the best order (2, 4, 1, 5, 3) is computed in $m(m + 1)/2$ = 15 steps by our exact DPT-List$_{TSR}$ algorithm to achieve the optimal result (726).

- In the first K_i selection, there are m $dFit_i$-results (in m = 5 steps) with two $min(dFits)$ = 0 (in K_2, K_3) and K_2 (min C_2) is selected (see conditions in Step 2 of Algorithm 10).
- In the second K_i selection, there are 4 $dFit_i$-results and K_4 ($min(dFit_4)$ = 0) is selected.
- In the third K_i selection, there are 3 $dFit_i$-results and K_1 ($min(dFit_1)$ = 0) is selected.
- In the fourth K_i selection, there are 2 $dFit_i$-results and K_5 ($min(dFit_5)$ = 0) is selected.
- In the fifth K_i selection, the last K_3 ($min(dFit_3)$ = 0) in the last step is selected.

Moreover, for critical decisions in some datasets, there are equal $min(dFit_i)$s in $K_{i'} - K_{i''}$. Then, three extra policies (Step 2 in Algorithm 10) are introduced to find the best of the three best results (for the good results as m! orders as much as possible).

Algorithm 10: multi-DPT-List$_{TSR}$ (the exact-fit (best) knapsack order).

Step 1: apply DPT-List$_{TSR}$ for (TP_i,TW_i) on each of m knapsacks in O(m[n^2, nC]) and O([n^2, nC]) in parallel ($p = m$).

Step 2: select best K_i with $min(dFit_i)$; $dFit_i = C_i - TW_i$ (i = 1, 2, . . . , m).

In Step 2, for critical $min(dFit_i)$, each of the three policies is applied.

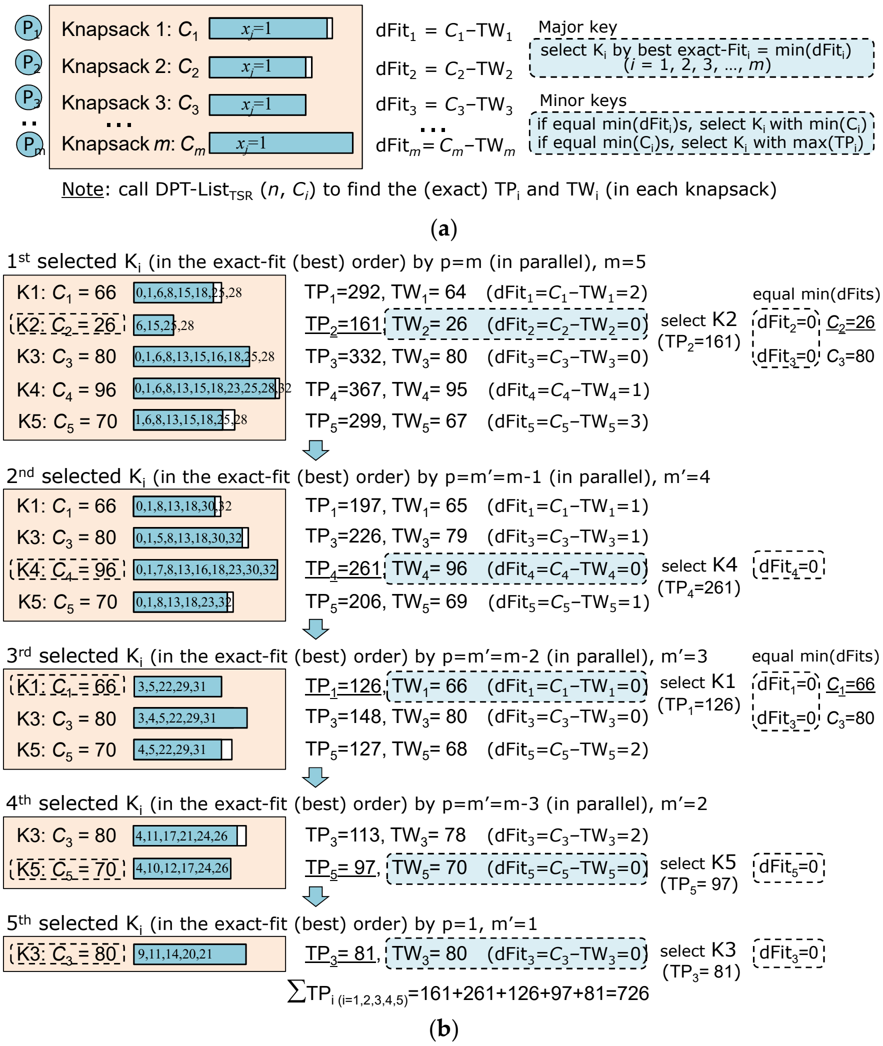
Policy 1: if (there are equal $min(dFit_i)$s), select best K_i with $min(C_i)$; if (there are equal $min(C_i)$s), select best K_i with $max(TP_i)$;

Policy 2: if (there are equal $min(dFit_i)$s), select best K_i with $max(TP_i/TW_i)$;
if (there are equal $max(TP_i/TW_i)$s), select best K_i with $min(C_i)$;
if (there are equal $min(C_i)$s), select best K_i with $max(TP_i)$;

Policy 3: if (there are equal $min(dFit_i)$s), select best K_i with $max(TP_i)$; if (there are equal $max(TP_i)$s), select best K_i with $min(C_i)$;

Step 3: update unselected $n^* = n - k$ and $m' = m - 1$.

Step 4: repeat Step 1–3 on n^* and m' until m' = 1.



(**a**) (**b**)

Figure 11. (**a**) The exact-fit policy for the best knapsack order and (**b**) an example of n = 33, m = 5, C = (66,26,80,96,70) to find the best order (2,4,1,5,3) in $m(m + 1)/2$ = 15 steps and the optimal result (726).

In parallel, the multi-DPT-List$_{TSR}$ (the exact-fit (best) order) can be processed in O(m[n^2, nC]) by p = m. However, O(mnC) in the worst case is not efficient for large m, n, and C. Thus, in Section 4.2, robust unbiased filtering (our key contribution) is presented in efficient O(m^2n) by p = 1 and O(mn) by p = m while retaining 99% exact precision.

4.2. Efficient Robust Unbiased Filtering for Polynomial Time Reduction

In our novel research track (Figure 8a), the contribution in polynomial time is achieved by robust unbiased filtering in O($m^2(n + C')$) or O(m^2n) on C' (<<C) < large n while retaining 99% optimal solutions. Our (fast and efficient) unbiased filtering can select the outstanding objects (from n objects), and only uncertain objects (n' < 300) are considered by the DPT-List$_{TSR}$ algorithm (in each knapsack). For the 0/1-mKP, the parameter (γ, α, β)-setting (in Equations (10)–(14)) was our key contribution to retain 99% optimal precision, as in our previous work (Algorithm 4) [1]. Usually, the critical and uncertain objects (γ, α, and β) could not be easily found. In this study, we performed the experiment on a variety of datasets (including the critical datasets) to classify objects into four groups (see Figure 12) before performing the efficient unbiased filtering. Variables (γ, α) refer to some critical objects (in Groups 1–2), another variable β refers to other critical objects (in Group 4), and most uncertain objects (U) are in Group 3.

$$n' = \gamma + \alpha + U + \beta < 300 \quad (10)$$

$$\gamma = min\ (10, 0.15 \times |\text{Group1}|);\ max\ \gamma = 10 \quad (11)$$

$$\alpha = min\ (25, 0.85 \times |\text{Group2}|);\ max\ \alpha = 25 \quad (12)$$

$$\beta = min\ (50, 0.70 \times |\text{Group4}|);\ max\ \beta = 50 \quad (13)$$

$$U = min\ (200, |\text{Group3}|);\ max\ U = 200 \quad (14)$$

From the four groups of object classification (in Figure 12), the dynamic critical region was studied to limit the critical/uncertain objects ($n' < 300$) after filtering while retaining 99% optimal precision. For large n, the variable n' is $\gamma + \alpha + \beta + U = 10 + 25 + 50 + 200 = 285$ since for large C there are a large number of filtering-in objects ($x_j = 1$) and for small C there are a large number of filtering-out objects ($x_j = 0$). Efficient filtering (in Algorithm 11: Line 3) is required (in each K_i) before applying the DPT-List$_{TSR}$ algorithm to n' and C_i'. Note: n' ((temporary) remaining objects after filtering) and n^* (remaining objects for next knapsack) are different. For example, Figure 13 shows the result of object classification for filtering ($n = 25$, $m = 2$, $C = (30, 40)$, $P = \{17, 10, 14, 18, 14, 15, 27, 11, 12, 16, 24, 13, 22, 26, 15, 16, 18, 22, 19, 24, 21, 13, 14, 11, 28\}$, and $W = \{11, 4, 14, 3, 7, 5, 4, 4, 10, 12, 6, 5, 7, 6, 8, 5, 11, 9, 5, 10, 8, 5, 3, 6, 8\}$) with four-group classification (P/W-rank in each group). Figure 14 demonstrates the result of filtering-in three objects (6, 13, 3) in knapsack K_1 ($C = 30$) and (temporary) filtering-out four objects (0, 9, 8, 2). For the remaining $n' = 18$ and $C' = 17$, the DPT-List$_{TSR}$ selects three objects (22, 10, 24). Then, there are remaining $n^* = 19$, including (0, 9, 8, 2). For knapsack K_2 ($C = 40$), the filtering selects three objects (18, 12, 15), and the DPT-List$_{TSR}$ selects five objects (5, 7, 11, 21, 1) from $n' = 14$, $C_2{}' = 18$. The result of our multi-DPT-List$_{TSR}$ + robust filtering is 256 (optimal). In addition, for the irregular datasets, our robust unbiased filtering can select some of n objects before packing the remaining n' (<300) by the DPT-List$_{TSR}$ in each knapsack. Figure 15 shows the optimal solution (726) by our efficient filtering, similar to Figure 11 (by our original multi-DPT-List$_{TSR}$). See the experimental results of regular and irregular datasets in Section 6.

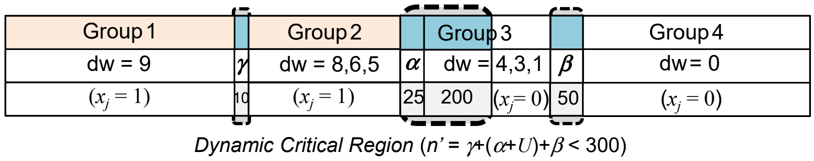


Figure 12. Four groups of object classification and efficient filtering for remaining $n' < 300$.

3 GH+ solutions and dw (dynamic weight) for filtering

X	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
P	17	10	14	18	14	15	27	11	12	16	24	13	22	26	15	16	18	22	19	24	21	13	14	11	28
W	11	4	14	3	7	5	4	4	10	12	6	5	7	6	8	5	11	9	5	10	8	5	3	6	8
P/W	1.5	2.5	1	6	2	3	6.8	2.8	1.2	1.3	4	2.6	3.1	4	2	3.2	1.6	2	4	2.4	2.6	2.6	4.7	1.8	3.5
GH1+	0	0	0	5	0	0	5	0	0	0	5	0	0	5	0	0	0	0	0	0	0	0	5	0	5
GH2+	0	0	0	0	0	0	3	0	0	0	0	0	3	3	0	0	0	0	3	0	0	0	0	0	3
GH3+	0	1	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0
dw	0	1	0	6	0	0	9	1	0	0	5	0	3	9	0	0	0	0	4	0	0	0	6	0	8

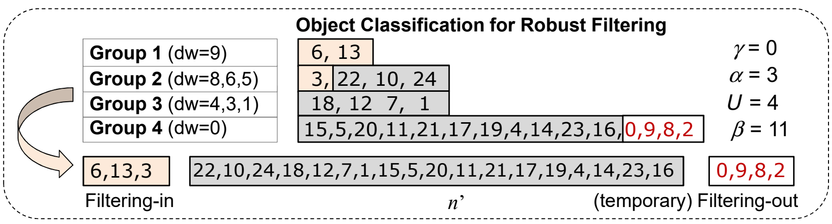


Figure 13. An example ($n = 25$, $m = 2$, $C = (30, 40)$) and object classification for knapsack$_1$ (K_1).

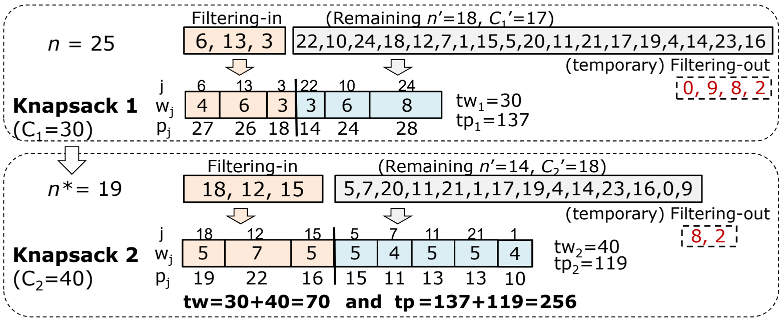
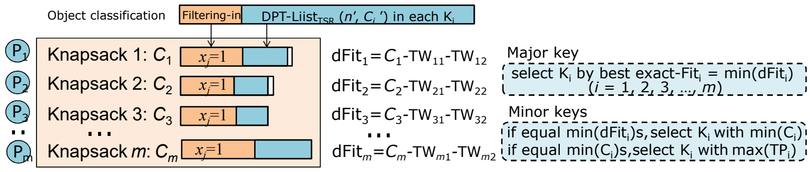
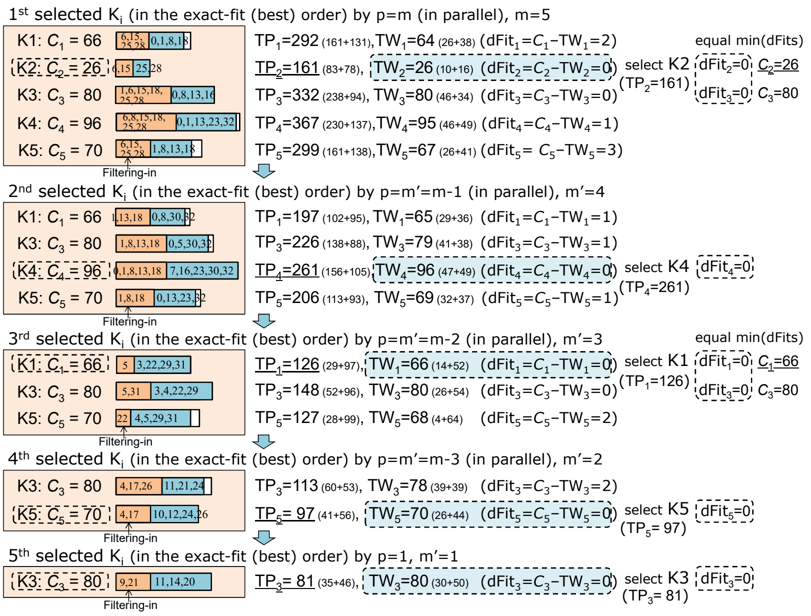


Figure 14. An example of robust unbiased filtering before applying DPT-List$_{\text{TSR}}$ on n' in K_1 and K_2.



(**a**)



(**b**)

Figure 15. (**a**) The exact-fit policy plus efficient filtering for the best knapsack order and (**b**) an example of $n = 33$, $m = 5$, $C = (66, 26, 80, 96, 70)$ to find the best order (2, 4, 1, 5, 3) in $m(m + 1)/2 = 15$ steps and the optimal result (726) by the multi-DPT-List$_{\text{TSR}}$ + robust unbiased filtering.

Algorithm 11: Multi-DPT-List$_{\text{TSR}}$ + robust unbiased filtering.

```
n* = n;
for (i = 1 to m) do
   do object classification and unbiased filtering (for Filter-tp) on n*;
   apply DPT-ListTSR (n′ < 300) on knapsack i (Ci′);
      call Algorithm 7 (preprocessing on remaining n′, Ci′);
      if (i < m) start = (max tp,max tw) else start = (max tp,min tw);
      call Algorithm 8 (X-tracking on n′ for solx from max tp);
      Total profit = Total profit + Filter-tp + max tp;
   update n* (exclude ki selected objects of knapsack i);
end for i;
return Total profit.
```

5. Analysis of Proposed Algorithms

The correctness of the DPT-List$_{TSR}$ algorithm for solving the 0/1-KP was proven in Section 5.1 and its complexity was analyzed in Section 5.2. For solving the 0/1-mKP, the high (optimal) precision (as m! orders) of the exact-fit (best) knapsack order was presented in Section 5.3. Finally, the 99% optimal precision of the robust unbiased filtering for solving the 0/1-KP and the 0/1-mKP were analyzed in Section 5.4.

5.1. Correctness of the DPT-List$_{TSR}$ Algorithm

The DPT-List$_{TSR}$ algorithm (in Section 3) was designed to solve the 0/1-KP in O([n^2, nC]) on the efficient lists, which can find the optimal solutions as the best DP (Algorithm 1: O(nC) on a 2D-array ($n \times C$)) before being applied in each of m knapsacks.

Our DPT-List$_{TSR}$ algorithm can reduce not only the redundant computing time but also the space consumption (of the basic DP: Algorithm 1) while retaining the correctness. Our focus is the DP transformation of the 2D array (nxC) to the efficient lists of e-nodes. Our preprocessing (Algorithm 7) employs two (temporary) F-lists (of objects $j-1$ and j) to inherit all worth e-nodes (of objects 0 to $j-1$) and compute new e-nodes (improved *tp values* by the current object j) before saving only the original e-nodes in B-list j.

To clarify our correct transformation, Figure 16 shows the construction of e-nodes (of F-lists j) in Figure 5 (n = 5). For object j = 1 (p_1 = 10, w_1 = 15), Figure 16a displays the F-list j construction. After the initial copy of two e-nodes (cn = (tp, d) = (0, 0) and (4, 8)) from F-list $j-1$ (while $cn.tp < p_1$ and $cn.d < w_1$) to F-list j, the rest of F-list j is fulfilled. For the first e-node en = (0, 0) of F-list $j-1$, a new e-node (10, 15) with d = 15 < C and tp = 10 > TP = 4 is added to the end of F-list j. For the next en = (4, 8), compute d = 8 + 15 = 23 > C (no new e-node is added). For object j = 3 (p_3 = 9, w_3 = 5), Figure 16b shows the F-list j construction in three steps. After the initial copy of two e-nodes (cn = (0, 0) and (6, 4)) from F-list $j-1$ (while $cn.tp < p_3$ and $cn.d < w_3$) to F-list j, the rest of F-list j is fulfilled. For en = (0, 0), a new e-node (9, 5) is added to F-list j. Second, for en = (6, 4), a new e-node (6 + 9, 4 + 5) = (15, 9) is added to F-list j. Third, for en = (10, 12), this remaining e-node is not inherited, whereas a new e-node (10 + 9, 12 + 5) = (19, 17) is added to F-list j. Note: Function "inherit remaining e-node" (in Algorithm 7: Line 6) is presented in Figure 16b; see the complex inherited results in Figure 6 (n = 15). Finally, our X-tracking (Algorithm 8) can find $solx[n]$ from the original e-nodes (on the B-lists in Figure 7), similar to the basic DP (on the 2D-array).

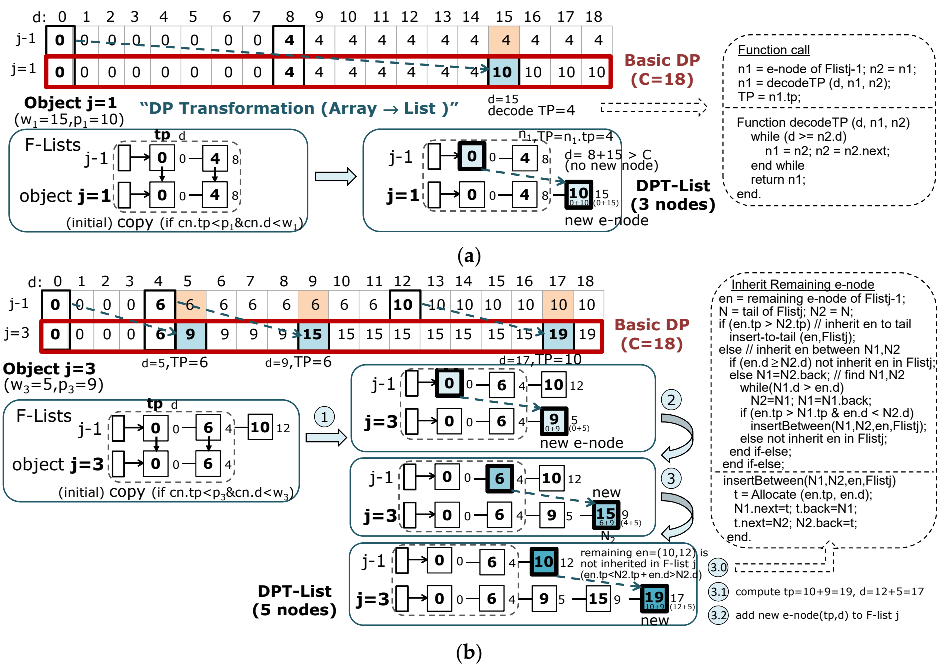


Figure 16. An example of the correct F-list j construction (n = 5 in Figure 5): (**a**) F-list j = 1 (add a new e-node to tail of F-list j) and (**b**) F-list j = 3 (add each of three new e-nodes to tail of F-list j).

5.2. Complexity Analysis of the DPT-List$_{TSR}$ Algorithm

The time complexity of our DPT-List$_{TSR}$ algorithm for solving the 0/1-KP is O([n^2, nC]), according to the efficient reduction of computing time and using space; see Figure 17 (our efficient time-space reduction). Our time complexity depends on the number of e-nodes of the (temporary) F-list j for all $j = 0, 1, 2, 3, \ldots, n-1$, where ∣F-list j∣ ≤ 2∣F-list $j-1$∣; see a simple example ($n = 5$) in Figure 5. The (initial) F-list j contains one e-node $(tp, d) = (0, 0)$. For object $j = 0$, there are at most two e-nodes (≤2 nodes). For object $j = 1$ (≤ 4 nodes) and for any j ($\leq 2 \times 2^{j-1}$ nodes), the time complexity of our DPT-List$_{TSR}$ algorithm can be the best, average, or worst cases, depending on the datasets. Figure 5 displays one of the best cases ($n = 5$, $C = 18$, e-nodes = 19, and original e-nodes = 7, reduced from $nC = 90$ elements). Thus, in this analysis, the best and worst cases can be derived as follows:

- Best case: Total steps (n objects ($j = 0$ to $n-1$)) are approximately $1 + 2 + 4 + \ldots + 2(j + 1) + \ldots + 2n \approx n(n + 1) = O(n^2)$.
- Worst case (rarely occurs): Total steps are approximately $1 + 2 + (\leq 4) + (\leq 8) + \ldots + (\leq 2 \times 2^j) + \ldots + (\approx n \times C/2) = O(nC)$.

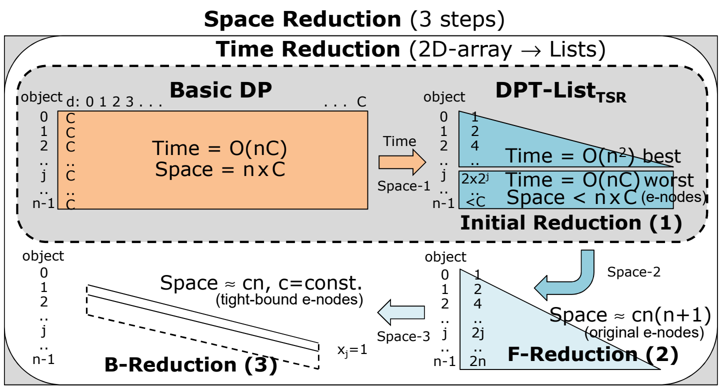


Figure 17. Time and space reduction of DPT-List$_{TSR}$ for the 0/1-KP.

The worst case (the e-nodes of F-list$_{n-1}$ = ∣F-list$_{n-1}$∣ ≈ C) hardly occurs due to some remaining e-nodes of F-list $j-1$ are not inherited to F-list j (see a clarified example in Figure 16b) and no additional new e-nodes in F-list j when considering some (worst) objects j (such as large w_j or tiny profit p_j) by two conditions: 1. $node.tp + p_j < tp[d]$ (no improved tp) and 2. $node.d + w_j > C$ (at $d + w_j$, object j cannot be packed in the knapsack), such as no additional new e-node for object $j = 4$ (in Figure 5). Figure 6 shows an example of a regular/average case ($n = 15$, $C = 40$, $nC = 600$, e-nodes = 223, and original e-nodes = 103). The time complexity of the average case arises in most datasets (≈(best + worst)/2 < $nC/2$). Since a weight w_j of object j can be $1 \leq w_j < C$, the average w_j is approximately C/n. For $w_j \geq C/n$, usually no e-node is added (because of the condition $node.d + w_j > C$).

For m knapsacks, the time complexity of our multi-DPT-list$_{TSR}$ algorithm for solving the 0/1-mKP is O(m[n^2, nC]) with any effective knapsack order (including top nine orders) and O(m^2[n^2, nC]) for the exact-fit (best) order in $m(m + 1)/2$ steps; see Section 5.3.

For space complexity, Figure 17 illustrates our three steps of space reduction: 1. e-nodes (<nC), 2. original e-nodes, and 3. tight bound of e-nodes; see Figure 5 ($n = 5$, $C = 18$) and Figure 7b ($n = 15$, $C = 40$). In our experiment, Section 6.1, displays the observed results ($n \leq 3000$), where after F-reduction the original e-nodes are a function of cn^2 (<n^3, c = a constant) and hence after B-reduction the original e-nodes are less than n^2.

5.3. High (Optimal) Precision (as m! Orders) of the Exact-Fit (Best) Knapsack Order

In our novel research track (Figure 8), we study by starting with the exact DP for the optimal solution in one knapsack to m knapsacks. In Section 3, we propose the DPT-List$_{TSR}$ algorithm to find the optimal solution in O([n^2, nC]) for each knapsack. In Section 4, we propose the efficient order reduction for m knapsacks (over m! orders), which are 1. the top nine effective orders in our multi-DPT-List$_{TSR}$ algorithm in O(m[n^2, nC]) for the regular datasets, and 2. the exact-fit (best) order in our multi-DPT-List$_{TSR}$ algorithm in O(m^2[n^2, nC]) for the irregular datasets, where the DPT-List$_{TSR}$ algorithm is applied in the internal and external effects (in each knapsack and among m knapsacks).

For the regular datasets, an effective order is increasing C_i; see Figure 18 (selecting k candidates (objects), $0 \leq j \leq n-1$, for m positions (knapsacks with capacity C_i, i = 1, 2, ..., m) in a company/organization, the profit p_j (knowledge), and the weight w_j (negative attitude/greedy weight)). For large m, the (fast) top nine effective orders (in Section 4.1.1) are presented and later improved by the LS of m permutations, where the partial LS (first nine permutations of each order) is concerned in O(m[n^2, nC]), see Table 1 (the proposed reduction orders and all possible m! orders (for $5 \leq m \leq 100$)).

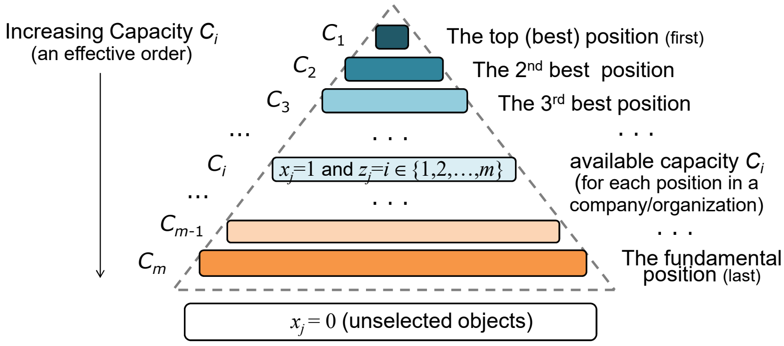


Figure 18. An example of increasing C_i (an effective knapsack order) for the 0/1-mKP.

Table 1. All possible (m!) knapsack orders and the proposed effective orders for the 0/1-mKP.

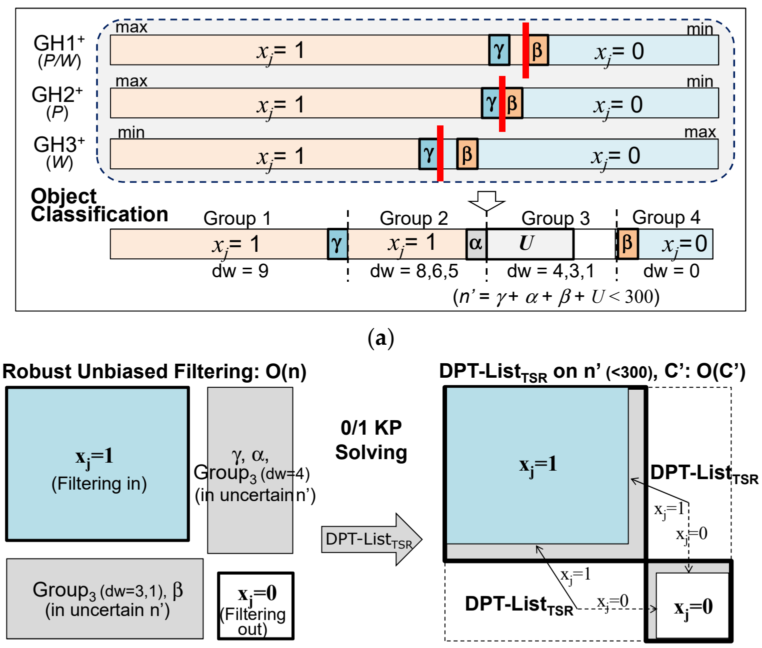
m Knapsacks	All Orders (m!)	Exact-Fit/Best (m(m + 1)/2)	Top (9) Orders	Partial LS Min (9 m, 9 × 9)	Full LS (9 m)
5	120	15	9	45	45
6	720	21	9	54	54
7	5040	28	9	63	63
8	40,320	36	9	72	72
9	326,880	45	9	81	81
10	3,268,800	55	9	81	90
20	20!	210	9	81	180
50	50!	1275	9	81	450
100	100!	5050	9	81	900

For the irregular datasets, we can use m! orders to achieve at least 99% optimal solutions but in exponential time O(m! [n^2, nC]). To reduce the time complexity and retain 99% optimal precision, the exact DPT-List$_{TSR}$ algorithm is applied for not only the internal effect (for the optimal solution in each knapsack) but also the external effect (for the best order among m knapsacks). For the external effect, the DPT-List$_{TSR}$ algorithm is used to find all exact-fit knapsacks before selecting the best knapsack and repeating the same process for the remaining objects and knapsacks. The exact-fit (best) order (Algorithm 10) is determined in $m(m + 1)/2$ steps in O(m^2[n^2, nC]). In particular, the best knapsack K_i is selected by the best exact-fit$_i$ = $min(dFit_i)$; $dFit_i = C_i - TW_i$, where the exact TP_i and TW_i are computed by the DPT-List$_{TSR}$ algorithm in each of the available knapsacks; see Figure 11 (n = 33, m = 5). Moreover, if there are equal $min(dFit_i)$s in more than one K_i (in the critical decision) in step 2 of Algorithm 10, then three proper policies are used to find the best of three best solutions. See the confirmed results (99% optimal solutions) in Section 6.3.

5.4. High (Optimal) Precision of the Robust Unbiased Filtering

The exact DP + unbiased filtering [1] can solve the 0/1-KP in O($n + C'$), $C' << C$ with at least 95% optimal precision. Thus, we can adopt the process of unbiased filtering for the

0/1-mKP. Initially, all objects are classified (into four groups) by dynamic weighting (dw), integrated from three effective ranks (P/W, P, W); see Figure 19a, where two parameters (α, β)) were defined to handle special objects (called outliers) in unbiased filtering [1] for the 0/1-KP with 95% optimal solutions. In this study, to achieve 99% optimal solutions for the 0/1-mKP, the (γ, α, β) parameters are introduced by studying all datasets (i.e., most datasets are regular (≈90%) and irregular datasets are ≈10%). In our robust unbiased filtering, the (γ, α, β, U) parameters are determined in Equations (10)–(14), where the critical objects are in low rank in Group 1 (γ) and Group 2 (α) and in top rank in Group 4 (β), and most uncertain objects (U) are in Group 3. Figure 19a shows that some objects (around the critical points in the three ranks (P/W, P, or W)) are the critical objects (γ, α, β). All uncertain/critical objects (n'< 300) can be solved by the DPT-List$_{TSR}$ algorithm in O(n + C'). Figure 19b shows the idea of 99% optimal precision (due to our robust unbiased filtering) in each knapsack of the 0/1-mKP; see in the observed results (in Section 6).



(**a**)

(**b**)

Figure 19. High (optimal) precision of robust unbiased filtering for each K_i of 0/1-mKP: (**a**) object classification and (**b**) robust unbiased filtering and DPT-List$_{TSR}$ for remaining n' < 300.

6. Experimental Results

Experiments were conducted to evaluate the DPT-List$_{TSR}$ algorithm and robust unbiased filtering for the 0/1-KP (Section 6.1) to ensure at least 99% optimal precision (in each knapsack) before applying to the 0/1-mKP (Sections 6.2 and 6.3) by the multi-DPT-List$_{TSR}$ algorithm (the best knapsack-order for m knapsacks) and robust unbiased filtering.

6.1. Results of the DPT-List$_{TSR}$ (One Knapsack) and Robust Unbiased Filtering

For solving the 0/1-KP, we generated a variety of random datasets (dynamic seeds) with a number of uniform distributions to obtain the profits and weights of n objects ($n \leq$ 10,000). The experiment was conducted to evaluate the performance of robust unbiased filtering. In the experimental results, our DPT-List$_{TSR}$ + robust unbiased filtering could find the exact solutions in each of the datasets ($n \leq$ 10,000), while there were 223 (of 10,000) datasets for which the recent TS$_{Reduction}$ + unbiased filtering [1] could not find the optimal solutions. Table 2 shows the empirical results of the first 23 of 223 special datasets (or irregular datasets), n = 12, 14, 21, 26, 39, . . . , 385 (observed on n = 5, 6, 7, . . . , 9999, 10,000).

Table 2. Experimental results of irregular datasets (23 of 223), observed from n = 5, 6, 7, . . . , 10,000.

		Total Weight (*soltw*)			Total Profit (*soltp*)		
n	C	DPT-List (Opt.)	DPT + rFilter	TSR + uFilter [1]	DPT-List (Opt.)	DPT + rFilter	TSR + uFilter [1]
12	96	**96**	96	92	**282**	282	280
14	112	**111**	111	109	**365**	365	362
21	168	**168**	168	167	**500**	500	498
26	208	**208**	208	203	**637**	637	636
39	312	**312**	312	312	**868**	868	866
45	360	**360**	360	360	**1177**	1177	1172
73	510	**510**	510	509	**1640**	1640	1639
80	559	**559**	559	558	**1712**	1712	1711
81	566	**566**	566	566	**1888**	1888	1886
143	1000	**1000**	1000	1000	**3084**	3084	3082
147	1028	**1028**	1028	1028	**3250**	3250	3239
155	1084	**1084**	1084	1083	**3440**	3440	3437
166	1161	**1161**	1161	1161	**3617**	3617	3616
182	1273	**1273**	1273	1273	**3967**	3967	3966
197	1378	**1378**	1378	1378	**4534**	4534	4533
199	1392	**1392**	1392	1391	**4561**	4561	4560
247	1481	**1481**	1481	1480	**4822**	4822	4821
276	1655	**1655**	1655	1655	**5446**	5446	5445
286	1715	**1715**	1715	1715	**5889**	5889	5888
316	1895	**1895**	1895	1895	**6266**	6266	6257
329	1973	**1973**	1973	1973	**6484**	6484	6469
360	2159	**2159**	2159	2159	**6985**	6985	6984
385	2309	**2309**	2309	2309	**7710**	7710	7709

Table 3 presents the optimal performance of our DPT-List$_{TSR}$ + robust unbiased filtering for at least 99% optimal solutions (on $n \leq 10{,}000$) compared to our previous work [1]. In this experiment, there exist irregular datasets ≈10% (from all 10,000 random datasets), where unbiased filtering (in TS$_{Reduction}$) [1] could handle ≈5% and our robust unbiased filtering (in DPT-List$_{TSR}$) could handle ≈9.9%. Table 4 displays our space reduction, observed on $n \leq 3000$ (with runtime < 1 min per n). Our F-reduction can save space 69–92%, and our B-reduction can save space 84–93%.

Table 3. Optimal precision of the DPT-List$_{TSR}$ + robust filtering ($n \leq 10{,}000$).

	DPT-List + Robust Filtering			TSR + Unbiased Filtering [1]		
n: Datasets	not Opt.	Optimal	Precision	not Opt.	Optimal	Precision
$5 \leq n \leq 100$	0	95	99.9%	9	86	90.0%
$5 \leq n \leq 200$	0	195	99.9%	16	179	92.0%
$5 \leq n \leq 500$	0	495	99.9%	23	472	95.4%
$5 \leq n \leq 2000$	0	1995	99.9%	28	1967	98.6%
$5 \leq n \leq 5000$	0	4995	99.9%	109	4886	97.8%
$5 \leq n \leq 10{,}000$	0	9995	99.9%	223	9772	97.8%

Note: 99% optimal solutions refer to "For 100 observed datasets, we could find 99 optimal solutions".

Table 4. Performance (percentage) of space reduction by the DPT-List$_{TSR}$ ($n \leq 3000$).

n	$n \times C$ (Full Space)	e-Nodes (1. Initial Reduction)		Original e-Nodes (2. F-reduction)		Tight-Bound e-Nodes (3. B-Reduction)	
5	90	13	86%	7	92%	6	93%
15	600	223	63%	103	83%	50	92%
50	17,450	6559	62%	2465	86%	1917	89%
100	69,900	35,852	49%	15,263	78%	10,849	84%
200	239,800	150,518	37%	66,883	72%	35,932	85%

Table 4. *Cont.*

n	$n \times C$ (Full Space)	e-Nodes (1. Initial Reduction)		Original e-Nodes (2. F-reduction)		Tight-Bound e-Nodes (3. B-Reduction)	
500	1,250,000	915,303	27%	374,729	70%	173,786	86%
1000	5,000,000	3,832,827	23%	1,566,414	69%	692,691	86%
1500	11,250,000	8,669,932	23%	3,364,525	70%	1,581,133	86%
2000	11,998,000	10,322,643	14%	3,293,641	73%	1,115,040	91%
2500	18,747,500	16,071,768	14%	5,338,435	72%	1,752,052	91%
3000	26,997,000	23,225,460	14%	7,497,627	72%	2,620,552	90%

6.2. Results of the Multi-DPT-List$_{TSR}$ (m Knapsacks) and Robust Unbiased Filtering

For solving the 0/1-mKP, the optimal performance of our multi-DPT-List$_{\mathrm{TSR}}$ algorithm with the proper (knapsack) orders (i.e., top nine orders, Latin squares, the exact-fit (best) order) was evaluated by comparison to the optimal solutions. In practice, the fast response time of our multi-DPT-List$_{\mathrm{TSR}}$ with robust unbiased filtering was observed, while retaining the high performance. In this experiment, a number of random datasets were generated for n ($\leq$10,000) objects and m ($\leq$100) knapsacks with a variation of capacities (i.e., $C_i \pm 10$, $C_i \pm 15$, $C_i \pm 20$, etc.). In addition, the benchmark datasets [34] were observed and the empirical results were compared to the optimal solutions.

In performance (total profit) evaluation, we focus on the investigation of 1. the exact-fit best (knapsack) order in our multi-DPT-List algorithm (for 99% optimal solutions in theory) and 2. the robust unbiased filtering (in polynomial time) to confirm 99% optimal solutions. We implemented our multi-DPT-List (the exact-fit best order) and the fast multi-DPT-List + robust filtering compared to the optimal solutions. For the practical polynomial-time evaluation, the fast response time of our multi-DPT-List + robust filtering was compared to the quick multi-GH$^+$ (a well-known heuristic algorithm).

For the performance comparison, the (known) optimal solutions of the 0/1-mKP (in Column 2 of Tables 5–10) can be computed by using a large knapsack ($C^s = \sum_{i=1}^{m} C_i$) by the exact DP or our DPT-List in the (regular and irregular) datasets.

The implemented programs of three main approaches (in this experiment) are

Exact	1.1 Multi-DPT-List (exact-fit best order)	$O(m^2[n^2,nC])$
Exact + Filtering	2.1 Fast multi-DPT-List + filtering (exact-fit best order)	$O(m^2n)$
	2.2 Fast multi-DPT-List + filtering (top 9 orders + partial LSs)	$O(mn)$
	2.3 Fast multi-DPT-List + filtering (top 9 orders)	$O(mn)$
Heuristic	3.1 Quick multi-GH (P/W rank) (top 9 orders)	$O(mn)$
	3.2 Improved multi-GH$^+$ (P/W rank) (top 9 orders)	$O(mn)$
	3.3 Improved multi-GH$^+$ (P/W rank) (top 9 + full LS orders)	$O(m^2n)$

Table 5. Results (total profits) of datasets with capacities $C \pm 10$ (m = 2).

m = 2	Optimal	mDPT-L $m!$ O($m[n^2, nC]$)	mDPT-L + Filter $m!$ O(mn)	mGH $m!$ O(mn)	mGH$^+$ $m!$ O(mn)
n		$m!$ = 2	$m!$ = 2	$m!$ = 2	$m!$ = 2
15	**315**	315	315	308	310
20	**420**	420	420	393	420
30	**800**	800	800	790	796
40	**1050**	1050	1050	1022	1047
50	**1019**	1019	1019	1006	1013
100	**2359**	2359	2359	2313	2357
200	**3878**	3878	3878	3860	3870
300	**6202**	6202	6202	6171	6196
400	**7686**	7686	7686	7654	7683
500	**9074**	9074	9074	9045	9072
1000	**18,038**	18,038	18,038	18,002	18,031

Table 6. Results (total profits) of datasets with capacities $C \pm 15$ (m = 3).

m = 3	Optimal	mDPT-L $m!$ O($m[n^2, nC]$)	mDPT-L + Filter $m!$ O(mn)	mGH $m!$ O(mn)	mGH⁺ $m!$ O(mn)
n		$m!$ = 6	$m!$ = 6	$m!$ = 6	$m!$ = 6
15	**327**	327	327	320	322
20	**466**	466	466	454	466
30	**840**	840	840	829	839
40	**1103**	1103	1103	1090	1099
50	**1067**	1067	1067	1031	1062
100	**2427**	2427	2427	2390	2426
200	**3949**	3949	3949	3908	3943
500	**9150**	9150	9150	9133	9148
1000	**18,112**	18,112	18,112	18,088	18,110
2000	**25,547**	25,547	25,547	25,524	25,544

Table 7. Results (total profits) of datasets with capacities $C \pm 20$ (m = 4).

m = 4	Optimal	mDPT-L $m!$ O($m[n^2, nC]$)	mDPT-L + Filter $m!$ O(mn)	mGH $m!$ O(mn)	mGH⁺ $m!$ O(mn)
n		$m!$ = 24	$m!$ = 24	$m!$ = 24	$m!$ = 24
20	**495**	495	495	490	490
30	**884**	884	884	884	884
40	**1200**	1200	1200	1187	1192
50	**1137**	1137	1136	1121	1136
60	**1504**	1504	1504	1472	1499
100	**2548**	2548	2548	2534	2546
200	**4098**	4098	4098	4071	4088
500	**9318**	9318	9318	9297	9314
1000	**18,284**	18,284	18,284	18,249	18,280
2000	**25,760**	25,760	25,760	25,735	25,752
3000	**38,941**	38,941	38,941	38,899	38,930

Table 8. Results (total profits) of datasets ($C \pm 20$), n = 1000–5000 (m = 6–50).

m	Optimal	mDPT-L O($m^2[n^2, nC]$)	mDPT-L + LS Filter O(m^2n)	mDPT-L + Filter O(mn)	mGH⁺ O(mn)	mGH⁺ + LS O(m^2n)
n = 1000		Best	9 m	9	9	9 m
6	**18,541**	18,541	18,541	18,541	18,535	18,535
7	**18,703**	18,703	18,703	18,703	18,684	18,693
8	**18,889**	18,889	18,889	18,889	18,879	18,884
9	**19,079**	19,079	19,079	19,079	19,068	19,071
n = 2000		Best	9 × 9	9	9	9 m
12	**27,769**	27,769	27,769	27,769	27,742	27,753
13	**28,154**	28,154	28,154	28,154	28,127	28,127
14	**28,547**	28,547	28,547	28,547	28,498	28,517
15	**28,948**	28,948	28,948	28,948	28,900	28,935
n = 5000		Best	9 × 9	9	9	9 m
20	**54,736**	54,736	54,736	54,736	54,680	54,695
30	**62,496**	62,496	62,496	62,496	62,410	62,450
40	**72,417**	72,417	72,417	72,417	72,323	72,331
50	**84,051**	84,051	84,051	84,051	83,943	83,963

Table 9. Results (total profits) of datasets ($C \pm 10$, 20), n = 9000 (m = 40–90).

m	Optimal	mDPT-L $O(m^2[n^2, nC])$	mDPT-L + LS Filter $O(m^2n)$	mDPT-L + Filter $O(mn)$	mGH⁺ $O(mn)$	mGH⁺ + LS $O(m^2n)$
$C_i \pm 10$		Best	9 × 9	9	9	9 m
40	**53,669**	53,669	53,669	53,669	53,494	53,520
50	**54,803**	54,803	54,803	54,803	54,538	54,655
60	**58,614**	58,614	58,614	58,614	58,425	58,450
70	**64,018**	64,018	64,018	64,018	63,689	63,784
$C_i \pm 20$		Best	9 × 9	9	9	9 m
40	**85,380**	85,380	85,380	85,380	85,203	85,286
50	**100,859**	100,859	100,859	100,859	100,683	100,724
60	**118,729**	118,729	118,729	118,729	118,525	118,616
70	**138,195**	138,195	138,195	138,195	137,963	138,038
80	**158,983**	158,983	158,983	158,983	158,722	158,857
90	**180,054**	180,054	180,054	180,054	179,873	179,981

Table 10. Results (total profits) of irregular datasets (m = 3–7, $n \leq$ 10,000).

$n \leq$ 10,000	Optimal	mDPT-L $O(m^2[n^2, nC])$	mDPT-L + LS Filter $O(m^2n)$	mDPT-L + Filter $O(mn)$	mGH⁺ $O(mn)$	mGH⁺ + LS $O(m^2n)$
m:n		m!	m!	m!	m!	m!
3:51	**1318**	1318	1317	1317	1315	1315
3:73	**1727**	1727	1725	1725	1725	1725
4:33	**752**	752	747	747	747	747
4:49	**1264**	1264	1263	1263	1254	1254
4:50	**1137**	1137	1136	1136	1136	1136
4:65	**1565**	1565	1563	1563	1563	1563
m:n		Best	9 m	9	9	9 m
5:89	**2366**	2366	2365	2365	2359	2359
7:77	**1834**	1834	1833	1833	1829	1830
7:138	**3263**	3263	3262	3262	3256	3256
7:148	**3780**	3780	3799	3799	3773	3777

First, we evaluated the performance of our mDPT-List and fast mDPT-List + filtering with m! orders (for small m = 2, 3, 4), compared to the optimal solutions. For $m \leq 4$, our approach can find the optimal solutions in most datasets; see Columns 3 and 4 in Tables 5–7. For $m > 4$, we investigated the effect of robust filtering plus the top nine effective orders and partial Latin squares (Columns 4 and 5 in Tables 8 and 9). For the regular datasets ($n \leq$ 10,000, $m \leq 100$), our mDPT-List + filtering (top nine orders) yielded 99% optimal solutions.

Second, we aimed to compare among the fast polynomial-time algorithms ($O(mn) - O(m^2n)$) by observing the effect of the top nine effective orders; see Columns 4–7 in Tables 8 and 9. For $m \leq 100$ and $n \leq$ 10,000, the results (total profits) of our mDPT-List + filtering (in Column 5) were compared to those of the quick mGH⁺ (P/W rank) in Columns 6–7 (response time < 1 s). For the regular datasets, our fast mDPT-List + filtering (in Column 5) yielded most optimal solutions, while the results of the quick mGH⁺ (in Column 6) and its improvement with LS of 9 m orders (in Column 7) were far from the optimal solutions, especially when using many knapsacks ($m > 10$). Note: GH (P/W rank) is frequently used in many meta-heuristic algorithms (i.e., GA, swarm, etc.) for the good initial solutions to solve the 0/1-KP and GH⁺ is used in unbiased filtering [1] (p. 199) and in robust unbiased filtering (in this study). In the comparison, we use the improved mGH⁺ with the Latin squares of top nine orders (for 9 m orders/iterations to emulate the evolution process of GA/swarm optimization). For most datasets, the mGH⁺ (9m orders) could not find the optimal solutions in each knapsack since it included uncertain object(s) in the solution.

However, it is not the problem in our robust unbiased filtering since all uncertain objects ($n' < 300$) were considered by the exact DPT-List with 99% optimal precision.

In our initial observation and analysis, for $m = 2$ ($n \leq 10{,}000$), the mDPT-List + filtering ($m!$ orders) in Table 5 yielded 100% optimal solutions. For $m = 3, 4$ ($n \leq 10{,}000$), our approach ($m!$ orders) in Tables 6 and 7 yielded 99.9% optimal solutions. Next, we found that (for the irregular datasets) the top nine orders were not sufficient to find the optimal solutions ≥99%, especially $m \geq 5$. Then, we investigated the effect of the LS of the top nine orders (see Column 4 in Tables 8–10). Moreover, Tables 10 and 11 report the irregular datasets found during the execution of each dataset ($n \leq 10{,}000$), where any dataset is called "irregular" when the top nine orders could not find the optimal solution. For $m \geq 5$, we performed an intensive study and experiment to observe each of $n \leq 10{,}000$ ($m \leq 100$) and found that a number of irregular datasets increased when m increased (see Column 6 in Table 11). Hence, the exact-fit (best) knapsack order is applied to solve this problem.

Table 11. Observed frequency of nonoptimal solutions (in $n \leq 10{,}000$ per m), $m = 5, 6, 7, \ldots, 53, 54$.

$n \leq$ 10,000	mDPT-L: O($m^2[n^2, nC]$)		mDPT-L + Filtering: O(m^2n)		
m	Best	9 m (LS)	Best	9 m (LS)	Top 9
5	0	0	0	1	2
6	0	0	0	1	3
7	0	1	0	3	6
8–14	0	0	0	2	1.6 (ave. per m)
15–19	0	0	0	0	1.6 (ave. per m)
20–24	0	0	0	0	2.6 (ave. per m)
25–29	0	0	0	0	4.8 (ave. per m)
30–34	0	0	0	0	5.4 (ave. per m)
35–39	0	0	0	0	13.8 (ave. per m)
40–44	0	0	0	0	31.4 (ave. per m)
45–49	0	0	0	0	34.8 (ave. per m)
50–54	0	0	0	0	69.2 (ave. per m)

Note: When observing the irregular datasets, using top 9 orders (Column 6) in our mDPT-List + filtering could not find the optimal solutions in approximate 69 datasets (in average) of $n \leq 10{,}000$, $m = 54$ in the (random) regular and irregular datasets, while using the best order (Column 4) could find all optimal solutions.

After performing the intensive study and comparison (on large $n \leq 10{,}000$), we found that for the irregular datasets, our mDPT-List + robust unbiased filtering (the exact-fit (best) order in O(m^2n)) could find at least 99% optimal solutions as those of the original mDPT-List (O($m^2[n^2,nC]$)); see a report of observed frequency of nonoptimal solutions (0%) of our approach in Table 11 (Column 4).

Finally, we performed an extra experiment to evaluate the performance of our mDPT-List on the benchmark datasets [34] available at http://or.dei.unibo.it/library (accessed on 13 June 2020). Tables 12 and 13 show the empirical results of our mDPT-List (the best knapsack order in O($m^2[n^2, nC]$)) and our fast mDPT-List + robust filtering (the best knapsack order in O(m^2n)), compared to the regular mDP (O(m^2nC)) and the optimal solutions.

For (n:m = 100:10) 10 datasets [34], the results (in Table 12) showed that our mDPT-List (without/with filtering (the best order, LS orders, top nine orders)) could find the optimal solutions (Columns 4–7), while the results (Column 8) of the quick mGH$^+$ (9 m orders in 9 m iterations) were not optimal.

Table 12. Results (total profits) of 10 benchmark datasets (n = 100, m = 10).

Research Approach		**Exact**		**Exact + Filtering**			**Heuristic**
		mDP	**mDPT-L**	**mDPT-L + Filter: O(m^2n)**			**mGH⁺**
n:m	**Optimal**	Best	Best	Best	9 *m*	9	9 *m*
100:10-1	**26,797**	26,797	26,797	26,797	26,797	26,797	26,763
100:10-2	**24,116**	24,116	24,116	24,116	24,116	24,116	24,093
100:10-3	**25,828**	25,828	25,828	25,828	25,828	25,828	25,812
100:10-4	**24,004**	24,004	24,004	24,004	24,004	24,004	23,977
100:10-5	**23,958**	23,958	23,958	23,958	23,958	23,958	23,933
100:10-6	**24,650**	24,650	24,650	24,650	24,650	24,650	24,614
100:10-7	**23,911**	23,911	23,911	23,911	23,911	23,911	23,886
100:10-8	**26,612**	26,612	26,612	26,612	26,612	26,612	26,579
100:10-9	**24,588**	24,588	24,588	24,588	24,588	24,588	24,565
100:10-10	**24,617**	24,617	24,617	24,617	24,617	24,617	24,591

Table 13. Results (total profits) of 20 benchmark datasets (200 ≤ n ≤ 500, 20 ≤ m ≤ 50).

Research Approach		**Exact**			**Exact + Filtering**		**Heuristic**
	Over Packing *		**mDPT-L O($m^2[n^2, nC]$)**		**mDPT-L + Filter O(m^2n)**		**mGH⁺ O(m^2n)**
n:m	**Optimal**	**Best of this study**	**Best +extra**	**Best**	**Best**	**9 *m***	**9 *m***
200:20-1	80,260 *	**80,205**	**80,205**	80,163	80,196	80,121	79,606
200:20-2	80,171 *	**80,122**	**80,122**	80,122	80,121	80,069	79,488
200:20-3	79,101 *	**79,083**	**79,083**	79,061	**79,083**	79,041	78,561
200:20-4	76,264 *	**76,208**	**76,208**	76,174	76,174	76,149	75,823
200:20-5	**79,619**	**79,619**	**79,619**	79,581	79,581	79,515	78,886
200:20-6	76,749 *	**76,711**	**76,711**	**76,711**	**76,711**	76,612	76,203
200:20-7	76,543 *	**76,474**	**76,474**	76,429	**76,474**	76,402	75,959
300:30-1	121,806 *	**121,756**	121,742	121,742	**121,756**	121,654	120,842
300:30-2	119,877 *	**119,828**	**119,828**	119,795	**119,828**	119,743	118,938
300:30-3	119,806 *	**119,762**	**119,762**	119,756	119,749	119,684	118,937
300:30-4	115,567 *	**115,556**	115,529	115,516	**115,556**	115,434	114,767
300:30-5	117,204 *	**117,175**	**117,175**	117,160	117,168	117,065	116,350
300:30-6	118,516 *	**118,493**	**118,493**	**118,493**	118,450	118,386	117,737
300:30-7	115,793 *	**115,752**	**115,752**	115,706	115,693	115,641	115,093
300:30-8	123,664 *	**123,624**	**123,624**	123,620	123,620	123,552	122,570
500:50-1	205,672 *	**205,645**	**205,645**	**205,645**	**205,645**	205,488	204,132
500:50-2	199,868 *	**199,781**	199,775	199,775	**199,781**	199,681	198,462
500:50-3	202,321 *	**202,286**	**202,286**	202,277	202,277	202,164	201,102
500:50-4	136,669 *	**136,657**	**136,657**	136,653	136,652	136,595	135,409
500:50-5	135,806 *	**135,796**	135,795	135,795	**135,796**	135,736	134,793

Note: The symbol * (in Column 2) means that the (extra) solution may be overpacking.

For (n:m = 200:20, 300:30, 500:50) 20 datasets [34], most optimal solutions of these critical datasets were unknown (see Table 13) since the DP-packing in one large knapsack ($C^s = \sum_{i=1}^{m} C_i$) may be overpacking. Figure 20a shows an example of overpacking, when some objects in the critical datasets (such as some valuable objects j (high p_j/w_j) but large w_j) cannot be packed in any knapsack i with capacity C_i, except in the extra space of one knapsack of large capacity C^s. In these critical datasets, our mDPT-List with the best order (in Columns 5 and 6) yielded good results, which were close to or equal to the optimal solutions and outperformed those of LSs (9m) of top nine orders (in Columns 7–8).

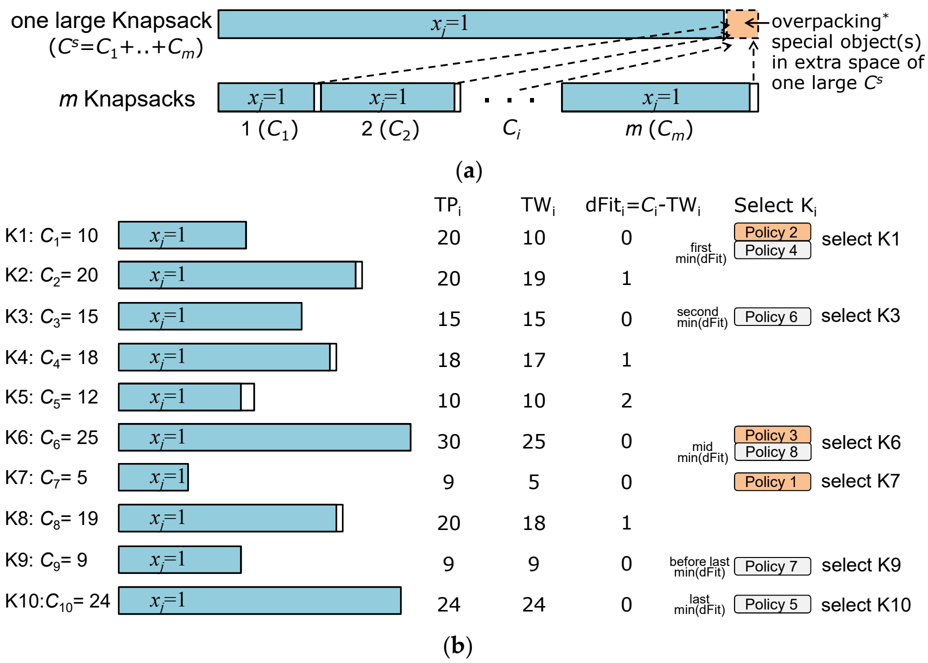


Figure 20. (**a**) Overpacking (in the extra space of one knapsack with large total-capacity C^s) in critical datasets and (**b**) an example of eight effective policies (to handle the critical decisions).

In our contribution, we focus on large n. The fast mDPT-List + filtering (top nine orders) in O(mn) is good for the regular datasets with 99% optimal solutions (Tables 5–9 and 12). For the irregular datasets (Table 11), our fast mDPT-List (the best order) + filtering in O(m^2n) can find 99% optimal solutions similar to our original mDPT-List (the best order). Thus, for the regular and irregular datasets, our fast mDPT-List + filtering (the best order) is sufficient to achieve 99% optimal solutions in polynomial time O(m^2n). Moreover, for the critical/special benchmark datasets, we have intensively studied by the experiment (in Section 6.3) to improve the solutions (Column 4 in Table 13).

6.3. Extra Experiment and Additional Improvement on Critical Datasets

To improve the results of the critical datasets (benchmark datasets [34]), we have to find all possible critical decisions, such as 1. nonunique Xs (in X-tracking (see an example in Figure 10) in Section 4.1.1) and 2. equal $min(dFits)$ in more than one knapsacks (in the exact-fit (best) order (see an example in Figure 11) in Section 4.1.2) and provide the right policies to handle them. Clearly, if there is only one $min(dFit_i)$, $dFit_i = C_i – TW_i$, we can select the best knapsack K_i directly for the best order. By the DP-packing, there may be many equal $min(dFit_i)$s in $K_{i'}–K_{i''}$ but only one K_i is selected (at a time), and this decision may cause the local optimal problem. To handle this problem, the top three effective policies are introduced in Algorithm 10, and the best of three best results is our final solution. However, to achieve the better results of these critical/special datasets, we add the other effective policies 4–8 in Algorithm 10 (step 2) to cover the other critical decisions; see an example in Figure 20b, i.e., select K_i with $min(dFit_i)$ at the first, last, second, before last, and mid policies (in policies 4–8). Figure 20b shows the detail of selecting the best K_i (m = 10 knapsacks, C_i = (10, 20, 15, 18, 12, 25, 5, 19, 9, 24)) with eight critical decisions (i.e., assume there are six $min(dFit_i)$s = 0 in K_i, i = 1, 3, 6, 7, 9, 10) for selecting the best K_i (in the best order) with one policy for one result ($solTP$). In this experiment, the improved results (max ($solTP_{i=1–8}$)) in Column 4 (Table 13) were stable under these eight policies. In each critical dataset, the results (Column 4) were improved due to the exact DPT-List packing plus the proper critical handling (by our eight policies for the best knapsack order).

In the regular comparison of our mDPT-List + robust unbiased filtering (the best knapsack order) on 20 critical datasets (in Table 13), our robust unbiased filtering (using top three policies) yielded (9 of 20) best results (Column 6), which outperformed the results

(Column 7) of using LSs of top nine orders (9 m). In the superior improvement of our eight effective policies (in the extra experiment), the extra mDPT-List yielded (16 of 20) best results (Column 4), while the other 4 of 20 best results (Column 3) were fulfilled by robust filtering due to the (unbiased) preselecting and the less problem of nonunique Xs in the X-tracking by DPT-List (on small $n' < 300$) in each knapsack. Obviously, for the unique X, our mDPT-List (with/without filtering) yields the same result.

In addition, the response times of three mDP algorithms (basic mDP, mDPT-List, and mDPT-List + filtering) were compared in this experiment (under the same 99% optimal precision). In theory, three different time complexities of these mDP algorithms are 1. O($m!nC$) in the basic mDP ($m!$ orders), 2. O($m^2[n^2, nC]$) in the mDPT-List (the best order), and 3. O(m^2n) in the fast mDPT-List + filtering (the best order). Due to our efficient unbiased filtering, the response time of our fast mDPT-List + filtering for n = 20,000 and m = 20 was less than one second, that of the best mDPT-List for n = 20,000 was 10 min, that of the basic mDP for n = 20,000 was more than one hour and so on for other large n.

Next to simplify the comparison and discussion (for the 0/1-mKP solving with the critical datasets), we employ the triple-right rule (right man, right place, and right time). Figure 21 displays the improvement of our multi-DPT-List$_{TSR}$ algorithm (our novel research track in Figure 8) to reach 99% optimal solutions in efficient time.

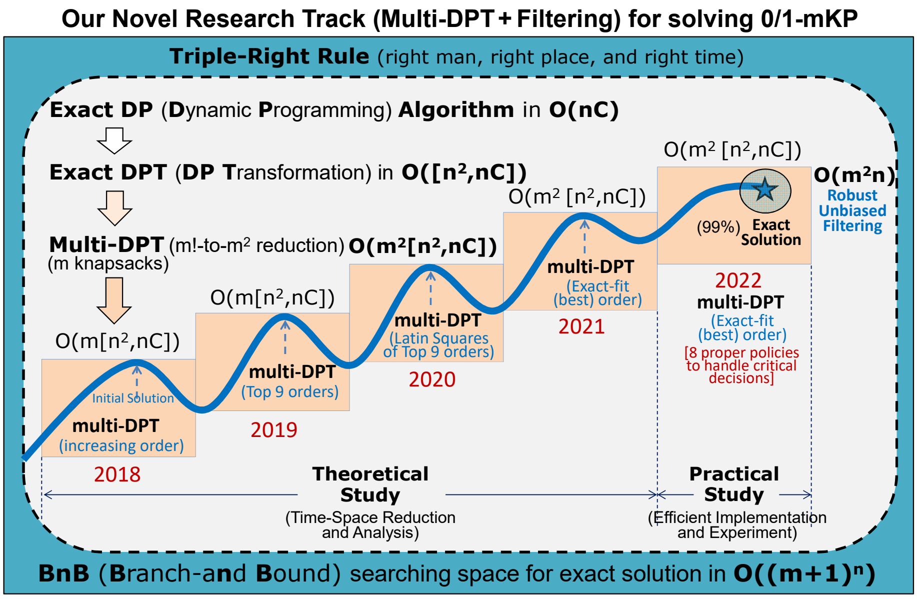


Figure 21. The improvement of our multi-DPT$_{TSR}$ algorithm for solving the 0/1-mKP (m knapsacks) when dealing with the critical datasets.

In theory, the exact BnB algorithm can find the exact solution of the 0/1-mKP but in O($(m+1)^n$) with the right man and right place but not the right time. In practice, the multi-GH$^+$ (with 9 m LS-orders) can find the good solutions in polynomial time but those solutions may not be optimal because it only confirms the right time rule. In this study (Figure 21), we study and apply the exact DPT (in theory) and the efficient unbiased filtering (in practice) to achieve triple-right packing (right man (object), right place (knapsack), and right time (O(m^2n))). Our contribution is the $m!$-to-m^2 reduction; see the highlight space of our improvement in Figure 21. This tight-bound reduction starts with the exact DPT for one knapsack (selecting the right object), uses the exact-fit (best) order for m knapsacks (putting the right object in the right knapsack), and ends with robust unbiased filtering (putting the right object in the right knapsack at the right time).

The comparison of our multi-DPT-List + robust filtering and the recent HyMKP [34] is demonstrated in Table 14. In practice (with large n ($\leq$10,000 in this experiment)), our multi-DPT-List + robust filtering yielded 99% optimal results in O(m^2n); see results in Tables 5–12, while the results of quick multi-GH⁺ in O(m^2n) were not optimal. For the HyMKP study [34], there is no available result for $n > 500$ since for large n the partial BnB (MULKNAP program) in the HyMKP (Algorithm 5) may not find the optimal solution in τ secs. Then, the reflect multi-graph MKP (with increasing n-weights (w_j) in Algorithm 6 (O(mnC)) can provide the comparable results to our multi-DPT-List (with top nine orders) in the regular datasets, according to the looping on weights and C, similar to the DP (Algorithm 1) for each of m knapsacks. For the irregular datasets, v-rounds of decompositions of HyMKP are used to improve the initial solution. However, (for large n) the process of Algorithm 6 may take a long time to reach the 99% optimal solutions due to its complexity O(mnC), while our exact-fit best (knapsack) order of multi-DPT-List + efficient filtering in O(m^2n) can find 99% optimal solutions as m! orders.

Table 14. Comparison of our multi-DPT-List + robust filtering and the mathematical HyMKP.

	For Regular and Irregular Datasets ($n \leq 10{,}000$, $m \leq 100$)	
Exact + Filtering	**Our multi-DPT-List + robust filtering (the exact-fit best order)** could find most optimal solutions (≥99%) in efficient response time (<1 s per n); see confirmed results in Tables 5–12.	**O(m^2n)**
Exact	**Mathematical HyMKP** [34] can execute in τ secs. with Algorithm 6 (reflect multi-graph MKP with decreasing n weights (w_j)) like the basic DP for each of m knapsacks. That initial solution can be improved by the knapsack decomposition in v iterations to find the optimal solution ($n \leq 500$) in τ secs. However, no available results for $n > 500$ in that study.	**O(mnC)**
Heuristic	**Multi-GH⁺ (Latin squares of top nine orders)** could find good solutions in efficient time (< 1 s) but they are not optimal (see the last column results in Tables 5–10 and Table 12). Note: LSs of top 9 orders could emulate 9 m iterations/evolutions in the GA/swarm optimization with good results (near optimal in each knapsack for small m).	**O(m^2n)**
	For critical and special benchmark datasets ($n \leq 500$) [34]	
Exact	**Partial BnB (in HyMKP)** [34]: The existing BnB (MULKNAP program) could find most optimal solutions (≥99.9%) in τ secs for $n \leq 500$.	**O($(m+1)^n$)**
Exact + Filtering	**Our multi-DPT-List + robust filtering (the best order):** For critical datasets in 0/1-mKP applications, we can adopt the MULKNAP program [34] for $n \leq 500$ in our approach to achieve 99.9% optimal solutions. For $n > 500$ we can apply our efficient multi-DPT-List + filtering in efficient time.	**O(m^2n)**

For $n \leq 500$ (in the critical datasets), the HyMKP model yielded 99.9% optimal solutions by the partial BnB (MULKNAP) program in τ secs. Thus, for $n \leq 500$ we can adopt that MULKNAP program in our approach for achieving 99.9% optimal solutions.

Finally, after achieving the good performance (99% optimal solutions) of our multi-DPT-List$_{TSR}$ + robust filtering in the efficient time O(m^2n), we can improve the time complexity to O(mn) in parallel (by using $p = m$ processors).

Moreover, to handle the critical datasets in parallel, we can achieve the global best result in parallel (by $p = m$ processors), such as Column 3 (in Table 13), by combining the local best result of the parallel multi-DPT-List$_{TSR}$ in O($m[n^2, nC]$)) in Column 4 and the local best result of the parallel multi-DPT-List$_{TSR}$ + robust filtering in O(mn) in Column 6 for the best of the best results (in Column 3) in O($m[n^2, nC]$)), which is efficient, especially in average ≈ O(mn^3) if $C = \max(C_i) \leq n^2$.

In practical 0/1-mKP applications (for large n), if the fast computing time is the most important factor (in the regular and irregular datasets), our multi-DPT-List + robust filtering in O(m^2n) or O(mn) in parallel ($p = m$) with 99% optimal solutions is good enough. However, if the high optimal performance is the most important factor (in the critical datasets and in the critical 0/1-mKP applications), the integration (of the original multi-DPT-List$_{TSR}$ and the fast multi-DPT-List$_{TSR}$ + robust filtering) provides higher precision (i.e., 99.9% optimal solutions) in efficient O($m[n^2, nC]$) in parallel ($p = m$) or O(mn^2) in the best case and O(mn^3) in average if $C \leq n^2$.

7. Conclusions

In this study, to solve the complex 0/1-mKP (m knapsacks) in polynomial time we introduced a novel research track with hybrid integration of DP transformation (for the optimal solution in each knapsack) and robust unbiased filtering (for polynomial time). First, the efficient DPT-List$_{TSR}$ algorithm was proposed to find the optimal solutions of the 0/1-KP in O($[n^2, nC]$) over O(nC) before being applied in the 0/1-mKP. Second, for solving the 0/1-mKP we proposed the multi-DPT-List$_{TSR}$ with the exact-fit (best) knapsack order (m!-to-m^2 reduction) with 99% optimal solutions in O($m^2[n^2, nC]$) over O($m![n^2, nC]$). Third (for large n, massive C), robust unbiased filtering was incorporated into our multi-DPT-List$_{TSR}$ to solve the 0/1-mKP in efficient O(m^2n) over O(mnC) of the recent HyMKP, while retaining 99% optimal solutions. The experiment was conducted to evaluate the performance of our multi-DPT-List + robust unbiased filtering (with 99% optimal solutions) on random and benchmark datasets ($n \leq$ 10,000, $m \leq 100$). Practically (for large m, n, and C), our multi-DPT-List$_{TSR}$ + robust unbiased filtering (O(m^2n)) could find 99% optimal solutions (as the original multi-DPT-List$_{TSR}$ (O($m^2[n^2, nC]$)) in polynomial time.

In our current research, we apply our multi-DPT-List$_{TSR}$ + robust unbiased filtering to solve the multi-container packing. In the future study, we will modify our unbiased filtering idea to solve another popular NP-hard problem (i.e., traveling salesman and logistic transportation, etc.) in efficient time with expected high optimal performance.

Author Contributions: Conceptualization, P.B. and J.W.; methodology, P.B.; software, P.B.; validation, P.B. and J.W.; formal analysis, J.W.; investigation, P.B.; writing—original draft preparation, P.B.; writing—review and editing, J.W.; visualization, P.B.; supervision, J.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the School of Science, King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok, Thailand, grant number [CW-011-2/2562].

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to deeply thank the School of Science, King Mongkut's Institute of Technology Ladkrabang (KMITL), Bangkok, Thailand for the Ph.D. research grant (CW-011-2/2562) to Patcharin Buayen. In addition, they would like to gratefully thank Phra-phrom Kunaporn (P.A. Pa-yut-tao) with his "Put-ta-tum" Buddhism-handbook for fulfilling their exact understanding and good decision-making. Last, they would like to gloriously honor their beloved King Bhumibol Adulyadej for the great inspiration in the optimistic study through positive thinking plus critical rethinking.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Buayen, P.; Werapun, J. Parallel time-space reduction by unbiased filtering for solving the 0/1-knapsack problem. *J. Parallel Distrib. Comput.* **2018**, *122*, 195–208. [CrossRef]
2. Aisopos, F.; Tserpes, K.; Varvarigou, T. Resource management in software as a service using the knapsack problem model. *Int. J. Prod. Econ.* **2013**, *141*, 465–477. [CrossRef]
3. Vanderster, D.C.; Dimopoulos, N.J.; Parra-Hernandez, R.; Sobie, R.J. Resource allocation on computational grids using a utility model and the knapsack problem. *Future Gener. Comput. Syst.* **2009**, *25*, 35–50. [CrossRef]

4. Bas, E. A capital budgeting problem for preventing workplace mobbing by using analytic hierarchy process and fuzzy 0-1 bidimensional knapsack model. *Expert Syst. Appl.* **2011**, *38*, 12415–12422. [CrossRef]
5. Camargo, V.C.B.; Mattiolli, L.; Toledo, F.M.B. A knapsack problem as a tool to solve the production planning problem in small foundries. *Comput. Oper. Res.* **2012**, *39*, 86–92. [CrossRef]
6. Fukunaga, A.S.; Korf, R.E. Bin completion algorithms for multicontainer packing, knapsack, and covering problems. *J. Artif. Intell. Res.* **2007**, *28*, 393–429. [CrossRef]
7. Rooderkerk, R.P.; van Herrde, H.J. Robust optimization of the 0-1 knapsack problem: Balancing risk and return in assortment optimization. *Eur. J. Oper. Res.* **2016**, *250*, 842–854. [CrossRef]
8. Ahmad, S.J.; Reddy, V.S.K.; Damodaram, A.; Krishna, P.R. Delay optimization using Knapsack algorithm for multimedia traffic over MANETs. *Expert Syst. Appl.* **2015**, *42*, 6819–6827. [CrossRef]
9. Wang, E.; Yang, Y.; Wu, J. A Knapsack-based buffer management strategy for delay-tolerant networks. *J. Parallel Distrib. Comput.* **2015**, *86*, 1–15. [CrossRef]
10. Wedashwara, W.; Mabu, S.; Obayashi, M.; Kuremoto, T. Combination of genetic network programming and knapsack problem to support record clustering on distributed databases. *Expert Syst. Appl.* **2016**, *46*, 15–23. [CrossRef]
11. Kellerer, H.; Pferschy, U.; Pisinger, D. Dynamic Programming. In *Knapsack Problem*; Springer: Berlin, Germany; New York, NY, USA, 2004; pp. 20–26.
12. Monaci, M.; Pferschy, U.; Serafini, P. Exact solution of the robust knapsack problem. *Comput. Oper. Res.* **2013**, *40*, 2625–2631. [CrossRef] [PubMed]
13. Rong, A.; Figueira, J.R.; Klamroth, K. Dynamic programming based algorithms for the discounted {0-1} knapsack problem. *Appl. Math. Comput.* **2012**, *218*, 6921–6933. [CrossRef]
14. Rong, A.; Figueira, J.R.; Pato, M.V. A two state reduction based dynamic programming algorithm for the bi-objective 0-1 knapsack problem. *Comput. Math. Appl.* **2011**, *62*, 2913–2930. [CrossRef]
15. Rong, A.; Figueira, J.R. Dynamic programming algorithms for the bi-objective integer knapsack problem. *Eur. J. Oper. Res.* **2014**, *236*, 85–99. [CrossRef]
16. Cunha, A.S.; Bahiense, L.; Lucena, A.; Souza, C.C. A new lagrangian based branch and bound algorithm for the 0-1 knapsack problem. *Electron. Notes Discret. Math.* **2010**, *36*, 623–630. [CrossRef]
17. Li, X.; Liu, T. On exponential time lower bound of knapsack under backtracking. *Theor. Comput. Sci.* **2010**, *411*, 1883–1888. [CrossRef]
18. Calvin, J.M.; Leung, J.Y.-T. Average-case analysis of a greedy algorithm for the 0/1 knapsack problem. *Oper. Res. Lett.* **2003**, *31*, 202–210. [CrossRef]
19. Truong, T.K.; Li, K.; Xu, Y. Chemical reaction optimization with greedy strategy for the 0-1 knapsack problem. *Appl. Soft Comput.* **2013**, *13*, 1774–1780. [CrossRef]
20. Balas, E.; Zemel, E. An algorithm for large zero-one knapsack problems. *Oper. Res.* **1980**, *28*, 1130–1154. [CrossRef]
21. Guastaroba, G.; Savelsbergh, M.; Speranza, M.G. Adaptive kernal search: A heuristic for solving mixed integer linear programs. *Eur. J. Oper. Res.* **2017**, *263*, 789–804. [CrossRef]
22. Lim, T.Y.; Al-Betar, M.A.; Khader, A.T. Taming the 0/1 knapsack problem with monogamous pairs genetic algorithm. *Expert Syst. Appl.* **2016**, *54*, 241–250. [CrossRef]
23. Sachdeva, C.; Goel, S. An improved approach for solving 0/1 knapsack problem in polynomial time using genetic algorithms. In Proceedings of the IEEE International Conference on Recent Advances and Innovations in Engineering, Jaipur, India, 9–11 May 2014.
24. Bansal, J.C.; Deep, K. A modified binary particle swarm optimization for knapsack problems. *Appl. Math. Comput.* **2012**, *218*, 11042–11061. [CrossRef]
25. Bhattacharjee, K.K.; Sarmah, S.P. Shuffled frog leaping algorithm and its application to 0/1 knapsack problem. *Appl. Soft Comput.* **2014**, *19*, 252–263. [CrossRef]
26. Moosavian, N. Soccer league competition algorithm for solving knapsack problem. *Swarm Evol. Comput.* **2015**, *20*, 14–22. [CrossRef]
27. Zhang, J. Comparative study of several intelligent algorithms for knapsack problem. *Procedia Environ. Sci.* **2011**, *11*, 163–168. [CrossRef]
28. Zou, D.; Gao, L.; Li, S.; Wu, J. Solving 0-1 knapsack problem by novel global harmony search algorithm. *Appl. Soft Comput.* **2011**, *11*, 1556–1564. [CrossRef]
29. Lv, J.; Wang, X.; Huang, M.; Cheng, H.; Li, F. Solving 0-1 knapsack problem by greedy degree and expectation efficiency. *Appl. Soft Comput.* **2016**, *41*, 94–103. [CrossRef]
30. Pavithr, R.S. Gursaran, Quantum inspired social evolution (QSE) algorithm for 0-1 knapsack problem. *Swarm Evol. Comput.* **2016**, *29*, 33–46. [CrossRef]
31. Zhao, J.; Huang, T.; Pang, F.; Liu, Y. Genetic algorithm based on greedy strategy in the 0-1 knapsack problem. In Proceedings of the 2009 Third International Conference on Genetic and Evolutionary Computing, Guilin, China, 14–17 October 2009.
32. Zhou, Y.; Chen, X.; Zhou, G. An improved monkey algorithm for 0-1 knapsack problem. *Appl. Soft Comput.* **2016**, *38*, 817–830. [CrossRef]

33. Sanchez-Diaz, X.; Ortiz-Bayliss, J.C.; Amaya, I.; Cruz-Duarte, J.M.; Conant-Pablos, S.E.; Terashima-Marin, H. A feature-independent hyper-heuristic approach for solving the knapsack problem. *Appl. Sci.* **2021**, *11*, 10209. [CrossRef]
34. Dell'Amico, M.; Delorme, M.; Iori, M.; Martello, S. Mathematical models and decomposition methods for the multiple knapsack problem. *Eur. J. Oper. Res.* **2019**, *274*, 886–899. [CrossRef]
35. Fukunaga, A.S. A branch-and-bound algorithm for hard multiple knapsack problems. *Ann. Oper. Res.* **2011**, *184*, 97–119. [CrossRef]
36. Martello, S.; Monaci, M. Algorithmic approaches to the multiple knapsack assignment problem. *Omega* **2020**, *90*, 102004. [CrossRef]
37. Angelelli, E.; Mansini, R.; Speranza, M.G. Kernal search: A general heuristic for the multi-dimensional knapsack problem. *Comput. Oper. Res.* **2010**, *37*, 2017–2026. [CrossRef]
38. Haddar, B.; Khemakhem, M.; Hanafi, S.; Wilbaut, C. A hybrid quantum particle swarm optimization for the multidimensional knapsack problem. *Eng. Appl. Artif. Intell.* **2016**, *55*, 1–13. [CrossRef]
39. Meng, T.; Pan, Q.-K. An improved fruit fly optimization algorithm for solving the multidimensional knapsack problem. *Appl. Soft Comput.* **2017**, *50*, 79–93. [CrossRef]
40. Wang, L.; Yang, R.; Ni, H.; Ye, W.; Fei, M.; Pardalos, P.M. A human learning optimization algorithm and its application to multi-dimensional knapsack problems. *Appl. Soft Comput.* **2015**, *34*, 736–743. [CrossRef]
41. Zhang, X.; Wu, C.; Li, J.; Wang, X.; Yang, Z.; Lee, J.-M.; Jung, K.-H. Binary artificial algae algorithm for multidimensional knapsack problems. *Appl. Soft Comput.* **2016**, *43*, 583–595. [CrossRef]
42. Gao, C.; Lu, G.; Yao, X.; Li, J. An iterative pseudo-gap enumeration approach for the multidimensional multiple-choice knapsack problem. *Eur. J. Oper. Res.* **2017**, *260*, 1–11. [CrossRef]
43. Wang, Y.; Wang, W. Quantum-inspired differential evolution with gray wolf optimizer for 0-1 knapsack problem. *Mathematics* **2021**, *9*, 1233. [CrossRef]