The Knapsack Problem with Conflicts: The Advantage of More than One Knapsack

AMBER KULP[1], YUN LU[2], MYUNG SOON SONG[2], FRANCIS J. VASKO[2]
[1]Department of Computer Science and Information Technology,
Kutztown University,
Kutztown, PA,
USA

[2]Department of Mathematics,
Kutztown University,
Kutztown, PA,
USA

Abstract: - The Knapsack Problem with Conflicts (KPC) is an extension of the classic Knapsack Problem (KP) that requires pairs of items that are in conflict cannot both be inserted into the knapsack. The KPC deals with one knapsack. Although we do not claim that this paper involves a new theory, this short paper will demonstrate for the first time in the operations research (OR) literature that there can be a significant advantage to using two knapsacks, but the advantage to using more than two knapsacks decreases rapidly. This is supported by an extensive empirical analysis solving 11,640 instances based on problems from the OR literature. The methodology of considering the advantage of using more than one knapsack when pairs of items can conflict has not been reported previously in the literature. Based on the authors' personal experience, this information can be very useful to OR practitioners.

Key-Words: - knapsack problems, knapsack problems with conflicts, knapsack problems with forfeits, operations research (OR) practice, integer programming software, Gurobi.

Received: May 9, 2025. Revised: June 11, 2025. Accepted: July 13, 2025. Published: November 24, 2025.

1 Introduction

An important extension of the classic knapsack problem (KP) is the knapsack problem with conflicts (KPC). In the KPC, in addition to filling the knapsack to optimize total profit without violating the capacity constraint, there are pairs of items that are in "conflict". For pairs of items that are in conflict, at most one of the items can be inserted into the knapsack. Recent papers [1], [2], [3] have shown that optimal solutions can be obtained for many KPC instances that are discussed in the literature.

However, except for one paper [4] that focused on heuristic solution strategies and small problem instances, only problems that consider a single KP are considered in the literature [1], [2], [3]. [5], [6], [7], [8], [9], [10]. Since the authors are familiar with industrial applications [11], that can involve more than one knapsack when solving a KPC, a natural question is: What is the advantage of using more than one knapsack? Unfortunately, in real-world applications, the trade-off between the benefits and costs of using more than one knapsack when solving the KPC can be difficult to quantify.

In this short paper, for the first time in the OR literature, using a fixed total capacity for all the knapsacks in a given problem, we explore the benefit in terms of increased objective function values when we go from a single knapsack to more than one knapsack when conflict pairs are present. Intuitively, we expect more knapsacks (but with total capacity fixed) to be beneficial, and we attempt to empirically quantify this benefit in terms of objective function values under several scenarios.

When more than one knapsack is used in the KPC, it is referred to as the multiple knapsack problem with conflicts (MKPC), [4]. We now give a mathematical formulation that we used in Gurobi for the MKPC that is slightly different from the one given in [4]. Let (G, E) be the conflict graph for a KPC where G is the set of vertices (items) and E is the set of edges (incompatible items have edges joining them). Let each item i ($1 \leq i \leq n$) have a positive integer profit p_i and a positive integer weight w_i, and $x_{ij} = 1$ if item i is inserted into knapsack j (j = 1, 2,…,m) and 0 otherwise. The capacity of knapsack j is c_j.

The multiple knapsack problem with conflicts (MKPC) is as follows:

Maximize $\sum_j \sum_i p_i \, x_{ij}$ (1)

Such that:

$\sum_i w_i x_{ij} \ \le \ c_j, j = 1 \dots m$ (2)

$x_{ij} + x_{kj} \le 1 \ \forall$ (i, k) ε E, $j = 1, \dots, m$ (3)

$\sum_j x_{ij} \ \le \ 1, \ i = 1, \dots, n$ (4)

$x_i \in \{0, 1\}, \qquad i = 1, \dots, n.$ (5)

The objective (1) is to maximize the sum of the profits of the selected items. Constraints (2) require that the knapsack capacities are not exceeded. Constraints (3) ensure that conflicting items cannot be placed in the same knapsack. Constraints (4) ensure that an item can be placed in at most one knapsack. Constraints (5) require the variables to be binary.

In the next section, we will provide a literature review of the KPC and the one paper that we are aware of that deals with the MKPC. This will be followed by a discussion of how we selected problem instances to test our hypothesis that more knapsacks are beneficial when solving a knapsack problem with conflicts. Next, we will provide an empirical analysis of the results obtained from using Gurobi to solve each of the 1680 selected test instances with different numbers of knapsacks, but such that the total knapsack capacity for a given problem is fixed. Gurobi will be called on to solve 11,640 problems. We will close this short paper with some observations concerning the benefit of having more than one knapsack when there are pairs of items that are incompatible and cannot both be inserted into the same knapsack.

2 Literature Review

In this section, the focus will be primarily on the KPC. However, we will also discuss the one paper [4] that we are aware of that deals with the MKPC. Since the literature on the knapsack problem and its extensions is very extensive, we suggest the reader start by referring to [12], [13], [14} for a good overview. It is interesting to note that the KPC has gone by several names in the OR literature. In 2002 [10], this problem was introduced as the disjunctively constrained knapsack problem (DCKP) and used heuristic and exact algorithms to solve it. In [10], an exact algorithm based on the branch-and-bound method is used that is based on dualizing the conflict constraints. In a 2006 paper [5], a local search-based algorithm for the DCKP was proposed. In a 2007 paper [6], the DCKP was solved using reduction strategies, and a branch-and-bound method was used to solve the reduced problem. In a 2009 paper [9], the problem is now referred to as the Knapsack Problem with Conflict Graphs (KPCG), and [9] develops an algorithm capable of producing exact solutions when G is either a chordal graph or a graph with bounded tree width. In still referring to this problem as the DCKP, a 2012 paper [7] and a 2014 paper [8] make use of scatter-search and an iterative rounding search-based heuristic to solve the DCKP, respectively. Both [1] and [2] present state-of-the-art exact algorithms for large instances of the KPC. In [1], a branch-and-bound algorithm is presented that is based on binary branching. In [2], a combinatorial branch-and-bound algorithm based on an n-ary branching scheme is proposed. This approach combines different procedures for pruning the branch-and-bound nodes built on different relaxations of the KPC. It is noted in both [1] and [2] that these highly specialized algorithms lose their advantage over general-purpose integer programming software for optimally solving knapsack problems with conflicts (KPC) as the conflict graph density decreases. For conflict graphs with densities below 0.1, it is felt that general-purpose integer programming software performs well. However, in a 2025 paper [3], it was shown that a large number of 4,800 KPC instances (most with conflict graph densities greater than 0.1) used in [1] and [2] could be solved efficiently using Gurobi with default parameters on a standard computer. Specifically, in [3], over 90% of the 4,800 KPC instances were solved to optimality, with 82% solved in an average of 3 seconds. Hence, [3] demonstrated that the general-purpose integer programming software Gurobi is capable of solving a large number of KPC instances from the literature.

The KPC deals with "hard" conflict constraints. Hence, for any pair of incompatible items, they cannot both be inserted into the knapsack, However, in a 2020 paper [15] the idea of "soft" conflict constraints was introduced in which pairs of incompatible items can both be inserted into the knapsack if a penalty is paid. This type of knapsack problem is referred to as the knapsack problem with forfeits (KPF). Besides the 2020 paper that introduced the KPF, the following paper, [16]. [17], [18], [19], [20] have presented solution approaches for the KPF.

All the papers mentioned above deal with conflicts between pairs of items and only one knapsack. There is only one paper [4] that the authors are aware of that considers the MKPC. In [4], the focus is on four heuristic solution procedures that are tested on 18 different MKPCs with 15 instances (the number of knapsacks is fixed in each instance) for each problem (a total of 18x15 = 270 instances). In [4], there is no discussion or analysis to determine the advantage of having different numbers of knapsacks. All these instances were generated by the author of [4], and we are still waiting for a reply to two requests we made for access to the 270 instances (first request sent 11/20/2024).

In this paper, we will focus strictly on using Gurobi to determine the benefit of having more than one knapsack when pairs of incompatible items are considered in the problem and the conflict constraints are “hard”. In other words, we are focusing only on the KPC. In the next section, empirical analysis will demonstrate the benefit of being able to use more than one knapsack with the total capacity fixed.

3 Empirical Analysis

3.1 The Problem Instances

The 4,800 KPC instances that were used in [1], [2], and [3] consisted of four data sets with correlated data that were referred to as C1, C3, C10, and CORRELATED, and four data sets with uncorrelated data that were referred to as R1, R3, R10, and RANDOM. The datasets C1, C3, and C10 each had 720 KPC instances, and CORRELATED had 240 KPC instances. And the datasets R1, R3, and R10 each had 720 KPC instances, and RANDOM had 240 KPC instances for a total of 4,800 KPC instances. The KPC instances in CORRELATED and RANDOM had the sparsest conflict graph densities, ranging from 0.001 to 0.05. The KPC instances in C1, C3, C10, R1, R3, and R10 had conflict graph densities ranging from 0.1 to 0.9. For more information on these KPC instances, [3].

Since we were interested in maintaining a fixed total capacity when more than one knapsack was used in a problem, out of the 4,800 KPC instances, we only considered instances where the knapsack capacity was at least 2000. For example, if a knapsack problem had a capacity of 3,000, when we considered it with two knapsacks, each of the two knapsacks had a capacity of 1,500. When three knapsacks were considered, each of the three knapsacks had a capacity of 1,000, and so forth.

Problem instances that were used in this computational analysis are summarized in Table 1 and Table 2. From Table 1, for instances with capacities of 2,000 or 3,000, we considered the impact of using up to six knapsacks. From Table 2, for instance, with capacities of 10,000, we considered the impact of using up to ten knapsacks.

Table 1. Knapsack Capacities—up to 6 knapsacks

CORRELATED--120 instances, RANDOM--120 instances. Total capacity per instance equals 2000	
Number of knapsacks	Capacity per knapsack
1	2,000
2	1,000, 1,000
3	666, 666, 667
4	500, 500, 500, 500
5	400, 400, 400, 400, 400
6	333, 333, 333, 333, 334, 334
C3--360 instances, R3--360 instances. Total capacity per instance equals 3000	
Number of knapsacks	Capacity per knapsack
1	3,000
2	1,500, 1,500
3	1,000, 1,000, 1,000
4	750, 750, 750, 750
5	600, 600, 600, 600, 600
6	500, 500, 500, 500, 500, 500

Source: created by the authors

Table 2. Knapsack Capacities—up to 10 knapsacks

C10--360 instances, R10--360 instances Total capacity per instance equals 10,000	
Number of knapsacks	Capacity per knapsack
1	10,000
2	5,000, 5,000
3	3,333, 3,333, 3,334
4	2,500, 2,500, 2,500, 2,500
5	2,000, 2,000, 2,000, 2,000, 2,000
6	1,666, 1,666, 1,667, 1,667, 1,667, 1,667
7	1428. 1428, 1428, 1429, 1429, 1429, 1429
8	1250, 1250, 1250, 1250, 1250, 1250, 1250, 1250
9	1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1112
10	1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000

Source: created by the authors

3.2 Analysis of Gurobi Solutions When Solving These 11,640 MKPC Instances

Unless otherwise stated, all Gurobi (12.0.3) runs are executed using default parameter settings on an

AMD Ryzen 5 7600X 6-Core Processor 4.70 GHz with a 64-bit operating system, x64-based processor using Windows 11. Because of the huge number of problems to be solved (11,640 instances), a time limit of 300 seconds was used for each problem. We used Gurobi because it had performed well in [3], and as new versions of Gurobi come out, it is very easy to upgrade and take advantage of improvements in the proprietary software.

Gurobi was able to solve to proven optimality 98% of the instances used from C3, CORRELATED, R3, and RANDOM. Since the instances in C10 and R10 have been shown [3] to be harder to solve, it is not surprising that, given the 300-second time limit, only 70% of the C10 instances used and 74% of the R10 instances used obtained proven optimums. However, we felt that the solutions generated by Gurobi would be valid to determine the benefit in terms of objective function values of using more than one knapsack. Most instances that were not proven optimum had relatively small mipgap values. In other words, the solutions obtained were guaranteed to be close to the optimums. Furthermore, the solution values that were not proven optimum were close enough to be of value in industrial applications such as [11].

Alternatively, even allowing the 30% of the C10 instances and the 26% of the R10 instances not proven optimal in 5 minutes to run for up to one hour for each of the 10 scenarios could potentially require a total execution time of 84 *days,* which we felt was not warranted.

The results of Table 3 and Table 4, as well as Figure 1 and Figure 2, vividly illustrate the advantage of using at least two knapsacks instead of one. In Table 3, for total capacities of both 2,000 and 3,000, using two knapsacks has a major advantage over just using one knapsack based on the improved objective function values. When the capacity is 10,000, the advantage of using more than two knapsacks is also evident. Table 4 clearly shows that for total capacities of 2,000 and 3,000, the benefit of increased objective function value decreases dramatically beyond two knapsacks. When the capacity is 10,000, the advantage of using more than two knapsacks does not decrease dramatically after two knapsacks. Figure 1 and Figure 2 visually display the information given in Table 3 and Table 4 for the C3 and C10 cases. These figures dramatically show the drop in additional benefits beyond the use of two knapsacks.

Table 3. Average objective function values based on the number of knapsacks

Number of knapsacks	CORRELATED	C3	C10
1	3305	2751	4784
2	3552	3083	7323
3	3612	3109	8687
4	3632	3110	9428
5	3632	3111	9856
6	3632	3111	10104
7			10217
8			10291
9			10320
10			10341
Number of knapsacks	RANDOM	R3	R10
1	8087	678	925
2	9398	894	1569
3	9676	969	1997
4	9737	996	2284
5	9737	1005	2481
6	9737	1008	2619
7			2715
8			2782
9			2829
10			2860

Source: created by the authors

Table 4. Percent increase in average objective function value when the number of knapsacks is increased by one

Number of knapsacks	CORRELATED	C3	C10
2	7.46%	12.07%	53.09%
3	1.70%	0.86%	18.61%
4	0.55%	0.03%	8.54%
5	0.00%	0.01%	4.53%
6	0.00%	0.00%	2.51%
7			1.12%
8			0.72%
9			0.29%
10			0.21%
Number of knapsacks	RANDOM	R3	R10
2	16.21%	31.72%	69.67%
3	2.96%	8.49%	27.25%
4	0.63%	2.71%	14.39%
5	0.00%	0.97%	8.62%
6	0.00%	0.30%	5.55%
7			3.67%
8			2.47%
9			1.69%
10			1.12%

Source: created by the authors

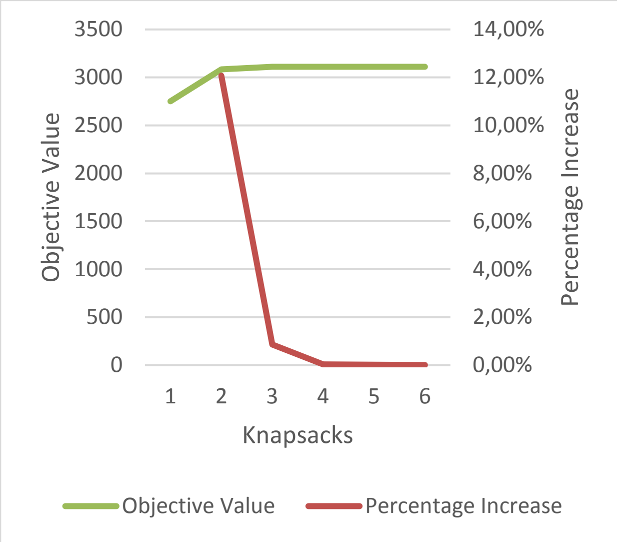

Fig. 1: C3
Source: created by the authors

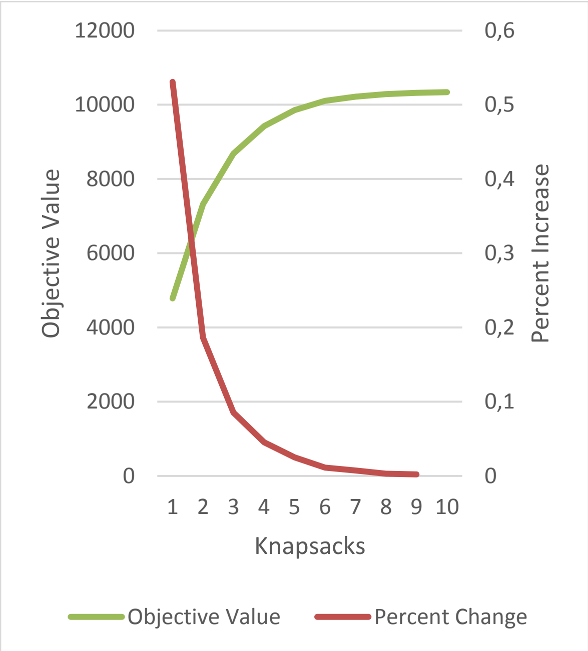

Fig. 2: C10
Source: created by the authors

4 Conclusion

The knapsack problem with conflicts (KPC) addresses the situation where pairs of items are incompatible and at most one item in a pair can be inserted into the knapsack. However, the KPC deals with only one knapsack, and there can be industrial applications that can handle more than one knapsack. In this short article, for a variety of scenarios, we show that the objective function values can be increased significantly when two knapsacks are used instead of one. However, the benefit in terms of increased objective function value increases slowly when more than two knapsacks are considered. Finally, in real-world applications, the benefits of having more than one knapsack must be weighed against logistical and productivity concerns related to dealing with more than one knapsack when pairs of incompatible items are involved.

References:

[1] Bettinelli, A., Cacciani, V., & Malaguti, E., A branch-and-bound Algorithm for the Knapsack Problem with Conflict Graph. *INFORMS Journal on Computing*, 29(3), 2017, pp. 457-473. https://doi.org/10.1287/ijoc.2016.0742.

[2] Coniglio, S., Furini, F., & San Segundo, P., A New Combinatorial Branch-and-bound Algorithm for the Knapsack Problem with Conflicts. *European Journal of Operational Research*, 289, 2021, pp. 435-455. https://doi.org/10.1016/j.ejor.2020.07.023.

[3] Song, M. S., Lin, P. H., Lu, Y., Vasko, F. J., Identifying Knapsack Problems with Conflicts that Are Difficult to Solve Optimally Using General-Purpose Integer Programming Software. *Operations Research Forum*. 6(88), 2025. https://doi.org/10.1007/s43069-025-00496-z.

[4] Basnet, C., Heuristics for the Multiple Knapsack Problem with Conflicts. *Int. J. Operational Research* 32(4), 2018, pp. 514-525.

[5] Hifi, M., & Michrafy, M., A Reactive Local Search-based Algorithm for the Disjunctively Constrained Knapsack Problem. *Journal of the Operational Research Society*, 57 (6), 2006, pp. 718–726. https://doi.org/10.1057/palgrave.jors.2602046.

[6] Hifi, M. , & Michrafy, M., Reduction Strategies and Exact Algorithms for the Disjunctively Constrained Knapsack Problem. *Computers & Operations Research*, 34 (9), 2007, pp. 2657–2673. https://doi.org/10.1016/j.cor.2005.10.004.

[7] Hifi, M. & Otmani, N., An Algorithm for the Disjunctively Constrained Knapsack Problem. *International Journal of Operational Research*, 13 (1), 2012, pp. 22–43.

[8] Hifi, M., An Iterative Rounding Search-based Algorithm for the Disjunctively Constrained Knapsack Problem. *Engineering Optimization*, 46 (8), 2014, pp. 1109–1122. https://doi.org/10.1080/0305215X.2013.819096.

[9] Pferschy, U. & Schauer, J. The Knapsack Problem with Conflict Graphs. *Journal of Graph Algorithms and Applications,* 13(2), 2009, pp. 233-249. https://doi.org/10.7155/jgaa.00186.

[10] Yamada, T., Kataoka, S. & Watanabe, K., Heuristic and Exact Algorithms for the Disjunctively Constrained Knapsack Problem. *Journal of Information Processing (JIP),* 43 (9), 2002, pp. 2864–2870.

[11] Vasko, F.J., Newhart, D.D., and Stott, K.L., A Hierarchical Approach for One-Dimensional Cutting Stock Problems in the Steel Industry that Maximizes Yield and Minimizes Overgrading, *European Journal of Operational Research*, 114(1), 1999, 72-82. https://doi.org/10.1016/S0377-2217(98)00035-6.

[12] Cacchiani, V., Lori, M., Locvatelli, A., Martello, S., Knapsack Problems - An Overview of Recent Advances. Part I: Single Knapsack Problems. *Computers & Operations Research* 143. 2022a, 105692. https://doi.org/10.1016/j.cor.2021.105692.

[13] Cacchiani, V., Lori, M., Locvatelli, A., Martello, S., Knapsack Problems-an Overview of Recent Advances. Part II: Multiple, Multidimensional, and Quadratic knapsack Problems. *Computers & Operations Research* 143. 2022b, 105693. https://doi.org/10.1016/j.cor.2021.105693.

[14] Kellerer, H., Pferschy, u., Pisinger, D., Knapsack Problems. In: Springer Nature Book Archives Millennium, Springer, 2004.

[15] Cerulli, R., D'Ambrosio, C., Raiconi, A., Vitale, G., 2020. The knapsack problem with forfeits. In: Combinatorial Optimization: 6th International Symposium, ISCO 2020, Montreal, QC, Canada, May, 2020. 4–6, Revised Selected Papers. Springer, Berlin, Heidelberg, pp. 263–272. https://doi.org/10.1007/978-3-030-53262-8_22.

[16] Capobianco, G., D'Ambrosio, C., Pavone, L., Raiconi, A., Vitale, G., Sebastiano, F., A hybrid metaheuristic for the knapsack problem with forfeits. *Soft Comput*. 26, 749–762, 2022. https://doi.org/10.1007/s00500-021-06331-x.

[17] Jovanovic, R., & Voss, S. Fixed set search matheuristic applied to the knapsack problem with forfeits. *Computers & Operations Research* 168. 106685. 2024.

[18] Zhou, Q., Hao, J.K., Jiang, Z. Z., & Wu, Q. Adaptive feasible and infeasible evolutionary search for the knapsack problem with forfeits. *Inti. Trans. In Op. Res*. 0, 1-30, 2024. https://doi.org/10.1111/itor.13512.

[19] Vieira, M.M., Nogueira, B., Pinheiro, R.G.S. An integrated ILS-VND strategy for solving the knapsack problem with forfeits. *Journal of Heuristics*. 30, 399-420, 2024. https://doi.org/10.1007/s10732-024-09532-3.

[20] Zhao, J., Hifi, M. A reinforcement learning-driven cooperative scatter search for the knapsack problem with forfeits. *Computers and Industrial Enginering,* 198 110713, 1-16, 2024. https://doi.org/10.1016/j.cie.2024.110713.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed to the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or the Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.