Making Gurobi Competitive with the Best Specialized Algorithms for the Multiple Knapsack Problem with Setups: Implications for OR Practitioners

PEI HUA LIN[2], YUN LU[1], MYUNG SOON SONG[1], FRANCIS J. VASKO[1]
[1]Department of Mathematics
Kutztown University
Kutztown, PA, 19530
U.S.A.
[2]Department of Computer Science
Kutztown University
Kutztown, PA, 19530
U.S.A.

Abstract: - An extension of the classic Knapsack Problem (KP) is the Multiple Knapsack Problem with Setups (MKPS). The MKPS has been used to model business applications that have previously been solved using approximate solution methods and implemented in industrial settings. In the operations research (OR) literature, there are a number of sophisticated solution approaches that, based on 360 MKPS instances typically used to test MKPS solution method performance, generate very good solutions. However, codes for these customized solution methods are typically not available for use by OR practitioners. In this paper, we use these same MKPS instances to demonstrate how well simple sequential increasing tolerance (SSIT) strategies combined with general-purpose integer programming software (Gurobi) can efficiently solve these MKPS instances. More importantly, it is shown both empirically and statistically that these simple approaches are as good as the best specialized methods in the literature. Hence, we present a methodology that is both easy for the OR practitioner to use and which gives results competitive with the best specialized MKPS solution methods in the literature without the need to generate algorithm-specific code. Finally, our approach is the only one that guarantees how close our solutions are to the optimums.

Key-Words: - multiple knapsack problem with setups; statistical analysis; operations research (OR) practice; simple sequential increasing tolerance (SSIT) strategies; Gurobi

Received: May 13, 2024. Revised: March 11, 2025. Accepted: April 14, 2025. Published: May 20, 2025.

1 Introduction

The classic knapsack problem (KP) and its many extensions have been used to model significant business and industrial applications [1], [2], [3], [4], [5], [6], [7], [8]. Some of the major extensions of the KP that deal with a single knapsack include: knapsack problems with setups [9], [10], [11], [12], [13], the multiple-choice knapsack problem [2], [14], [15], the knapsack problem with conflict constraints [16], [17], [18], [19], [20], the multiple-choice knapsack problem with setups [21], [22], [23], [24], [25], the multidimensional knapsack problem [26], [27], [28], and the quadratic knapsack problem [29], [30], [31], [32]. Many of these extensions have a version that involves multiple knapsacks [33], [34], [35], [36]. An overview of the many extensions of the basic KP that involve a single knapsack are discussed in [37] and extensions that involve multiple knapsacks are discussed in [38].

One extension that has many industrial production planning applications is the multiple knapsack problem with setups (MKPS). This is the problem of inserting items that are divided into classes (sometimes called families) into multiple knapsacks such that items from the same family are inserted into at most one knapsack. If items are inserted from a class, then there is a class cost incurred and a class capacity consumption incurred as well. One of the authors of this paper actively used this concept to formulate and develop customized solution approaches to solve steel mill production planning problems that were implemented in the 1990s.

However, as discussed in [39], the first priority of the operations research (OR) group that developed and implemented these systems was to satisfy the client and to internally document the models and systems that were developed. Less than 50% of the novel optimization models and systems developed by the research-based OR group discussed in [39] were documented in OR journal articles (only about 60 journal publications).

The following mathematical formulation is the standard one that appears in the literature. The Multiple Knapsack Problem with Setups (MKPS) is a generalization of the Knapsack Problem with Setup to multiple knapsacks with each class (also called families) i that can be activated and assigned to at most one knapsack. Consider a set of T knapsacks ($t \in \{1,\ldots,T\}$), such that each knapsack t has capacity b_t, and a set of N classes of items. Each class $i \in \{1,\ldots,N\}$ is composed of n_i items. Let d_i (a nonnegative integer) be the capacity consumption of class i and let f_{it} (a positive integer) be the setup cost of class i in knapsack t. Each item j of class i, that is, $j \in \{1,\ldots,n_i\}$, is characterized by a capacity consumption a_{ij} a positive integer and a knapsack dependent profit c_{ijt} also a positive integer related to the placement in knapsack $t \in \{1,\ldots,T\}$. The objective is to maximize the total profit of the selected items, respecting that the capacity consumption of the selected items assigned to each knapsack t does not exceed the corresponding capacity b_t, while considering the related classes' setup costs and capacity consumption. The variable y_{it} is equal to 1 if class i is activated and assigned to knapsack t and 0 otherwise. The variable x_{ijt} is equal to 1 if item j of class i is placed in knapsack t and 0 otherwise.

The MKPS can be formulated as a 0-1 integer linear program as follows for $\forall i \in \{1, \ldots, N\}$, $\forall j \in \{1, \ldots, n_i\}, \forall t \in \{1, \ldots, T\}$:

$$Maximize \ \ z = \sum_{t=1}^{T} \sum_{i=1}^{N} (\sum_{j=1}^{n_i} c_{jit} x_{ijt} - f_{it} y_{it}) \quad (1)$$

$$\text{s.t.} \quad \sum_{i=1}^{N} \left(d_i y_{it} + \sum_{j=1}^{n_i} a_{ij} x_{ijt} \right) \le b_t \quad (2)$$

$$x_{ijt} \le y_{it} \quad (3)$$

$$\sum_{t=1}^{T} y_{it} \le 1 \quad (4)$$

$$x_{ijt}, y_{it} \in \{0, 1\} \quad (5)$$

The objective function (1) maximizes the profit of selected items minus the fixed setup costs of activated classes. Constraints (2) guarantee that for each knapsack $t \in \{1,\ldots,T\}$, the sum of the total weight of related placed items and related classes' setup capacity consumption does not exceed the related capacity b_t. Constraints (3) require that each item is placed in a knapsack only if it belongs to a class that has been activated and assigned to the same knapsack. Constraints (4) ensure that any class i is assigned to at most one knapsack. Constraints (5) ensure that the decision variables are binary. Note that the MKPS contains $T + T \times n + n$ constraints and $T \times N + T \times n$ variables where $n = \Sigma n_i$.

When an OR practitioner is faced with solving a business application that is formulated in part or in total as an MKPS, there are several solution options: (1) the OR practitioner can code a problem-specific algorithm either based on a solution method in the literature or developed by the practitioner taking advantage of application-specific knowledge or both, (2) try to obtain the computer code for a specialized solution algorithm that appears in the literature, or (3) use in a straightforward manner with all default parameter settings general-purpose integer programming software that the corporation has purchased and uses to solve optimization problems.

The problem with the first approach is that it involves extensive algorithm development, coding, testing, and validation efforts which can be costly and time consuming. For industrial applications, this approach is particularly unattractive if reasonably competitive solutions can be obtained using general-purpose integer programming software already purchased and in use by the corporation. The problem with the second approach is that typically such computer codes are not available for industrial use. At most, some specialized codes, such as the Combo code [40] for the classic knapsack problem are available for research purposes only. Additionally, even if such codes were available there would be no technical support obtainable. The problem with the third approach is that when the software is used in a default manner with a tight termination tolerance, typically 0.0001, then the execution times can be excessive.

In contrast to these three options, there have been a number of recent publications that discuss how general-purpose integer programming software (e.g., Gurobi and CPLEX) can be used in an iterative fashion. In other words, instead of executing Gurobi or CPLEX for a maximum of one hour at the default tolerance of 0.0001, an OR practitioner or researcher would define a sequence of tolerances and execution times for each tolerance with each tolerance looser than the previous one. For a particular application, based on some preliminary experimentation and how quickly a solution was needed, the user would decide on the most appropriate sequence of tolerances and maximum execution times for each sequence [41].

For a variety of binary integer programming problems such as the Set K-Covering Problem [42], the Generalized Assignment Problem [43], and the

Multiple-Choice Knapsack Problem with Setups [44] solutions have been efficiently generated such that they are guaranteed to be very close to the optimums (typically guaranteed to be within 0.1% of the optimums). An overview of this technique referred to as the simple sequential increasing tolerance (SSIT) strategy is provided in [41]. Although to-date many of the successful applications of the SSIT strategy have been for variations and extensions of the Knapsack Problem, the authors of this paper are actively exploring other potential applications of the SSIT strategy. However, there have been a number of unsuccessful applications of the SSIT strategy (most unpublished at this time). In general, the SSIT strategy fails when even with looser termination tolerances the execution times are excessive or the bounds are so large that they are not meaningful. Such bounds are usually not acceptable in practice. We typically consider a SSIT strategy to be a practical success if it can guarantee that its solutions are within 1% of the optimums. Actual SSIT strategy applications typically guarantee that its solutions are within 0.1% of the optimums. This guarantee is in sharp contrast to metaheuristics which provide no guarantees on solution quality! There is no a priori guarantee that SSIT can be used successfully to solve a particular binary integer programming problem. Hence there is the need to determine its appropriateness on a case-by-case basis (including the topic of this paper, the MKPS). This is not unlike other solution strategies (e.g. genetic algorithms, scatter search, tabu search, simulated annealing, GRASP, etc.) that are applied to problems on a case-by-case basis. The specific SSIT strategies used to solve the MKPS will be outlined later in this paper.

In the OR literature there are 360 MKPS instances that are available [45] for researchers to empirically test their MKPS solution approaches. The primary contribution of this paper is not to present any new theory or methodology. As discussed above SSIT is not a new solution methodology. It was first discussed in [46]. To date SSIT has been successfully applied to a number of binary integer programming problems, but it has also failed for others. Simply stated, we consider the SSIT strategy to fail when no tight bound on the optimum can be determined within an acceptable time for the given application. Although not discussed in the literature at this time, the authors have been unable to successfully use a SSIT strategy to solve the set covering problem with conflicts on sets. Hence, its usefulness needs to be demonstrated on a case-by-case basis. Instead, the primary focus of this paper is to show that by creatively using the general-purpose integer programming software Gurobi one can quickly obtain solutions that are empirically and statistically on pare with the best specialized MKPS algorithms in the literature. Furthermore, these solutions are guaranteed to be very close to the optimums. The authors feel this practitioner-focused contribution fills a vital gap: making robust MKPS solving accessible to real-world users via a commercial solver. Additionally, this paper makes a significant and timely contribution by empirically demonstrating that Simple Sequential Increasing Tolerance (SSIT) strategies combined with Gurobi can solve the MKPS nearly as well—or better—than state-of-the-art, highly specialized algorithms. This is both a novel and practical methodology for solving MKPS instances. Finally, although the Gurobi code is proprietary, if default parameter settings do not yield desired results, Gurobi's tuning tool is very helpful at optimizing parameter settings.

Hence the focus of this paper is to establish a strategy that is easy for OR practitioners to use that will efficiently generate solutions to MKPS instances that are guaranteed to be very close to the optimums. Specifically, this paper will demonstrate: (1) that 360 MKPS instances available in the literature can be efficiently solved using simple sequential increasing tolerance (SSIT) strategies combined with general-purpose integer programming software, (2) which SSIT strategy provides the tightest solution bounds. In other words, which SSIT strategy will *guarantee* solutions closest to the optimums, (3) empirically and statistically that these SSIT strategies in conjunction with the Gurobi integer programming software are as good as the best specialized methods in the literature, but are easier for the OR practitioner to use.

In the next section, the solution methods for the MKPS that appear in the literature will be overviewed. After this discussion, the results of using these solution methods to solve 360 MKPS instances [45] will be given. This will be followed by an analysis of how SSIT can best be used with general-purpose integer programming software to solve the 360 MKPS test instances from the literature [45]. Next, empirical and statistical analyses will be used to compare solution method performances on these 360 MKPS instances. This will be followed by a discussion of the benefits to the OR practitioner who uses SSIT strategies with general-purpose integer programming software. Finally, some conclusions and implications for OR practitioners will be given.

2 Literature Review

In this section the focus is exclusively on the MKPS. For a thorough review of other extensions of the KP, we suggest the reader start by consulting [37] and [38]. The MKPS was formally stated and first dealt with in [21]. In [21] a heuristic based on branch-and-bound was proposed to solve the MKPS, but this approach was only effective for small instances. In [45] 360 MKPS instances were defined which are the current benchmark for testing MKPS solution methods. Additionally, in [45] a reduction method called Mred and a matheuristic based on Tabu Search (MTS) were discussed. The MTS iteratively decomposes the MKPS into a series of less complex subproblems. In [47] a constructive heuristic called RBH that was inspired by Mred was developed. Also, in [47] a matheuritic was developed and denoted as VND&IP that combines variable neighbourhood descent (VND) with integer programming (IP). They compare their results to results obtained from executing CPLEX for up to one hour. In [48] a Lagrangean relaxation (Lag) approach is used that takes advantage of the problem structure. In [49] an enhanced hybrid method (EHM) that combines the solution related to the mixed integer relaxation and a series of KPs is developed. It has not been tested on uncorrelated MKPS instances. Finally, [50] developed a solution approach denoted as LP&DP-VNS that uses the linear programming (LP) relaxation and dynamic programming (DP) to enhance a variable neighbourhood search (VNS). The LP-VNS is used to decompose the MKPS into T-independent classic KPs and DP is used to optimally solve each KP. This solution approach is stochastic and the authors make ten runs and select the best. We will denote the solution approach of running LP&DP-VNS ten times and selecting the best solution obtained as the answer as LP&DP-VNS10. It should be noted that all of the solution methods mentioned above are tailored to solve the MKPS. More importantly, none of these approaches provide guarantees on how close their solutions are to the optimums. In fact, LP&DP-VNS10 (the best performing approach) is stochastic and reports the best of 10 solutions generated. Finally, to our knowledge, none of the computer codes associated with these methods are available either to researchers or OR practitioners.

In the next section, we will present empirical results from executing CPLEX for up to one hour, Lag, EHM, RHB, MTS, Mred, VND&IP, and LP&DP-VNS10 to solve the 360 MKPS instances made available in [45]. All these results are taken from detailed results provided in [50] including computer execution times. Then these same 360 MKPS instances will be solved using several SSIT scenarios combined with general-purpose integer programming software (primarily Gurobi). It is important to note that all general-purpose integer programming software results involve using SSIT strategies. This is because preliminary results using both Gurobi and CPLEX with only the default termination tolerance many times resulted in excessive computation times. This was not surprising because CPLEX results for the MKPS previously reported in the literature tended to require excessive computation times.

3 Empirical Results Using MKPS Instances [45]

3.1 Description of the 360 MKPS Test Instances

The benchmark instances consist of 360 MKPSs and can be obtained from [45]. These 360 MKPS instances are partitioned into three sets of 120 instances each. Each set contains instances with T in {5,10,15,20}, and N in {10,20,30}. There are 120 uncorrelated instances with n_i values selected from [60,100] such that: $c_{ijt} = r_1 + 10$, $a_{ij} = r_2 + 10$, $d_i = (r_3 +10)x35$, and $f_{it} = (r_4 + 10)x35$. There are 240 strongly correlated instances. For 120 of these instances n_i values are selected from [40,60] and for the other 120 instances the n_i values are selected from [90,110]. The weight of an item j belonging to a family i is uniformly distributed in [10, 10,000] and the profit associated with each item is defined by $c_{ijt} = a_{ij} + e_1$ with e_1 uniformly distributed in [0,10] There are 10 instances for each combination of (T,N,n_i). For more background on these 360 instances, see [45].

3.2 Empirical Results for Eight Solution Methods from the Literature

Table 1 provides average objective function values and average execution times (in seconds) for eight solution methods (CPLEX for up to one hour, Lag, EHM, RHB, MTS, Mred, VND&IP, and LP&DP-VNS10) over 120 instances in each of the three categories described previously. The average objective function values were calculated from the 360 objective function values provided in the literature [50]. The execution times were reported in [50]. As stated earlier, LP&DP-VNS10 is the result of running LP&DP-VNS ten times and reporting the best solution obtained. The execution times for LP&DP-VNS10 are the sum of the ten execution times. For each of the three sets of 120 MKPS instances LP&DP-VNS10 had the best objective function values. For the 120 strongly correlated MKPS (n_i [40,60]) instances the Lag solution method tied LP&DP-VNS10 based on average objective function values, but the Lag method required less execution time. Many of these results are very close and hence the need for the statistical analyses that will be discussed in the next section. It should be noted that the results from using CPLEX in default mode (termination tolerance of 0.0001) typically resulted in long execution times.

Method	Avg Obj	Avg Time
120 uncorrelated MKPS instances		
LP&DP-VNS10	249718	88
VND&IP	249246	49
Mred	232454	1
MTS	249300	43
RHB	232454	0.3
CPLEX 1H	249557	1763
Lag	249706	47
EHM	No data	No data
120 strongly correlated MKPS instances (n_i [40,60])		
LP&DP-VNS10	674758	60
VND&IP	674756	4
Mred	674756	7
MTS	674757	82
RHB	671489	.7
CPLEX 1H	674541	2941
Lag	674758	41
EHM	674757	21
120 strongly correlated MKPS instances (n_i [90,110])		
LP&DP-VNS10	1195180	107
VND&IP	1195179	1
Mred	1195178	9
MTS	1195178	138
RHB	1190171	0.9
CPLEX 1H	1193582	3135
Lag	1195126	444
EHM	1195179	109

Table 1: Average objective function values and execution times for solution methods from the literature. *The times are on a PC with 3.3 GHZ Intel core i3 as reported in [50]. Note: The Lag and EHM solution methods are not analyzed statistically because their details for the 360 instances are not available.

3.3 Empirical Results for SSIT Scenarios

As stated earlier, the main goal of this paper is to demonstrate that solutions to MKPS instances obtained by using SSIT strategies and the general-purpose integer programming software Gurobi are as good as solutions obtained by the best specialized solution methods available in the literature. An additional benefit of the approach presented in this paper is that it guarantees how close its solutions are to the optimums.

For the many companies (as of March 12, 2024 Gurobi claims over 2500 industrial customers spanning 40 industries world-wide) that have invested in Gurobi software being able to easily use this software to solve industrial applications formulated as MKPS can be a significant time and cost savings. The faster a model is developed, solved and implemented the faster the benefits of its use are realized. Additionally, the customer support and continual new improved versions of such software are very tangible benefits to the users.

Although the SSIT strategy can be combined with any general-purpose integer programming software, in general, we have experienced excellent results using both Gurobi and CPLEX. However, based on the CPLEX results reported in the literature, we focused mainly on using Gurobi for solving the MKPS. Using Gurobi or CPLEX in a default mode to solve an integer programming problem means that either a mipgap value less than 0.0001 is achieved or the maximum time has been reached where for a maximization problem mipgap = [best upper bound - best current solution]/(best current solution). The SSIT strategy loosens the acceptable mipgap value as the software is executing (see [41] for a general discussion of the use of SSIT).

When we started our analysis, we were using Gurobi version 10.0.0. When Gurobi 11.0.0 became available we used that version. All CPLEX runs were with version 22.1.1.0. All executions of Gurobi and CPLEX were performed on a macOS 13.1, M1 Chip, 8-Core Processor, and 8GB RAM.

Below are the details for the SSIT scenarios used, represented by stage - version, [(time, tolerance)] and the results are summarized in Table 2. Other than adjusting the mipgap termination value, the only other parameter that is adjusted is MIPFocus which is discussed below.

For example, the strategy labelled Gurobi10SSIT1D means that the MKPS is solved using Gurobi version 10.0 with all default (D) parameter settings, and the following steps:

Step 1: Use the default mipgap of 0.0001 for up to 60 seconds of execution time.

Step 2: At 60 seconds of execution time, if the software had not terminated because the mipgap was not less than 0.0001, then open the termination mipgap to 0.001 and execute the software for 60 seconds.

Step 3: At 120 seconds of execution time, if the software had not terminated because the mipgap was not less than 0.001, then open the termination mipgap

to 0.005 and execute the software for another 60 seconds.
Step 4: At 180 seconds of total execution time, if the software had not terminated because the mipgap was not less than 0.005, then terminate the program. The final mipgap will be greater than or equal to 0.005.

Uncorrelated

Gurobi10SSIT1D - Gurobi 10, [(60, 0.0001), (60, 0.001), (60, 0.005)]
Gurobi11SSIT2D - Gurobi 11, [(60, 0.0001), (120, 0.001), (180, 0.005), (180, 0.008)]
Guribi11SSIT2MF3 - Gurobi 11 with MIPFocus=3, [(60, 0.0001), (120, 0.001), (180, 0.005), (180, 0.008)]

Strongly correlated [40,60]

Gurobi10SSIT1D - Gurobi 10, [(60, 0.0001), (60, 0.001), (60, 0.005)]
Gurobi11SSIT2D - Gurobi 11, [(60, 0.0001), (120, 0.001), (180, 0.005), (180, 0.008)]
Guribi11SSIT2MF3 - Gurobi 11 with MIPFocus=3, [(60, 0.0001), (120, 0.001), (180, 0.005), (180, 0.008)]

Strongly correlated [90,110]

Gurobi10SSIT1D - Gurobi 10, [(60, 0.0001), (60, 0.001), (60, 0.005)]
Gurobi10SSIT2D - Gurobi 10, [(60, 0.0001), (120, 0.001), (180, 0.005), (180, 0.008)]
Gurobi11SSIT2D - Gurobi 11, [(60, 0.0001), (120, 0.001), (180, 0.005), (180, 0.008)]
Guribi11SSIT2MF3 - Gurobi 11 with MIPFocus=3, [(60, 0.0001), (120, 0.001), (180, 0.005), (180, 0.008)]
Gurobi11SSIT3D - Gurobi 11, [(180, 0.0001), (120, 0.001), (120, 0.005)]
Gurobi11SSIT3MF3 - Gurobi 11 with MIPFocus=3, [(180, 0.0001), (120, 0.001), (120, 0.005)]
CPLEXSSIT2D - CPLEX, [(60, 0.0001), (120, 0.001), (180, 0.005), (180, 0.008)]

In Table 2 we report the average objective function values, average execution times (in seconds), average mipgap values (%) at program termination, and maximum mipgap values (%) at program termination. Note that there is no mipgap information available for the Table 1 results. Our SSIT analysis started simple (Gurobi10SSIT1D) with three different mipgap values executed for 60 seconds each. The objective function values, times, and mipgap values were very good for the uncorrelated and strongly correlated [40,60] instances. Since a mipgap of 0.0001 (0.01%) is considered to be close enough to be optimum for practical purposes, we see from Table 2 that this simple SSIT strategy works well for 240 of the 360 MKPS instances with optimums obtained in less than 60 seconds for many of the 240 instances.

Method	Avg Obj	Avg Time	Avg mipgap	Max mipgap
120 uncorrelated MKPS instance				
Guribi10SSIT1D	249723	8	0.006	0.084
Guribi11SSIT2D	249723	8	0.006	0.081
Gurobi11SSIT2MF3	249723	11	0.006	0.020
120 strongly cor. MKPS instance [40,60])				
Guribi10SSIT1D	674721	18	0.008	0.033
Guribi11SSIT2D	674733	20	0.007	0.011
Gurobi11SSIT2MF3	674728	24	0.008	0.010
120 strongly cor. MKPS instance [90,110])				
Guribi10SSIT1D	1195087	53	2.764	58.073
Guribi10SSIT2D	1195104	95	1.961	49.854
Guribi11SSIT2D	1195093	64	0.269	19.823
Gurobi11SSIT2MF3	1195066	51	0.016	0.415
Guribi11SSIT3D	1195112	56	0.836	49.704
Gurobi11SSIT3MF3	1195106	54	0.008	0.094
CPLEXSSIT2D	1193820	162	2.516	28.443

Table 2: Average objective function values and execution times for solution methods *The times are on a macOS 13.1, M1 Chip, 8-Core Processor, and 8GB RAM.

The Gurobi10SSIT1D results for the strongly correlated [90,110] case were not as good as for the previous two cases. Although the average objective function values were comparable to the results for this case using solution methods from the literature, we found the average and maximum mipgap values to be unacceptable. Keep in mind that by using a SSIT strategy, we are trying to get *guaranteed* tight bounds on our solutions compared to the optimums. For this case, there were a few MKPS instances that although they had very good objective function values, their final mipgap values were very high. This means that Gurobi found a very good solution but was unable to reduce the best upper bound value thus not guaranteeing that the solution was close to the optimum.

Our first attempt to reduce the mipgap values was to lengthen the execution time (Gurobi10SSIT2D) but as can be seen in Table 2 this produced minimal improvement. Next, Gurobi version 11 was released and we tried two scenarios (Gurobi11SSIT2D and Gurobi11SSIT2MF3). The Gurobi11SSIT2D was able to reduce the average final mipgap to 0.269%. However, the maximum mipgap was still too large at 19.823%. We next decided to adjust one key Gurobi input parameter.

Gurobi has a large number of input parameters that can be set by the user. However, we have found that the default settings provided by Gurobi usually (not always) work very well. Using the default parameter settings, Gurobi will adjust its solution strategy based on its analysis of the particular problem that is input. One parameter that we have found helpful to adjust on occasion is MIPFocus. The Gurobi MIP solver strives to achieve a balance between finding new feasible solutions and proving that the current solution is optimal. Setting MIPFocus = 1 focuses on finding a feasible solution quickly. Setting MIPFocus = 2 focuses on proving optimality, and setting MIPFocus = 3 focuses on obtaining a tight bound.

Hence, by setting the MIPFocus parameter value to 3 (Gurobi11SSIT2MF3), we focused on proactively reducing the upper bound. Gurobi11SSIT2MF3, while maintaining high-quality objective function values, now significantly reduced both the final average mipgap value and the final maximum mipgap value. We tried one more scenario (SSIT3) which maintained a tight mipgap termination value for 180 seconds instead of 60 seconds. For SSIT3 with a default MIPFocus value of 0, the improvement was not significant (Gurobi11SSIT3D). However, for SSIT3 with MIPFocus = 3 (Gurobi11SSIT3MF3), the results were outstanding. Specifically, with average execution time of only 54 seconds, the final average mipgap was only 0.008% and the final maximum mipgap was 0.094%. In other words, our solutions were *guaranteed* to be extremely close to the optimums. Although all six Gurobi SSIT scenarios achieved relatively the same objective function values, reduction in the final mipgap values was achieved by setting MIPFocus = 3. Hence, the Gurobi11SSIT3MF3 strategy not only gives excellent objective function values, but guarantees that these solutions are very close to the optimums. The fact that our approach actually *guarantees* that our solutions are close to the optimums is what sets our approach apart from other MKPS solution approaches that appear in the literature.

When Gurobi version 11 was released, we ran two SSIT strategies (see Table 2) using this version for the uncorrelated case and the strongly correlated case [40,60], but there was basically no difference from the Gurobi10SSIT1D results. In spite of poor CPLEX performance reported in the literature, for the strongly correlated [90,110] case, we ran one SSIT scenario with CPLEX. However, the results were disappointing in all respects (average objective function value, execution time, final average mipgap, and final maximum mipgap). Table 2 gives the specifics for the CPLEX scenario.

Based on the results from Tables 1 and 2, we will now empirically compare the objective function values obtained from the best solution method from the literature, LP&DP-VNS10, with the best objective function values obtained using Gurobi.

3.4 Empirical Comparison of Objective Function Values

For each of the three categories of 120 MKPS instances (120 uncorrelated instances, 120 strongly correlated [40,60] instances, and 120 strongly correlated [90,110] instances), we now compare the average objective function values for LP&DP-VNS10 (best results from the literature) with the Gurobi with SSIT scenarios. It is important to note that the Gurobi with SSIT scenarios provide guaranteed bounds on how close their solutions are to the optimums. This is not true for any of the approaches in the literature. In fact, as stated previously, LP&DP-VNS10 (the best performing approach) is stochastic and reports the best of 10 solutions generated.

For the 120 uncorrelated instances, all three Gurobi scenarios had an average objective function value of 249723 and the LP&DP-VNS10 average objective function value was 249718. Hence the Gurobi results were slightly better than the best results in the literature and the average Gurobi execution time was between 8 and 11 seconds depending on the SSIT-Gurobi scenario that was used. Furthermore, the SSIT-Gurobi solutions were guaranteed to be very close to the optimums with an average mipgap of only 0.006%.

For the 120 strongly correlated (n_i [40,60]) instances, Gurobi11SSIT2D had an average objective function value of 674733 and the LP&DP-VNS10 average objective function value was 674758. Hence the Gurobi results deviated from the best literature results by only 0.0037%. Furthermore, average Gurobi execution time was 20 seconds and the Gurobi11SSIT2D solutions were guaranteed to be very close to the optimums with an average mipgap of only 0.007% and a maximum mipgap of 0.011%.

For the 120 strongly correlated (n_i [90,110]) instances, Gurobi11SSIT3MF3 had an average objective function value of 1195106 and the LP&DP-VNS10 average objective function value was 1195180. Hence the Gurobi results deviated from the best literature results by only 0.0062%. Furthermore, average Gurobi execution time was 54 seconds and the Gurobi11SSIT3MF3 solutions were guaranteed to be very close to the optimums with an

average mipgap of only 0.008% and a maximum mipgap of 0.094%.
In the next section, all solution methods from the literature for which all 360 MKPS solutions were available (results for Lag and EHM were not available) are statistically analyzed along with our SSIT results. The analyses are divided into the same three categories: uncorrelated, strongly correlated [40,60], and strongly correlated [90,110] that were discussed above.

4 Statistical Analyses

In this section, we conducted statistical analyses on the 360 MKPS instances (120 uncorrelated instances, 120 strongly correlated [40,60] instances, and 120 strongly correlated [90,110] instances).
For the 120 uncorrelated instances and the 120 strongly correlated [40,60] instances, we compared three SSIT scenarios (Gurobi10SSIT1D, Gurobi11SSIT2D, and Guribi11SSIT2MF3) from Table 2 to six solution methods from the literature (LP&DP-VNS10, VND&IP, Mred, MTS, RHB, and CPLEX 1H), respectively.
For the 120 strongly correlated [90,110] instances, we compared seven SSIT scenarios (Gurobi10SSIT1D, Gurobi10SSIT2D, Gurobi11SSIT2D, Gurobi11SSIT2MF3, Gurobi11SSIT3D, Gurobi11SSIT3MF3, and CPLEXSSIT2D) from Table 2 to the six solution methods from the literature described above.
Significant differences among the solution methods were detected using a repeated measures ANOVA model for all three groups of instances (120 uncorrelated instances, 120 strongly correlated [40,60] instances, and 120 strongly correlated [90,110] instances). Tukey's pairwise comparison was conducted for each group as a follow-up test [51].
Sections 4.1, 4.2, and 4.3 illustrate the summary of Tukey's comparisons in the three groups of instances. Note that the objective function gets smaller in order of A, B, and C and methods that do not share a letter are significantly different. The common significance level at 1% was applied in the analyses.

4.1 Case - 120 Uncorrelated Instances

Table 3 shows no statistically significant difference between the three SSIT scenarios (Gurobi10SSIT1D, Gurobi11SSIT2D, and Guribi11SSIT2MF3) and the four solution methods from the literature. (LP&DP-VNS10, CPLEX 1H, MTS, and VND&IP)

Method	N	Mean	Grouping	
Gurobi11SSIT2D	120	249723	A	
Gurobi11SSIT2MF3	120	249723	A	
Gurobi10SSIT1D	120	249723	A	
LP&DPVNS10	120	249718	A	
CPLEX1H	120	249557	A	
MTS	120	249300	A	
VND&IP	120	249246	A	
RHB	120	232454		B
Mred	120	232454		B

Means that do not share a letter are significantly different.

Table 3: Comparison of the solution methods with the uncorrelated instances using the Tukey Method and 99% Confidence

4.2 Case - 120 Strongly Correlated [40,60] Instances.

In Table 4, there is no statistically significant difference between the three SSIT scenarios (Gurobi10SSIT1D, Gurobi11SSIT2D, and Guribi11SSIT2MF3) and the four solution methods from the literature (LP&DP-VNS10, MTS, VND&IP, and Mred).

Method	N	Mean	Grouping	
LP&DPVNS10	120	674758	A	
MTS	120	674757	A	
VND&IP	120	674756	A	
Mred	120	674756	A	
Gurobi11SSIT2D	120	674733	A	
Gurobi11SSIT2MF3	120	674728	A	
Gurobi10SSIT1D	120	674721	A	
CPLEX1H	120	674541	A	
RHB	120	671489		B

Means that do not share a letter are significantly different.

Table 4: Comparison of the solution methods with the strongly correlated [40,60] instances using the Tukey Method and 99% Confidence

4.3 Case - 120 Strongly Correlated [90,110] Instances.

Table 5 indicates no statistically significant difference between the six Gurobi SSIT scenarios (Guribi10SSIT1D, Guribi10SSIT2D, Guribi11SSIT2D, Gurobi11SSIT2MF3, Guribi11SSIT3D, and Gurobi11SSIT3MF3) and the four solution methods from the literature (LP&DP-NS10, VND&IP, MTS, and Mred).

Method	N	Mean	Grouping		
LP&DPVNS10	120	1195180	A		
VND&IP	120	1195179	A		
MTS	120	1195178	A		
Mred	120	1195178	A		
Gurobi11SSIT3D	120	1195112	A		
Gurobi11SSIT3MF3	120	1195106	A		
Gurobi10SSIT2D	120	1195104	A		
Gurobi11SSIT2D	120	1195093	A		
Gurobi10SSIT1D	120	1195087	A		
Gurobi11SSIT2MF3	120	1195066	A		
CPLEXSSIT2D	120	1193820		B	
CPLEX1H	120	1193582		B	
RHB	120	1190171			C

Means that do not share a letter are significantly different.

Table 5: Comparison of the solution methods with the strongly correlated [90,110] instances using the Tukey Method and 99% Confidence

In summary, the Gurobi SSIT approaches demonstrated statistical parity with the best specialized methods in the literature, as evidenced by our analyses in sections 4.1, 4.2, and 4.3.

5. Benefits to the OR Practitioner: Using SSIT Strategies with Commercial Integer Programming Software

Although as discussed earlier, SSIT strategies are not always successful at providing tightly bounded solutions in a timely manner, when they are successful, they have a number of significant benefits. Although the Gurobi search procedures are proprietary, the user can make use of the Gurobi fine-tuning option to adjust parameter settings for a particular application. In the authors' experience with solving approximately 50,000 BIP instances, typically default parameter settings have worked very well. Obviously, in using a SSIT strategy, the termination mipgap value is adjusted (loosened). The only other parameter that we have found to impact the guarantee on how close the solution was to the optimum was MIPFocus. As demonstrated in Table 2 of this paper, good solutions can be obtained, but there can still be a large gap between the final best solution and the final upper bound (maximization). Again, as discussed in Section 3.3, setting MIPFocus = 3 can significantly reduce this gap without any negative impact on solution quality or execution time. This guarantees a tight bound on the solution found by the SSIT and Gurobi methodology. However, the authors of this paper are currently studying the situation when adjusting Gurobi parameters does not adequately reduce the final mipgap value in the allotted execution time. Their results will be detailed in a future paper.

When the SSIT strategy is successful, it allows the OR practitioner to avoid the time-consuming and costly development of problem-specific algorithms, computer codes, and testing. Additionally, for commercial software such as Gurobi and CPLEX, there is a wealth of examples that come with these packages. These templates often only require limited adjustments before they are ready to run domain specific BIPs. Furthermore, practitioners can quickly find answers to many software specific questions in online forums and manuals. Finally, when industrial systems use SSIT, the performance of these systems is "automatically" improved when new versions of the optimization software are installed.

6. Conclusions and Implications for OR Practitioners

Because important industrial applications can be modelled as the multiple knapsack problem with setups (MKPS), operations research (OR) practitioners that are called on to solve MKPSs need to be able to generate either optimal solutions or solutions that are guaranteed to be near-optimal in an efficient manner. The best solution methods available from the literature that are highly-specific to solving the MKPS can require considerable development, testing, and implementation time and effort for the OR practitioner. Conversely, the straightforward use of general-purpose integer programming software can require extensive computer execution times. In this paper, using 360 MKPS instances commonly used to test MKPS solution methods, it was shown that a simple sequential increasing tolerance (SSIT) strategy combined with general-purpose integer programming software could be used in a straightforward manner to efficiently generate either optimal solutions or solutions that are guaranteed to be near-optimal in an efficient manner.

Furthermore, the SSIT solution scenarios combined with the Gurobi software package were statistically compared to the best specialized solution methods. These comparisons demonstrated that the SSIT scenarios with Gurobi produced solutions competitive with the best specialized solution methods in very reasonable PC execution times (average execution times of less than a minute on a standard PC for the largest MKPSs). Finally, by setting the Gurobi parameter MIPFocus = 3 (this value focuses on reducing the upper bound in a maximization problem), a SSIT with Gurobi strategy not only generated excellent solutions quickly, but guaranteed that the solutions generated were extremely close to the optimums (at most 0.094% from the optimums). These guaranteed bounds are not provided by any approaches for the MKPS that appear in the literature.

References

[1] Martello, S., Toth, P., (1990). Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, Inc., New York, NY, USA.

[2] Kellerer, H., Pferschy, U., Pisinger, D., (2004). Knapsack Problems. In: Springer Nature Book Archives Millennium, Springer.

[3] Hifi, M., M'Hallah, R., (2012). Special issue on knapsack problems and applications. *Comput. Oper. Res*. 39, 1–2.

[4] Lahyani, R., Khemakhem,M., &Semet,F. (2017). A unified matheuristic for solving multi-constrained traveling salesman problems with profits. *EURO Journal on Computational Optimization,*5(3),393–422.

[5] Chebil, K., & Khemakhem, M. (2015). A dynamic programming algorithm for the knap sack problem with setup. *Computers & Operations Research,*64,40–50.

[6] Khemakhem, M., &Chebil, K. (2016). A tree search-based combination heuristic for the knapsack problem with setup. *Computers & Industrial Engineering,* 99, 280–286.

[7] McLay, L.A. (2006). Designing aviation security systems: Theory and practice. Dissertation Urbana, IL: University of Illinois at Urbana-Champaign.

[8] Yanchun, Y. (2006). Knapsack problems with setup. Dissertation USA: Auburn University

[9] Chajakis, E., Guignard, M., (1994). Exact algorithms for the setup knapsack problem. *INFOR* 32, 124–142.

[10] Yang, Y., Bulfin, R., (2009). An exact algorithm for the knapsack problem with setup. International Journal of Operational Research 5, 280–291.

[11] Amiri, A., Barkhi, R., (2020). A Lagrangean based solution algorithm for the multiple knapsack problem with setups. *Computers & Industrial Engineering* 153, 107089.

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

[13] Pferschy, U., Scatamacchia, R., (2017). Improved dynamic programming and approximation results for the knapsack problem with setups. *International Transactions in Operational Research* 25, 667–682.

[14] Bednarczuk, E.M., Miroforidis, J., Pyzel, P., (2018). A multi-criteria approach to approximate solution of multiple-choice knapsack problem. Comput. Optim. Appl. 70, 889–910.

[15] He, C., Leung, J.Y.-T., Lee, K., Pinedo, M.L. (2016). An improved binary search algorithm for the multiple-choice knapsack problem. *RAIRO Oper. Res*. 50, 995–1001.

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

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

[18] Coniglio, S., Furini, F., & San Segundo, P. (2021). A new combinatorial branch-and-bound algorithm for the Knapsack Problem with Conflicts. *European Journal of Operational Research*, 289, 435-455. http://doi.org/10.10116/ejor.2020.07.023.

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

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

[21] Yang, Y., (2006) Knapsack problems with setup. PhD thesis, Auburn University, Auburn, AL. USA.

[22] Adouani, Y., Jarboui, B., & Masmoudi, M. (2019). A variable neighborhood search with integer programming for the zero-one multiple-choice knapsack problem with setup. In: Sifaleras A, Salhi S, Brimberg J (eds). Variable Neighborhood Search. ICVNS 2018 Lecture Notes in Computer Science, 11328, 152-166. https://doi.org/10.1007/978-3-030-15843-9_13

[23] Adouani, Y., Masmoudi, M., & Jarboui, B. (2022). Efficient Mat Heuristic for Solving the Multiple-Choice Knapsack Problem with Setup. Conference paper in 3rd Asia Pacific International Conference on Industrial Engineering and Operations Management, Johor Bahru, Malaysia. https://doi.org/10.46254/AP03.20220384.

[24] Boukhari, S., & Hifi, M. (2024). An exact algorithm for the multiple-choice Knapsack problem

with setups. *Computers & Industrial Engineering* 188, 1-14. https://doi.org/10.1016/j.cie.2023.109818
[25] Adouani, Y., Masmoudi, M., Jarray, F., Jarboui, B. (2024). Iterative integer linear programming-based heuristic for solving the multiple-choice knapsack problem with setups. *Expert Systems with Applications*, 242, 1-8. https://doi.org/10.1016/j.eswa.2023.122835
[26] J. Puchinger, GR. Raidl, and U. Pferschy. (2010). The multidimensional knapsack problem: structure and algorithms, *INFORMS J. Comput.*, vol. 22, pp. 250-265.
[27] A. Rezoug and D. Boughaci. (2016). A self-adaptive harmony search combined with a stochastic local search for the 0–1 multidimensional knapsack problem, *Int J Bio Inspired Comput.,* vol. 8(4), pp. 234–239.
[28] A. Rezoug, M. Bader-El-Den, and D. Boughaci. (2018). Guided genetic algorithm for the multidimensional knapsack problem, *Memetic Computing*, vol. 10, pp. 29-42.
[29] Gallo, G., Hammer, P.L., Simeone, B. (1980). Quadratic knapsack problems, *Math. Program. Stud*. 12: 132–149.
[30] Pisinger, D. The quadratic knapsack problem — a survey. (2007). *Discrete Appl. Math*. 155: 623–648.
[31] Caprara, A., Pisinger, D., Toth, P. (1998). Exact solution of the quadratic knapsack problem. *INFORMS J. Comput*. 11: 125 137.
[32] Fomeni, F.D., Kaparis, K., Letchford, A.N. (2022). A cut-and-branch algorithm for the Quadratic Knapsack Problem. *Discrete Optimization*. 44: 1-18. https://doi.org/10.1016/j.dispot.2020.100579.
[33] M. Dell'Amico, M. Delorme, M. Lori, S. Martello. (2019). Mathematical Models and Decomposition Methods for the Multiple Knapsack Problem, *Eur. J. Oper. Res*, vol. 274, 886–899.
[34] D. Pisinger. (1999). An Exact Algorithm for Large Multiple Knapsack Problems, *Eur. J. Oper. Res*. vol. 114, no. 3, pp. 528–541.
[35] Detti, P. (2021). A new upper bound for the multiple knapsack problem. *Comput. Oper. Res.* 129, 105210.
[36] Martello, S., Monaci, M. (2020). Algorithmic approaches to the multiple knapsack assignment problem. *Omega* 90, 102004.
[37] Cacchiani, V., Lori, M., Locvatelli, A., Martello, S.n (2022a). Knapsack problems-an overview of recentadvances. Part I: single knapsack problems. *Computers & Operations Research* 143. 105692.
[38] Cacchiani, V., Lori, M., Locvatelli, A., Martello, S. (2022b). Knapsack problems-an overview of recent advances. Part II: multiple, multidimensional, and quadratic knapsack problems. *Computers & Operations Research* 143. 105693.
[39] Stott, K. S., Huegler, Newhart, D. D., Vasko, F. J. (2022). Neither Fish nor Fowl: Historical Retrospectives of a Research-based OR/MS Group in an Integrated Steel Company, *ORMS Today*. https://doi.org/10.1287/orms.2022.03.05
[40] Martello, S., Pisinger, D., Toth, P. (1999). Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45, 414-424.
[41] Vasko, F.J., Lu, Y., Song, M.S. (2023). Solving Hard Combinatorial Optimization Problems with General Purpose Integer Programming Software: A Guide for OR Practitioners, *Inside OR*, No. 627, 13.
[42] McNally, B., Lu, Y., Shively-Ertas, E., Song, M. S., Vasko, F. J. (2021). A Simple and Effective Methodology for Generating Bounded Solutions for the Set K-Covering and Set Variable K-Covering Problems: A Guide for or Practitioners, *Review of Computer Engineering Research*, https://doi.org/10.18488/journal.76.2021.82.76.95
[43] Dellinger, A., Lu, Y., McNally, B., Song, M. S., Vasko, F. J. (2021). A Simple and Efficient Technique to Generate Bounded Solutions for the Generalized Assignment Problem: A Guide for OR Practitioners, *Research Reports on Computer Science*, https://ojs.wiserpub.com/index.php/RRCS/article/view/1039
[44] Song, M.S., Lu, Y., Rando, D., Shively-Ertas, E., Vasko, F. J. (2025)., Statistical analyses of solution methods for the multiple-choice knapsack problem with setups: Implications for OR practitioners, *Expert Systems with Applications*, 262, https://doi.org/:10.1016/eswa.2024.125622
[45] Lahyani, r., Chebil, K., Khemakhem, M., Coelho, L. C. (2019). Matheuristics for solving the multiple knapsack problem with setup. *Computers & Industrial Engineering* 129, 76-89.
[46] McNally, B. (2021). A simple sequential increasing tolerance matheuristic that generates bounded solutions for combinatorial optimization problems. Master's Thesis, Kutztown University of Pennsylvania, USA.
[47] Adouani, Y., Jarboui, B., Masmoudi, M. (2020). Efficient matheuristic for the generalized multiple knapsack problem with setup. *European Journal of Industrial Engineering* 14, 1-27.
[48] Amiri, A., Barkhi, R. (2020). A Lagrangean based solution algorithm for the multiple knapsack problem with setups. *Computers & Industrial Engineering* 153, 107089.

[49] Boukhari, S., Dahmani, I., Hifi,. M. (2022). Computational power of a hybrid algorithm for solving the multiple knapsack problem with setup. *Intelligent Computing*. Springer, Cham. 154-168.
[50] Masmoudi, M., Adouani, Y., Jarboui, B. (2024). LP relaxation and dynamic programming enhancing VNS for the multiple knapsack problem with setup. *International Transactions in Operational Research* 31, 1890-1916. DOI:10.1111/itor.13213
[51] Tukey, J. (1949). Comparing individual means in the analysis of variance, *Biometrics*, vol. 5, pp. 99-114, 1949. Available at: https://doi.org/10.2307/3001913

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

The authors equally contributed in 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 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.