Received: 3 April 2023 | Revised: 8 June 2023 | Accepted: 12 June 2023 | Published online: 14 June 2023

Journal of Computational and Cognitive Engineering
2024, Vol. 3(1) 1–7
DOI: 10.47852/bonviewJCCE3202921

RESEARCH ARTICLE

Efficiently Generating Bounded Solutions for Very Large Multiple Knapsack Assignment Problems

Francis J. Vasko[1,*], **Yun Lu**[1], **Emre Shively-Ertas**[2] **and Myung Soon Song**[1]

[1]*Department of Mathematics, Kutztown University, USA*

[2]*Computer Science Department, Kutztown University, USA*

Abstract: The multiple knapsack assignment problem (MKAP) is an interesting generalization of the multiple knapsack problem which has logistical applications in transportation and shipping. In addition to trying to insert items into knapsacks in order to maximize the profit of the items in the knapsacks, the MKAP partitions the items into classes and only items from the same class can be inserted into a knapsack. In the literature, the Gurobi integer programming software has solved MKAPs with up to 1240 variables and 120 constraints in at most 20 min on a standard PC. In this article, using a standard PC and iteratively loosening the acceptable tolerance gap for 180 MKAPs with up to 20,100 variables and 1120 constraints, we show that Gurobi can, on average, generate solutions that are guaranteed to be at most 0.17% from the optimums in 43 s. However, for very large MKAPs (over a million variables), Gurobi's performance can be significantly improved when an initial feasible solution is provided. Specifically, using from the literature, a heuristic and 42 MKAP instances with over 6 million variables and nearly 90,000 constraints, Gurobi generated solutions guaranteed to be, on average, within 0.21% of the optimums in 10 min. This is a 99% reduction in the final solution bound (gap between the best Gurobi solution and the best upper bound) compared to the approach without initial solution inputs. Hence, a major objective of this article is to demonstrate for what size MKAP instances providing Gurobi with an initial heuristic solution significantly improves performance in terms of both execution time and solution quality.

Keywords: multiple knapsack assignment problem, Gurobi integer programming software, simple sequential increasing tolerance methodology, initial feasible solution

1. Introduction

The 0-1 multiple knapsack problem has been widely studied in the operations research (OR) literature with classic works by Martello and Toth (1990) and Kellerer et al. (2004). Additional key results are discussed in Chekuri and Khanna (2005), Fukunaga (2008), Fukunaga (2011) Fukunaga and Korf (2007), Lalami et al. (2012), Samir et al. (2015), and Yamada and Takeoka (2009). Please see Dell'Amico et al. (2019), Lalonde et al. (2022), Sur et al. (2022), and Shively-Ertas et al. (2023) for recent results on the multiple knapsack problem.

For the multiple knapsack problem, the idea of partitioning the items into classes and allowing only items of the same class to be inserted into a knapsack was first introduced by Kataoka and Yamada (2014) and referred to as the multiple knapsack assignment problem (MKAP). In order to generate solutions guaranteed to be close to the optimums for MKAP with as many as several million variables and nearly 90,000 constraints, a strategy that will focus on using the Gurobi integer programming software in an iterative manner that takes advantage of an initial feasible heuristic solution (sometimes called a warm start) input to Gurobi will be discussed. Additionally, we will demonstrate that warm starting Gurobi with an initial feasible solution does not significantly improve Gurobi's performance when solving MKAPs until the problem size exceeds a million variables. In other words, when solving MKAPs available from the literature (Martello & Monaci, 2020) with less than 800,000 variables, warm starting Gurobi with an initial feasible solution does not provide a significant benefit in either solution quality or execution time.

It is common in the OR literature and in OR practice for integer programming software to be used in a "single pass" mode in which the default gap (difference between the upper bound and the best solution for a maximization problem) tolerance is not modified during the software execution. Recent research (discussed in Section 2) has documented that automatically loosening the acceptable gap when executing integer programming software can result in obtaining solutions that are guaranteed to be very close to optimum in a timely manner for a variety of binary integer programming (BIP) problems. This strategy is called the simple sequential increasing tolerance (SSIT) matheuristic. However, all previous research has used SSIT in a cold start mode (no initial feasible solution provided to the software). This article is the first to demonstrate when warm starting SSIT provides a significant benefit in solution quality and execution time. Because SSIT does not require the time commitment for algorithm development,

*Corresponding author: Francis J. Vasko, Department of Mathematics, Kutztown University, USA. Email: vasko@kutztown.edu computer code generation, and testing, it can be particularly beneficial to OR practitioners who need to solve and implement solutions to real-world problems in a cost-effective manner.

1.1. Mathematical programming formulation

We will now provide a mathematical programming formulation for the MKAP discussed in Martello and Monaci (2020). Specifically, we assume that all item weights are positive and without loss of generality that all input data are positive integers. Let $N = \{1, 2, \ldots, n\}$ denote the set of items, $M = \{1, 2, \ldots, m\}$ denote the set of knapsacks, w_j and p_j denote the weight and profit of item $j \in N$, respectively, and the capacity of knapsack $i \in M$ is c_i. However, items are divided into r mutually disjoint subsets of items S_k where $k \in K = \{1, \ldots, r\}$. For each knapsack $i \in M$ and item $j \in N$, let x_{ij} be a binary variable taking the value one if and only if item j is inserted into knapsack i. Similarly, for each knapsack $i \in M$ and class $k \in K$, let y_{ik} be a binary variable taking the value one if and only if knapsack i is assigned to class k. The MKAP mathematical formulation is

$$\max \sum_{j \in N} p_j \sum_{i \in M} x_{ij} \tag{1}$$

$$\sum_{i \in M} x_{ij} \leq 1 \qquad j \in N \tag{2}$$

$$\sum_{k \in K} y_{ik} \leq 1 \qquad i \in M \tag{3}$$

$$\sum_{j \in S_k} w_j x_{ij} \leq c_i y_{ik} \qquad i \in M,\ k \in K \tag{4}$$

$$x_{ij} \in \{0,\ 1\} \qquad i \in M,\ j \in N \tag{5}$$

$$y_{ik} \in \{0,\ 1\} \qquad i \in M,\ k \in K. \tag{6}$$

The objective function (1) maximizes the sum of the profits of the items inserted into the knapsacks. Constraints (2) require that each item is inserted into at most one knapsack, and constraints (3) require that each knapsack is assigned to at most one item class. For each knapsack i, the associated constraints (4) require that: (i) only items of the class assigned to knapsack i (if any) can be inserted into it and (ii) the capacity of the knapsack is not exceeded. This model has $mn + mr$ binary variables and $n + m + mr$ constraints.

1.2. Background and applications

We now review the existing MKAP literature. Kataoka and Yamada (2014) discuss how the MKAP is encountered in marine cargo planning to allocate ships to destinations. Zhen et al. (2018) also use the MKAP as a core problem in a different maritime shipping application involving a barge transport system. Dimitrov et al. (2017) modeled the emergency relocation of items using single trips as two special cases of the MKAP. They were able to use the distinct structures of these specific applications to develop efficient solution approaches.

Kataoka and Yamada (2014) developed a very fast constructive heuristic (referred to as KY) that works well at generating solutions for large MKAPs. They also provide an upper bound on their KY solution which allows the user to measure the quality of their solution without knowing the optimum. However, Kataoka and Yamada (2014) note that "the quality of the heuristic solution is unsatisfactory for these SMALL instances, with relative errors sometimes higher than 100%." Although, in this article, MKAP solutions that are close to the optimums for a variety of instance sizes will be discussed, OR practitioners should be aware that there are MKAP instances with as few as 50 items that are difficult to solve (Kataoka & Yamada, 2014) using integer programming software. Kataoka and Yamada (2014) graciously shared their KY computer code which we used to generate starting feasible solutions as input for Gurobi. Lalla-Ruiz and Voß (2015) used a biased random-key genetic algorithm to solve the MKAP and tested their Genetic Algorithm (GA) on MKAPs with up to 1240 variables and 120 constraints. Martello and Monaci (2020) introduced a constructive heuristic and metaheuristic refinement (referred to as MM) to generate solutions for the MKAP. Also, Martello and Monaci (2020) introduced 3660 new MKAP instances which they made available to us for test purposes.

In the next section, we will outline a strategy of inserting Kataoka and Yamada's (2014) KY heuristic as a starting solution (warm start) for Gurobi and then iteratively loosening the solution tolerance to efficiently obtain solutions guaranteed to be close to the optimums. We will also discuss the MKAP instances made available to us by Martello and Monaci (2020). We will use a subset of these instances (a total of 594 MKAPs) to demonstrate that warm starting Gurobi with the KY heuristic provides a computational advantage when solving very large (over a million variables) MKAPs.

2. A Kataoka and Yamada Initial Feasible Solution and Iterative Gurobi Methodology

After inputting the problem objective function and constraints, the most straightforward way to use Gurobi to solve an integer programming problem is to use all the default parameter settings (including the default tolerance of 0.01%) and setting a maximum execution time. We will refer to this as the base case use of Gurobi and, in this case, no feasible solution is input as a starting point (referred to as a "cold start") for the Gurobi solution process. If Gurobi terminates before the maximum execution time is reached, then the tolerance of 0.01% has been achieved. In other words, for a maximization problem the difference between the best solution generated and the final upper bound will be less than 0.01% (the answer is considered optimum). However, if Gurobi terminates because the maximum execution time was reached, the tolerance of 0.01% was not achieved. In this case, the quality of the best solution generated can be measured by comparing the final upper bound to the best solution generated and calculating the gap = 100 × (final upper bound − best solution)/(final upper bound). If the execution time is considered excessive or the gap is too large, then one alternative is to try to fine-tune some of Gurobi's 57 parameters in hopes of improving Gurobi's performance when solving this problem. The difficulty with trying to fine-tune Gurobi parameters is the large number of possible settings (over 2×10^{34} possible parameter settings). To aid in fine-tuning the Gurobi parameters, Gurobi provides an automatic parameter tuning tool which will automatically try many different parameter settings and return the most promising settings. However, for a large integer program, this function may need to execute for 12–24 h or longer and there is no guarantee that the best parameter settings found will significantly reduce execution time or the gap.

Alternatively, McNally (2021) found that for several difficult BIP problems, using all default parameter settings except for the tolerance, solutions guaranteed to be close to the optimums could be quickly generated if the gap tolerance was *automatically* loosened during the execution of the integer programming software. He called this strategy the simple sequential increasing tolerance methodology or SSIT. SSIT is considered a matheuristic because it uses math programming combined with a heuristically determined sequence of tolerances and execution times. SSIT takes advantage of the power of the general-purpose exact solution software and requires no problem-specific algorithm. In other words, when solving a BIP, the OR practitioner does not need to develop an algorithm and computer code specific to that problem. Instead, the OR practitioner simply needs to input the mathematical formulation of the problem into the integer programming software. This can be a tremendous time and cost savings. OR practitioners using the SSIT strategy with commercial integer programming software to solve industrial applications will see their models' performances automatically improve when newer versions of the general-purpose software are implemented. In contrast, if an OR practitioner develops and codes a problem-specific solution algorithm and implements it in an industrial computing platform, the only way its performance will improve is if a faster computing platform is implemented.

When SSIT is being considered for solving a BIP problem, the first step is to solve several sample problems using the integer programming software with the default tolerance (0.0001 for Gurobi) that will be used. By studying the software engine logs, the OR practitioner can decide what tolerances and execution times to use for each tolerance. If the problems solve quickly with the software, no SSIT strategy may be required. For example, suppose after some preliminary experimentation, the user decided that instead of the fixed 0.0001 tolerance, the following sequence of tolerances would be used: 0.0001 for only 1 min, if no suitable solution was obtained in 1 min, then the tolerance would be automatically loosened to 0.001 for 1 min. Again, if Gurobi did not terminate before the end of 2 min total execution time, the tolerance would be loosened to 0.005 for 4 min. If Gurobi did not terminate before the end of 6 min of total execution time, then the tolerance would be loosened to 0.01 and executed for 4 min. Only if Gurobi required the full 10 min of execution time would the final gap be greater than 1%. Additionally, if the solution time was less than 1 min, then an optimal solution was obtained. Successful applications of SSIT to solve several BIPs have been documented in the literature. For example, McNally et al. (2021) used SSIT to solve the set k-covering and set variable k-covering problems (SVKCPs), and Shively-Ertas et al. (2023) used SSIT to solve the multiple knapsack problem. For all these applications, SSIT was cold started (no initial feasible solution inputted) and typically generated solutions guaranteed to be within 0.10% of the optimums in about a minute on standard PCs. The real advantage to using SSIT with integer programming software is its ability to quickly generate solutions guaranteed to be close to the optimums. For example, McNally et al. (2021), for 65 SVKCPs, compared a SSIT strategy to running the integer programming software (CPLEX) with the 0.0001 default tolerance. The SSIT solutions were off from the solutions obtained using the default tolerance by only 0.04% (statistically insignificant). However, the SSIT execution time was only 1.2% of the required running time in the default mode—a very significant reduction in execution time from 1055 s to only 12 s.

SSIT strategies (tolerances and execution times for each tolerance) are robust in that they are tailored to the problem and user needs. An industrial application that needs to be solved quickly in a real-time production environment will use a SSIT strategy different from a production planning application that is only solved once per day. In fact, when SSIT was used to solve the very difficult multi-demand multidimensional knapsack problem (Dellinger et al., 2022), three different SSIT strategies were used based on the predicted difficulty of the problem being solved. Up to now, there have been no published examples in which SSIT was used to solve BIPs and an initial feasible solution was provided to the integer programming software. More details about SSIT can be found in McNally et al. (2021).

Having obtained the constructive heuristic (KY) from Kataoka and Yamada (2014), we try to determine when warm starting Gurobi with the KY heuristic solution combined with using a SSIT strategy would result in significantly better results (reduced gap and execution time) compared to just cold starting Gurobi with a SSIT strategy. Our results are documented in the next section.

3. Presentation of Results

3.1. Martello and Monaci's 3660 MKAP test instances

In addition to 360 small MKAP instances from the literature (Kataoka & Yamada, 2014), Martello and Monaci (2020) generated 3660 MKAP instances that are divided into 6 sets (SET2, SET3, SET4, SET5, SET6, and SET7). SET1 is omitted from our analyses because all instances are small and similar to the instances in SET7. Martello and Monaci (2020) graciously allowed us to access these 3660 MKAP instances. Specifications of these MKAP instances are summarized in Tables 1 and 2. For each combination of r, m, n, and family (correlation classes—uncorrelated, weakly correlated, and strongly correlated), there are 10 instances of each MKAP. Also, SET3 and SET6 have different rho values (0.25, 0.50, 0.75), and SET5 has different R values (100, 1000, 10,000). The rho value is one of two parameters that are used to determine the capacity of the knapsack based on the sum of the item weights—the smaller the rho value, the smaller

Table 1
MKAP data sets obtained from Martello and Monaci

Data set	Number of instances	*r* values	*m* values	*n* values
SET2	600	2, 5	10, 20	20, 40, 60, 80, 100, 200, 400, 600, 800, 1000
SET3	360	50, 100	200, 400, 800	4000, 8000
SET4	360	50, 100	200, 400, 800	4000, 8000
SET5	1080	50, 100	200, 400, 800	4000, 8000
SET6	720	50, 100	200, 400, 800	4000, 8000
SET7	540	2	10, 20	20, 40, 60

Table 2
MKAP data sets obtained from Martello and Monaci

		Maximum size of MKAP instances	
Data set	Brief description	Number of variables	Number of constraints
SET2	Increasing n values	20,100	1120
SET3	Large, uncorrelated instances with different rho values	6,480,000	88,800
SET4	Large instances of different correlation families	6,480,000	88,800
SET5	Large instances of different correlation families and different values of R	6,480,000	88,800
SET6	Large binary and uncorrelated instances with different rho values	6,480,000	88,800
SET7	New small instances	1240	120

the capacity of the knapsack. R denotes the range of w_j, i.e., the weight w_j is distributed uniformly random over the integer interval [1, R]. Extensive details of how these MKAP instances are generated are provided in Martello and Monaci (2020). As noted earlier, the number of binary variables in a MKAP is mn + mr and the number of constraints is n +m + mr. To experimentally determine the feasibility of using a Gurobi-based solution strategy to solve these MKAP instances, we decided that a proper subset of these 3660 instances would be adequate. In our analyses, we will start with the small instances first (SET7), next the medium-sized instances (SET2), and then the large instances (SET3, SET4, SET5, and SET6). For SET7, there are 54 categories with 10 instances in each category. For our experiments, for SET7, we randomly chose three instances from each category for a total of 162 (54 × 3) MKAP instances. For SET2, there are 60 categories with 10 instances in each category. For our experiments, for SET2, we randomly chose three from each category for a total of 180 (60 × 3) MKAP instances. For SET3, SET4, SET5, and SET6 which contained large MKAP instances, we only randomly selected one MKAP instance from each category for a total of 252 very large MKAP instances.

3.2. Gurobi results using 594 MKAP instances

All executions of Gurobi (9.5) were on a PC with specifications: an AMD Ryzen 7 3700x 8-Core Processor and 16 GB RAM on Windows 11 Home 64-bit and 4 software threads.

To analyze the 162 MKAP instances selected from SET7, we considered two scenarios both with Gurobi cold started. Based on some preliminary computations, we determined that there was no advantage to warm starting Gurobi for these small MKAPs so only cold starts were used. First, a base-case scenario in which Gurobi was executed for a maximum of 1200 s with all default parameter settings (tolerance = 0.0001). The second scenario was a SSIT scenario with the following tolerances and execution times: 0.001 for 60 s, 0.005 for 180 s, 0.01 for 180 s, and 0.02 for 180 s. Note that the goal here is not to obtain proven optimal solutions (within 0.01% tolerance) but to obtain tightly bounded solutions relatively quickly. The results are summarized in Tables 3 and 4. Since these are small MKAPs with at most 1240 binary variables and 120 constraints, it is not surprising that excellent results were achieved by both scenarios. These instances were also successfully solved in Martello and Monaci (2020). The Wilcoxon signed rank test shows that the difference of objective function values between the base case and the SSIT scenario is statistically significant at $\alpha = 5\%$. However, the advantage of using SSIT is that it required only 8% of the execution time of the base case, and solutions were still guaranteed to be at most 0.1% from the optimums. If time is a concern in solving a real-world MKAP application, then this SSIT strategy is very advantageous. However, if a better solution bound than, on average, 0.09% is required, then an alternative SSIT strategy that starts with a tighter tolerance can be used and still requires significantly less time than the base case.

The medium-sized MKAP instances in SET2 have at most 20,100 binary variables and 1120 constraints. For each of the two scenarios analyzed in Tables 3 and 4 (SET7), we now do two sub scenarios—warm and cold starts. We were interested in determining the benefit of warm starting Gurobi with the KY heuristic which we had obtained from Kataoka and Yamada (2014). The MKAP solutions generated by the Kataoka and Yamada (2014) constructive heuristic (KY) required negligible

Table 3
SET7 base case summary results by correlation families: Maximum values, $n = 60$, $m = 20$, $r = 2$, #variables = 1240, # constraints = 120

Correlation class	Number of instances	Time (s)	Average gap
Uncorrelated	54	248	0.04%
Weakly correlated	54	250	0.03%
Strongly correlated	54	333	0.07%
Overall	162	277	0.05%

Table 4
SET7 SSIT strategy summary results by correlation families: Maximum values, $n = 60$, $m = 20$, $r = 2$, #variables = 1240, # constraints = 120

Correlation class	Number of instances	Time (s)	Average gap
Uncorrelated	54	15	0.09%
Weakly correlated	54	21	0.09%
Strongly correlated	54	29	0.10%
Overall	162	21	0.09%

time to execute. The results for these four scenarios are summarized in Tables 5 and 6. A comparison in base cases shows negligible advantage to warm starting Gurobi for these 180 medium-sized MKAP instances. Similar results hold when comparing the SSIT scenarios. Base case results were guaranteed, on average, to be at most 0.10% (cold start) and 0.09% (warm start) from the optimums with the SSIT scenarios guaranteed, on average, to be at most 0.17% (cold start) and 0.14% (warm start) from the optimums with the SSIT scenarios requiring only about 6% of the execution time of the base cases (6.3% for the cold start and 5.7% for the warm start). Again, if time is a concern in solving a real-world MKAP application, then the SSIT strategy is very advantageous.

Up to this point, warm starting Gurobi with the Kataoka and Yamada (2014) constructive heuristic has had no significant impact on Gurobi performance. However, we will now consider the large MKAP instances that make up SET3, SET4, SET5, and SET6. The smallest instances in these data sets contain 810,000 binary variables and 14,200 constraints. The largest instances in these data sets contain 6,480,000 binary variables and 88,800 constraints. Preliminary empirical analyses indicated that using the base case scenario for these instances could be very time consuming. Specifically, the base case could require 10 times (or more) as long to execute in order to obtain results similar to SSIT. Because of the size of these instances, we increased the execution time of SSIT for each tolerance. Specifically, when using SSIT to solve these large MKAP instances, the execution times are 600, 600, 300, and 300 s for tolerances 0.001, 0.005, 0.01, and 0.02, respectively. The warm start and cold start Gurobi SSIT results for the 252 MKAP instances selected from SET3, SET4, SET5, and SET6 are summarized in Tables 7 and 8. The results are summarized by data sets, correlation families, and the number of variables in the MKAP. Both the average final gaps by class and the maximum final gap for a MKAP in the class are reported.

In all cases, the warm start runs considerably improve the Gurobi performance. It is interesting to note that for these 252 MKAP instances, the final Gurobi objective function value, on average, is only improved over the initial KY objective function value by

Table 5
SET2 cold start summary results by correlation families: Maximum values, $n = 1100$, $m = 20$, $r = 5$, #variables = 20,100, # constraints = 1120

		Cold start		Warm start	
Correlation class	Number of instances	Time (s)	Gap*	Time (s)	Gap*
Uncorrelated	60	591	0.07%	36	0.15%
Weakly correlated	60	780	0.10%	38	0.16%
Strongly correlated	60	690	0.13%	56	0.21%
Overall	180	680	0.10%	43	0.17%

Table 6
SET2 warm start summary results by correlation families: Maximum values, $n = 1100$, $m = 20$, $r = 5$, #variables = 20,100, # constraints = 1120

		Cold start		Warm start	
Correlation class	Number of instances	Time (s)	Gap*	Time (s)	Gap*
Uncorrelated	60	566	0.07%	29	0.13%
Weakly correlated	60	707	0.07%	30	0.13%
Strongly correlated	60	682	0.13%	53	0.18%
Overall	180	652	0.09%	37	0.14%

*These are average over all MKAPs in the correlation class

Table 7
SET3, SET4, SET5, and SET6 SSIT cold start summary results: Maximum values, $n = 8000$, $m = 800$, $r = 100$, #variables = 6,480,000, # constraints = 88,800

SET	# instances	Time(s)	Average gap	Maximum gap
SET3	36	1554	9.59%	129%
SET4	36	1498	6.29%	70%
SET5	108	1470	4.39%	83%
SET6	72	1451	6.71%	128%
Correlation family				
Uncorrelated	120	1542	7.50%	129.0%
Weakly correlated	48	1457	3.73%	41.0%
Strongly correlated	48	1421	4.74%	46.0%
Binary	36	1388	6.17%	128.0%
#variables				
0.8 million to 3 million	126	1418	1.44%	4.7%
More than 3 million to 6 million	84	1491	1.83%	5.4%
More than 6 million	42	1649	28.43%	129.0%
SETS 3–6	252	1481	6.07%	129.0%

Table 8
SET3, SET4, SET5, and SET6 SSIT warm start summary results: Maximum values, $n = 8000$, $m = 800$, $r = 100$, #variables = 6,480,000, # constraints = 88,800

SET	# instances	Time(s)	Average gap	Maximum gap
SET3	36	621	0.35%	2.6%
SET4	36	644	0.29%	1.1%
SET5	108	609	0.28%	1.3%
SET6	72	786	0.75%	4.8%
Correlation family				
Uncorrelated	120	589	0.30%	3.0%
Weakly correlated	48	492	0.21%	1.2%
Strongly correlated	48	795	0.40%	1.3%
Binary	36	1002	1.18%	4.8%
#Variables				
0.8 million to 3 million	126	617	0.38%	4.7%
More than 3 million to 6 million	84	773	0.61%	4.8%
More than 6 million	42	600	0.21%	2.7%
SETS 3–6	252	667	0.43%	4.8%

Table 9
SSIT summary improvement results by number of variables: Maximum values, $n = 8000$, $m = 800$, $r = 100$, #variables = 6,480,000, # constraints = 88,800

Number of variables	Number of instances	Gap reduction	Execution time reduction
0.8 million to 3 million	126	43%	56%
More than 3 million to 6 million	84	67%	48%
More than 6 million	42	99%	64%

0.07% with a maximum improvement of 5%. Hence, the high quality of the initial KY solution input to Gurobi allows it to efficiently search the solution space. Among all sets 3–6, SET3 demonstrated the most improvement from using the warm start. Specifically, the execution time was reduced by 60% and the gap was reduced by 96%. Among all correlation families, the weakly correlated family demonstrated the most improvement from using the warm start in terms of execution time reduction (66%), but the uncorrelated family demonstrated the most improvement from using the warm start in terms of gap reduction (96%). The maximum SSIT execution time was 1800 s and only 12 warm start instances required the full 1800 s. However, there were 79 cold start instances that required the full 1800 s. The largest final gaps were 4.8% for the warm start case and 129% for the cold start case.

In Table 9, the gap reductions and execution time reductions when warm starts are used instead of cold starts are summarized based on MKAP instance size. The most dramatic warm start improvement in Gurobi performance is when the number of variables exceed 6 million. In this case, warm starting Gurobi with the SSIT strategy obtained solutions that were guaranteed, on average, to be within 0.21% of the optimums in 10 min on a standard PC. This was an average gap reduction of 99% with an execution time reduction of 64%.

4. Conclusion and Future Work

Previously in the literature (Martello & Monaci, 2020), Gurobi had been used with no initial feasible solution (cold start) and all default parameter values to solve MKAPs with up to 1240 binary variables and 120 constraints. In this article, by solving 594 MKAP instances from the literature, we determined *when* warm starting Gurobi with a feasible solution generated by a constructive heuristic of Kataoka and Yamada (2014) provided a significant computational and solution quality advantage compared to cold starting Gurobi. Additionally, we demonstrated that using a strategy (SSIT) that automatically iteratively loosens the Gurobi tolerance parameter, bounded solutions for MKAPs could be generated quickly. Moreover, when using Gurobi to solve very large MKAP instances (over a million variables), the substantial benefit of combining the SSIT strategy with inputting a feasible solution generated by a constructive heuristic of Kataoka and Yamada (2014) is clearly established. Specifically, solutions for 42 MKAPs from the literature (Martello & Monaci, 2020) with over 6 million binary variables and nearly 90,000 constraints were generated, on average, in 10 min on a standard PC and the solutions were guaranteed, on average, to be at most 0.21% from the optimums. This is a 99% solution quality improvement compared to using Gurobi with cold start.

Furthermore, it is interesting to note that when solving MKAP instances from the literature with up to 20,100 variables (the next largest MKAP instances in the literature have 810,000 variables), inputting an initial feasible solution to Gurobi provided no computational advantage over just cold starting Gurobi. An interesting future research topic would be to determine if there is an advantage to warm starting Gurobi when trying to solve MKAPs with between 20,100 and 810,000 binary variables. However, the first step would be to appropriately define these MKAPs.

Although it has been demonstrated that Gurobi with the SSIT strategy can be used successfully to quickly generate bounded solutions for a variety of MKAP instance sizes, as stated earlier, OR practitioners should be aware that there are small MKAP instances (50 variables, for example) that can be difficult to solve. Additionally, although not currently documented in the OR literature, the authors are aware of very hard 0-1 knapsack problems with as few as 400 variables that even after 6 h of Gurobi execution time on a standard PC still have a gap of about 3%. The authors are currently exploring if Gurobi can be used to generate bounded solutions (guaranteed within 1% of the optimum) in a timely manner for such difficult problems. Moreover, the authors are actively trying to characterize when a 0-1 knapsack problem is difficult to solve.

Finally, the real motivation for this work was to demonstrate how OR practitioners can effectively and efficiently use commercial integer programming software, like Gurobi, in a relatively straightforward manner to solve industrial size problems. Specifically, since SSIT does not require the time commitment for algorithm development, computer code generation, and testing, it can be particularly beneficial to OR practitioners who need to solve and implement solutions to real-world problems in a cost-effective manner.

Acknowledgements

We are sincerely grateful to Seiji Kataoka and Takeo Yamada for providing us with their C-code for their constructive heuristic (referred to as KY in our article) for the MKAP. We are also very grateful to Silvano Martello and Michelle Monaci for allowing us to access the MKAP instances discussed in their 2020 *Omega* article. Additionally, we are grateful to Silvano Martello and Michelle Monaci for sharing the MKAP solutions from the algorithms discussed in their article.

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

References

Samir, B. A. L. B. A. L., Yacine, L., & Mohamed, B. (2015). Local search heuristic for multiple knapsack problem. *International Journal of Intelligent Information Systems*, *4*(2), 35–39.

Chekuri, C., & Khanna, S. (2005). A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, *35*(3), 713–728. https://doi.org/10.1137/S0097539700382820

Dell'Amico, M., Delorme, M., Iori, M., & Martello, S. (2019). Mathematical models and decomposition methods for the multiple knapsack problem. *European Journal of Operational Research*, *274*(3), 886–899. https://doi.org/10.1016/j.ejor.2018.10.043

Dellinger, A., Lu, Y., Song, M. S., & Vasko, F. J. (2022). Generating bounded solutions for multi-demand multidimensional knapsack problems: A guide for operations research practitioners. *International Journal of Industrial Optimization*, *3*(1), 1–17. https://doi.org/10.12928/ijio.v3i1.5073

Dimitrov, N. B., Solow, D., Szmerekovsky, J., & Guo, J. (2017). Emergency relocation of items using single trips: Special cases of the multiple knapsack assignment problem. *European Journal of Operational Research*, *258*(3), 938–942. https://doi.org/10.1016/j.ejor.2016.09.004

Fukunaga, A. S. (2008). Integrating symmetry, dominance, and bound-and-bound in a multiple knapsack solver. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 5th International Conference*, 5, 82–96. https://doi.org/10.1007/978-3-540-68155-7_9

Fukunaga, A. S. (2011). A branch-and-bound algorithm for hard multiple knapsack problems. *Annals of Operations Research*, *184*(1), 97–119. https://doi.org/10.1007/s10479-009-0660-y

Fukunaga, A. S., & Korf, R. E. (2007). Bin completion algorithms for multicontainer packing, knapsack, and covering problems. *Journal of Artificial Intelligence Research*, *28*, 393–429. https://doi.org/10.1613/jair.2106

Kataoka, S., & Yamada, T. (2014). Upper and lower bounding procedures for the multiple knapsack assignment problem. *European Journal of Operational Research*, *237*(2), 440–447. https://doi.org/10.1016/j.ejor.2014.02.014

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems*. Germany: Springer.

Lalami, M. E., Elkihel, M., El Baz, D., & Boyer, V. (2012). A procedure-based heuristic for 0-1 multiple knapsack problems. *International Journal of Mathematics in Operational Research, 4*(3), 214–224. https://doi.org/10.1504/IJMOR.2012.046684

Lalla-Ruiz, E., & Voß, S. (2015). A biased random-key genetic algorithm for the multiple knapsack assignment problem. In *Learning and Intelligent Optimization: 9th International Conference*, 218–222. https://doi.org/10.1007/978-3-319-19084-6_19

Lalonde, O., Cote, J. F., & Gendron, B. (2022). A branch-and-price algorithm for the multiple knapsack problem. *INFORMS Journal on Computing*, *34*(6), 3134–3150. https://doi.org/10.1287/ijoc.2022.1223

Martello, S., & Monaci, M. (2020). Algorithmic approaches to the multiple knapsack assignment problem. *Omega*, *90*, 102004. https://doi.org/10.1016/j.omega.2018.11.013

Martello, S., & Toth, P. (1990). *Knapsack problems: Algorithms and computer implementations*. USA: John Wiley & Sons.

McNally, B. (2021). *A simple sequential increasing tolerance matheuristic that generates bounded solutions for combinatorial optimization problems*. Master's Thesis, Kutztown University of Pennsylvania.

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*, *8*(2), 76–95. https://doi.org/10.18488/journal.76.2021.82.76.95

Shively-Ertas, E., Lu, Y., Song, M., & Vasko, F. (2023). Using general-purpose integer programming software to generate bounded solutions for the multiple knapsack problem: A guide for or practitioners. *International Journal of Industrial Optimization*, *4*(1), 16–24. https://doi.org/10.12928/ijio.v4i1.6446

Sur, G., Ryu, S. Y., Kim, J., & Lim, H. (2022). A deep reinforcement learning-based scheme for solving multiple Knapsack problems. *Applied Sciences*, *12*(6), 3068. https://doi.org/10.3390/app12063068

Yamada, T., & Takeoka, T. (2009). An exact algorithm for the fixed-charge multiple knapsack problem. *European Journal of Operational Research*, *192*(2), 700–705. https://doi.org/10.1016/j.ejor.2007.10.024

Zhen, L., Wang, K., Wang, S., & Qu, X. (2018). Tug scheduling for hinterland barge transport: A branch-and-price approach. *European Journal of Operational Research*, *265*(1), 119–132. https://doi.org/10.1016/j.ejor.2017.07.063

How to Cite: Vasko, F. J., Lu, Y., Shively-Ertas, E., & Song, M. S. (2024). Efficiently Generating Bounded Solutions for Very Large Multiple Knapsack Assignment Problems. *Journal of Computational and Cognitive Engineering, 3*(1), 1–7. https://doi.org/10.47852/bonviewJCCE3202921