UAD Universitas Ahmad Dahlan

IJIO
(*International Journal of Industrial Optimization*)
ISSN 2723-3022 (online) | 2714-6006 (printed)
VOL 4, No. 1, 16-24
https://doi.org/10.12928/ijio.v4i1.6446

Using general-purpose integer programming software to generate bounded solutions for the multiple knapsack problem: a guide for or practitioners

Emre Shively-Ertas [1], Yun Lu [2], Myung Soon Song [2], Francis J. Vasko [2,*]
[1] Department of Computer Science and Information Technology, Kutztown University, USA
[2] Department of Mathematics, Kutztown University, USA
*Corresponding Author: vasko@kutztown.edu

ARTICLE INFO

ABSTRACT

Article history
Received: August 10, 2022
Revised: September 13, 2022
Accepted: November 1, 2022

Keywords
Bounded Solutions;
Integer Programming;
Multiple Knapsack Problem;
Simple Sequential Increasing;
Tolerance (SSIT).

An NP-Hard combinatorial optimization problem that has significant industrial applications is the Multiple Knapsack Problem. If approximate solution approaches are used to solve the Multiple Knapsack Problem there are no guarantees on solution quality and exact solution approaches can be intricate and challenging to implement. This article demonstrates the iterative use of general-purpose integer programming software (Gurobi) to generate solutions for test problems that are available in the literature. Using the software package Gurobi on a standard PC, we generate in a relatively straightforward manner solutions to these problems in an average of less than a minute that are guaranteed to be within 0.16% of the optimum. This algorithm, called the Simple Sequential Increasing Tolerance (SSIT) algorithm, iteratively increases tolerances in Gurobi to generate a solution that is guaranteed to be close to the optimum in a short time. This solution strategy generates bounded solutions in a timely manner *without* requiring the coding of a problem-specific algorithm. This approach is attractive to management for solving industrial problems because it is both cost and time effective and guarantees the quality of the generated solutions. Finally, comparing SSIT results for 480 large multiple knapsack problem instances to results using published multiple knapsack problem algorithms demonstrates that SSIT outperforms these specialized algorithms.

INTRODUCTION

We will use the description of the Multiple Knapsack Problem given in [1]. Specifically, given a set of m containers (knapsacks) with capacity c_i ($i = 1,\ldots,m$) and a set of n objects (items) with profit p_j and weight w_j ($j = 1,\ldots,n$), the Multiple Knapsack Problem requires that items are to be inserted into knapsacks such that the total weight of the items in a knapsack does not exceed its capacity, and the overall profit of the selected items is a maximum. In the Operations Research (OR) literature both the multidimensional knapsack problem (MKP) which involves only one knapsack but multiple dimensional constraints and the multiple knapsack problem which is the focus of this article and involves multiple knapsacks are typically abbreviated as MKP. However, in this article we will abbreviate the multiple knapsack problem as MKSP to indicate that there are several knapsacks. In other words, MKSP will be used throughout this article to stand for the multiple knapsack problem which requires that items must be inserted into multiple knapsacks such that the knapsack capacities are not exceeded and that the total profit of the inserted items is maximized.

The MKSP has important and varied applications in business and industry. A few examples of applications that can be formulated as MKSPs include: Ketyko et al. [2] formulated a multi-user computation offloading problem as a MKSP. To maximize the profit of user equipment, it considers user equipment as items, requested CPU as weights, and CPU capacities of mobile edge computing server as capacities of knapsacks. Labbe et al. [3] formulated a multiprocessor scheduling problem as a MKSP, and Cappanera et al. formulated the placement of Virtual Network Functions as an MKSP by [4]. Also, there are many cutting stock problems that can be formulated as MKSPs. An example from the steel industry is given in [5]. Specifically, cutting customer orders for structural products from long steel beams can be formulated as a MKSP where the long steel beams are the knapsacks.

When OR practitioners are faced with the need to solve and implement solutions to industrial or business applications that can be formulated as MKSPs, the expense in terms of the personnel time commitment on algorithm development, computer code generation and testing, and solution implementation can be significant and costly. Because numerous industrial applications can be mathematically formulated as MKSPs, OR practitioners need simple, effective, and efficient solution approaches to solve these problems and to implement the solutions in a cost-effective manner. Additionally, an important management goal is that the solutions generated are guaranteed to be close to the optimums.

The purpose of this article is to present a viable alternative for OR practitioners that need to solve MKSPs. Specifically, the main contribution of this article is to demonstrate a strategy for solving MKSPs that does not require problem-specific logic and computer coding but executes quickly and provides solutions that are guaranteed to be close to the optimums thus making it very attractive for industrial use. Instead of using either approximate or exact solution approaches that require algorithm-specific computer coding for the MKSP, the major benefit of this approach is that it iteratively makes use of general-purpose integer programming software. MKSP solutions that are guaranteed to be very close to the optimums will be quickly generated using this procedure, called the Simple Sequential Increasing Tolerance (SSIT) algorithm. In fact, when compared to published algorithms specifically designed to solve MKSPs, SSIT will outperform these methods when tested on 480 large and difficult MKSP instances.

Statistical analyses will demonstrate that this iterative approach provides the same high-quality solutions compared to when Gurobi is executed in a default mode but SSIT requires much less execution time.

1. Mathematical Formulation

We will use the formulation given in [6]. Let the binary decision variable x_{ij} be 1 if item j is placed in container i, and 0 otherwise. Note p_j, w_i and c_i are defined in Section 1. Then the 0-1 MKSP can be formulated as the integer program below, where constraint (2) encodes the capacity constraint for each container, and constraint (3) ensures that each item is assigned to at most one container. In other words, the goal of an MKSP is to insert items into knapsacks such that the total profit of the items inserted is maximized.

$$\text{Maximize } \Sigma_i \ \Sigma_j \ p_j \ x_{ij} \tag{1}$$

$$\text{Subject to } \Sigma_i \ w_j \ x_{ij} \ \leq c_i \ \ i = 1, \dots., m \tag{2}$$

$$\Sigma_i \ x_{ij} \leq 1 \ \ j = 1, \dots, n \tag{3}$$

$$x_{ij} \in \{0,1\} \ for \ all \ i, j \tag{4}$$

2. Overview of Existing Work

The existing literature on the MKSP is extensive. This includes two chapters in the two books [7] and [8]. Below we will briefly mention some of the most recent articles and suggest the reader consult [1] and [9] for more details.

A classic exact algorithm for the MKSP is Martello and Toth's [7] branch-and-bound algorithm MTM which includes surrogate relaxation and lower bounds. Another classic branch-and-bound based algorithm is due to Pisinger [10] called MULKNAP which employs surrogate relaxation for upper bounds, and the subset-sum problem for lower bounds. Martello and Toth [7] proposed MTHM which is composed of a greedy mapping, rearrangement using reordering, swapping items between two knapsacks, and replacing one item with a subset of unassigned items. Dell'Amico et al. [1] developed a hybrid exact algorithm which combines MULKNAP with decomposition methods.

Fukunaga [11] and Fukunaga and Tazoe [12] presented several genetic algorithm (GA) approaches and tested them on instances with up to 300 items and 100 knapsacks. They obtained consistent improvement in terms of solution quality with respect to MTHM, although at the expense of much larger execution times. Sur et al. [9] used a deep reinforcement learning-based scheme in which the deep neural networks (DNN) model is extensively trained with various combinations of random items and knapsacks. The trained DNN model has the capability of solving diversified MKSPs with untrained instance sets of items and knapsacks.

It is important to note that all of the solution methods mentioned above require problem-specific logic and computer coding. In contrast, this article will demonstrate that SSIT can quickly provide solutions that are guaranteed to be close to the optimums and does not require problem-specific logic and computer coding. The rest of this article is organized as follows. In Section 2 we give an overview of the simple sequential increasing tolerance method. In Section 3 we analyze the results of using SSIT to solve MKSPs available in the literature. Finally, Section 4 summarizes our results and suggests future work.

METHOD

Trying to have the best of two worlds (a method that is both fast and guarantees solution quality) is the motivation [13] behind the SSIT algorithm. More benefits of SSIT are detailed in [19]. Successful applications of SSIT to solve several binary integer programs (BIP) have been documented in the literature [14-19]. For these applications, SSIT typically generates solutions guaranteed to be within 0.1% of the optimums in about 60 seconds on standard PCs.

Although the SSIT applications mentioned above have been successful at quickly generating solutions guaranteed to be close to the optimums for a number of generalizations of the classic knapsack problem, it does not follow that SSIT will necessarily perform well on all knapsack generalizations. For example, preliminary analyses by the authors have indicated that SSIT performs poorly for large instances of the multiple knapsack assignment problem. Hence, one cannot assume that SSIT works well for all generalizations of the knapsack problem.

SSIT can be considered a multi-pass algorithm in which the program terminates if the goal tolerance is met. If it is not met, then the tolerance is "loosened" and the current best solution is used as input for the next step in the solution process.

The pseudo code below summarizes the SSIT algorithm for a generic COP. More details about SSIT can be found in McNally et al. [14].

SSIT Algorithm [14]

```
1.  Begin
2.  Input the number of phases $N$
3.  Input tolerance $T\_i$ and maximum execution time $t\_j$ for phases $i = 1,...,N$
4.  Input COP details
5.  Run integer programming software program to solve COP
6.  For $1 \le i \le N-1$
7.  IF integer programming software running time in phase $i$ is less than $t\_i$ or $i = N$, FINISH
```

```
8.  ELSE
9.  Take best solution obtained from phase $i$ and save it as $SOL\_i$ SOL_i
10. Run integer programming software program with $SOL\_j$ as the warm start and tolerance
    $T\_\{i+1\}$ and maximum execution time $t\_\{i+1\}$
11. $i = i + 1$
12. LOOP through step 7-11 until FINISH
```

It is important to note that there is no need to “optimize” either the number of tolerances used or their values as well as the execution times for each tolerance. These values are both user and problem specific and can be easily adjusted to meet the users’ needs.

RESULTS AND DISCUSSION

To test how well SSIT could in a timely manner generate solutions for MKSPs guaranteed to be close to the optimums, we made use of 342 of the 2100 MKSP instances discussed in [1]. To our knowledge these are the most comprehensive and largest MKSP instances available to researchers. Dell’Amico et al. [1] used these 2100 MKSP instances to test the performance of the algorithms discussed in [1]. In particular, Dell’Amico et al. [1] were testing the performance of their exact method Hybrid MKP (Hy-MKP). This method involves the following components: instance reduction, capacity lifting, item dominance, knapsack-based decomposition, and reflect-model decomposition. Hy-MKP provided the best solutions compared to the other algorithms tested by Dell’Amico et al. [1] on these 2100 MKSP instances. Please see [1] for more details. These 2100 MKSP instances are contained in five data sets: called SMALL (180 instances), FK_1, FK_2, FK_3, and FK_4 each containing 480 instances. SMALL contains 18 subsets of 10 instances each and each FK data set contains 24 subsets of 20 instances each. Summary information for these five data sets is given in Table 1.

Table 1. Characteristics of the 5 sets of instances

Set	n/m	Item weights	Correlation between profits and weights
SMALL	{20/10, 40/10, 60/10, 20/20, 40/20, 60/20}	[1-1000]	Uncorrelated, weakly, strongly
FK_1	{60/30, 45/15, 48/12, 75/15, 60/10, 100/10}	[10-1000]	Uncorrelated, weakly, strongly, subset-sum
FK_2	{120/60, 90/30, 96/24, 150/340, 120/20, 200/20}	[10-1000]	Uncorrelated, weakly, strongly, subset-sum
FK_3	{180/90, 135/45, 144/36, 225/45, 180/30, 300/30}	[10-1000]	Uncorrelated, weakly, strongly, subset-sum
FK_4	{300/150, 225/75, 240/60, 375/75, 300/50, 500/50}	[10-1000]	Uncorrelated, weakly, strongly, subset-sum

In order to determine if SSIT could be used effectively to quickly generate bounded solutions for MKSPs, we did not feel the need to solve all 2100 instances. Instead from each of the 18 sets in SMALL that contained 10 instances each, we randomly chose 3 for test purposes. In each of the 24 sets in FK_1 to FK_4, that contained 20 instances each, we randomly chose 3 for test purposes. This gives a total of 18x3 + 4x24x3 = 342 MKSPs for empirical testing. So, we have 3 instances from each of the 114 categories that are composed of 6 different n/m ratios (2, 3, 4, 5, 6, and 10) and either 3 or 4 correlation classes. For more details on how these instances were generated plus details on the correlation classes, we recommend the reader consult [1]. The largest instances are in FK_4 with 300 items and 150 knapsacks for a total of 45,000 binary variables.

1. SSIT Results for the 342 MKSP Instances

In order to use SSIT to solve MKSPs, a sequence of increasing tolerances and corresponding maximum execution times must be specified for Gurobi. Based on limited empirical experimentation, the authors decided to try two different SSIT sequences both with total maximum execution time of 600 seconds. The first SSIT scenario (SSIT1) favored faster execution times with the following four tolerances and maximum times per tolerance: 0.001 for 60 seconds, 0.005 for 180 seconds, 0.01 for 180 seconds, and 0.02 for 180 seconds. SSIT2 favored tighter bounds on the solutions with the following four tolerances and maximum times per tolerance: 0.0001 for 60 seconds, 0.0005 for 180 seconds, 0.001 for 180 seconds, and 0.005 for 180 seconds. Although the maximum possible execution time is 600 seconds, the actual average execution time was 38.5 seconds for SSIT1 and 192.1 seconds for SSIT2. These tolerances and execution times are very reasonable for most industrial applications**.** However, the user can adjust the number of tolerances as well as their values. Also, the user determines the maximum execution times at each tolerance and they may be different by tolerance. To demonstrate the benefit of using SSIT instead of just executing Gurobi at the default tolerance of 0.0001, these 342 MKSPs were solved with Gurobi for a maximum of 1200 seconds at the default tolerance of 0.0001 (referred to as the BASE CASE).

In addition to the tolerances and execution times as specified above, the execution was restricted to four software threads. All other software parameters had their default settings. The computer used for all executions of Gurobi had the following specifications: an AMD Ryzen 7 3700X 8-Core Processor and 16 GB RAM on Windows 11 Home 64-bit. The results of executing SSIT to solve the 342 MKSP instances are summarized in Tables 2.

Table 2. Average performance of Gurobi on MKSP

Problems	**Base Case**		**SSIT1**		**SSIT2**	
	Time(s)	**Gap(%)**	**Time(s)**	**Gap(%)**	**Time(s)**	**Gap(%)**
90 uncorrelated	712.5	0.112%	44.8	0.184%	245.2	0.132%
90 weakly correlated	763.5	0.142%	64.4	0.232%	263.9	0.166%
90 strongly correlated	813.3	0.060%	32.7	0.121%	187.8	0.073%
72 subset-sum	183.5	0.011%	4.9	0.074%	40.4	0.013%
342 total	642.2	0.085%	38.5	0.157%	192.1	0.100%

In Table 2, the 342 MKSPs were partitioned by correlation class. Historically [20], when the weights and profits are strongly correlated the execution time increases. It is important to note that the Gap (%) column gives the *farthest* away the solutions can be without knowing the exact value of the optimums. Except for the SSIT1 results for the 90 weakly correlated MKSPs (0.232%), all deviations from the optimums were under 0.2**%.** In all three cases, Gurobi solved the subset-sum instances much quicker and with very tight bounds compared to the other correlation classes. In fact, for the 72 subset-sum instances, in the BASE CASE, Gurobi found the proven optimums (within 0.01% tolerance) for 64 instances. Simply executing Gurobi (BASE CASE) for up to 1200 seconds at the default tolerance of 0.0001 resulted in an average execution time of 642.2 seconds with a maximum deviation from the optimum of 0.085%. However, the maximum deviations for SSIT1 and SSIT2 are very close at 0.157% and 0.100%, but only require 6% and 30% of the execution time of the BASE CASE respectively. In Section 3.4, statistical analyses will demonstrate, regardless of correlation class, that there is no statistically significant difference between the BASE CASE results and the SSIT2 results. Thus, SSIT2 provides the same quality results with only 30% of the execution time. The fact that both the solutions are guaranteed to be very close to the optimums and the short execution times makes the SSIT algorithm very advantageous for solving industrial problems.

Table 3 show the distributions of the tolerances at which SSIT1 and SSIT2 terminate. From Table 3 we see that for SSIT1 97.4% of the 342 MKSPs terminated with a tolerance less than 0.005. For SSIT2 99.1% of the 342 MKSPs terminated with a tolerance less than 0.005. Hence, the vast majority of the 342 MKSPs have solutions very close to the optimums.

Table 3. Tolerance termination of Gurobi on MKSP

Problems			**SSIT1 Tolerances/time**			
			0.001 60s	**0.005 180s**	**0.01 180s**	**0.02 180s**
90 uncorrelated			37	51	2	0
90 weakly correlated			33	51	6	0
90 strongly correlated			56	33	0	1
72 subset-sum			71	1	0	0
342 total			197	136	8	1
Percentage			57.6%	39.8%	2.3%	0.3%
			SSIT2 Tolerances/time			
Problems	**0.0001 60s**	**0.0005 180s**	**0.001 180s**	**0.005 180s**	**Greater than 0.005**	
90 uncorrelated	35	3	3	48	1	
90 weakly correlated	29	4	6	50	1	
90 strongly correlated	28	15	28	18	1	
72 subset-sum	54	14	4	0	0	
342 total	146	36	41	116	3	
Percentage	42.7%	10.5%	12.0%	33.9%	0.9%	

In Subsection 3, statistical analyses will be used to show the relationship among the BASE CASE, SSIT1, and SSIT2 results based on objective function values.

2. MKSP SSIT results compared to other metaheuristics

As OR practitioners, the authors appreciate the guaranteed bounds (0.16% averaged over 342 MKSP instances for SSIT1), the short execution times, and the ease of implementation that requires no algorithm specific computer coding that are associated with SSIT. However, there may be readers that are interested in seeing how the SSIT solutions obtained in the last section compare to MKSP solution methods published in the literature. To achieve this comparison, we decided to use SSIT1 and SSIT2 to solve all 480 MKSP instances in FK_4 (previously only 72 FK_4 instances were solved by SSIT1 and SSIT2). There was no need to include the BASE CASE because SSIT2 has been shown to generate similar solutions in only 30% of the time. FK_4 was chosen because Dell'Amico et al. [1] state that the instances in FK_4 appear to be "much more difficult" than the instances in SMALL, FK_1, FK_2, and FK_3. In Table 6 of Dell' Amico et al. [1], they report results for the six most competitive methods for FK_4 instances. However, three of the methods, Arc-flow model, Reflect model+priority, and Reflect-based decomposition, were not included in our comparison because all three of these methods were unable to solve 80 FK_4 instances in the maximum allowed time of 1200 seconds. The three methods that we did compare SSIT1 and SSIT2 to were Pisinger's MULKNAP, the Classical model, and the hybrid Hy-MKP model because these methods were able to solve all 480 FK_4 instances. Details of these methods are given in [1]. Table 4 summarizes the performance of SSIT1, SSIT2 and these three state-of-the-art methods specifically designed to solve MKSPs.

Table 4. Comparison of SSIT1 and SSIT2 with competitive MKSP solution methods averaged over all 480 FK_4 instances

Solution method	**Gap (%)**	**Comparable Time (s)*****
SSIT1*	0.236%	60.2
SSIT2*	0.161%	264.2
Pisinger's MULKNAP**	0.285%	354.2
Classical method**	0.388%	751.5
Hy-MKP**	8.193%	215.2

*Computer specifications: All executions of Gurobi were on a computer with the following specifications: an AMD Ryzen 7 3700X 8-Core Processor and 16 GB RAM on Windows 10 Home 64-bit.
**Computer specifications: Intel Xeon E5530, 2.4 GHz with 24 GB of memory, running under a Linus Ubuntu 14.04 LTS 64-bit, using a single core.
***to make the PC execution times approximately comparable, the original Dell'Amico et al. [1] times were multiplied by 0.75.

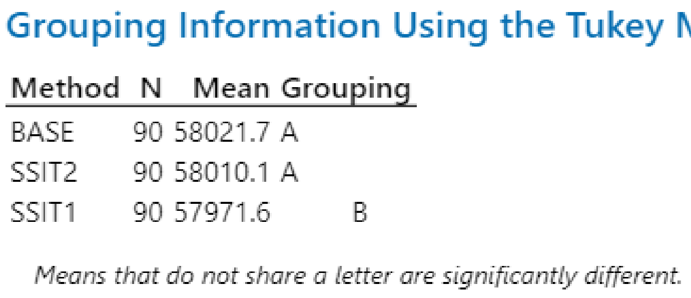
Table 4 shows that both SSIT1 and SSIT2 outperformed these three solution methods specifically designed to solve MKSPs. Although it is difficult to give a good comparable time estimate, given how many factors can affect the speed of these computations, we believe a conservative estimate is achieved by multiplying Dell' Amico et al. [1] times by 0.75. In this case, only Hy-MKP is slightly faster than SSIT2 (SSIT1 is by far the fastest), but Hy-MKP has a much larger gap.

3. Statistical Analyses

In this section, the 342 MKSP instances (90 uncorrelated instances, 90 weakly correlated' instances, 90 strongly correlated instances, and 72 submit-sum instances) are used for statistical analyses. The two SSIT methods (SSIT1 and SSIT2) are compared to the BASE CASE method in terms of the objective function.

Since significant differences among the three methods are detected from the repeated measures ANOVA model, Tukey's pairwise comparison is conducted for each correlation class as a follow-up test. (Tukey [21]) In every correlation class, there is no statistically significant difference between the BASE CASE and SSIT2, but there is a statistically significant difference between the BASE CASE (or SSIT2) and SSIT1. This means SSIT1 objective function values are statistically less than the BASE CASE (or SSIT2) objective function values in average with the 99% confidence.

Figure 1 illustrates the summary of Tukey's comparison in the uncorrelated class. Note that the objective function is getting smaller in order of A and B, and methods that do not share a letter are significantly different. All other three correlation classes (weakly correlated, strongly correlated, and submit-sum) show similar results.



Grouping Information Using the Tukey Method and 99% Confidence

Method	N	Mean	Grouping	
BASE	90	58021.7	A	
SSIT2	90	58010.1	A	
SSIT1	90	57971.6		B

Means that do not share a letter are significantly different.

Figure 1. Comparison of the three methods (BASE CASE, SSIT1, and SSIT2)

CONCLUSION

The primary contribution of this article is it demonstrated that the SSIT algorithm that iteratively uses commercial integer programming software with no algorithm-specific code effectively solved a total of 750 (480 + 342 – 72) MKSP instances from the literature. The solutions generated by SSIT are guaranteed to be within tight bounds of the optimums. Furthermore, the particular application and users' needs determine what SSIT strategy to use. To illustrate, in this article two SSIT strategies were defined—one that favored shorter execution times (SSIT1) and one that favored tighter bounds on the solution (SSIT2). For the 342 MKSP test instances initially discussed, on average, SSIT1 generated solutions within 0.16% of the optimums in 39 seconds on a standard PC and SSIT2 generated solutions within 0.10% of the optimums in 192 seconds on the same standard PC. The user would need to determine which strategy was more appropriate for the application at hand. Additionally, in order to make a fair comparison among SSIT1and SSIT2 and three MKSP solution methods in the literature, all 480 of the difficult FK_4 MKSP instances were solved by both SSIT1 and SSIT2 (72 had been solved initially as part of the 342 instances). Both SSIT1 and SSIT2 outperformed these solution methods specifically designed to solve the MKSP.

OR practitioners can quickly develop integer programming models using default software parameter values and templates with no need for problem-specific algorithms. An added benefit is that newer versions of the software will "automatically" improve the performance of the application.

The authors believe that significant evidence has been presented to indicate that SSIT provides a viable alternative both in terms of solution quality and execution time required for OR practitioners that need to solve MKSPs for an industrial application. Based on this successful application of SSIT and other SSIT successes reported in the literature, the authors plan to test the performance of SSIT for solving other difficult-to-solve combinatorial optimization problems to provide additional evidence of when SSIT should be used to solve COP. However, as will be illustrated in future articles, there are COP that SSIT has difficulty solving.

REFERENCES

[1] M. Dell'Amico, M. Delorme, M. Lori, S. Martello, "Mathematical Models and Decomposition Methods for the Multiple Knapsack Problem", Eur. J. Oper. Res, vol. 274, 886–899, 2019.

[2] I. Ketykó, L. Kecskés, C. Nemes, C. L. Farkas, "Multi-user Computation Offloading as Multiple Knapsack Problem for 5G Mobile Edge Computing", European Conference on Networks and Communications (EuCNC), Athens, Greece, pp. 225–229, 27–30 June 2016.

[3] M. Labbe, G. Laporte, and S. Martello, "Upper Bounds and Algorithms for the Maximum Cardinality Bin Packing Problem", Eur. J. Oper. Res., vol. 149, pp. 490–498, 2003.

[4] P. Cappanera, F. Paganelli, F. Paradiso, "VNF Placement for Service Chaining in a Distributed Cloud Environment with Multiple Stakeholders", Comput. Commun. vol. 133, pp. 24–40, 2019.

[5] F. J. Vasko, D. D. Newhart, K. L. Stott," A Hierarchical Approach for One-dimensional Cutting Stock Problems in the Steel Industry that Maximizes Yield and Minimizes Overgrading", Eur. J. Oper. Res, vol. 114, no. 1, pp. 72-82, 1999.

[6] A. S. Fukunaga, "A Branch-and-bound Algorithm for Hard Multiple Knapsack Problems", Annals of Operations Research, vol. 184, no. 1, pp. 97–119, 2011.

[7] S. Martello, P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Chichester, 1990. (available online at www.or.deis.unibo.it).

[8] H. Kellerer, D. Pisinger, U. Pferschy, *Knapsack Problems.* Springer, Berlin, 2004.

[9] G. Sur, S. Y. Ryu, J. Kim, H. Lim, "A deep reinforcement learning-based scheme for solving multiple knapsack problems", Appl. Sci. vol. 12, pp. 3068, 2022. https://doi.org/ 10.3390/app12063068

[10] D. Pisinger, "An Exact Algorithm for Large Multiple Knapsack Problems", Eur. J. Oper. Res. vol. 114, no. 3, pp. 528–541, 1999.

[11] A. S. Fukunaga, "A New Grouping Genetic Algorithm for the Multiple Knapsack Problem", Proceedings of the 2008 IEEE Congress on Evolutionary Computation, Hong Kong, China, pp. 2225–22321, 6 June 2008.

[12] A. S. Fukunaga, S. Tazoe, "Combining Multiple Representations in a Genetic Algorithm for the Multiple Knapsack Problem", 2009 IEEE Congress on Evolutionary Computation, pp. 2423–2430, 2009.

[13] B. McNally, A Simple Sequential Increasing Tolerance Metaheuristic that Generates Bounded Solutions for Combinatorial Optimization Problems. Master's Thesis, Kutztown University of Pennsylvania, 2021.

[14] B. McNally, Y. Lu, E. Shively-Ertas, M.S. Song, F. J. Vasko, “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, vol. 8, no. 2, pp. 76-95, 2021.

[15] Y. Lu, B. McNally, E. Shively-Ertas, F. J. Vasko, “A Simple and Efficient Technique to Generate Bounded Solutions for the Multidimensional Knapsack Problem: a Guide for OR Practitioners”, International Journal of Circuits, Systems, and Signal Processing, vol. 15, pp. 1650-1656, 2021.

[16] A. Dellinger, Y. Lu, B. McNally, M. S. Song, F. J. Vasko,” A Simple and Efficient Technique to Generate Bounded Solutions for the Generalized Assignment Problem: a Guide for OR Practitioners”, Research Reports on Computer Science, pp. 13-34, 2021.

[17] M. S. Song, B. Emerick, Y. Lu, F. J. Vasko, “When to Use Integer Programming Software to Solve Large Multi-Demand Multidimensional Knapsack Problems: A Guide for Operations Research Practitioners,” Engrg. Optim vol. 54, no. 5, pp. 894-906, 2022.

[18] A. Dellinger, Y. Lu, M.S. Song, F. J. Vasko, “Generating Bounded Solutions for Multi-demand Multidimensional Knapsack Problems: a Guide for Operations Research Practitioners,” International Journal of Industrial Optimization, vol. 3, no. 1, pp. 1-21, 2022.

[19] A. Dellinger, Y. Lu, M. S. Song, F. J. Vasko, “Simple Strategies that Generate Bounded Solutions for the Multiple-choice Multi-dimensional Knapsack Problem: A Guide for OR Practitioners”, International Transactions in Operational Research, vol. 0, pp. 1-17, 2022.

[20] F. J. Vasko, “A Computational Note on the Martello-Toth Knapsack Algorithm”, European Journal of Operational Research, vol. 73, no. 1, pp. 169-171, 1994.

[21] J. Tukey, “Comparing Individual Means in the Analysis of Variance”, *Biometrics* 1949, vol. 5, pp. 99-114, 1949.