INTERNATIONAL JOURNAL OF CIRCUITS, SYSTEMS AND SIGNAL PROCESSING
DOI: 10.46300/9106.2021.15.178 Volume 15, 2021

A Simple and Efficient Technique to Generate Bounded Solutions for the Multidimensional Knapsack Problem: a Guide for OR Practitioners

Yun Lu
Department of Mathematics
Kutztown University
Kutztown, PA, U.S.A.
lu@kutztown.edu

Bryan McNally
Department of Computer Science and Information Technology
Kutztown University
Kutztown, PA, U.S.A.
bmcna842@live.kutztown.edu

Emre Shively-Ertas
Department of Computer Science and Information Technology
Kutztown University
Kutztown, PA, U.S.A.
eshiv901@live.kutztown.edu

Francis J. Vasko
Department of Mathematic
Kutztown University
Kutztown, PA, U.S.A.
vasko@kutztown.edu

***Abstract*— The 0-1 Multidimensional Knapsack Problem (MKP) is a NP-Hard problem that has important applications in business and industry. Approximate solution approaches for the MKP in the literature typically provide no guarantee on how close generated solutions are to the optimum. This article demonstrates how general-purpose integer programming software (Gurobi) is iteratively used to generate solutions for the 270 MKP test problems in Beasley's OR-Library such that, on average, the solutions are guaranteed to be within 0.094% of the optimums and execute in 88 seconds on a standard PC. This methodology, called the simple sequential increasing tolerance (SSIT) matheuristic, uses a sequence of increasing 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. The SSIT results (although guaranteed within 0.094% of the optimums) when compared to known optimums deviated only 0.006% from the optimums—far better than any published results for these 270 MKP test instances.**

***Keywords*— matheurstic; Gurobi; Multidimensional Knapsack Problem; bounded solutions; simple sequential increasing tolerance matheuristic.**

I. Introduction

Since the 0-1 Multidimensional Knapsack Problem (MKP) is NP-hard and most real-world applications are typically large in scale, exact solution approaches are usually not appropriate. Nevertheless, the MKP has numerous real-world direct applications or sub-problem applications that need to be solved. It is important for operations research (OR) practitioners that there are simple, effective and efficient solution approaches available to solve these problems. However, it is even more important that OR practitioners can guarantee the quality of the solutions that are presented to management for implementation. This is true regardless if the OR practitioner is called on to assist with a critical strategic planning issue or needs to implement an optimization module in a production system that is executed daily.

In general, given a set of test problem instances, approximate solution method results are compared to optimal or best-known results that were determined by executing an exact algorithm for a long period of time—sometimes up to 24 hours or more! Researchers assume that, if an approximate solution method performs well on a *limited* set of problem instances, it will perform well on other problems. This is the weakness of using approximate solution methods with no guaranteed bounds on solution quality.

Some algorithms developed to solve NP-hard combinatorial optimization problems (COP) make use of commercial integer programming software to solve small or moderate-sized subproblems. On the other hand, for decades, OR practitioners have generated feasible solutions to industrial applications of COP by executing commercial integer programming software for long execution times. In this article, a procedure that iteratively uses commercial integer programming software with no algorithm-specific code required is documented and applied to solve 270 MKPs that are commonly used to test MKP algorithms. This procedure, called the simple sequential increasing tolerance (SSIT) matheuristic, will be shown to quickly generate MKP solutions that are guaranteed to be very close to the optimums. This multi-pass matheuristic is used in conjunction with an integer programming software (Gurobi) package and employs a sequence of increasing tolerances that are used with the integer programming software. If a goal tolerance bound on the solution is not achieved in a user-defined time interval, the best solution found at this interval is then input as a starting solution for the next time interval with looser tolerance.

SSIT was first discussed in [16], and one key feature of the SSIT solutions is that they are *guaranteed* to be within a tight tolerance of the optimum. Another important feature of SSIT is its iterative use of *any* general-purpose integer programming software (Gurobi is used in this article, but other software packages could be used just as easily). These features combined with a user-defined sequence of loosening tolerances and maximum execution times for each tolerance makes SSIT a very flexible solution methodology. SSIT is considered a matheuristic because it uses math programming combined with a heuristically determined sequence of tolerances and execution times. Since SSIT takes advantage of the power of a general-purpose exact solution method, and it require no problem-specific algorithm. Because there is no

problem-specific algorithm to code, the amount of time required to implement a solution to an industrial application is greatly reduced. Furthermore, a unique feature of SSIT is that implemented applications will automatically improve in performance as newer versions of the general-purpose software are implemented for the application.

In the next section, a mathematical formulation for the MKP will be discussed as well as the relevant literature will be reviewed. Next, a brief overview of the SSIT matheuristic will be provided. This will be followed by empirical results obtained from using the SSIT matheuristic to solve 270 MKP test problems available in Beasley's OR-Library. The SSIT results will then be compared to published results for the MKP. This article will close with some conclusions and suggested future work.

II. Literature Background on the 0-1 Multidimensional Knapsack Problem

A. Mathematical Programming Formulation

The mathematical formulation for the 0-1 Multidimensional Knapsack Problem is:

Maximize

$$Z = \sum p_j x_j \quad (1)$$

Subject to

$$\sum a_{ij} x_j \leq b_i \quad i = 1, \ldots m \quad (2)$$

$$x_j \in \{0,1\} \quad j = 1, \ldots n \quad (3)$$

Decision variables are binary where $x_j = 1$ means that item j is packed in the knapsack, and $x_j = 0$ otherwise. Each item j requires a_{ij} units of resource consumption in the ith knapsack constraint and yields p_j units of profit upon inclusion in the knapsack. The goal is to find a subset of items that yields maximum profit without exceeding the resource capacities (the b_i s).

B. Mathematical Programming-based Solution Approaches

The solution approaches based on mathematical programming can be either exact (guarantees a solution within a close tolerance of the optimum) or can be employed in a heuristic manner. If exact, this usually means that if the program terminates before the maximum allowed time limit, the best solution found is within a tight tolerance of the optimum. For example, the default tolerance for Gurobi (version 9.1) is T = 0.0001.

A number of mathematical programming-based solution methods for the MKP have been suggested in the literature with some recent (since 2010) examples applied to solve the MKP discussed here. An exact solution approach was developed by [4] that employees a multi-level search strategy which uses different enumeration schemes at each level. Specifically, it combines resolution search, branch-and-bound, and a depth-first search. [19] use an integer linear programming metaheuristic with new core problem concepts. The core problem is solved with a truncated CPLEX execution or a memetic algorithm. [2] identify restricted sets of promising items (kernel or core) and use an exact solution procedure on these sub-problems. Initially, the continuous relaxation of the MKP is solved on the complete set of available items. Information from this solution is then used to identify the initial kernel. Next, a sequence of integer linear program (ILP) sub-problems are solved, where each sub-problem is restricted to the present kernel and to a subset of other items. Each ILP sub-problem may find better solutions with respect to the previous one and identify further items to insert into the kernel. Reference [7] present improved core problem based heuristics for the MKP. A basic LP-based core problem approach is presented when execution time is limited. Also, a partial enumeration core problem approach is presented that is strongly parallelizable. [15] present an exact algorithm called CORAL (for CORe Algorithm). This paper solves a number of sub-problems limited to a subset of variables and attempts to quickly find good quality lower bounds to speed up the search and to quickly fix as many variables as possible to their provably optimal values. Reference [9] shows how the Global Lifted Cover Inequalities (GLCI) cuts can improve on the core problem heuristic. The GLCIs are presented in the general lifting framework and several variants are proposed. A two-level core problem heuristic is also proposed for very large instances of the MKP. If the methods mentioned above are exact, then they tend to require substantial execution times for large problems. If the methods are heuristic in nature, then *after-the-fact* gaps between the best upper bound and best solution can be calculated—but not a priori specified.

When an OR practitioner is faced with solving a large industrial MKP, all of the methodologies mentioned above are relatively complex to be implemented. A considerable amount of time can be required for computer coding and testing of the selected algorithm, so computer execution time for the actual application can be significant.

C. Bio-inspired Solution Approaches

There are many approximate bio-inspired solution approaches available from the literature. Some recent (since 2010) examples applied to solve the MKP include: harmony search (HS)-based approaches by [12] and [20], particle swarm optimization (PSO)-based approaches by [14] and [10], a shuffled complex evolution algorithm by [3], a fruit fly optimization algorithm by [17], a guided genetic algorithm (GGA) approach by [21], and a teaching-learning based optimization (TLBO) by [11]. In order to solve the MKP, earlier papers discussed a genetic algorithm by Chu and Beasley (1998) and heuristic approaches by [18], [1] and [5]. A classic paper [8] discussed both probabilistic and worst-case analyses for the MKP. A recent survey of the MKP, which contains 167 references, is given by [13].

If an OR practitioner wants to use one of these approximate solution methodologies, the practitioner would be required to code and test the solution approach. More importantly, all the above-mentioned solution procedures explicitly designed to solve the MKP provide **no** guarantees on solution quality! This is in sharp contrast to the SSIT matheuristic because SSIT uses strictly general purpose software and *does* provide bounds on the generated solutions with no problem-specific coding required.

III. Overview of the Simple Sequential Increasing Tolerance (SSIT) Matheuristic

The motivation ([16]) behind the simple sequential increasing tolerance (SSIT) matheuristic is to try to have the best of two worlds. Namely, SSIT makes use of state-of-the-art optimization software (such as CPLEX or Gurobi) combined with loosening tolerances to obtain solutions that are guaranteed within *known and relatively tight* tolerances of the optimum in a timely manner. By using commercially available and state-of-the-art optimization software instead of

highly complex specialized codes for the particular combinatorial optimization problem (COP), SSIT can be used in a straightforward manner by both OR practitioners as well as researchers with no problem-specific coding required. The SSIT matheuristic is very flexible and robust because the user can specify the number of tolerances as well as their specific values based on their needs. The maximum execution time for each tolerance is also specified based on the specific needs of the user.

As indicated earlier, SSIT can be considered a multi-pass methodology 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 "loosened" tolerance allows the branch-and-bound tree in the commercial software to be pruned more quickly. The worst-case scenario for SSIT is that it does not terminate until the sum of the maximum execution times for each tolerance is reached. In this case only, the software gap at termination will indicate how close the best SSIT solution is to the optimum instead. Specifically, for a minimization COP, the optimization software provides the gap between the best lower bound and the best solution.

The pseudo code below summarizes the SSIT methodology for a generic COP.

SSIT MATHEURISTIC

1. Begin
2. Input the number of phases N
3. Input tolerance T_i and maximum execution time t_i for phases i=1, ..., N
4. Input COP details
5. Run integer programming software program to solve COP
6. For 1<=i<=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.
10. Run integer programming software program with SOL_i 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**.

The flow chart of SSIT is also provided in the Figure 1 below.

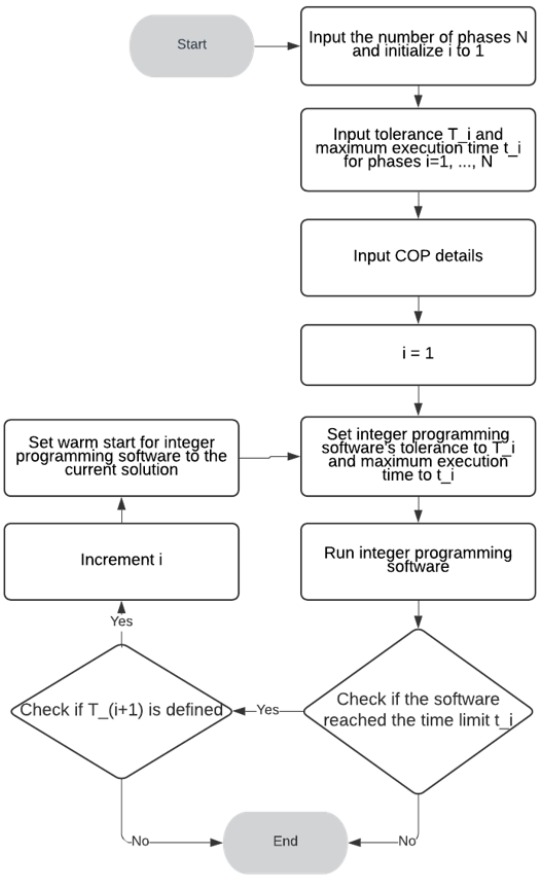

Figure 1: SSIT flowchart

The benefit of SSIT using general purpose integer programming software such as CPLEX or Gurobi and, at the same time, requiring no problem-specific coding is significant. For the SSIT problems discussed in this article, all the software default settings were kept except the time and tolerance per SSIT pass. In particular, the OR practitioner or researcher does not need to develop code or test a problem-specific algorithm. Furthermore, practitioners will find that there is a wealth of examples that come with most optimization software (definitely CPLEX and Gurobi), which are ready to run out of the box. These templates often only require a few adjustments before they are ready to run domain specific combinatorial optimization problems. Practitioners can also quickly find answers to many software specific questions in the online forums and extensive manuals. Additionally, for industrial systems that use SSIT, the performance of these systems is "automatically" improved when new versions of the optimization software are installed. SSIT saves the practitioner time writing extensive code and testing different parameter settings with its ability to quickly find templates and models for various problems and to run a problem with pre-defined defaults that work well with many problems.

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!

Although it is common for OR practitioners to use commercial software at the default tolerance for a fixed amount of time and use the best solution generated when the

execution time "runs out", SSIT provides an alternate to this approach that will be shown to provide bounded solutions quickly.

IV. Empirical Results Using Beasley's 270 MKPs

A. The 270 MKP Instances

To test and compare solution approaches for the MKP, [6] defined 270 MKPs which are available to researchers in Beasley's OR-Library. These 270 problems are divided into 9 datasets with 30 problems in each dataset. In each dataset, the first 10 problems have a tightness ratio of 0.25, the next 10 problems have a tightness ratio of 0.50 and the last 10 problems have a tightness ratio of 0.75. A tightness ratio implies the size of the right hand side value compared to the sum of the variable coefficients for that constraint. For example, a tightness ratio of 0.25 implies that the right hand side of the knapsack constraint is 0.25 times the sum of the variable coefficients for that constraint. The problems consist of either 100, 250 or 500 variables and the number of knapsack constraints are either 5, 10 or 30 for a total of nine datasets—one for each combination of the number of variables and number of constraints.

B. SSIT Results for the 270 MKP Instances

In order to use the SSIT matheuristic to solve MKPs, a sequence of increasing tolerances and corresponding maximum execution times must be specified for Gurobi. Based on limited empirical experimentation, the authors found the following sequence of tolerances and maximum execution times to work very well for these MKPs: tolerances T = 0.0001, 0.001, 0.003, 0.005, 0.007, and 0.009 with maximum execution times of 60 seconds for T= 0.0001 and 120 seconds for all the other tolerances. Maximum possible execution time is 660 seconds, but the actual average execution time was 88 seconds. As practitioners, the authors view these tolerances and execution times as very acceptable for most industrial applications. However, the robustness of SSIT is that 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. This flexibility is really the metaheuristic part of the SSIT matheuristic.

Except for tolerances and execution times as specified above, all other software parameters will have their default settings. 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.

The results of executing SSIT to solve the 270 MKP instances are summarized in Tables I, II, and III. As an example, detailed results for Dataset 3 are given in Table 4. Detailed results for all 270 MKPs are available upon request. In Table I average results over the 30 MKPs in each of the 9 datasets are given. The guaranteed maximum deviation from the optimum column shows the *farthest* away the SSIT solutions can be without knowing the exact value of the optimums. Over all 270 MKPs, the average guaranteed farthest deviation the SSIT solutions are from the optimums is 0.094%. However, comparing these 270 SSIT solutions to the known optimums or best-known solutions, the SSIT solutions, on average, actually only deviated 0.006% from the optimums. Additionally, these very impressive results required, on average, only 88 seconds of execution time on a standard PC. In particular, the execution times were 180 seconds or less for about 90% of the MKPs.

Again, from Table 1, one can see that the most difficult MKPs for SSIT to solve were in Datasets 8 and 9. Even in these cases, the average guaranteed deviations from the optimums were only 0.359% and 0.250%, and required execution time to solve for Data Sets 8 and 9 are 285.7 and 217.0 seconds respectively.

TABLE I. Summary of SSIT results for the 270 MKPs

Data Set	Number of variables	Number of constraints	Average guaranteed maximum deviation from optimum (%)	Average actual deviation from optimum (%)	Average execution time (seconds)
1	100	5	0.000	0.000	0.4
2	250	5	0.007	0.000	6.1
3	500	5	0.014	0.001	35.1
4	100	10	0.001	0.000	2.2
5	250	10	0.104	0.001	108.4
6	500	10	0.080	0.011	100.2
7	100	30	0.028	0.004	37.8
8	250	30	0.359	0.013	285.7
9	500	30	0.250	0.022	217.0
Overall			0.094	0.006	88.1

Tables II and III show the distributions of the tolerances at which SSIT terminated by data sets and by constraint tightness ratios respectively. From these tables, one can see that 50% of the 270 MKPs terminated in under 60 seconds when the tolerance of 0.0001 was in effect. The power of SSIT is evident in that by loosening the tolerance 90.7% of the 270 MKPs terminated when a tolerance of 0.003 or smaller was in effect. From Table 2 it can be observed that only Datasets 8 and 9 had MKPs that terminated at tolerances greater than 0.003 with 15 MKPs from Dataset 8 and 10 MKPs from Dataset 9. From Table 3 it can be observed that of the 25 MKPs that terminated at tolerances greater than 0.003, 20 of them (80%) had a tightness ratio of 0.25 and 5 had a tightness ratio of 0.50. Of the 25 MKPs that terminated at a tolerance greater than 0.003, 15 terminated at the tolerance of 0.005, 9 at a tolerance of 0.007, and only one at a tolerance of 0.009. The one MKP that terminated at a tolerance of 0.009, had a final gap (guaranteed deviation from the optimum) of 0.0073 and terminated as soon as the tolerance of 0.007 was loosened to 0.009. The 10 MKPs that terminated at tolerances greater than 0.005 were all from Dataset 8 with a tightness ratio of 0.25. For the few problems with termination tolerances greater than 0.003. if tighter guaranteed bounds were required, the execution times for the tolerances could be increased.

TABLE II. SUMMARY OF SSIT TERMINATION TOLERANCES DISTRIBUTIONS BY DATASETS

	Tolerances					
Dataset	**0.0001**	**0.001**	**0.003**	**0.005**	**0.007**	**0.009**
1	30					
2	30					
3	20	10				
4	30					
5	1	20	9			
6		20	10			
7	24	5	1			
8			15	5	9	1
9		1	19	10		
Overall	135	56	54	15	9	1
Percentage (%)	50.0	20.7	20.0	5.6	3.3	0.4

TABLE III. SUMMARY OF SSIT TERMINATION TOLERANCES DISTRIBUTIONS BY TIGHTNESS RATIO

	Tolerances					
Tightness ratio	**0.0001**	**0.001**	**0.003**	**0.005**	**0.007**	**0.009**
.25	39	12	19	10	9	1
.50	45	24	16	5		
.75	51	20	19			
Overall	135	56	54	15	9	1
Percentage (%)	50.0	20.7	20.0	5.6	3.3	0.4

Table IV details the SSIT solution process for the 30 MKPs in Dataset 3. As mentioned previously, detailed solutions tolerance-by-tolerance for each of the 270 MKPs are available upon request. As can be observed in Table 4, 20 of the MKPs terminated in under 60 seconds when the guaranteed maximum deviation from the optimums dropped below the 0.0001 tolerance. The other 10 MKPs reached 60 seconds of execution time and then when the tolerance was loosened to 0.001, all 10 terminated immediately because, at 60 seconds of execution time, the guaranteed maximum deviation from the optimums was less than 0.001.

TABLE IV. DATASET 3 WITH 500 VARIABLES 5 CONSTRAINTS

Problem	**Tolerance**	**Obj Fn**	**Time (seconds)**	**Final Gap**
1	0.0001	120148	60.00	
	0.001	120148	0.00	0.0003
2	0.0001	117879	13.57	0.00009
3	0.0001	121131	60.00	
	0.001	121131	0.00	0.00017
4	0.0001	120794	60.00	
	0.001	120794	0.00	0.00032
5	0.0001	122319	24.39	0.00010
6	0.0001	122024	60.00	
	0.001	122024	0.00	0.00022
7	0.0001	119127	60.00	
	0.001	119127	0.00	0.00029
8	0.0001	120568	30.74	0.00010
9	0.0001	121575	60.00	
	0.001	121575	0.00	0.00033
10	0.0001	120711	60.00	
	0.001	120711	0.00	0.00032
11	0.0001	218428	40.22	0.00010
12	0.0001	221202	29.74	0.00009
13	0.0001	217542	60.00	
	0.001	217542	0.00	0.00013
14	0.0001	223560	60.00	
	0.001	223560	0.00	0.00015
15	0.0001	218966	9.44	0.00010
16	0.0001	220530	41.22	0.00010
17	0.0001	219989	32.35	0.00010
18	0.0001	218215	25.24	0.00010
19	0.0001	216976	52.72	0.00010
20	0.0001	219719	60.00	
	0.001	219719	0.00	0.00012
21	0.0001	295828	3.58	0.00010
22	0.0001	308086	15.48	0.00010
23	0.0001	299796	8.02	0.00010
24	0.0001	306480	25.92	0.00010
25	0.0001	300342	9.93	0.00010
26	0.0001	302562	22.41	0.00010
27	0.0001	301339	6.05	0.00009
28	0.0001	306454	8.63	0.00008
29	0.0001	302828	18.45	0.00008
30	0.0001	299904	34.20	0.00010

In the next section, these SSIT results for these 270 MKP instances will be compared to 11 bio-inspired metaheuristics from the literature. However, it is important to note that none of these 11 metaheuristics provide any guarantees on solution quality.

C. MKP SSIT Results Compared to Other Metaheuristics

Although, as OR practitioners, the authors appreciate the guaranteed bounds that the SSIT matheuristic provides (unless the sum of the maximum times is exceeded which did not happen for any of the 270 MKPs), there may be readers that are interested in seeing how the SSIT solutions obtained in the previous subsection compare to 11 published metaheuristics for the MKP.

In Table II of [11], results of solving the 270 MKPs discussed in this article using 11 metaheuristics are summarized. For each of the 11 metaheuristics, this table contains the average deviations from the optimum or best known solutions for each of the nine datasets discussed in this article. The 11 metaheuristic solution procedures listed in

Table II of [11] and cited earlier in this article are Teaching-Learning Based Optimization (TLBO), guided GA, a GA, Primal Effective Capacity Heuristic (PECH), MAG and VZ, two solution approaches using Lagrange multipliers, PIR, a dual surrogate relaxation heuristic with a branch and bound component, Shuffled Complex Evolution (SCE), CB, a GA augmented with a feasibility and constraint operator, New Reduction (Pirkul) NRP operates a lagrangian dual relaxation on MKP, and the Modified Choice Function-Late Acceptance Strategy (MCF). The reader should consult [21] or [11] for more details. The SSIT average deviations from optimum or best-known solutions along with corresponding results for these 11 metaheuristics are summarized in Table V.

TABLE V. DEVIATION FROM OPTIMUM (%)

# Cons -Vars	SSIT	TLBO	GGA	G A	PECH	MAG	VZ	PI R	SCE	CB	NR P	MCF
5-100	0.000	0.42	0.54	1.2	4.24	8.47	7.6	1.0	2.4	.59	.53	.68
5-250	0.000	1.46	0.56	1.9	4.03	5.1	4.6	.31	3.03	.14	.24	.26
5-500	0.001	2.59	0.54	2.1	3.83	3.37	3.0	.12	3.33	.94	.08	.12
10-100	0.000	0.68	0.73	1.5	4.57	10.77	10.6	2.1	4.77	.05	1.1	1.12
10-250	0.001	1.4	0.63	1.9	3.27	7.63	6.7	.67	5.03	.3	.49	.48
10-500	0.011	1.97	0.57	1.9	2.9	6.03	4.97	.29	5.33	.14	.19	.26
30-100	0.004	0.83	1.08	1.6	3.97	11.89	11.1	4.9	6.27	1.7	1.45	2.06
30-250	0.013	1.13	1.23	1.8	2.7	8.83	7.8	2.0	6.33	.68	.8	.99
30-500	0.022	1.33	2.14	1.9	2.1	6.87	6.27	1.0	6.67	.35	.49	.59
Overall Average	**0.006**	1.31	0.89	1.8	3.51	7.66	6.95	1.4	4.8	.54	.6	.73

The SSIT results are given to 3 decimal places because of their small values. SSIT significantly outperformed all other metaheuristics. Over all 270 MKPs, the average deviation from the optimum was only 0.006%. Of the 11 metaheuristics listed in Table 5, the one closest to the SSIT result was the Chu-Beasley (classic) genetic algorithm with an average deviation from the optimum of 0.54%. Even the SSIT guaranteed bound of 0.094% is far better than any published metaheuristic results. It is important to note that SSIT is a *deterministic* solution method and the 11 metaheuristic results are taken from previously published results. Hence, any statistical analysis is unnecessary for this comparison. However, the real benefit of SSIT is its ability to guarantee that its generated solutions are within a typically very small percentage of the optimum—0.094% for these 270 MKPs.

V. SUMMARY AND FUTURE WORK

Some algorithms developed to solve NP-hard combinatorial optimization problems (COP) make use of commercial integer programming software to solve small or moderate-sized subproblems. On the other hand, for decades OR practitioners have generated feasible solutions to industrial applications of COP by executing commercial integer programming software for long execution times. In this article, a procedure that iteratively uses commercial integer programming software with no algorithm-specific code required is used to solve 270 multidimensional knapsack problem (MKP) instances from the literature. This procedure called the simple sequential increasing tolerance (SSIT) matheuristic was empirically shown to quickly generate MKP solutions that are guaranteed to be very close to the optimums—on average the SSIT solutions were guaranteed within 0.094% of the optimums and it required only 88 seconds on a standard PC. When compared to the known optimums or the best-known solutions for these 270 MKPs, the SSIT solutions actually deviated only 0.006% from the optimums. Both the SSIT guaranteed maximum deviation from the optimums of 0.094% and the actual SSIT deviations from the optimums of 0.006% are far better than any metaheuristic results that appear in the literature.

This multi-pass matheuristic was used in conjunction with the integer programming software Gurobi and employs a sequence of increasing tolerances that allows Gurobi to quickly generate solutions guaranteed to be close to the optimums. If a goal tolerance bound on the solution is not achieved in a user-defined time interval, the best solution found at this time interval is then input as a starting solution for the next time interval with looser tolerance.

In addition to SSIT finding bounded solutions quickly, its use of general-purpose integer programming software is a significant benefit to OR practitioners. Specifically, it allows OR practitioners to quickly develop SSIT models using default software parameter values and templates with no need for problem-specific algorithms. Based on the particular application, the user has the flexibility to set the number of tolerances as well as their values. Additionally, the user determines the maximum execution time for each tolerance. Furthermore, OR practitioners who implement SSIT in an industrial application that is executed routinely, have the added advantage that the performance of their application will continue to "automatically" improve as new versions of the commercial software are implemented.

Finally, since the SSIT matheuristic is a general-purpose strategy for solving combinatorial optimization problems, the authors plan to test the performance of SSIT on solving other difficult-to-solve combinatorial optimization problems.

REFERENCES

[1] Y. Akcay, H. Li, and SH Xu, "Greedy algorithm for the general multidimensional knapsack problem," Ann Oper Res., vol.150, pp. 17-29, 2007.

[2] E. Angelelli, R. Mansini, and MG. Speranza, "Kernel search: a general heuristic for the multi-dimensional knapsack problem," Comput. Oper. Res., vol 37, pp. 2017-2026, 2010.

[3] MDV. Baroni and FM. Varejao, "A shuffled complex evolution algorithm for the multidimensional knapsack problem," Iberoamerican Congress on Pattern Recognition, Springer, pp. 768-775, 2015.

[4] S. Boussier, M. Vasquez, Y. Vimont, S. Hanafi, and P. Michelon, "A multi-level search strategy for the 0-1 multidimensional knapsack problem," Discr. App. Math., vol. 158, pp. 97-109, 2010.

[5] V. Boyer, M. Elkihel, D. El Baz, "Heuristics for the 0-1 multidimensional knapsack problem," European Journal of Operational Research, vol. 199, pp. 658-664, 2009.

[6] P. Chu and J. Beasley, "A genetic algorithm for the multidimensional knapsack problem," Journal of Heuristics, vol 4, pp. 63-86, 1998.

[7] F. Della Croce and A. Grosso, "Improved core problem based heuristics for the 0/1 multi-dimensional knapsack problem," Comput. Oper. Res., vol. 39, pp. 27-31, 2012.

[8] AM. Frieze and MRB. Clarke, "Approximation algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses," European Journal of Operations Research, vol. 15, pp.100-109, 1984.

[9] H. Gu, "Improving problem reduction for 0-1 multidimensional knapsack problems with valid inequalities," Comput. Oper. Res., vol. 71, pp. 82-89, 2016.

[10] K. Kang, "A Fast Particle Swarm Optimization algorithm for large scale multidimensional knapsack problems," Journal of Computational Information Systems, vol. 8(7), pp. 2709-2716, 2012.

[11] Z. Kern, Y. Lu, and FJ. Vasko, "An OR practitioner's solution approach to the multidimensional knapsack problem," International Journal of Industrial Engineering Computations, vol. 11(1), pp.1-10, 2020.

[12] X. Kong, L. Gao, H. Ouyang, and S. Li, "Solving large-scale multidimensional knapsack problems with a new binary harmony

search algorithm," Computers & Operations Research, vol. 63, pp. 7-22, 2015.
[13] S. Laabadi, M. Naimi, H. El Amri, and B. Achchab, "The 0/1 multidimensional knapsack problem and its variants: a survey of practical models and heuristic approaches," American Journal of Operations Research, vol. 8, pp. 395-439, 2018.
[14] S. Labed, A. Gherboudj, and S. Chikhi, "A modified hybrid particle swarm optimization algorithm for multidimensional knapsack problem," International Journal of Computer Applications, vol. 34(2), pp. 11-16, 2011.
[15] R. Mansini and MG. Speranza, "CORAL: an exact algorithm for the multidimensional knapsack problem," INFORMS J. Comput., vol. 24, pp. 399-415, 2012.
[16] B. McNally, "A simple sequential increasing tolerance matheuristic that generates bounded solutions for combinatorial optimization problems," Master's Thesis, Kutztown University of Pennsylvania, 2021.
[17] T. Meng and QK. Pan, "An improved fruit fly optimization algorithm for solving the multidimensional knapsack problem," Applied Soft Computing., vol. 50, pp. 79-93, 2017.
[18] RJ. Moraga, WG. DePuy, and GE. Whitehouse, "Meta-RaPS approach for the 0-1 multidimensional knapsack problem," Computers & Industrial Engineering, vol. 43, pp. 83-96, 2005.
[19] J. Puchinger, GR. Raidl, and U. Pferschy, "The multidimensional knapsack problem: structure and algorithms," INFORMS J. Comput., vol. 22, pp. 250-265, 2010.
[20] A. Rezoug and D. Boughaci, "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, 2016.
[21] A. Rezoug, M. Bader-El-Den, and D. Boughaci, "Guided genetic algorithm for the multidimensional knapsack problem," Memetic Computing, vol. 10, pp. 29-42, 2018.