UAD Universitas Ahmad Dahlan

IJIO
(*International Journal of Industrial Optimization*)
ISSN 2723-3022 (online) | 2714-6006 (printed)
VOL 3, No. 1, 1-17
https://doi.org/10.12928/ijio.v3i1.5073

Generating bounded solutions for multi-demand multidimensional knapsack problems: a guide for operations research practitioners

Anthony Dellinger [1], Yun Lu [2], Myung Soon Song [2], Francis J. Vasko [2,*]
[1] Computer Science Department, Kutztown University, United States of America
[2] Department of Mathematics, Kutztown University, United States of America
*Corresponding Author: vasko@kutztown.edu

ARTICLE INFO

Article history
Received: October 28, 2021
Revised: March 4, 2022
Accepted: March 6, 2022

Keywords
Simple sequential increasing tolerance methodology;
Integer programming software;
Classification trees;
Bounded solutions;
Operations research practice.

ABSTRACT

A generalization of the 0-1 knapsack problem that is hard-to-solve both theoretically (NP-hard) and in practice is the multi-demand multidimensional knapsack problem (MDMKP). Solving an MDMKP can be difficult because of its conflicting knapsack and demand constraints. Approximate solution approaches provide no guarantees on solution quality. Recently, with the use of classification trees, MDMKPs were partitioned into three general categories based on their expected performance using the integer programming option of the CPLEX® software package on a standard PC: Category A—relatively easy to solve, Category B—somewhat difficult to solve, and Category C—difficult to solve. However, no solution methods were associated with these categories. The primary contribution of this article is that it demonstrates, customized to each category, how general-purpose integer programming software (CPLEX in this case) can be iteratively used to efficiently generate bounded solutions for MDMKPs. Specifically, the simple sequential increasing tolerance (SSIT) methodology will iteratively use CPLEX with loosening tolerances to efficiently generate these bounded solutions. The real strength of this approach is that the SSIT methodology is customized based on the particular category (A, B, or C) of the MDMKP instance being solved. This methodology is easy for practitioners to use because it requires no time-consuming effort of coding problem specific-algorithms. Statistical analyses will compare the SSIT results to a single-pass execution of CPLEX in terms of execution time and solution quality.

INTRODUCTION

The 0-1 multi-demand multidimensional knapsack problem (MDMKP) involves filling a knapsack such that the value of the items inserted in the knapsack is maximized while a number of knapsack constraints (≤) and demand constraints (≥) are not violated. Since the MDMKP is a generalization of the classic 0-1 knapsack problem, it is easy to show that it is NP-hard, and thus large instances are typically solved using approximate solution approaches such as heuristics or metaheuristics. Because MDMKPs have both demand and knapsack constraints that directly conflict with each other, they can be very challenging to solve. Furthermore, there are a number of practical applications [1] like obnoxious and semi-obnoxious facility location [2], and capital budgeting and portfolio selection [3] that typically require the solution of large MDMKPs. Hence, motivated by the need to solve these applications, approximate solution approaches are usually discussed in the literature.

Recently, Song et al. [4] used classification tree analyses combined with 1620 MDMKPs discussed in the literature to categorize MDMKPs into one of three categories: Category A—relatively easy to solve, Category B—somewhat difficult to solve, and Category C—difficult to solve based on their performance using the integer programming option of the CPLEX® software package on a standard PC. These 1620 problems were chosen because they are discussed in the literature and readily available to researchers for empirical test purposes. For these analyses, the CPLEX tolerance was loosened (from 0.0001 (default) to 0.001) for all executions of CPLEX with the maximum running time of an hour. Although execution times will generally vary with PC, if an MDMKP was classified in Category A, then the expected execution time would be several minutes on average. If an MDMKP was classified in Category B, the expected execution time would be an average hour. However, if an MDMKP was classified in Category C, then CPLEX would not be expected to find either feasible solutions or solutions within the 0.001 tolerance within an execution time of one hour.

This article aims to apply the simple sequential increasing tolerance (SSIT) methodology [5] to generate bounded solutions for these 1620 test MDMKP instances in shorter execution times than executing CPLEX using the default tolerance 0.0001 for up to one hour on a standard PC. It is important to note that SSIT will be customized based on whether the MDMKP is in Category A, B, or C. Hence, category information is critical to the specific solution strategy used. These 1620 MDMKPs are composed of 810 MDMKPs defined in Lu and Vasko [6] and 810 MDMKPs defined by Cappanera and Trubian [1]. These two sets of 810 MDMKPs are identical except that the demand constraint right-hand-sides (RHS) defined in [6] are only 10% of the values of the demand constraints defined in [1]. Thus, the demand constraints of [6] are looser than the original demand constraints in [1].

Approximate solution approaches for the MDMKP include a nested-tabu-search heuristic [1], an adaptive search method [7], a scatter search scheme [8], an alternating control tree (ACT) search framework [9], a dominance principle-based heuristic [10], a two-stage solution-based tabu search approach [11], and a core-based methodology [12]. It is important to note that when using typical heuristic or metaheuristic methods to solve MDMKPs, these approaches provide no a priori (up front) guarantee on the quality of the solution. Classically in the literature, these approximate solution methods measure how good their solutions are after the fact by comparing their empirical results to optimum or best-known results. The optimum or best-known results were generated typically consuming considerable computer time using an integer programming optimizer such as CPLEX.

The next section will provide the mathematical formulation of a multi-demand multidimensional knapsack problem. Then the nature of the 1620 test problems will be defined. Next, the classification tree used in Song et al. [4] will be reviewed and modified so that MDMKPs are partitioned into four categories; easy (Category A), somewhat difficult (Category B), difficult (Category C), and very difficult to solve (Category D). Then the motivation for SSIT will be outlined. This will be followed by the solution of these 1620 MDMKPs using SSIT tailored to the problem category. Statistical analyses will compare the SSIT results to a single-pass execution of CPLEX in terms of execution time and solution quality. Finally, several conclusions are drawn, and implications for operations research practitioners are discussed.

METHOD

1. Mathematical Formulation of a Multi-Demand Multidimensional Knapsack Problem

The MDMKP is an extension of the 0-1 knapsack problem with multiple knapsack and demand constraints. The knapsack constraints represent “capacity” type constraints, and the demand constraints represent minimum “fill” requirements for the knapsack. Although there are

typically several knapsack and demand constraints, there is only one knapsack that is being "filled" with items for this formulation. Also, in the MDMKP, the objective function coefficients $\{p_i\}_{1\leq i\leq n}$ are not constrained in sign.
Mathematical Formulation of the MDMKP

$$\text{Max} \quad \sum_{i=1}^{n} p_i x_i \tag{1}$$
$$\text{s.t.} \quad \sum_{i=1}^{n} w_{ik} x_i \leq c_k, \qquad k \in \{1, \dots, P\} \tag{2}$$
$$\sum_{i=1}^{n} w_{il} x_i \leq d_l, \qquad l \in \{P+1, \dots, P+Q\} \tag{3}$$
$$x_i \in \{0,1\}, \qquad i \in \{1, \dots, n\} \tag{4}$$

where $c_k > 0$ for $k \in \{1, \dots, P\}$, $d_l > 0$ for $l \in \{P+1, \dots, P+Q\}$, $w_{ik} \geq 0$ and $w_{il} \geq 0$ for $i \in \{1, \dots, n\}, k \in \{1, \dots, P\}$ and $l \in \{P+1, \dots, P+Q\}$.

Constraint set (2) is the classic knapsack constraints (there are P of them) which can be viewed as capacity constraints on the knapsack. Constraint set (3) is the demand constraints (there are Q of them), representing minimum fill requirements for the knapsack. Constraint set (4) ensures that the variables take only zero or one values. If the variable x_i is one, then the item is inserted into the knapsack. If the variable x_i is zero, then the item is not inserted into the knapsack. If only constraint sets (2) and (4) are included, then the problem is referred to as the Multidimensional Knapsack Problem (MKP). Additionally, the objective function coefficients in an MKP are all positive, whereas the objective coefficients $\{p_i\}_{1\leq i\leq n}$ in an MDMKP can be either positive or mixed (both positive and negative).

2. MDMKP Test Instances

The 1620 MDMKPs are categorized in [4] as either easy, moderate, or difficult to solve in this article using the SSIT methodology tailored to the particular problem category. In [1], the authors use 270 MKP instances developed by Chu and Beasley [13] as a base to define 810 MDMKP instances (also available on Beasley's OR-Library [14]).

How Cappanera and Trubian [1] defined their 810 MDMKP instances will now be given. The number of ≤ constraints m was set to 5, 10 or 30 as in [13], while the number of ≥ constraints q was dependent on m. For a fixed value of m, possible values for q were 1, truncate(m/2), and m, respectively. The number of variables n was set to 100, 250, or 500 as in [13]. Objective function coefficients were either all positive or had mixed (both positive and negative) values. Nine datasets were defined based on the number of variables and constraints in the problem. There are 90 problem instances in each of the nine datasets for a total of 810 MDMKP instances. Within each dataset six 'cases' are defined based on the value of q and whether objective function coefficients were strictly positive or mixed. Defined for knapsack constraints, the tightness ratio is the relationship between the constraint coefficients' sum and the constraint's right-hand-side value. For example, a knapsack constraint with a 0.25 tightness ratio implies that the right-hand side value of the constraint is 0.25 times the sum of the constraint coefficients. Each case has 15 problem instances, five at each of the three tightness ratios: 0.25, 0.50, and 0.75 for knapsack constraints.

For reporting results in this article, the following cases are defined:
Case 1 has q = 1 and positive objective function coefficients,
Case 2 has q = truncate(m/2) and positive objective function coefficients,
Case 3 has q = m and positive objective function coefficients,
Case 4 has q = 1 and mixed objective function coefficients,
Case 5 has q = truncate(m /2) and mixed objective function coefficients,
Case 6 has q = m and mixed objective function coefficients.

In this article, the modified versions of these 810 MDMKPs discussed in Lu and Vasko [6] are also solved with SSIT. In Lu and Vasko [6], the right-hand-sides of all demand constraints were set to 10% of their original values as defined by Cappanera and Trubian [1]. The right-hand-sides for the demand constraints were reduced to 10% of their original values because Lu and Vasko [6] were using these problems as a basis to solve MDMKPs with added choice constraints. It was necessary to reduce the right-hand-sides of the demand constraints to 10% of their original values to obtain feasible solutions for these problems (see [6] for more details). In this article, the 810 MDMKPs from [6] are considered to have loose demand constraints for obvious reasons. The original 810 MDMKPs from [1] are considered to have tight demand constraints.

3. Classification Trees

In [4], a classification tree was constructed using the built-in function fitctree in MATLAB from the Statistical and Machine Learning Toolbox [15]. The following four numerical variables were input: Var for the number of variables in the MDMKP, Dim for the number of knapsack constraints in the MDMKP, Dem for the number of demand constraints in the MDMKP, and Dim_T for the tightness of the knapsack constraints (0.25, 0.50, 0.75) in the MDMKP. There were also two categorical variables: Obj for if the objective function coefficients were positive (P) or mixed (M) in the MDMKP, and DEM_T for if the demand constraints were tight (T) or loose (L) in the MDMKP. The classification tree used in [4] is given in Figure 1. For more details, see [4].

The goal is to use the decision tree in Figure 1 to predict if an MDMKP falls into one of three categories:
Category A—CPLEX can obtain a solution for an MDMKP that is guaranteed within 0.1% of optimum and requires less than 300 seconds on a standard PC (times will vary by PC).
Category B---CPLEX can obtain a solution for a MDMKP that is guaranteed within 0.1% of optimum but may require up to an hour on a standard PC (times will vary by PC).
Category C---CPLEX will be unsuccessful at finding either feasible solutions or solutions guaranteed within 0.1% of optimum in a reasonable amount of time, about an hour (times will vary by PC).
Using the decision tree from Figure 1, the 1620 MDMKPs discussed in this article are classified as follows: 1050 MDMKPs (65%) are classified in Category A, 330 MDMKPs (20%) are classified in Category B, and 240 (15%) are classified in Category C.

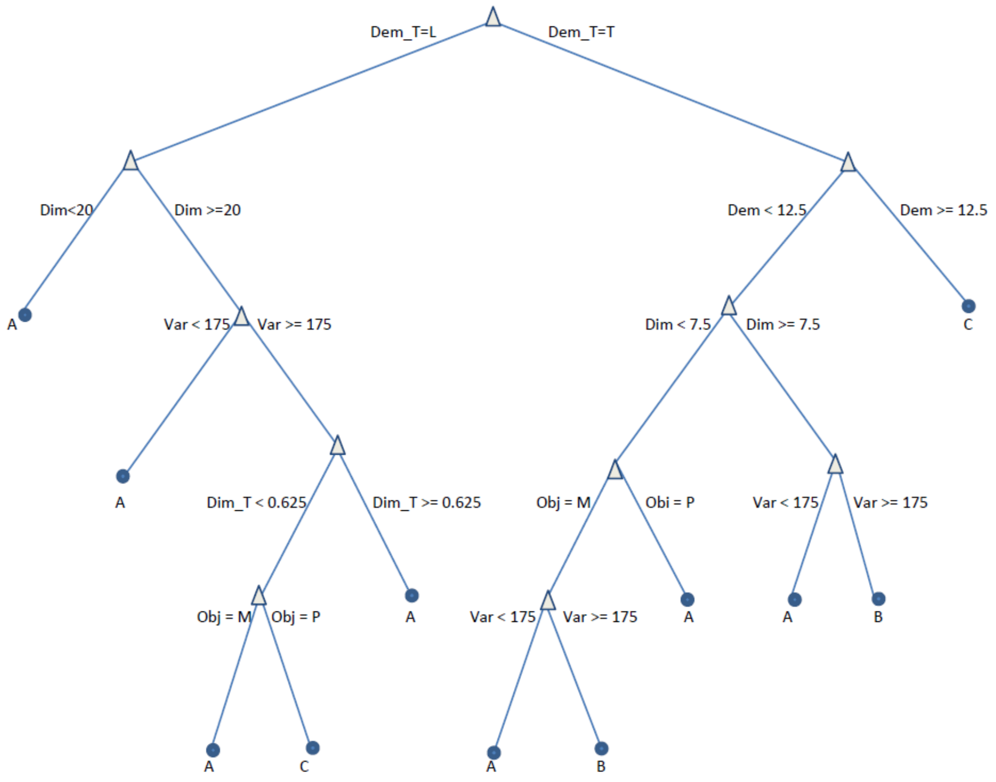

Figure 1. The decision tree [4] to categorize MDMKPs in categories A, B, or C

The tree predicts long computing times (Category C) based on two distinct paths, one path simpler than the other. The simple path predicts that an MDMKP will be in Category C if it contains tight demand constraints (Dem_T = T) with at least 15 demand constraints (DEM > 12.5). Hence, tight constraints make it difficult for CPLEX to obtain a bounded solution in a reasonable amount of time once there are at least 15 demand constraints. Note that for these MDMKPs, any problem having 15 demand constraints will have 30 knapsack constraints.

It has been observed by several researchers [1], [6] that it is difficult even to obtain feasible solutions for the problems in Cappanera and Trubian's [1] dataset 7, cases 3 and 6. The authors of this article conjecture that the difficulty with solving these MDMKPs is because there is a large number of both demand (30) and knapsack (30) constraints, but the number of variables is "small" (only 100). This conjecture is supported by the fact that there is no problem obtaining feasible solutions for cases 3 and 6 in datasets 8 and 9, which also have 30 demand and 30 knapsack constraints but have 250 and 500 variables, respectively.

These 30 MDMKPs are classified as Category C, but because they are so difficult to solve, Category C will be partitioned into two subsets. Category D will consist of the 30 very difficult MDMKPs in dataset 7, cases 3 and 6 [1]. All the other MDMKPs in Category C will remain in Category C. Figure 2 shows the "manually" adjusted decision tree. The reason for defining Category D is that the solution strategy that will be recommended to solve Category D problems will be specifically tailored for operations research practitioners faced with the need to solve such MDMKP for real-world applications. Before discussing SSIT strategies specific to each category of MDMKPs (A, B, C, and D), an overview of the SSIT solution strategy will be provided.

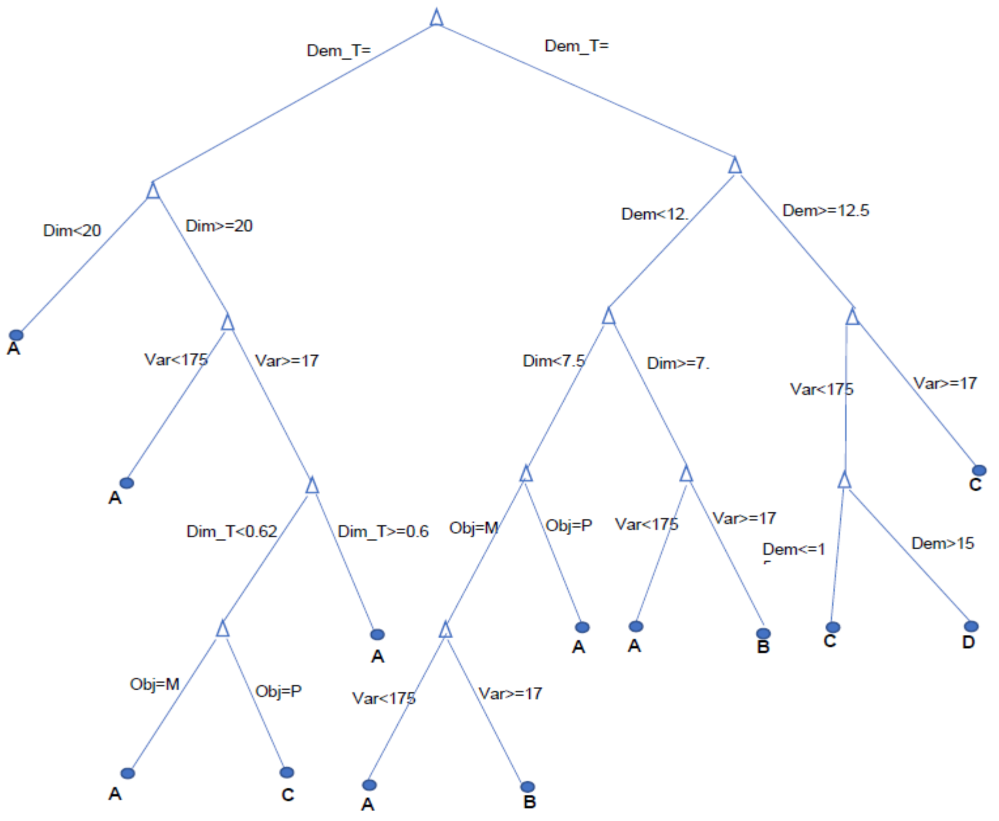

Figure 2. Modified decision tree to categorize MDMKPs in categories A, B, C, or D

4. Overview of the Simple Sequential Increasing tolerance Matheuristics

The motivation [5] behind the simple sequential increasing tolerance (SSIT) methodology is to try to have the best of two worlds. Namely, to use 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, but in a timely manner. By using commercially available, state-of-the-art optimization software instead of highly complex specialized codes for the particular binary integer program (BIP) being solved, SSIT can be used straightforwardly by both OR practitioners and researchers.

Successful applications of SSIT to solve several BIPs have been documented in the literature. McNally et al. [16] was able to solve 135 set K-covering problems (SKCP) from the literature on a standard PC in an average of 67 seconds obtained an average guaranteed bound of 0.13% from the optimums using the Gurobi software package. Statistical analyses demonstrated that these SSIT results were as good as the best published results from algorithms specifically designed to solve SKCPs. Also, using Gurobi, Lu et al. [17] employed the SSIT methodology to quickly (average of 88 seconds on a standard PC) generate solutions guaranteed to be, on average, within 0.09% of the optimum on 270 multidimensional knapsack problems (MKP) instances commonly used in the literature. These results are far better than other published metaheuristic results for the MKP. Dellinger et al. [18] employed SSIT to quickly (average of 63 seconds on a standard PC) generate solutions guaranteed to be within 0.08% of the optimum on 51 generalized assignment problem (GAP) instances commonly used in the literature. These results are very competitive with the best published solution methods for the GAP. Additionally, in [18], both Gurobi and CPLEX were used to solve the GAP instances, and there was no statistically significant difference in solution quality or execution times between Gurobi and CPLEX.

The SSIT methodology is very flexible and robust because the user can specify how many tolerances and their specific values based on the needs of the particular application being solved. The maximum execution times are also specified based on the particular application. SSIT can be considered a multi-pass methodology in which the program terminates if the goal (tolerance) is met within the time allowed. If it is not completed, the tolerance is "loosened". The current best solution is used as input for this next step in the solution process. 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, instead of the solution generated being within a user pre-defined tolerance of the optimum, the software gap at termination will indicate how close the best SSIT solution is to the optimum. Specifically, for a minimization BIP, the optimization software provides the gap between the best lower bound and the best solution. Although SSIT is very intuitive, this is the first article to discuss and quantify the benefits of using SSIT specifically to solve MDMKPs.

The pseudo-code and flowchart [16] below summarize the SSIT methodology for a generic BIP.

SSIT Methodology [16]

```
1. Input the number of phases N
2. Input tolerance T_i and maximum execution time t_i for phases i=1, ..., N
3. Input BIP details
4. Run integer programming software program to solve BIP
5. For 1<=i<=N-1,
6. IF integer programming software running time in phase i is less than t_i, FINISH
7. ELSE,
8. take best solution obtained from Phase i and save it as SOL_i.
9. Run integer programming software program with SOL_i as the warm start and
   tolerance T_{i+1} and maximum execution time t_{i+1}.
10. i=i+1
11. LOOP through step 7-11 until FINISH.
```

The benefit of SSIT using general-purpose integer programming software such as CPLEX or Gurobi is significant, especially to an OR practitioner. For the SSIT problems discussed in this article, all the CPLEX default settings were kept, except the time and tolerance per the SSIT procedure were modified. In particular, the OR practitioner or researcher does not need to develop and code a problem-specific algorithm. Furthermore, practitioners will find a wealth of examples that come with most optimization software (definitely CPLEX and Gurobi). These models are ready to run out of the box. These templates often only require a few changes before they are ready to run domain-specific binary optimization programs. Practitioners can also quickly find answers to many software specific questions in the online forums and extensive manuals.

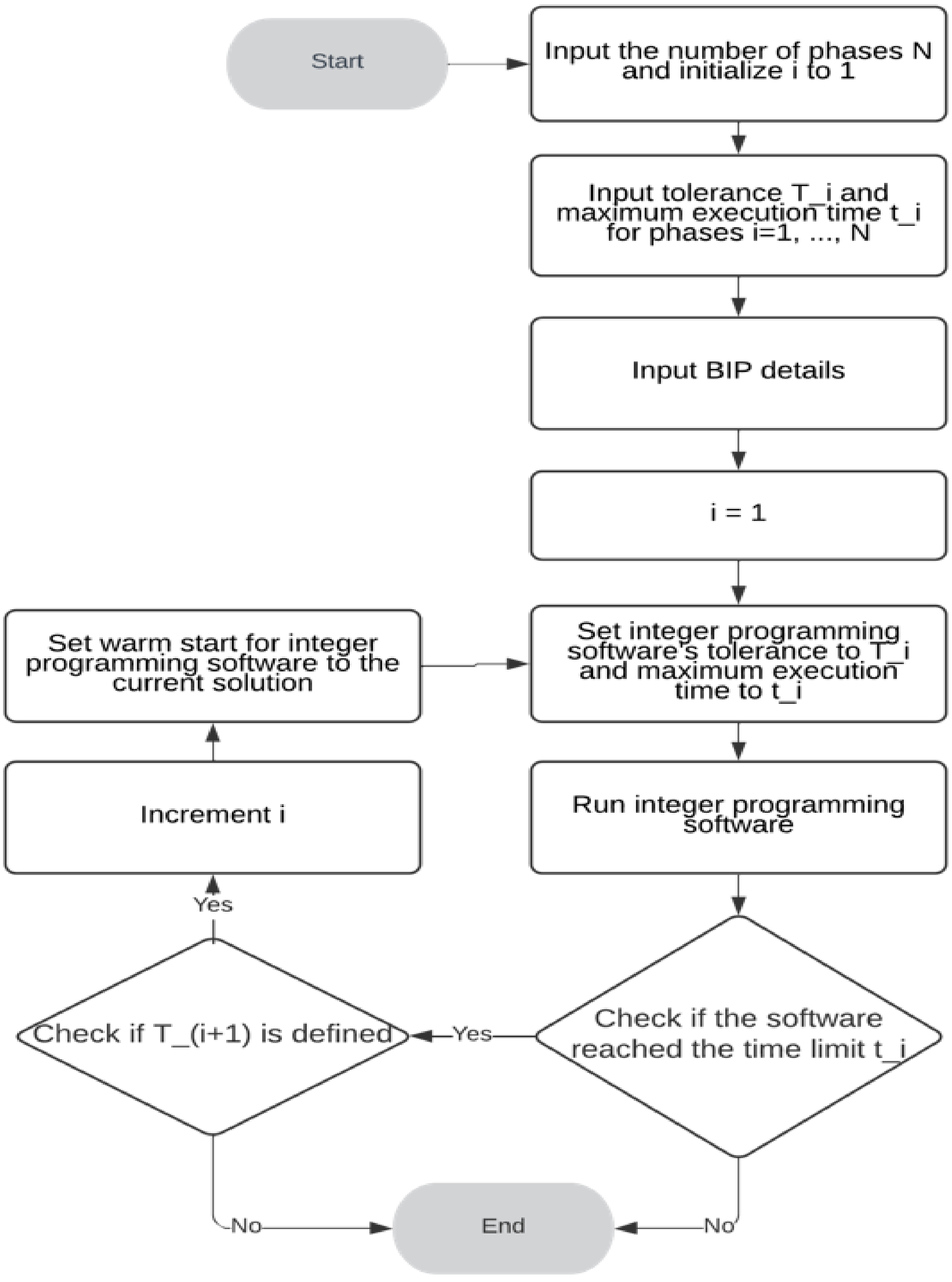
Start
Input the number of phases N and initialize i to 1
Input tolerance T_i and maximum execution time t_i for phases i=1, ..., N
Input BIP details
i = 1
Set integer programming software's tolerance to T_i and maximum execution time to t_i
Set warm start for integer programming software to the current solution
Run integer programming software
Increment i
Yes
Check if T_(i+1) is defined
Yes
Check if the software reached the time limit t_i
No
End
No

Figure 3. SSIT flowchart [16]

The ability to quickly find templates and models for various problems and the ease of getting a problem running with pre-defined defaults that work well with many problems saves the practitioner time writing extensive code and testing different parameter settings. Additionally, for industrial systems that use SSIT, the performance of these systems is “automatically” improved when new versions of the optimization software are installed.

A few scenarios will now be given to illustrate the nature of SSIT when applied to solve MDMKPs. More details are provided in [5]. It is important to remember that there is no need to “optimize” either the number of tolerances used or their values and the execution times for each tolerance. These values are both user and problem specific and can be easily adjusted to meet the users’ needs! This point will now be illustrated with a few examples.

Suppose that the OR practitioner is trying to solve MDMKPs in Category A, i.e., that should be relatively easy to solve with CPLEX. Then a tolerance-execution time sequence of 0.0001 for 60 seconds, 0.001 for 180 seconds, 0.003 for 180 seconds, 0.007 for 180 seconds, and 0.01 for 180 seconds might be appropriate.

In a worst-case scenario, SSIT would terminate in 780 seconds, and the CPLEX gap between the best solution found and the tightest upper bound would be reported. However, as shown in the next section, most (84%) of the 1050 MDMKPs in Category A terminated when the second tolerance (0.001) was initiated.

Now suppose that the OR practitioner is trying to solve MDMKPs in Category C, i.e., that should be rather difficult to solve. Then a tolerance-execution time sequence of 0.005 for 180 seconds, 0.01 for 600 seconds, 0.02 for 600 seconds, and 0.05 for 600 seconds might be appropriate. Now, in a worst-case scenario, SSIT would terminate in 1980 seconds and the CPLEX gap between the best solution found and the tightest upper bound would be reported.

As the difficulty of the MDMKPs increase (based on category), the OR practitioner should expect that the execution times will increase and that bounds on the solutions generated will get looser. However, there are three major benefits for the OR practitioner who uses SSIT customized for each category to solve MDMKPs. First, there is absolutely no need for the time-consuming task of developing and coding an algorithm specifically designed to solve MDMKPs. Second, SSIT will provide bounds that guarantee how close the solution is to the optimum. Lastly, if the OR practitioner is not satisfied with the SSIT solution, the OR practitioner can adjust the tolerance-execution time sequence to either reduce total execution time or to improve the bound on the solution.

RESULTS AND DISCUSSION

In this section, the 1620 MDMKP test instances will be used to evaluate the performance of different SSIT strategies designed specifically for MDMKPs in each of the four categories: A, B, C, and D. These results will be compared with a simple strategy commonly used by OR practitioners; specifically, inputting the problem to CPLEX using all default parameter values and executing it up to some maximum time (one hour). If the maximum time is reached, the best solution obtained is the answer and its quality is measured by the final CPLEX gap between the best upper bound and the best solution generated. It’s important to note that compared to just using CPLEX at the default tolerance, the increasing tolerances in SSIT allow CPLEX to be more aggressive when pruning the branch-and-cut tree, thus decreasing the gap faster. All executions of CPLEX were on a compute server with the following specifications: an Intel(R) Xeon(R) CPU E5-2640 v3 processor, 32 GiB of RAM and CentOS Linux 7. The threads parameter was set to one in all cases.

1. SSIT Applied to Solve 1050 Category A MDMKPs

This subsection will report results using SSIT to solve the 1050 Category A MDMKPs. The SSIT results will be compared to the results obtained by executing CPLEX for up to one hour using the default tolerance of 0.0001. The SSIT strategy suggested in Section 6 for MDMKPs predicted to be in Category A will be used. Specifically, the tolerance-execution time sequence of 0.0001 for 60 seconds, 0.001 for 180 seconds, 0.003 for 180 seconds, 0.007 for 180 seconds, and 0.01 for 180 seconds will be used.

In a worst-case scenario, SSIT would terminate in 780 seconds and the CPLEX gap between the best solution found and the tightest upper bound would be reported. The results for these 1050 MDMKPs are summarized in Tables 1. The results of using the simple strategy for the Category A MDMKPs are summarized in Tables 2. These tables contain, by dataset and case (see problem definitions in Section Method), the number of problems solved, the average execution times, and the average gaps between the best upper bound and the best solutions generated.

For these 1050 Category A MDMKPs, SSIT had an average execution time of 102 seconds and an average gap from the optimum of 0.152%. For the simple strategy the average execution time was 711 seconds and the average gap from the optimum was 0.034%. The SSIT execution times were only 14% of the execution times of the simple strategy and still had an average gap less than 0.2%.

Table 1. Summary of SSIT results for 1050 category A MDMKPs datasets 1 to 9

	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case 5**	**Case 6**
Dataset 1						
# problems	30	30	30	30	30	30
Aver time	12.5	9.5	19.4	4.6	16.4	78.3
Aver gap	0.013%	0.010%	0.016%	0.007%	0.012%	0.090%
Dataset 2						
# problems	30	30	30	15	15	15
Aver time	81.1	91.3	103.0	0.3	0.5	0.6
Aver gap	0.076%	0.081%	0.093%	0.009%	0.007%	0.006%
Dataset 3						
# problems	30	30	30	15	15	15
Aver time	80.5	84.8	64.5	2.0	12.9	14.2
Aver gap	0.068%	0.077%	0.069%	0.009%	0.016%	0.013%
Dataset 4						
# problems	30	30	30	30	30	30
Aver time	156.0	252.8	271.6	80.4	320.7	146.0
Aver gap	0.189%	0.422%	0.412%	0.072%	0.698%	0.359%
Dataset 5						
# problems	15	15	15	15	15	15
Aver time	118.6	174.5	169.9	2.7	44.2	28.0
Aver gap	0.085%	0.184%	0.165%	0.010%	0.044%	0.024%
Dataset 6						
# problems	15	15	15	15	15	15
Aver time	120.9	128.7	124.9	21.5	70.5	76.1
Aver gap	0.122%	0.123%	0.127%	0.026%	0.070%	0.072%
Dataset 7						
# problems	30	15	15	30	15	15
Aver time	373.1	30.6	32.2	257.9	1.0	0.9
Aver gap	0.943%	0.022%	0.028%	0.506%	0.005%	0.002%
Dataset 8						
# problems	5	5	5	15	15	15
Aver time	48.5	166.0	153.9	56.8	138.8	123.3
Aver gap	0.046%	0.174%	0.100%	0.041%	0.163%	0.183%
Dataset 9						
# problems	5	5	5	15	15	15
Aver time	60.3	142.1	68.6	112.0	145.7	140.4
Aver gap	0.087%	0.108%	0.097%	0.139%	0.155%	0.162%

To get a better feel for how SSIT performed on these 1050 MDMKPs, Table 3 provides the number of MDMKPs that terminate at each of the tolerances. It can be observed from this table that 84% of the 1050 MDMKPs terminated by the time the tolerance was loosened to 0.001. Also, only 41 (4%) MDMKPs reached the time limit and had a gap greater than 0.01.

Table 2. Summary of simple strategy results for 1050 category A MDMKPs datasets 1 to 9

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Dataset 1						
# problems	30	30	30	30	30	30
Aver time	12.0	9.6	17.1	4.6	13.0	36.6
Aver gap	0.010%	0.010%	0.010%	0.007%	0.006%	0.005%
Dataset 2						
# problems	30	30	30	15	15	15
Aver time	685.9	709.9	873.5	0.3	0.5	0.6
Aver gap	0.011%	0.013%	0.023%	0.009%	0.007%	0.006%
Dataset 3						
# problems	30	30	30	15	15	15
Aver time	1692.5	2033.4	1868.1	2.0	14.9	15.9
Aver gap	0.023%	0.032%	0.026%	0.009%	0.010%	0.007%
Dataset 4						
# problems	30	30	30	30	30	30
Aver time	402.0	503.5	513.7	49.5	665.8	129.1
Aver gap	0.022%	0.018%	0.016%	0.007%	0.088%	0.005%
Dataset 5						
# problems	15	15	15	15	15	15
Aver time	275.1	1665.7	1555.2	2.6	49.7	23.5
Aver gap	0.010%	0.021%	0.052%	0.010%	0.007%	0.006%
Dataset 6						
# problems	15	15	15	15	15	15
Aver time	2556.0	3177.2	3381.5	34.3	643.6	1182.7
Aver gap	0.053%	0.065%	0.069%	0.010%	0.010%	0.018%
Dataset 7						
# problems	30	15	15	30	15	15
Aver time	1784.1	22.9	21,1	470.8	1.0	0.9
Aver gap	0.471%	0.010%	0.010%	0.042%	0.005%	0.002%
Dataset 8						
# problems	5	5	5	15	15	15
Aver time	72.9	1981.9	1058.1	64.9	362.3	623.1
Aver gap	0.010%	0.017%	0.010%	0.010%	0.007%	0.021%
Dataset 9						
# problems	5	5	5	15	15	15
Aver time	2188.6	3600	3600	1549.8	2204.6	1200.4
Aver gap	0.017%	0.072%	0.049%	0.027%	0.072%	0.100%

Table 3. Terminating tolerances for the 1050 category A MDMKPs

Tolerance	0.0001	0.001	0.003	0.007	0.01	Time limit
# MDMKPs	637	244	85	31	12	41

2. SSIT Applied to Solve 330 Category B MDMKPs

This subsection will report results using SSIT to solve the 330 Category B MDMKPs. The SSIT results will be compared to the results obtained by executing CPLEX for up to one hour using the standard default tolerance of 0.0001. The SSIT strategy employed considers that the Category B problems are more complex than the Category A MDMKPs. Specifically, for Category B MDMKPs the tolerance-execution time sequence of 0.001 for 180 seconds, 0.003 for 180 seconds, 0.005 for 180 seconds, 0.008 for 180 seconds, 0.01 for 300 seconds and 0.02 for 300 seconds will be used. In a worst-case scenario, SSIT would terminate in 1320 seconds and the CPLEX gap between the best solution found and the tightest upper bound would be reported. For Category B, because of the expected difficulty of the problems, the tolerances have been loosened and the maximum execution time increased.

The results for these 330 MDMKPs are summarized in Table 4. The results from using the simple strategy for the Category B MDMKPs are summarized in Table 5. These tables contain, by dataset and case (see problem definitions in Section 3), the number of problems solved, the average execution times, and the average gaps between the best upper bound and the best solutions generated. It should be noted that blank entries in Tables 4 and 5 indicate no Category B MDMKPs in that dataset-case combination. For example, there are no entries for Cases 1, 2, and 3 in Dataset 2. For datasets like 1, 4, and 7 that have no MDMKPs in Category B, these datasets do not appear at all in Tables 4 and 5.

For these 330 Category B MDMKPs, SSIT had an average execution time of 464 seconds and an average gap from the optimum of 0.645%. For the simple strategy, the average execution time was 3363 seconds, and the average gap from the optimum was 0.439%. Similar to Category A results, the SSIT execution times were only 14% of the execution times of the simple strategy and had an average gap of less than 0.7%.

Table 4. Summary of SSIT results for 330 category B MDMKPs datasets 2, 3, 5, 6, 8, and 9

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Dataset 2						
# problems				15	15	15
Aver time	-	-	-	138.1	272.1	517.5
Aver gap				0.206%	0.371%	0.690%
Dataset 3						
# problems				15	15	15
Aver time	-	-	-	128.1	190.6	364.7
Aver gap				0.164%	0.219%	0.472%
Dataset 5						
# problems	15	15	15	15	15	15
Aver time	359.2	559.6	509.6	443.2	816.6	978.9
Aver gap	0.491%	0.740%	0.716%	0.584%	1.300%	1.361%
Dataset 6						
# problems	15	15	15	15	15	15
Aver time	239.6	269.4	231.7	263.2	574.0	712.9
Aver gap	0.239%	0.334%	0.300%	0.338%	0.692%	1.019%
Dataset 8						
# problems	15			15		
Aver time	748.3	-	-	733.8	-	-
Aver gap	1.180%			1.172%		
Dataset 9						
# problems	15			15		
Aver time	558.2	-	-	589.9	-	-
Aver gap	0.622%			0.703%		

Table 5: Summary of simple strategy results for 330 category B MDMKPs datasets 2, 3, 5, 6, 8, and 9

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Dataset 2						
# problems				15	15	15
Aver time	-	-	-	697.6	2800.0	2728.8
Aver gap				0.010%	0.095%	0.349%
Dataset 3						
# problems				15	15	15
Aver time	-	-	-	3036.4	3600	3600
Aver gap				0.068%	0.129%	0.297%
Dataset 5						
# problems	15	15	15	15	15	15
Aver time	3600	3600	3600	3514.6	3600	3600
Aver gap	0.304%	0.511%	0.469%	0.332%	0.911%	1.087%
Dataset 6						
# problems	15	15	15	15	15	15
Aver time	3600	3600	3600	3600	3600	3600
Aver gap	0.170%	0.233%	0.220%	0.228%	0.547%	0.864%
Dataset 8						

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
# problems	15			15		
Aver time	3600	-	-	3600	-	-
Aver gap	0.873%			0.825%		
Dataset 9						
# problems	15			15		
Aver time	3600	-	-	3600	-	-
Aver gap	0.552%			0.576%		

Analogous to the results in Table 3, Table 6 provides the number of MDMKPs that terminate at each tolerance. It can be observed from this table that 59% of the 330 MDMKPs terminated by the time the tolerance that was loosened to 0.005. Also, only 4 (1.2%) MDMKPs reached the time limit and had a gap greater than 0.02.

Table 6: Terminating tolerances for the 330 category B MDMKPs

Tolerance	**0.001**	**0.003**	**0.005**	**0.008**	**0.01**	**0.02**	**Time limit**
# MDMKPs	16	98	82	60	28	42	4

3. SSIT Applied to Solve 210 Category C MDMKPs

This subsection will report results using SSIT to solve the 210 Category C MDMKPs. Keep in mind that Song et al. [4] found it very difficult to obtain solutions for Category C problems even after one hour of execution time at a tolerance of 0.001. As done previously, the SSIT results will be compared to the results obtained by executing CPLEX for up to one hour using the standard default tolerance of 0.0001. The SSIT strategy used was mentioned in the previous section and takes into account that the Category C problems are even more complex than the Category B MDMKPs. Specifically, for Category C MDMKPs the tolerance-execution time sequence of 0.005 for 180 seconds, 0.01 for 600 seconds, 0.02 for 600 seconds, and 0.05 for 600 seconds will be used. In a worst-case scenario, SSIT would terminate in 1980 seconds, and the CPLEX gap between the best solution found and the tightest upper bound would be reported. For Category C, because of the expected difficulty of the problems, the tolerances have been loosened, and the maximum execution time increased.

The results for these 210 MDMKPs are summarized in Table 7. The results from using the simple strategy for the Category C MDMKPs are summarized in Table 8. These tables contain, by dataset and case (see problem definitions in Section 3), the number of problems solved, the average execution times, and the average gaps between the best upper bound and the best solutions generated. The blank entries in Tables 7 and 8 indicate no Category C MDMKPs in that dataset-case combination. None of the MDMKPs from datasets 1, 2, 3, 4, 5, and 6 are in Category C.

For these 210 Category C MDMKPs, SSIT had an average execution time of 1048 seconds and an average gap from the optimum of 4.016%. The average execution time was 3546 seconds for the simple strategy, and the average gap from the optimum was 3.193%. The SSIT execution times were only 30% of the execution times of the simple strategy and had an average gap of about 4%. Although being guaranteed to be within 4% of the optimums might not seem like a tightly bound and the CPLEX execution time of 1048 seconds (over 17 minutes) might seem substantial, one must keep in mind that the Category C problems were identified in [4] as problems that could **not** be solved in 3600 seconds with CPLEX at a fixed tolerance of 0.001.

Analogous to the results in Tables 3 and 6, Table 9 provides the number of MDMKPs that terminate at each tolerance. It can be observed from this table that 45% 0f the 210 MDMKPs terminated by the time the tolerance was loosened to 0.02. Also, only 48 (23%) MDMKPs reached the time limit and had a gap greater than 0.05.

Table 7: Summary of SSIT results for 210 category C MDMKPs
datasets 7, 8, and 9

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Dataset 7						
# problems		15			15	
Aver time	-	1603.5	-	-	1980.0	-
Aver gap		5.942%			14.547%	
Dataset 8						
# problems	10	25	25		15	15
Aver time	76.3	684.2	874.8	-	1717.2	1940.1
Aver gap	0.632%	1.916%	2.518%		5.887%	9.847%
Dataset 9						
# problems	10	25	26		15	15
Aver time	14.7	524.6	658.8	-	1224.2	1578.6
Aver gap	0.420%	1.125%	1.550%		3.129%	4.317%

Table 8: Summary of simple strategy results for 210 category C MDMKPs
datasets 7, 8, and 9

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Dataset 7						
# problems		15			15	
Aver time	-	3600	-	-	3600	-
Aver gap		4.200%			12.172%	
Dataset 8						
# problems	10	25	25		15	15
Aver time	2467.2	3600	3600	-	3600	3600
Aver gap	0.125%	1.340%	2.036%		4.967%	7.987%
Dataset 9						
# problems	10	25	26		15	15
Aver time	3600	3600	3600	-	3600	3600
Aver gap	0.231%	0.852%	1.183%		2.658%	3.455%

Table 9: Terminating tolerances for the 210 Category C MDMKPs

Tolerance	0.005	0.01	0.02	0.05	Time limit
# MDMKPs	44	21	30	67	48

4. SSIT Applied to Solve 30 Category D MDMKPs

It is important to note that the goal of this article is to demonstrate how a commercial software package like CPLEX (or Gurobi or others) can be used in an iterative manner to efficiently generate guaranteed bounded solutions for MDMKPs. A methodology has been presented that requires **no** algorithm development and coding by the OR practitioner. This solution approach called the simple sequential increasing tolerance (SSIT) methodology in conjunction with the software package CPLEX has been tailored to solve MDMKPs based on the difficulty classification of the MDMKP. For 1620 MDMKP test instances, SSIT has successfully solved 1590 (98%) of these MDMKPs in a manner that can easily be used by OR practitioners and anyone needing to solve MDMKPs.

Now what strategy can be easily used by OR practitioners to solve the 30 most difficult problems? Other researchers such as Arntzen et al [7] and Lai et al. [11] have successfully developed highly complex MDMKP-specific algorithms, but these are nontrivial to computer code and typically require parameter fine-tuning. The question remains: how do OR practitioners solve MDMKPs that are classified in Category D?

For OR practitioners that need to solve such problems and implement their solutions, reporting to management that the integer programming software did not find a feasible solution after four hours of execution time is not acceptable (the authors of this article executed several

Category D problems using both a SSIT strategy and the simple strategy discussed in this article for four hours without obtaining a feasible solution). One could try to execute either a SSIT strategy or simply execute CPLEX for a longer time, like 12 or 24 hours. However, there is no guarantee of finding a feasible solution for a given real-world problem even with the long execution time. The authors suggest an approach that is common among OR practitioners in this situation and is documented in Vasko et al [19]. The OR practitioner would go back to the client and discuss if any of the constraints could be loosened. As illustrated in [19], this process is iterative requiring several discussions with the client before the final parameters for the model are determined.

To simulate this situation for the 30 Category D MDMKPs, all right-hand sides of the demand constraints were reduced by 10% and executed using the SSIT strategy for Category C. All 30 problems now obtained feasible bounded solutions. Next, the original demand right-hand sides were only reduced by 5% and again all 30 problems now obtained feasible bounded solutions. This illustrates a practical way to successfully handle real-world applications modeled as MDMKPs and classified in Category D.

In the next subsection, statistical analyses will compare the SSIT results to the simple single-pass execution of CPLEX in terms of execution time and solution quality for MDMKPs in categories A, B, and C—98% of the test instances.

5. Statistical Analysis

To compare SSIT with the simple strategy in terms of execution time (the elapsed time) and solution quality (the lowest gap), statistical analyses are conducted for the 1590 MDMKPs in Categories A, B, and C. To properly compare SSIT and the simple strategy, the one-way repeated measures multivariate analysis of variance (MANOVA) is adopted because the two response variables (the elapsed time and the lowest gap) are correlated of each other (if they are not correlated, the analysis of variance (ANOVA) could have been used), and all instances are used two times for both SSIT and the simple strategy [20]. Three separate analyses by category are conducted at the common significant level of 0.05.

Tables 10, 11, and 12 (for Categories A, B, and C, respectively) illustrate that the lowest gap obtained by using SSIT is significantly greater than that of the simple strategy, but the elapsed time of SSIT is significantly smaller than that of the simple strategy in all categories. (The corresponding p-values are almost 0). In short, there is a trade-off between the elapsed time and the smallest gap as we expected.

Table 10: Comparison between SSIT and the simple strategy in Category A

	Tests of Within-Subjects Contrasts						
Source	Measure	method	Type III Sum of Squares	df	Mean Square	F	Sig.
method	gap	Linear	.001	1	.001	123.460	.000
	time	Linear	194746100.946	1	194746100.946	262.541	.000
Error (method)	gap	Linear	.006	1049	5.907E-006		
	time	Linear	778119575.525	1049	741772.713		

Table 11: Comparison between SSIT and the simple strategy in Category B

	Tests of Within-Subjects Contrasts						
Source	Measure	method	Type III Sum of Squares	df	Mean Square	F	Sig.
method	gap	Linear	.001	1	.001	430.087	.000
	time	Linear	1386774685.305	1	1386774685.305	4853.659	.000
Error (method)	gap	Linear	.001	329	1.631E-006		
	time	Linear	94001006.891	329	285717.346		

Table 12: Comparison between SSIT and the simple strategy in Category C

	Tests of Within-Subjects Contrasts `						
Source	Measure	method	Type III Sum of Squares	df	Mean Square	F	Sig.
method	gap	Linear	.007	1	.007	96.530	.000
	time	Linear	655129998.402	1	655129998.402	2283.641	.000
Error (method)	gap	Linear	.015	209	7.366E-005		
	time	Linear	59957832.290	209	286879.580		

CONCLUSION

In Song et al. [4] multi-demand multidimensional knapsack problems (MDMKPs) were classified into three general categories based on their predicted difficulty to solve using CPLEX. Category A consists of problems that are expected to be easy to solve, Category B consists of problems that are somewhat difficult to solve, and Category C consists of problems that are difficult to solve on a standard PC. However, no specific strategies for efficiently generating solutions guaranteed to be close to the optimums were given.

In this article, Category C problems were further partitioned into either difficult to solve (Category C) problems or problems that are very difficult to solve or even get feasible solutions (Category D). Next, solution strategies customized to each of the four problem categories were developed. These strategies are based on a procedure that iteratively uses commercial integer programming software (CPLEX in this article) with no algorithm-specific code required. This multi-pass methodology is used in conjunction with any integer programming software package and employs a sequence of increasing tolerances that is used with the integer programming software. If a goal bound on the solution is not achieved in a user-defined time interval, the best solution found at one tolerance is then input as a starting solution for the next looser tolerance.

This procedure called the simple sequential increasing tolerance (SSIT) methodology has been empirically shown, regardless of which category an MDMKP was classified, to generate bounded solutions much quicker than just using CPLEX in a single-pass execution mode. Although SSIT is very intuitive, this is the first article to discuss and quantify the benefits of using SSIT specifically to solve MDMKPs. Specifically, for the 1050 MDMKPs (65%) in Category A, SSIT was able to, on average, generate solutions bounded within 0.152% of the optimums in 102 seconds. This SSIT result is an 86% reduction in CPLEX execution time over a single-pass CPLEX run (maximum time of 3600 seconds) which required, on average, 711 seconds and obtained solutions guaranteed within 0.034% of the optimums. For the 330 MDMKPs (20%) in Category B, SSIT was able to, on average, generate solutions bounded within 0.645% of the optimums in 464 seconds. This SSIT result is an 86% reduction in CPLEX execution time over a single-pass CPLEX run (maximum time of 3600 seconds) which required, on average, 3363 seconds and obtained solutions guaranteed within 0.439% of the optimums. Finally, for the 210 MDMKPs (13%) in Category C, SSIT was able to, on average, generate solutions bounded within 4.016% of the optimums in 1048 seconds. This SSIT result is a 70% reduction in CPLEX execution time over a single-pass CPLEX run (maximum time of 3600 seconds) which required, on average, 3546 seconds and obtained solutions guaranteed within 3.193% of the optimums.

The implications of the strategies discussed in this article are significant for OR practitioners. Suppose that an OR practitioner is charged with developing and implementing a solution for a real-world application requiring MDMKPs. Instead of trying to code, test, and implement a state-of-the-art algorithm from the literature, the OR practitioner would first determine the category classification of the MDMKP using the decision tree diagram in Figure 2 in this article. Next, if the MDMKP was classified in either Category A, B, or C, then the OR practitioner would use the SSIT strategy specific to that category in conjunction with the best integer programming software available to solve the MDMKP. If the MDMKP was classified in Category D, then the OR practitioner could try the SSIT strategy for Category C, but if unsuccessful in getting a feasible solution in an acceptable amount of time (situation dependent), then the OR practitioner should suggest to the client that modifications are needed to the problem formulation. After modifications acceptable to the client are made, the modified MDMKP can be solved using the SSIT strategy for Category C. It has been experienced that modifications of the model formulation will be an iterative process. Finally, an additional benefit of using SSIT is that the application's performance will continue to "automatically" improve as new versions of the commercial software are implemented.

REFERENCES

[1] P. Cappanera, and M. Trubian, "A Local-search-based heuristic for the demand-constrained multidimensional knapsack problem." *INFORMS Journal on Computing* 17(1): 82-98, 2005. Doi:10.1287/ijoc.1030.0050.

[2] P. Cappanera, G. Gallo, and F. Maffioli, "Discrete facility location and routing of obnoxious activities." *Discrete Applied Mathematics* 13: 3–28, 2004. doi:10.1016/S0166-218X(03)00431-1.

[3] G. J. Beaujon, S. P. Marin, and G. C. McDonald, "Balancing and optimizing a portfolio of r&d projects". *Naval Research Logistics* 48(1):18–40, 2001. doi: 10.1002/1520-6750(200102)48:1<18::AID-NAV2>3.0.CO;2-7

[4] M. S. Song, B. Emerick, Y. Lu, and F. J. Vasko., "When to use integer programming software to solve large Multi-demand multidimensional knapsack problems: a guide for operations research practitioners," In *Engineering Optimization*, 1-13, 2021. doi: 10.1080/0305215X.2021.1933965

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

[6] Y. Lu, and F. J. Vasko., "A Comprehensive empirical demonstration of the impact of choice constraints on solving generalizations of the 0–1 knapsack problem using the integer programming option of CPLEX®." *Engineering Optimization*. 52(9): 1632-1644, 2020. doi: 10.1080/0305215X.2019.1658748

[7] H. A. Arntzen, L. M. Hvattum, and A. Lokketangen, "Adaptive memory search for multidemand multidimensional knapsack problems." *Computers & Operations Research* 33(9): 2508-2525, 2006. doi.org/10.1016/j.cor.2005.07.007.

[8] L. M. Hvattum, and A. Løkketangen, "Experiments using scatter search for the multidemand multidimensional knapsack problem." In: *Metaheuristics. operations research/computer science interfaces series 39*. vol 39. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-71921-4_1

[9] L. M. Hvattum, H. Arntzen, A Løkketangen, and F. Glover, "Alternating control tree search for knapsack/covering problems." *Journal of Heuristics* 16 (3):239–258, 2010. doi: 10.1007/s10732-008-9100-4.

[10] S. R. Balachandar, and K. Kannan, "A new heuristic approach for knapsack/covering problem." *International Journal of Mathematical Sciences and Applications* 1(2): 593-606, 2011. http://ijmsa.yolasite.com/resources/16-may.pdf.

[11] X. Lai, J. K. Hao, J.K, and D. Yue, "Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem." *European Journal of Operational Research* 274: 35-48, 2019. doi: 10.1016/j.ejor.2018.10.001

[12] S. Al-Shihabi. "A novel core-based optimization framework for binary integer programs-the multi-demand multidimesional knapsack problem as a test problem", *Operations Research Perspectives* 8: 100182, 2021.

[13] P. Chu, and J. E. Beasley, “A Genetic algorithm for the multidimensional knapsack problem.” *Journal of Heuristics* 4: 63-86, 1998.

[14] J. E. Beasley, “OR-Library”. Brunel University of London. Accessed August 15, 2016. http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html.

[15] G. James, D. Witten, T. Hastie, and T. Tibshirani, *An Introduction to Statistical Learning: with Applications in R.*1st edition. *Springer,* 2013*.*

[16] B. McNally, Y. Lu, E. Shively-Ertas, M. S. Song, and 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* 8(2) 76-95, 2021. doi:10.18488/journal.76.2021.82.76.95.

[17] Y. Lu, B. McNally, E. Shively-Ertas, and 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,* 15, 1650-1656, 2021. doi: 10.46300/9106.2021.15.178.

[18] A. Dellinger, Y. Lu, B. McNally, M. S. Song, and 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,* 1, 13-34, 2021.

[19] F. J. Vasko, F. E. Wolf, and K. L. Stott, ″Optimal selection of ingot sizes via set covering″. *Operations Research* 35(3): 346-353, 1987. doi:10.1287/opre.35.3.346

[20] T. W. Anderson. *An Introduction to Multivariate Statistical Analysis.* 3rd edition*.* Hoboken, NJ: John Wiley, 2003.