Research Reports on Computer Science
https://ojs.wiserpub.com/index.php/RRCS/

UNIVERSAL WISER PUBLISHER

Research Article

A Simple and Efficient Technique to Generate Bounded Solutions for the Generalized Assignment Problem: A Guide for OR Practitioners

Anthony Dellinger[1], Yun Lu[2], Bryan McNally[1], Myung Soon Song[2], Francis J. Vasko[2*]

[1]Computer Science Department, Kutztown University, Kutztown, PA, USA
[2]Department of Mathematics, Kutztown University, Kutztown, PA, USA
E-mail: vasko@kutztown.edu

Received: 14 July 2021; **Revised:** 22 September 2021; **Accepted:** 25 October 2021

Abstract: The generalized assignment problem (GAP) is a NP-hard problem that has a large and varied number of important applications in business and industry. Approximate solution approaches for the GAP do not require excessive computation time, but typically there are no guarantees on solution quality. In this article, a methodology called the simple sequential increasing tolerance (SSIT) matheuristic that iteratively uses *any* general-purpose integer programming software is discussed. This methodology uses a sequence of increasing tolerances in conjunction with optimization software to generate a solution that is guaranteed to be within a specified percentage of the optimum in a short time. SSIT requires no problem-specific coding and can be used with any commercial optimization software to generate bounded solutions in a timely manner. Empirically, SSIT is tested on 51 GAP instances (24 medium and 27 large) in the literature. The performance of CPLEX versus Gurobi on these 51 GAP test instances is also statistically analyzed.

***Keywords*:** matheurstic, CPLEX, Gurobi, generalized assignment problem, bounded solutions

1. Introduction

The generalized assignment problem (GAP) is an interesting NP-hard combinatorial optimization problem (COP) that can be considered both as an assignment problem as well as a knapsack-type problem. As an assignment problem, the GAP seeks an optimal assignment of n jobs to m agents where each agent has a maximum time to process jobs. Alternatively, the GAP can be viewed as a knapsack-type problem in which each of n items must be inserted into exactly one of m knapsacks. Each of the n items has a weight specific to each knapsack and each knapsack has a maximum weight that cannot be exceeded. Both the minimization and maximization type objective functions are used for the GAP. In this article, without loss of generality, the focus will be on the minimization GAP. The mathematical formulation for the GAP will be described in terms of the knapsack interpretation.

Define the binary variable x_{ij} equal to 1 if and only if item j is assigned to knapsack i. The formulation of the minimization case of the GAP is the following.

DOI: https://doi.org/10.37256/rrcs.1120221039

$$\text{Minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{1}$$

$$\text{Subject to} \quad \sum_{j=1}^{n} a_{ij}x_{ij} \le b_i, \qquad i = 1, \ldots, m \tag{2}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad j = 1, \ldots, n \tag{3}$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, \ldots, m \quad \text{and} \quad j = 1, \ldots, n \tag{4}$$

where c_{ij} is the cost of assigning item j to knapsack i, b_i is the capacity of knapsack i and a_{ij} is the weight of item j if assigned to knapsack i. The objective function (1) is to minimize the total assignment cost of items (jobs) to knapsacks (agents). Constraints (2) denote the capacity availability restriction of each knapsack (agent) and are simply referred to as the capacity constraints. Constraints (3) ensure that each item (job) is assigned to exactly one knapsack (agent) and are known as semi-assignment constraints. Finally, constraints (4) ensure that each decision variable is binary.

The GAP formulation has been used to model and solve a variety of real-world problems in many areas including scheduling, transportation and routing, telecommunications, production planning, location, supply chain and logistics, and computer processor allocation. An extensive discussion of real-world applications of the GAP can be found in Oncan [1]. In Oncan [1] Table 1, entitled "Some of the real-life applications of the GAP", lists information pertaining to 36 real-world applications of the GAP.

Since there are numerous real-world applications of the GAP, it is important to provide operations research (OR) practitioners with simple, effective and efficient solution approaches to solve these problems. However, it is even more important that OR practitioners can present to management for implementation, systems that generate guaranteed high-quality solutions. This is true regardless if the OR practitioner is called on to assist with a critical strategic planning issue or 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 [2] 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 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. This procedure, called the simple sequential increasing tolerance (SSIT) matheuristic, will be shown, using 51 GAP from the literature, to quickly generate solutions that are guaranteed to be very close to the optimums. This multi-pass matheuristic is used in conjunction with an integer programming software (both CPLEX and Gurobi were tested) package and employs a sequence of increasing tolerances that are 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.

SSIT was first discussed in McNally [3], 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 (CPLEX and Gurobi will both be 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, no problem-specific algorithm is required. 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.

The rest of this article is organized as follows. In Section 2, the relevant generalized assignment problem literature will be reviewed. The reader should be aware that there is a large number of articles in the literature that deal with extensions and topics peripheral to the GAP. These are not discussed in the article. In Section 3, a brief overview of the SSIT matheuristic will be provided. In Section 4, empirical results from applying the SSIT matheuristic to solve 51 GAP instances with both the CPLEX and Gurobi software packages will be presented. In Section 5, a statistical analysis comparing the performances of CPLEX and Gurobi will be discussed. In Section 6, this article will close with conclusions and suggested future work.

2. Relevant literature on the GAP

2.1 *Mathematical programming-based solution approaches*

The solution approaches based on mathematical programming can be either exact or can be employed in a heuristic manner. Note that the best solution found by exact approaches is within a tight tolerance of the optimum if the program terminates before the maximum allowed time limit. A number of mathematical programming-based solution methods for the GAP have been suggested in the literature. Below are some recent examples applied to solve the GAP.

Woodcock and Wilson [4] discussed a hybrid methodology that combines tabu search with a branch-and-bound strategy to solve the GAP. The methodology uses commercial software (Xpress-MP) to solve sub-problems generated with the guidance of the tabu search. The linear programming relaxation of GAP is part of the branch-and-bound strategy and helps suggest which binary variables should be used. Posta and Ferlande [2] presented a variable fixing approach to solve the GAP. This is an exact procedure that reformulates the GAP into a sequence of decision problems that are solved using variable-fixing rules. The decision problems are solved using a depth-first lagrangian branch-and-bound method that makes use of variable-fixing rules to prune the search tree. Munapo et al. [5] proposed a transportation branch-and-bound algorithm for solving the GAP. This approach uses a transportation technique to solve sub-problems in the branch-and-bound algorithm. Sadykov et al. [6] used a column generation-based heuristic to solve the GAP. This heuristic uses a sophisticated combination of dual stabilization, piecewise linear penalty functions, Wentge's dual smoothing, and sub-gradient directional smoothing. Sadykov et al. [7] discussed the benefits of incorporating diving methods into primal heuristics for branch-and-price procedures. Their methodology is used to solve the GAP and a number of other combinatorial optimization problems. Bando and Kawasaki [8] show that the core of a GAP satisfies two types of stability properties. Their results add to the importance of the core in an assignment problem where agents' preferences may not be quasilinear. When an OR practitioner is faced with solving a large industrial GAP, all of the methodologies mentioned above are relatively complex to implement in order to solve a real-world GAP application. A considerable amount of time can be required for computer coding and testing of the algorithm selected and then computer execution time for the actual application can be significant.

2.2 *Approximate solution approaches*

There are many classic approximate solution approaches in the literature. For example, the classic application of a genetic algorithm (GA) to solve the GAP was proposed by Chu and Beasley [9]. A hybrid GA which included local search was proposed by Ahmed [10]. Zhi-Bin et al. [11] discuss how to effectively implement a parallel genetic algorithm in the graphics processing unit (GPU) environment. Specifically, the authors show how a GA can be efficiently parallelized in a GPU environment to solve the GAP. An Artificial Bee Colony (ABC) algorithm was applied to solve the GAP by Baykasoglu et al. [12]. A path relinking with ejection chains approach was proposed for solving the GAP by Yagiura et al. [13]. This algorithm generates new solutions by combining two or more reference solutions. It uses an ejection chain approach which is embedded in a neighborhood construction to create more complex and powerful moves.

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 GAP 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.

3. Overview of the simple sequential increasing tolerance matheuristic

The motivation 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, state-of-the-art optimization software instead of highly complex specialized codes for the particular 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 Figure 1.

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, and 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.

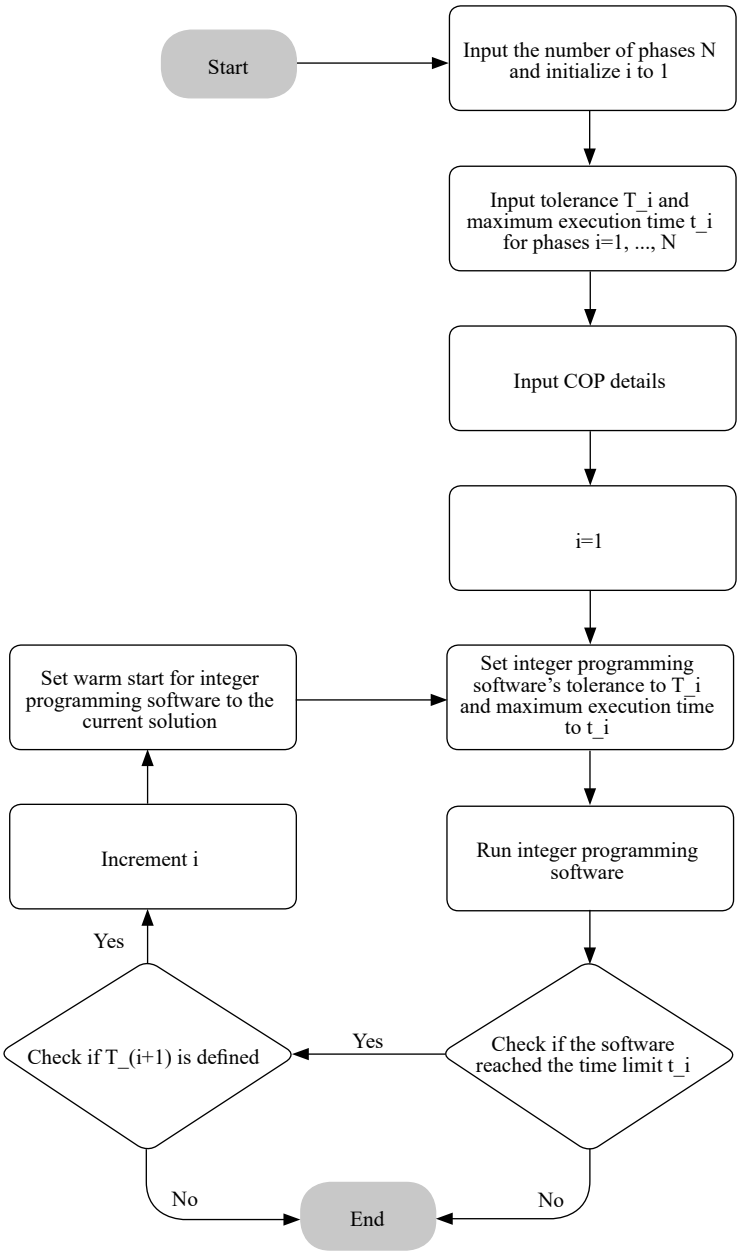

Figure 1. SSIT flowchart

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! A few scenarios will now be given to illustrate the nature of SSIT. More details are provided in McNally [3].

Suppose that, for a particular application, getting a solution that is guaranteed to be very close to the optimum is important and execution time is not top priority. In this case, the tolerance sequence might be set to 0.0001, 0.001, 0.004, 0.007, and 0.01 with the maximum execution time sequence of 300, 300, 300, 300, and 200 seconds respectively for a total possible execution time of 1,400 seconds. Furthermore, suppose, for example, a COP is solved with these SSIT settings and it terminates after 10 seconds at a tolerance of 0.007. This means that the best solution is guaranteed within 0.7% of the optimum and required a total of 910 seconds of execution time. Because SSIT terminated after only 10 seconds at a tolerance of 0.007, the user might want to see if this solution bounds the optimum within a tighter bound

of 0.4%. Checking this is very simple to do. Simply set the optimization software tolerance at 0.004 and the maximum execution time at say 1,200 seconds, and input the best solution found so far. If the optimization software terminates in less than 1,200 seconds, then the best solution found will be guaranteed to be within 0.4% of the optimum. On the other hand, if the software terminates because the maximum time limit of 1,200 seconds was reached, then the user still has a solution that is guaranteed to be within 0.7% of the optimum from the previous SSIT runs. Additionally, the software gap at termination will indicate how close the best SSIT solution is to the optimum, which will be less than 0.7%, but greater than 0.4%.

Alternatively, suppose a large amount of computations are being performed in a nightly production planning run and only 600 seconds are allocated per problem. In this case, a tolerance sequence of 0.001, 0.005, 0.01, 0.02, and 0.05 with the maximum execution time sequence of 200, 100, 100, 100, and 100 seconds respectively would be reasonable, since it requires at most a total execution time of 600 seconds.

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.

4. GAP empirical results

4.1 *Problem instances*

In this article, we will test SSIT on 51 GAP instances (types B, C, D, and E) that are available from Yagiura et al. [13]. These GAP instances are used by researchers to test algorithm performance. Type A instances from Beasley's OR-Library are not included here because they are small and easy to solve optimally with current software packages. Types D and E are more difficult to solve because the c_{ij} and a_{ij} are inversely correlated.

MEDIUM GAP instances: Total of 24 instances of types B, C, D and E with n (number of items or jobs) up to 200, where each type consists of six instances. Among them, types B, C and D instances were taken from Beasley's OR-Library [http://people.brunel.ac.uk/~mastjjb/jeb/orlib/gapinfo.html] and type E instances were taken from Yagiura's GAP instances [http://www-or.amp.i.kyoto-u.ac.jp/~yagiura/gap/].

LARGE GAP instances: Total of 27 instances of types C, D and E with n (number of items or jobs) up to 1600, where each type consists of nine instances. These instances were taken from Yagiura's GAP instances [http://www-or.amp.i.kyoto-u.ac.jp/~yagiura/gap/].

Details of these 51 GAP instances are given in Table 1.

The first number in the problem label is the number of items or jobs and the second number is the number of knapsacks or agents. Note that the number of variables in these 51 instances range from 500 to 128,000 and the number of constraints range from 105 to 1,680.

4.2 *SSIT results for the GAP*

In order to use the SSIT matheuristic to solve GAPs, a sequence of increasing tolerances and corresponding maximum execution times must be specified and an integer programming software package must be selected. The purpose of this article is to demonstrate that the SSIT matheuristic works effectively with any given integer programming software package. Obviously, the better the selected optimization software is in terms of generating good solutions quickly, the better the solution the SSIT matheuristic generates. Additionally, as new versions of the selected software are implemented, the performance of industrial applications will be improved with no input required from the OR practitioner who developed the application—another advantage for using SSIT.

In this article, the two leading optimization software packages: CPLEX (12.9) and Gurobi (9.1) will be used to solve the 51 GAP test instances in the three cases below.

BASE CASE: The 51 GAP instances are solved with a tolerance of 0.0001 and a maximum execution time of 3,600 seconds.

CASE 1: The 51 GAP instances are solved with the following five-step SSIT strategy (tolerances, time): 0.0001 for 150 seconds, 0.001 for 150 seconds, 0.004 for 300 seconds, 0.007 for 300 seconds, and 0.01 for 300 seconds.

Table 1. 51 GAP instances

24 medium sized problem instances			27 large sized problem instances		
Problem	# Variables	# Constraints	Problem	# Variables	# Constraints
B-100-5	500	105	C-400-10	4000	410
B-100-10	1000	110	C-400-20	8000	420
B-100-20	2000	120	C-400-40	16000	440
B-200-5	1000	205	C-900-15	13500	915
B-200-10	2000	210	C-900-30	27000	930
B-200-20	4000	220	C-900-60	54000	960
C-100-5	500	105	C-1600-20	32000	1620
C-100-10	1000	110	C-1600-40	64000	1640
C-100-20	2000	120	C-1600-80	128000	1680
C-200-5	1000	205	D-400-10	4000	410
C-200-10	2000	210	D-400-20	8000	420
C-200-20	4000	220	D-400-40	16000	440
D-100-5	500	105	D-900-15	13500	915
D-100-10	1000	110	D-900-30	27000	930
D-100-20	2000	120	D-900-60	54000	960
D-200-5	1000	205	D-1600-20	32000	1620
D-200-10	2000	210	D-1600-40	64000	1640
D-200-20	4000	220	D-1600-80	128000	1680
E-100-5	500	105	E-400-10	4000	410
E-100-10	1000	110	E-400-20	8000	420
E-100-20	2000	120	E-400-40	16000	440
E-200-5	1000	205	E-900-15	13500	915
E-200-10	2000	210	E-900-30	27000	930
E-200-20	4000	220	E-900-60	54000	960
			E-1600-20	32000	1620
			E-1600-40	64000	1640
			E-1600-80	128000	1680

CASE 2: The 51 GAP instances are solved with the following four-step SSIT strategy (tolerances, time): 0.001 for 300 seconds, 0.004 for 300 seconds, 0.007 for 300 seconds, and 0.01 for 300 seconds.

The Base Case is a methodology commonly used by OR practitioners in which the software is allowed to run up to a certain time. If the maximum time is reached, then the best solution is provided to the OR practitioner, and the gap between the lower bound and the best solution generated (minimization) is an after-the-fact measure of the solution quality. The Case 1 and 2 tolerances and execution times were based on some preliminary empirical analysis of the 51 test instances and both have a maximum execution time of 1,200 seconds. The Case 1 strategy focuses on tighter bounds starting with a tolerance of 0.0001. The Case 2 strategy has a 'looser' starting tolerance and is expected to require less execution time than Case 1 but might have looser bounds on the solutions. These two SSIT cases are compared with the Base Case to demonstrate the benefit of using the SSIT matheuristic. Remember that the SSIT provides OR practitioners with a methodology that iteratively uses commercial integer programming software to generate tightly bounded solutions in a short time.

The 51 GAP instances will be solved a total of six times—three times (Base Case, Case 1, and Case 2) with CPLEX and three times with Gurobi. The threads parameter was set to 4 in all cases. Except for tolerances and execution times as specified above, all other software parameters will have their default settings. All executions of CPLEX and Gurobi

were on a compute server with the following specifications: an Intel(R) Xeon(R) CPU E5-2640 v3 processor, 32 GB of RAM and CentOS Linux 7. The results of the Base Case are given in Table 2. Case 1 results for CPLEX and Gurobi are given in Tables 3 and 4 respectively. Case 2 results for CPLEX and Gurobi are given in Tables 5 and 6 respectively.

From Table 2, one can observe that using the simple Base Case strategy of running the optimization software at a tolerance of 0.0001 for up to an hour gave similar results for both CPLEX and Gurobi (a statistical comparison of the performance of CPLEX compared to Gurobi will be discussed in Section 5). Specifically, the average solution times per problem for CPLEX and Gurobi were 1,143 and 1,224 seconds respectively. Also, the average guaranteed bounds on the solutions from the optimums were 0.040% and 0.030% respectively. For CPLEX there were 15 GAP instances that required the full 3,600 seconds and for Gurobi there were 16 GAP instances that required the full 3,600 seconds. In Tables 3, 4, 5, and 6, each problem is provided one row for each required tolerance to solve the problem. For example, in Table 3(B), problem D-400-20 has three rows because this problem terminates at the 0.004 tolerance when the gap is reduced to 0.00164. However, problem D-1600-20 has only two rows because it terminates at the 0.001 tolerance when the gap is reduced to 0.00067.

The Case 1 results in Tables 3 and 4 clearly demonstrate the benefits to OR practitioners of choosing SSIT over the Base Case approach. For both the CPLEX and the Gurobi results Case 1 execution times are reduced by 91% (Table 7 will give a complete summary). At the same time the guaranteed bounds on the optimums are still very tight at an average bound of 0.060% and 0.045% for CPLEX and Gurobi respectively. The Case 1 solution times and solution qualities are shown to be excellent. However, Case 2 could be used instead if there is a need to further reduce execution time. Its results can be found in Tables 5 and 6.

Table 2(A). Base Case results for the 24 medium instances

	CPLEX			**Gurobi**		
Problem	Objective function	Time (sec)	Final GAP	Objective function	Time (sec)	Final GAP
B-100-5	1843	0.32	0.000%	1843	0.29	0.000%
B-100-10	1407	0.12	0.000%	1407	0.10	0.000%
B-100-20	1166	0.12	0.000%	1166	0.18	0.000%
B-200-5	3552	0.41	0.000%	3552	0.44	0.000%
B-200-10	2827	1.69	0.000%	2827	0.93	0.000%
B-200-20	2339	0.29	0.000%	2339	0.33	0.000%
C-100-5	1931	0.17	0.000%	1931	0.18	0.000%
C-100-10	1402	0.28	0.000%	1402	0.43	0.000%
C-100-20	1243	0.44	0.000%	1243	0.42	0.000%
C-200-5	3456	0.36	0.000%	3456	0.55	0.000%
C-200-10	2806	1.75	0.000%	2806	1.81	0.000%
C-200-20	2391	1.68	0.000%	2391	1.96	0.000%
D-100-5	6353	25.87	0.000%	6353	11.64	0.000%
D-100-10	6348	3600.00	0.051%	6348	3600.01	0.047%
D-100-20	6207	3600.00	0.608%	6211	3600.01	0.580%
D-200-5	12742	1213.05	0.000%	12743	170.08	0.010%
D-200-10	12435	3600.00	0.083%	12437	3600.01	0.080%
D-200-20	12260	3600.00	0.274%	12249	3600.02	0.163%
E-100-5	12681	2.04	0.010%	12681	1.24	0.010%
E-100-10	11577	8.59	0.010%	11577	2.90	0.010%
E-100-20	8436	8.96	0.000%	8436	7.77	0.000%
E-200-5	24930	1.32	0.010%	24930	0.18	0.000%
E-200-10	23307	11.04	0.010%	23307	5.44	0.010%
E-200-20	22379	6.70	0.010%	22379	5.13	0.010%

Table 2(B). Base Case results for the 27 large instances

	CPLEX			Gurobi		
Problem	Objective function	Time (sec)	Final GAP	Objective function	Time (sec)	Final GAP
C-400-10	5597	1.45	0.000%	5597	1.26	0.000%
C-400-20	4782	8.47	0.000%	4782	14.98	0.000%
C-400-40	4244	2.41	0.000%	4244	5.63	0.000%
C-900-15	11340	284.30	0.000%	11340	14.75	0.010%
C-900-30	9982	410.23	0.000%	9982	255.07	0.000%
C-900-60	9326	3600.00	0.012%	9327	3600.18	0.011%
C-1600-20	18803	75.52	0.010%	18803	1118.25	0.010%
C-1600-40	17145	1352.91	0.010%	17146	3600.16	0.012%
C-1600-80	16287	3600.00	0.025%	16287	3600.30	0.025%
D-400-10	24970	3600.00	0.047%	24965	3600.02	0.024%
D-400-20	24588	3600.00	0.121%	24591	3600.03	0.130%
D-400-40	24456	3600.00	0.438%	24440	3600.07	0.372%
D-900-15	55420	3600.00	0.033%	55415	3600.07	0.024%
D-900-30	54880	3600.00	0.089%	54879	3600.09	0.086%
D-900-60	54736	3600.00	0.338%	54613	3600.17	0.113%
D-1600-20	97855	3600.00	0.034%	97853	3600.13	0.031%
D-1600-40	97246	3600.00	0.145%	97180	3600.19	0.077%
D-1600-80	97242	3600.00	0.214%	97111	3600.36	0.079%
E-400-10	45749	10.94	0.010%	45748	1.307	0.010%
E-400-20	44877	9.37	0.010%	44877	7.87422	0.010%
E-400-40	44562	184.61	0.010%	44561	184.506	0.010%
E-900-15	102429	3.84	0.010%	102421	11.4457	0.010%
E-900-30	100435	18.37	0.010%	100436	30.3832	0.010%
E-900-60	100157	333.45	0.010%	100155	2243.15	0.010%
E-1600-20	180661	24.97	0.010%	180660	6.88894	0.010%
E-1600-40	178306	40.21	0.010%	178308	69.7459	0.010%
E-1600-80	176835	250.65	0.010%	176835	654.628	0.010%

Table 3(A). Case 1 CPLEX SSIT results for 24 medium GAP instances

Problem	Tolerance	Objective function	Time (sec)	Final GAP
B-100-5	0.0001	1843	0.32	0%
B-100-10	0.0001	1407	0.12	0%
B-100-20	0.0001	1166	0.32	0%
B-200-5	0.0001	3552	0.41	0%
B-200-10	0.0001	2827	1.70	0%
B-200-20	0.0001	2339	0.29	0%
C-100-5	0.0001	1931	0.17	0%
C-100-10	0.0001	1402	0.28	0%
C-100-20	0.0001	1243	0.43	0%
C-200-5	0.0001	3456	0.36	0%
C-200-10	0.0001	2806	1.75	0%
C-200-20	0.0001	2391	1.68	0%
D-100-5	0.0001	6353	25.87	0%
D-100-10	0.0001	6351	150.00	
	0.001	6351	150.00	
	0.004	6351	0.01	0.217%
D-100-20	0.0001	6209	150.00	
	0.001	6209	150.00	
	0.004	6209	300.00	
	0.007	6209	0.01	0.665%
D-200-5	0.0001	12748	150.00	
	0.001	12748	0.51	0.05%
D-200-10	0.0001	12435	150.00	
	0.001	12435	147.84	0.1%
D-200-20	0.0001	12267	150.00	
	0.001	12267	150.00	
	0.004	12267	0.01	0.4%
E-100-10	0.0001	11577	8.59	0.010%
E-100-20	0.0001	8436	8.96	0.000%
E-200-5	0.0001	24930	1.32	0.008%
E-200-10	0.0001	23307	11.04	0.010%
E-200-20	0.0001	22379	6.70	0.007%

Table 3(B). Case 1 CPLEX SSIT results for large GAP instances C and D

Problem	Tolerance	Objective function	Time (sec)	Final GAP
C-400-10	0.0001	5597	1.45	0%
C-400-20	0.0001	4782	8.47	0%
C-400-40	0.0001	4244	2.41	0%
C-900-15	0.0001	11348	150.00	
	0.001	11348	0.69	0.016%
C-900-30	0.0001	9988	150.00	
	0.001	9988	0.36	0.017%
C-900-60	0.0001	9332	150.00	
	0.001	9332	0.32	0.038%
C-1600-20	0.0001	18803	75.32	0.010%
C-1600-40	0.0001	17154	150.00	
	0.001	17154	0.96	0.028%
C-1600-80	0.0001	16293	150.00	
	0.001	16293	0.66	0.025%
D-400-10	0.0001	24980	150.00	
	0.001	24980	0.13	0.056%
D-400-20	0.0001	24596	150.00	
	0.001	24596	150.00	
	0.004	24596	0.01	0.164%
D-400-40	0.0001	24505	150.00	
	0.001	24488	150.00	
	0.004	24488	300.00	
	0.007	24488	0.01	0.669%
D-900-15	0.0001	55452	150.00	
	0.001	55452	0.74	0.039%
D-900-30	0.0001	54906	150.00	
	0.001	54906	150.00	
	0.004	54906	0.01	0.158%
D-900-60	0.0001	54765	150.00	
	0.001	54765	150.00	
	0.004	54765	0.02	0.367%
D-1600-20	0.0001	97906	150.00	
	0.001	97906	11.72	0.067%
D-1600-40	0.0001	97253	150.00	
	0.001	97253	150.00	
	0.004	97253	0.02	0.154%
D-1600-80	0.0001	97255	150.00	
	0.001	97255	150.00	
	0.004	97255	0.04	0.365%

Table 3(C). Case 1 CPLEX SSIT results for large GAP instances E

Problem	Tolerance	Objective function	Time (sec)	Final GAP
E-400-10	0.0001	45749	10.95	0.009%
E-400-20	0.0001	44877	9.38	0.004%
E-400-40	0.0001	44593	150.00	
	0.001	44593	0.87	0.015%
E-900-15	0.0001	102429	3.84	0.010%
E-900-30	0.0001	100497	18.37	0.008%
E-900-60	0.0001	100241	150.00	
	0.001	100241	0.37	0.038%
E-1600-20	0.0001	180661	24.97	0.010%
E-1600-40	0.0001	178306	40.21	0.008%
E-1600-80	0.0001	176983	150.00	
	0.001	176983	106.36	0.080%

Table 4(A). Case 1 Gurobi SSIT results for 24 medium GAP instances

Problem	Tolerance	Objective function	Time (sec)	Final GAP
B-100-5	0.0001	1843	0.25	0%
B-100-10	0.0001	1407	0.08	0%
B-100-20	0.0001	1166	0.16	0%
B-200-5	0.0001	3552	0.38	0%
B-200-10	0.0001	2827	0.78	0%
B-200-20	0.0001	2339	0.32	0%
C-100-5	0.0001	1931	0.19	0%
C-100-10	0.0001	1402	0.41	0%
C-100-20	0.0001	1243	0.41	0%
C-200-5	0.0001	3456	0.60	0%
C-200-10	0.0001	2806	1.69	0%
C-200-20	0.0001	2391	1.86	0%
D-100-5	0.0001	6353	8.33	0%
D-100-10	0.0001	6351	150.00	
	0.001	6351	150.00	
	0.004	6351	0.01	0.126%
D-100-20	0.0001	6220	150.00	
	0.001	6220	150.00	
	0.004	6220	300.00	
	0.007	6220	300.00	
	0.01	6220	0.01	0.723%
D-200-5	0.0001	12744	150.00	
	0.001	12744	0.01	0.016%
D-200-10	0.0001	12445	150.00	
	0.001	12440	150.00	
	0.004	12440	0.01	0.121%
D-200-20	0.0001	12288	150.00	
	0.001	12285	150.00	
	0.004	12260	0.01	0.261%
E-100-5	0.0001	12681	0.55	0.010%
E-100-10	0.0001	11577	2.96	0.010%
E-100-20	0.0001	8436	6.36	0.000%
E-200-5	0.0001	24930	0.18	0.008%
E-200-10	0.0001	23307	6.07	0.010%
E-200-20	0.0001	22379	5.48	0.001%

Table 4(B). Case 1 Gurobi SSIT results for large GAP instances C and D

Problem	Tolerance	Objective function	Time (sec)	Final GAP
C-400-10	0.0001	5597	1.19	0%
C-400-20	0.0001	4782	13.36	0%
C-400-40	0.0001	4244	5.26	0%
C-900-15	0.0001	11340	14.03	0.01%
C-900-30	0.0001	9988	150.00	
	0.001	9988	0.03	0.011%
C-900-60	0.0001	9328	150.00	
	0.001	9328	0.02	0.032%
C-1600-20	0.0001	18804	150.00	
	0.001	18804	0.04	0.011%
C-1600-40	0.0001	17147	150.00	
	0.001	17147	0.02	0.017%
C-1600-80	0.0001	16287	150.00	
	0.001	16287	0.04	0.025%
D-400-10	0.0001	24975	150.00	
	0.001	24980	0.01	0.064%
D-400-20	0.0001	24615	150.00	
	0.001	24608	150.00	
	0.004	24608	0.01	0.199%
D-400-40	0.0001	24468	150.00	
	0.001	24460	150.00	
	0.004	24447	300.00	
	0.007	24447	0.01	0.401%
D-900-15	0.0001	55415	150.00	
	0.001	55415	0.01	0.023%
D-900-30	0.0001	54896	150.00	
	0.001	54895	150.00	
	0.004	54895	0.01	0.155%
D-900-60	0.0001	54660	150.00	
	0.001	54639	150.00	
	0.004	54639	0.02	0.161%
D-1600-20	0.0001	97855	150.00	
	0.001	97855	0.01	0.034%
D-1600-40	0.0001	97220	150.00	
	0.001	97200	92.58	0.098%
D-1600-80	0.0001	97251	150.00	
	0.001	97229	150.00	
	0.004	97229	0.01	0.201%

Table 4(C). Case 1 Gurobi SSIT results for large GAP instances E

Problem	Tolerance	Objective function	Time (sec)	Final GAP
E-400-10	0.0001	45748	1.52	0.010%
E-400-20	0.0001	44877	8.03	0.010%
E-400-40	0.0001	44568	150.00	
	0.001	44568	0.01	0.020%
E-900-15	0.0001	102421	12.16	0.010%
E-900-30	0.0001	100436	32.45	0.010%
E-900-60	0.0001	100164	150.00	
	0.001	100164	0.01	0.022%
E-1600-20	0.0001	180660	5.99	0.010%
E-1600-40	0.0001	178308	67.48	0.010%
E-1600-80	0.0001	176910	150.00	
	0.001	176910	0.01	0.054%

Table 5(A). Case 2 CPLEX SSIT results for 24 medium GAP instances

Problem	Tolerance	Objective function	Time (sec)	Final GAP
B-100-5	0.001	1843	0.18	0100%
B-100-10	0.001	1407	0.04	0.099%
B-100-20	0.001	1166	0.05	0.092%
B-200-5	0.001	3553	0.24	0.093%
B-200-10	0.001	2828	0.95	0.098%
B-200-20	0.001	2340	0.13	0.000%
C-100-5	0.001	1931	0.08	0.100%
C-100-10	0.001	1402	0.20	0.100%
C-100-20	0.001	1243	0.27	0.067%
C-200-5	0.001	3457	0.10	0.092%
C-200-10	0.001	2807	0.57	0.196%
C-200-20	0.001	2392	1.04	0.100%
D-100-5	0.001	6355	1.74	0.099%
D-100-10	0.001	6351	300.00	
	0.004	6351	0.01	0.196%
D-100-20	0.001	6209	300.00	
	0.004	6209	300.00	
	0.007	6209	0.01	0.650%
D-200-5	0.001	12748	17.51	0.071%
D-200-10	0.001	12435	297.84	0.087%
D-200-20	0.001	12267	300.00	
	0.004	12267	0.01	0.334%
E-100-5	0.001	12688	0.58	0.100%
E-100-10	0.001	11582	2.62	0.093%
E-100-20	0.001	8439	4.96	0.070%
E-200-5	0.001	24936	0.32	0.037%
E-200-10	0.001	23323	0.81	0.093%
E-200-20	0.001	22396	4.88	0.083%

Table 5(B). Case 2 CPLEX SSIT results for 27 large GAP instances

Problem	Tolerance	Objective function	Time (sec)	Final GAP
C-400-10	0.001	5599	0.27	0.075%
C-400-20	0.001	4784	2.03	0.092%
C-400-40	0.001	4246	1.14	0.089%
C-900-15	0.001	11348	0.69	0.092%
C-900-30	0.001	9988	6.36	0.084%
C-900-60	0.001	9332	20.46	0.092%
C-1600-20	0.001	18813	1.03	0.077%
C-1600-40	0.001	17154	7.0	0.074%
C-1600-80	0.001	16293	11.66	0.061%
D-400-10	0.001	24980	23.13	0.090%
D-400-20	0.001	24596	300.00	
	0.004	24596	0.01	0.154%
D-400-40	0.001	24505	300.00	
	0.004	24488	300.00	
	0.007	24488	0.01	0.568%
D-900-15	0.001	55452	17.74	0.091%
D-900-30	0.001	54906	300.00	
	0.004	54906	0.01	0.137%
D-900-60	0.001	54765	300.00	
	0.004	54765	0.02	0.391%
D-1600-20	0.001	97906	161.72	0.086%
D-1600-40	0.001	97253	300.00	
	0.004	97253	0.02	0.152%
D-1600-80	0.001	97255	300.00	
	0.004	97255	0.04	0.227%
E-400-10	0.001	45788	1.72	0.097%
E-400-20	0.001	44912	5.02	0.082%
E-400-40	0.001	44593	42.87	0.082%
E-900-15	0.001	102486	1.21	0.066%
E-900-30	0.001	100497	3.11	0.074%
E-900-60	0.001	100241	146.93	0.097%
E-1600-20	0.001	180820	1.24	0.099%
E-1600-40	0.001	178430	9.26	0.079%
E-1600-80	0.001	176983	256.36	0.093%

Table 6(A). Case 2 Gurobi SSIT results for 24 medium GAP instances

Problem	Tolerance	Objective function	Time (sec)	Final GAP
B-100-5	0.001	1843	0.27	0.100%
B-100-10	0.001	1407	0.08	0.099%
B-100-20	0.001	1166	0.17	0.092%
B-200-5	0.001	3552	0.20	0.093%
B-200-10	0.001	2827	0.47	0.098%
B-200-20	0.001	2339	0.24	0.000%
C-100-5	0.001	1931	0.16	0.100%
C-100-10	0.001	1402	0.43	0.100%
C-100-20	0.001	1243	0.34	0.067%
C-200-5	0.001	3458	0.41	0.092%
C-200-10	0.001	2807	0.53	0.196%
C-200-20	0.001	2392	0.98	0.100%
D-100-5	0.001	6357	1.41	0.099%
D-100-10	0.001	6350	235.73	0.095%
D-100-20	0.001	6207	300.00	
	0.004	6207	300.00	
	0.007	6207	0.01	0.516%
D-200-5	0.001	12751	1.23	0.094%
D-200-10	0.001	12436	32.55	0.088%
D-200-20	0.001	12277	300.00	
	0.004	12277	0.01	0.399%
E-100-5	0.001	12685	0.74	0.079%
E-100-10	0.001	11582	3.31	0.086%
E-100-20	0.001	8443	7.67	0.095%
E-200-5	0.001	24935	0.10	0.052%
E-200-10	0.001	23315	0.21	0.090%
E-200-20	0.001	22396	2.25	0.098%

Table 6(B). Case 2 Gurobi SSIT results for 27 large GAP instances

Problem	Tolerance	Objective function	Time (sec)	Final GAP
C-400-10	0.001	5599	0.53	0.089%
C-400-20	0.001	4784	1.39	0.084%
C-400-40	0.001	4247	1.49	0.094%
C-900-15	0.001	11346	0.76	0.071%
C-900-30	0.001	9989	3.66	0.090%
C-900-60	0.001	9332	11.25	0.096%
C-1600-20	0.001	18806	2.25	0.032%
C-1600-40	0.001	17154	2.28	0.082%
C-1600-80	0.001	16297	14.94	0.086%
D-400-10	0.001	24979	6.40	0.088%
D-400-20	0.001	24612	300.00	
	0.004	24612	0.01	0.215%
D-400-40	0.001	24505	300.00	
	0.004	24488	300.00	
	0.007	24488	0.01	0.393%
D-900-15	0.001	55452	17.74	0.079%
D-900-30	0.001	54906	300.00	
	0.004	54906	0.01	0.098%
D-900-60	0.001	54765	300.00	
	0.004	54765	0.02	0.172%
D-1600-20	0.001	97906	161.72	0.095%
D-1600-40	0.001	97253	300.00	
	0.004	97253	0.02	0.101%
D-1600-80	0.001	97255	300.00	
	0.004	97255	0.04	0.125%
E-400-10	0.001	45765	0.58	0.052%
E-400-20	0.001	44912	5.17	0.082%
E-400-40	0.001	44595	10.07	0.094%
E-900-15	0.001	102510	0.71	0.091%
E-900-30	0.001	100503	8.97	0.081%
E-900-60	0.001	100208	33.17	0.071%
E-1600-20	0.001	180810	3.51	0.093%
E-1600-40	0.001	178454	17.42	0.094%
E-1600-80	0.001	176961	22.99	0.091%

Tables 5 and 6 show the results of using the SSIT Case 2 scenario to solve the 51 GAP instances. For Case 2, the average execution times are 85.4 seconds for CPLEX and 62.5 seconds for Gurobi respectively. On average, the solutions are guaranteed to be within 0.085% and 0.080% of the optimums for CPLEX and Gurobi respectively. It is up to the user to decide which case is more appropriate for the application being solved. Table 7 summarizes the results of these three cases (Base Case, Case 1, and Case 2).

Table 7 clearly validates the benefit of SSIT compared with the Base Case approach. This appears to be true regardless of the chosen software (CPLEX or Gurobi). The average time reduction in using CPLEX is 91% for Case 1 and 93% for Case 2 respectively. The average time reduction in using Gurobi is 91% and 95% for Case 1 and Case 2 respectively.

Table 7. Summary comparison of Base Case, Case 1, and Case 2 for both CPLEX and Gurobi

	CPLEX		Gurobi	
Case	Guaranteed bound	Time (sec)	Guaranteed bound	Time (sec)
Base Case	0.040%	1143	0.030%	1224
Case 1	0.060%	108	0.045%	115
Case 2	0.085%	85	0.080%	63

4.3 *GAP SSIT results compared to other results in the literature*

Although the authors (as OR practitioners) appreciate the guaranteed bounds that the SSIT matheuristic provides, there may be readers who are interested in seeing how the SSIT solutions compare to other available results in the literature. First of all, due to their large sizes, the 51 GAP testing instances in this article were not used in many metaheuristic approaches. Several approaches in the literature tend to use smaller GAP instances for testing purposes. For example, Ahmed [10] used test problems with at most 200 items. Although there are no guarantees on solution quality, the metaheuristic of Yagiura et al. [13] is the best performing metaheuristic on these 51 GAP instances. To determine the quality of the Yagiura et al. [13] solutions, these results are compared to the best results from the Posta and Ferland [2] variable fixing approach, the column generation approach of Sadykov et al. [6], and the diversified diving method of Sadykov et al. [7]. Although both Sadykov et al. [6] and Sadykov et al. [7] do not provide computer hardware details, they report execution times that can exceed 10,000 seconds. The average deviation of the Yagiura et al. [13] metaheuristic results from the best known results mentioned above is 0.022%. However, the quality of the Yagiura et al. [13] metaheuristic results are not known until these results are compared to the best known results which required considerable computer resources. The Yagiura et al. [13] is a sophisticated algorithm and computer code. Furthermore, if any industrial GAP application is solved by coding and using the Yagiura et al. [13] metaheuristic, there is absolutely no guarantee on the solution quality. This is in sharp contrast to the SSIT matheuristic being used to solve an industrial GAP application. SSIT benefits include: no problem-specific computer code would be required, there would be a guaranteed bound on the solution quality, and improved performance is automatic as newer versions of the software become available.

In Sadykov et al. [6, 7], the authors report that they have found an optimal solution for D-900-15 to be **54404**. They did in fact find the optimal solution, but the optimal objective function is incorrect. In Table 2 of Sadykov et al. [6], a solution is given for D-900-15 which is reported to have an objective function value of 54404 which is lower than the dual bound of 55404. When the authors of this article evaluated the Table 2 solution, its objective function was 55404 which agrees with the lower bound, hence is the optimum. So, the optimal value for D-900-15 should be reported as 55404.

5. Statistical comparison of CPLEX and Gurobi results for the 51 GAP instances

In this section, we will conduct a statistical analysis with the 51 GAP instances to compare the performance of CPLEX to Gurobi in terms of time and gap. The paired t-tests [14] is used for case-by-case comparisons between CPLEX and Gurobi.

5.1 *Comparison of CPLEX and Gurobi - Base Case*

Table 8 summarizes the results from the paired t-test to compare the time (or gap) of CPLEX to the counterpart of Gurobi in Base Case (Case 0). At the significance level of 0.01, there is no significant difference between CPLEX and Gurobi in terms of time (or gap) since its p-value of 0.218 (or 0.090) is greater than 0.01.

Table 8. Summary of paired t-test - Base Case (Case 0)

Time

Descriptive Statistics

Sample	N	Mean	StDev	SE Mean
time0_cplex	51	1143	1622	227
time0_gurobi	51	1224	1660	232

Test

Null hypothesis	H_0: μ_difference = 0
Alternative hypothesis	H_1: μ_difference ≠ 0

T-value	P-value
-1.25	0.218

Gap

Descriptive Statistics

Sample	N	Mean	StDev	SE Mean
gap0_cplex	51	0.000395	0.000940	0.000132
gap0_gurobi	51	0.000295	0.000777	0.000109

Test

Null hypothesis	H_0: μ_difference = 0
Alternative hypothesis	H_1: μ_difference ≠ 0

T-value	P-value
1.73	0.090

5.2 *Comparison of CPLEX and Gurobi - Case 1*

Table 9 summarizes the results from the paired t-test to compare the time (or gap) of CPLEX to the counterpart of Gurobi in Case 1. At the significance level of 0.01, the test results indicate that the time (or gap) of CPLEX is not statistically different from the time (or gap) of Gurobi, since its p-value of 0.323 (or 0.076) is considerably greater than 0.01.

Table 9. Summary of paired t-test - Case 1

Time

Descriptive Statistics

Sample	N	Mean	StDev	SE Mean
time1_cplex	51	107.7	145.7	20.4
time1_gurobi	51	115.4	172.3	24.1

Test

Null hypothesis	H_0: μ_difference = 0
Alternative hypothesis	H_1: μ_difference ≠ 0

T-value	P-value
-1.00	0.323

Gap

Descriptive Statistics

Sample	N	Mean	StDev	SE Mean
gap1_cplex	51	0.000600	0.001295	0.000181
gap1_gurobi	51	0.000446	0.001019	0.000143

Test

Null hypothesis	H_0: μ_difference = 0
Alternative hypothesis	H_1: μ_difference ≠ 0

T-value	P-value
1.81	0.076

5.3 *Comparison of CPLEX and Gurobi - Case 2*

Lastly, Table 10 summarizes the results from the paired t-test to compare the time (or gap) of CPLEX to the counterpart of Gurobi in Case 2. At the significance level of 0.01, we cannot find a significant time (or gap) difference between CPLEX and Gurobi since its p-value of 0.013 (or 0.103) is greater than 0.01.

Table 10. Summary of paired t-test - Case 2

Time

Descriptive Statistics

Sample	N	Mean	StDev	SE Mean
time2_cplex	51	85.4	154.0	21.6
time2_gurobi	51	62.5	132.8	18.6

Test

Null hypothesis	H_0: μ_difference = 0
Alternative hypothesis	H_1: μ_difference ≠ 0

T-value	P-value
2.59	0.013

Gap

Descriptive Statistics

Sample	N	Mean	StDev	SE Mean
gap2_cplex	51	0.000847	0.001026	0.000144
gap2_gurobi	51	0.000734	0.000795	0.000111

Test

Null hypothesis	H_0: μ_difference = 0
Alternative hypothesis	H_1: μ_difference ≠ 0

T-value	P-value
1.66	0.103

To sum up, no significant difference in time or performance (gap) between CPLEX and Gurobi is found in all three cases.

6. Summary and future work

Some algorithms developed to solve NP-hard combinatorial optimization problems 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. This procedure called the SSIT matheuristic has been empirically shown, using 51 GAP from the literature, to quickly generate solutions that are guaranteed to be very close to the optimums. This multi-pass matheuristic is used in conjunction with an integer programming software (both CPLEX and Gurobi were tested) 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.

Regardless of which software package (Gurobi or CPLEX) was used or which SSIT scenario was used (two were tested), this methodology was able, on average, for the 51 GAP instances, to generate solutions guaranteed to be within 0.1% of the optimums in under two minutes execution time on a standard PC. 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.

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.

The SSIT matheuristic is an effective alternative for OR practitioners to generate GAP solutions in an industrial setting. This article demonstrates that the SSIT matheuristic is able to generate guaranteed bounded solutions (on average, within 0.1% of the optimum) for 51 GAP test instances in only about 10% of the running time for executing CPLEX or Gurobi with the max time of 3600 seconds at a tolerance of 0.0001. Furthermore, statistically, there was no significant difference between CPLEX and Gurobi when solving these 51 GAP instances.

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 using different commercial integer programming software packages.

Conflict of interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Oncan T. A Survey of the Generalized Assignment Problem and its Applications. *INFOR*. 2007; 45(3): 123-141. Available from: doi: 10.3138/infor.45.3.123.

[2] Posta M, Ferland JA. An exact method with variable fixing for the generalized assignment problem. *Computational Optimization Applications*. 2012; 52: 629-644. Available from: doi: 10.1007/s10589-011-9432-0.

[3] McNally B. *A Simple sequential increasing tolerance matheuristic that generates bounded solutions for combinatorial optimization problems*. Master's Thesis. Kutztown University of Pennsylvania; 2021.

[4] Woodcock AJ, Wilson JM. A hybrid tabu search/branch & bound approach to solving the generalized assignment problem. *European Journal of Operational research*. 2010; 207(2): 566-578. Available from: doi: 10.1016/j.ejor.2010.05.007.
[5] Munapo E, Lesaona M, Nyamugure P. A transportation branch and bound algorithm for solving the generalized assignment problem. *International Journal of System Assurance Engineering and Management*. 2015; 6: 217-223. Available from: doi: 10.1007/s13198 015 03439.
[6] Sadykov R, Vanderbeck F, Pessoa A, Uchoa E. Column generation based heuristic for the generalized assignment problem. *XLVII Simposio Brasileiro de pesquisa Operacional*. Port de Galinhas, Brazil; 2015. p.3624-3631.
[7] Sadykov R, Vanderbeck F, Pessoa A, Tahiri I, Uchoa E. Primal heuristics for branch-and-price: the assets of diving methods. *INFORMS Journal on Computing*. 2019; 31(2): 251-267. Available from: doi: 10.1287/ijoc.2018.0822.
[8] Bando KA, Kawasaki RB. Stability properties of the core in a generalized assignment problem. *Games and Economic Behavior*. 2021; 130: 211-223. Available from: doi: 10.2139/ssrn.3782247.
[9] Chu PC, Beasley JE. A genetic algorithm for the generalized assignment problem. *Computers and Operations Research*. 1997; 24(1): 17-23. Available from: doi: 10.1016/S0305-0548(96)00032-9.
[10] Ahmed ZH. Performance analysis of hybrid genetic algorithms for the generalized assignment problem. International *Journal of Computer Science and network Security.* 2019; 19(9): 216-222. Available from: http://paper.ijcsns.org/07_book/201909/20190925.pdf [Accessed 25th October 2021].
[11] Zhi-Bin H, Guang-Tao F, Dan-Yang D, Chen X, Zhe-Lun D, Zhi-Tao D. Novel parallel hybrid genetic algorithms on the GPU for the generalized assignment problem. *The Journal of Supercomputing*. 2021: 1-24. Available from: doi: 10.1007/s11227-021-03882-6.
[12] Baykasoglu A, Ozbaku L, Tapkan P. Artificial Bee colony algorithm and its application to generalized assignment problem. In: Chan FTS, Tiwari MK (eds.) *Swarm Intelligence: Focus on Ant and Particle Swarm Optimization.* Vienna, Austria: I-Tech Education and Publishing; 2007. p.113-144.
[13] Yagiura M, Ibaraki T, Glover F. A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research*. 2006; 169(2): 548-569. Available from: doi: 10.1016/j.ejor.2004.08.015.
[14] Kutner M, Nachtsheim CJ, Neter J, Li W. *Applied Linear Statistical Models (5th edition)*. Burr Ridge, IL: McGraw Hill; 2013.