Two-stage multiple binary knapsack problem with uncertain profit in the second stage

Hossein Tohidi
Advisor: Dr. Osman Ozaltin

Abstract

An stochastic multiple binary knapsacks problem is studied in this paper and an algorithm is developed to solve this problem based on integer L-shaped method. Another method is also implemented by combining the Benders Decomposition algorithm and integer L-shaped method. These two algorithms are implemented in Python using Gurobi Optimizer. Then, 13 different instances are randomly generated and solved by proposed algorithms. The result proves the effectiveness of the combined method in solving larger instances in a timely manner. Also a sensitivity analysis is done to investigate the impact of the growth in the problem's parameters in the performance of the integer L-shaped algorithm.

Keywords: stochastic integer programming; integer L-shaped method; decomposition methods; multiple binary knapsack.

1 Introduction

The knapsack problem is a widely used and studied combinatorial optimization and NP-Hard problem. The binary knapsack problem can be defined as below:

Given a set of items with their own profit and weight and a knapsack, the objective is to find a subset of items so as to maximize the profit without exceeding the capacity of knapsack.Knapsack problem is a well studied discrete optimization problem, and it has application in many areas including project selection [3], resource distribution [4], investment decision making [5], and etc.

A well-known generalization of single binary knapsack problem is multiple binary knapsack problem. This problem is described as follows. Given a set of items with their own profit and weight and a set of knapsacks with their own capacities, the objective is to find item subsets for each knapsack so as to maximize the profit without exceeding the capacity of each knapsack.

The first examples of multiple knapsack problem is studied by Lorie and Savage [6] and by Manne and Markowitz [7]. Though the multiple knapsack problem is a generalization of the single case, the situation is quite different when several constraints are taken into account.

The first exact algorithms for multiple knapsack problem developed during the sixties by Gilmore and Gomery [8]. However, the early effective procedures developed in the 1980s [9, 10, 11]. Different solution methods including dynamic programming and its variants, branch-and-bound approach, and special enumeration techniques and reduction schemes is used to find exact solutions for this problem.

Even the recent advances on the branch-and-cut method have made possible the solution of middle size multiple knapsack problem instances, heuristic methods remain a competitive alternative, specially when the number of constraints is large. They used different solution methodologies including specific local search [13], multi-start strategies [14], branch-and-bound early termination [15], greedy algorithms [12], meta-heuristics [16], and etc.

In this project, our focus is on a stochastic multiple binary knapsack problem. This uncertainty can be captured in any part of a real world problem. In this problem the uncertainty is assumed to be in the profit of a subset of items with known and discrete probability distribution.

Several stochastic variants of the knapsack problem is studied in the literature [17, 18, 19]. Both heuristic approaches and exact approaches are proposed for this problem. In this project, the objective is to use the Integer L-shaped method to find exact solutions for the problem. This method is well studied in the literature for different optimization problems [1, 2].

In this paper, it is assumed that there are two sets of items, named set $\mathcal{M}$ and set $\mathcal{K}$ and multiple knapsacks with different capacities. The objective is to find a subset of items from both lists to fit in each knapsack and maximize the total profit. In this problem, the items in set $\mathcal{K}$ have uncertain profits with a known discrete distribution. This problem can be formulated in extensive form to maximize the profit from set $\mathcal{M}$ plus the expected profit from set $\mathcal{K}$. The number of constraints in this formulation increases as the number of scenarios increases.

This problem has an special structure, in which a block of constraints repeated for each scenario; therefore, it is possible to formulate it in a two-stage form. Our objective in this project is to use this structure to solve the problem using Integer L-shaped method, iteratively. Using this approach, the expected profits from the set $\mathcal{K}$ is replaced by an auxiliary variable θ and its all associate constraints will be removed. The remaining of the formulation is called the first stage problem or Master Problem. The second stage problems or subproblems are made for each scenario in which the total second stage profit in each scenario is maximized. The key idea of the Integer L-shaped method is to overestimate θ and try to improve this estimation iteratively by introducing optimality cuts.

This problem has another nice property which makes the life easier and this is the relative complete recourse property. It means that a feasible solution for the master problem is also feasible for all subproblems.

In this paper, two algorithms with two different optimality cuts are developed and implemented in Python 2.7 and solved using Gurobi 7.5.1. The effectiveness of these two algorithms are examined by introducing 13 randomly generated numerical instances.

The first optimality cut that is used in this study, was initially introduced by Laporte and Louveaux in 1993. This cut is called Integer L-shaped Cut, hereafter. The second cut is based on the Benders cut and it uses the dual information of the subproblems in order to be computed.

The rest of this report is organized in 5 sections. In section 3, problem formulation is given. Section 4 will provide the reader with two proposed algorithms, integer L-shaped algorithm, and single-cut Benders algorithm. Result and discussion are given in section 5; and finally in section 6 the conclusion is made.

2 Problem Statement

In this section, the binary knapsack problem, multiple binary knapsack and finally stochastic multiple binary knapsack problem are closely studied and the mathematical formulation for each problem is given.

2.1 Binary knapsack problem

Assume there is a set of m items. Each item i has its own weight and profit. there is also a knapsack with the finite capacity. Now the following set can be defined:

$\mathcal{M} = 1, 2, ..., m$ set of items

The following parameters also can be defined:

a_i weight associated with item $i \in \mathcal{M}$
c_i profit associated with item $i \in \mathcal{M}$
b capacity of knapsack

The following decision variable will be used in the mathematical formulation:

$$x_i = \begin{cases} 1 & \text{if item i is selected} \\ 0 & \text{otherwise} \end{cases}$$

The resulting mathematical model then becomes:

$$\text{maximize} \sum_{i \in \mathcal{M}} c_i x_i \tag{1}$$

$$\text{s.t.} \sum_{i \in \mathcal{M}} a_i x_i \leq b \tag{2}$$

$$x_i \in \{0, 1\}, \quad i \in \mathcal{M} \tag{3}$$

The objective function (1) maximizes total profit. The constraint (2) ensures that the capacity of knapsack is not violated. Finally, the set of constraints (3) make sure variables are binary.

2.2 Multiple binary knapsacks problem

Assume that there is a set of m items. Each item i has its own weight and profit. There is also a set of n knapsacks. Each knapsack j has a finite capacity. Now, the following sets can be defined:

$\mathcal{M} = 1, 2, ..., m$ set of items

$\mathcal{N} = 1, 2, ..., n$ set of knapsacks

The following parameters also can be defined:

a_i weight associated with item $i \in \mathcal{M}$
c_i profit associated with item $i \in \mathcal{M}$
b_j capacity associated with knapsack $j \in \mathcal{N}$

The following decision variable will be used in model:

$$x_{ij} = \begin{cases} 1 & \text{if item i is assigned to knapsack j} \\ 0 & \text{otherwise} \end{cases}$$

The resulting mathematical model then becomes:

$$\text{maximize} \sum_{i \in \mathcal{M}} c_i x_{ij} \tag{4}$$

$$\text{s.t.} \sum_{i \in \mathcal{M}} a_i x_{ij} \leq b_j \quad \forall j \in \mathcal{N} \tag{5}$$

$$\sum_{j \in \mathcal{N}} x_{ij} \leq 1 \quad \forall i \in \mathcal{M} \tag{6}$$

$$x_{ij} \in \{0, 1\}, \quad i \in \mathcal{M}, \quad j \in \mathcal{N} \tag{7}$$

The objective function (4) maximizes total profit. The set of constraints (5) ensure that the capacity of each knapsack is not violated. The set of constraints (6) make sure that each item is assigned to at most one knapsack . Finally, constraints (7) make sure variables are binary.

2.3 Stochastic multiple binary knapsack problem

Assume that there is a set of m items (first stage items). Each item in this set has its own deterministic weight and profit. Also, there is another set of K items (second stage items). Each item in this second set has its own weight

and profit. The weight is deterministic for these items; however, the profit is stochastic (uncertain). We also have a set of n knapsacks. Each knapsack has a finite capacity. Finally, we have a set of Ω scenarios. Each scenario represents a realization of uncertain parameter. Then, the following sets can be defined:

$\mathcal{M} = 1, 2, ..., m$ set of items with deterministic weight and profit

$\mathcal{N} = 1, 2, ..., m$ set of knapsacks

$\mathcal{K} = 1, 2, ..., k$ set of items with deterministic weight and stochastic profit

$\Omega = 1, 2, ..., \omega$ set of scenarios

We also define the following parameters:

a_i weight associated with item $i \in \mathcal{M}$
c_i profit associated with item $i \in \mathcal{M}$
w_k weight associated with item $k \in \mathcal{K}$
p^ω probability associated with scenario $\omega \in \Omega$
q_k^ω realization of profit associated with item $k \in \mathcal{K}$ based on scenario $\omega \in \Omega$
b_j capacity associated with knapsack j

The following decision variables will be used in model:

$$x_{ij} = \begin{cases} 1 & \text{if item i from list } \mathcal{M} \text{ is assigned to knapsack j} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{kj}^\omega = \begin{cases} 1 & \text{if item k from list } \mathcal{K} \text{ is assigned to knapsack j in scenario } \omega \\ 0 & \text{otherwise} \end{cases}$$

The resulting mathematical model then becomes:

$$\text{maximize} \sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{N}} c_i x_{ij} + \sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}} p^{\omega} q_k^{\omega} y_{kj}^{\omega} \tag{8}$$

$$\text{s.t.} \sum_{i\in\mathcal{M}} a_i x_{ij} \leq b_j \quad \forall j \in \mathcal{N} \tag{9}$$

$$\sum_{j\in\mathcal{N}} x_{ij} \leq 1 \quad \forall i \in \mathcal{M} \tag{10}$$

$$\sum_{i\in\mathcal{M}} a_i x_{ij} + \sum_{k\in\mathcal{K}} \omega_k y_{kj}^{\omega} \leq b_j \quad \forall j \in \mathcal{N}, \forall \omega \in \Omega \tag{11}$$

$$\sum_{j\in\mathcal{N}} y_{kj}^{\omega} \leq 1 \quad \forall k \in \mathcal{K}, \forall \omega \in \Omega \tag{12}$$

$$x_{ij} \in \{0,1\}, \quad i \in \mathcal{M}, \quad j \in \mathcal{N} \tag{13}$$

$$y_{ij}^{\omega} \in \{0,1\}, \quad k \in \mathcal{K}, \quad j \in \mathcal{N} \tag{14}$$

This model is called extensive form of the problem. The objective function (8) maximizes the total profit of first stage items and the expected profit of second stage items.The set of constraints (9) make sure that the capacity of each knapsack is not violated. The set of constraints (10) make sure that each item from first stage is assigned to at most one knapsack. The set of constraints (11) make sure that for each scenario the each knapsack is not violated. The set of constraints (12) make sure that for each scenario, each item from second stage is assigned to at most one knapsack. Finally, constraints (13) and (14) say that all variables are binary.

This model has an special structure, in which a block of constraints repeated for different scenarios (Fig 1).

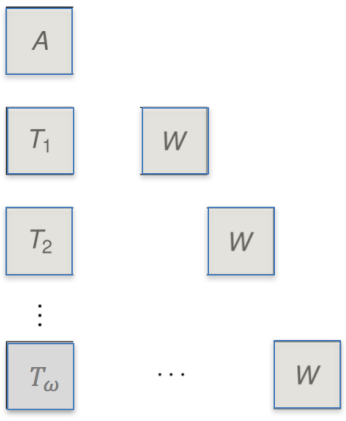


Fig1.Special structure of the problem

In Fig 1, matrix A represents a block of constraints that has only x variables. Constraints (9) and (10) are in this category. Then, other block of constraints exist in which there is a link between x and y variables. Matrix T in Fig 1 represents x's coefficients and matrix W is the y's coefficients for each scenario. i.e. a block of constraints is repeated for each scenario. Constraints (11) and (12) are in this category. Note that in (12) x's coefficients are all zero. Using the special structure of the problem, it can be reformulated in a two-stage form as follows:

$$\text{maximize} \sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{N}} c_i x_{ij} + \mathcal{Q}(x) \tag{15}$$

$$\text{s.t.} \sum_{i\in\mathcal{M}} a_i x_{ij} \leq b_j \quad \forall j \in \mathcal{N} \tag{16}$$

$$\sum_{j\in\mathcal{N}} x_{ij} \leq 1 \quad \forall i \in \mathcal{M} \tag{17}$$

$$x_{ij} \in \{0,1\}, \quad i \in \mathcal{M}, \quad j \in \mathcal{N} \tag{18}$$

where

$$\mathcal{Q}(x) = \mathbb{E}[Q(x,\omega)]$$

and

$$\mathcal{Q}(x,w) = \text{maximize} \sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{N}} q_k^{\omega} y_{kj}^{\omega} \tag{19}$$

$$\text{s.t.} \sum_{i\in\mathcal{M}} a_i x_{ij} + \sum_{k\in\mathcal{K}} \omega_k y_{kj}^{\omega} \leq b_j \quad \forall j \in \mathcal{N}, \forall \omega \in \Omega \tag{20}$$

$$\sum_{j\in N} y_{kj}^{\omega} \leq 1 \quad \forall k \in \mathcal{K}, \forall \omega \in \Omega \tag{21}$$

$$y_{ij}^{\omega} \in \{0,1\}, \quad k \in \mathcal{K}, \quad j \in \mathcal{N} \tag{22}$$

The problem described by model (15-18) is called first stage problem or the Master Problem, and the problem described by model (12-15) is called second stage problem or the Subproblem.

In the following section, the solution approaches will be discussed.

3 Solution approach

3.1 Integer L-shaped Algorithm

To solve the proposed problem, an algorithm is developed based on Integer L-Shaped method that is introduced in [2]. Note that the given problem has the complete recourse property; i.e. any feasible solution for the first stage is feasible for second stage. This property make the algorithm easier as no feasibility cuts need to be generated.

According to Gilbert Laporte [2] definition, this problem can be classified in B/B/D category; meaning that, both first stage and second stage variables are binary and the number of scenario is finite (Ω is not a continues set). The idea of integer L-shaped method is to relax second stage constraints, consider θ as an over estimator for $\mathcal{Q}(x)$ and successively add cuts (optimality cuts) in the (x, θ)-space to better approximate the shape of $\mathcal{Q}(x)$. In the following section, the Integer L-shaped method is summarized:

Table 1. Integer L-shaped Algorithm

1: Compute upper bound U of $\mathcal{Q}$
2: Set counter $v = 0$
3: If $v = 0$, then ignore θ in the objective function. otherwise, add θ to the objective function of the master problem. [Note: This variable (θ) represents the expected value of second stage decision]. Then the objective function would be as follows: $max \sum_i \sum_j c_i x_{ij} + \theta$
4: Solve the first stage problem; declaring x as binary variable. Lets (x^*, θ^*) to be the optimal solution to the master problem.
5: Plug in x^* in the second stage problem and solve it for $\omega \in \Omega$ and compute the expected objective function value of the second stage problem θ_r.
6: if $\theta_r = \theta$ then stop and report the optimal master problem solution. Otherwise go to step 7.
7: Initialize the optimality cut function and add the generated cut to the master problem. v=v+1, then go to 3.

This procedure is depicted in the following flowchart, assuming $\Omega = \{1, 2, 3\}$

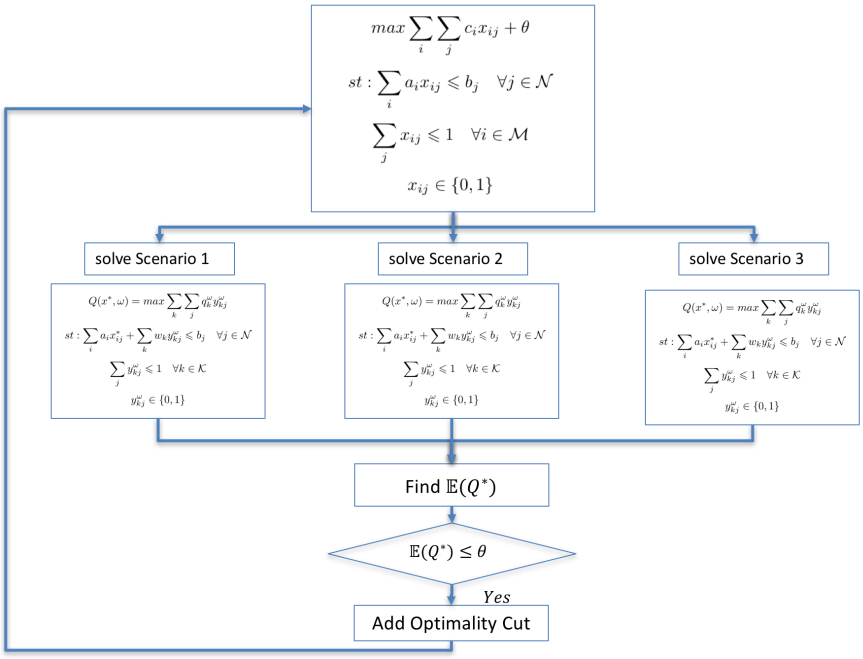


Fig2.Integer L-shaped algorithm, assuming $\Omega = \{1, 2, 3\}$

Integer L-shaped Optimality Cut:
As shown in the flowchart of the Integer L-shaped Algorithm, in each iteration an optimality cut should be added to the master problem in order to improve the estimation of the expected second stage profit or θ. In this study, the cut is generated using the idea presented by Laporte and Louavex. The cut is as following:

$$\theta \leqslant (\theta_r - U)(\sum_{(i,j)\in S_r} x_{ij} - \sum_{(i,j)\notin S_r} x_{ij} - |S_r| + 1) + U$$

$\theta_r = \mathbb{E}(Q^*)$ is the expected profits from the second stage problems; $|S_r|$ is the cardinality of S_r which is a set of (i, j) where $x_{ij} = 1$; and U is the upper bound on the θ and it can be find by plugging $x_{ij} = 0$ in the subproblems $U = E(Q^*(x = 0))$. As it has been proven by Laporte and Louavex, this inequality is a VI and the algorithm will be converged in finite number of iteration.

Proof: The quantity $\sum_{(i,j)\in S_r} x_{ij} - \sum_{(i,j)\notin S_r} x_{ij}$ is always less than or equal to $|S_r|$. It takes the value of $|S_r|$ only when x is the r^th feasible solution. Now, when $\sum_{(i,j)\in S_r} x_{ij} - \sum_{(i,j)\notin S_r} x_{ij} = |S_r|$, the inequality reduced to $\theta \leqslant \theta_r$ and otherwise it will be less than or equal to U.

The L-shaped Algorithm with the introduced cuts will be converged to optimal solution in finite number of iterations. The proof for this argument is that the number of feasible solution for the first stage is finite as all x's and y's coefficients are positive. This argument is also necessary in finding a bounded value for U.

3.2 Single-cut Benders Algorithm

Although the Integer L-shaped cut is a VI and it is guaranteed to help the algorithm to get converged to optimal solution in finite number of iterations, in practice, the number of iterations increases by increasing the instance's size. Remember that the master problem is a pure IP problem. Hence, by increasing the number of iterations and as a result adding more and more cuts to it, the solution time will be increased, significantly. Therefore, in this section, another cut will be introduced based on the Benders Decomposition idea. The Benders algorithm, also known as Non-Integer L-shaped method, is using the dual information of subproblems in order to find the optimality cuts. In this problem, since the subproblems are pure IP, the dual information is not available, unless this integrability constraints get relaxed.
Here is the Benders algorithm that is used in this study:

Table 2. Benders Algorithm

1: Set counter $v = 0$
2: If $v = 0$, then ignore θ in the objective function. otherwise, add θ to the objective function of the master problem. The objective function would be as follows: $max \sum_i \sum_j c_i x_{ij} + \theta$
3: Solve the first stage problem; declaring x as binary variable. Lets (x^*, θ^*) to be the optimal solution to the master problem.
4: Plug in x^* in the second stage problem and solve the LP-relaxation of each subproblems for $\omega \in \Omega$ and compute the expected objective function value $\bar{\theta_r}$ and optimal dual variables π_ω.
5: if $\theta <= \bar{\theta_r}$ then stop and report the optimal master problem solution. Otherwise go to step 6.
7: Initialize the optimality cut function and add the generated cut to the master problem. v=v+1, then go to 3.

Benders Optimality Cut:
Each subproblem is in the form of:

$$max\ QY$$
$$st:\ WY \leqslant h - TX^*$$
$$Y \geqslant 0$$

It should be noted that all subproblem's constraints can be formulated in this way, including the constraints in the form of $y_{jk}^{\omega} \leqslant 1$ which must be added after relaxing the problem.
Then, the dual of this problem can be formulated in this form:

$$min\ \pi(h - TX^*)$$
$$st:\ W\pi \geqslant Q$$
$$\pi \geqslant 0$$

Using this formulation, and assuming p_ω to be the associated probability of each scenario, the optimality cut will be as following:

$$\theta \leqslant p_\omega \pi_\omega (h_\omega - T_\omega X)$$

Note that the RHS of the expression is expectation of objective function of the dual problem and in the optimality it will be equal to the expected objective function value of the primal problem. So, For any given vector X, θ can be at most be equal to the RHS (in the optimality) and otherwise it will be lower than RHS. Hence, the given inequality is a VI. for more information about this VI refer to [20].

After all, y^* might not be integer as the integrality requirements have been relaxed in the subproblems. Therefore, the following algorithm is proposed (assuming $\Omega = \{1, 2, 3\}$).

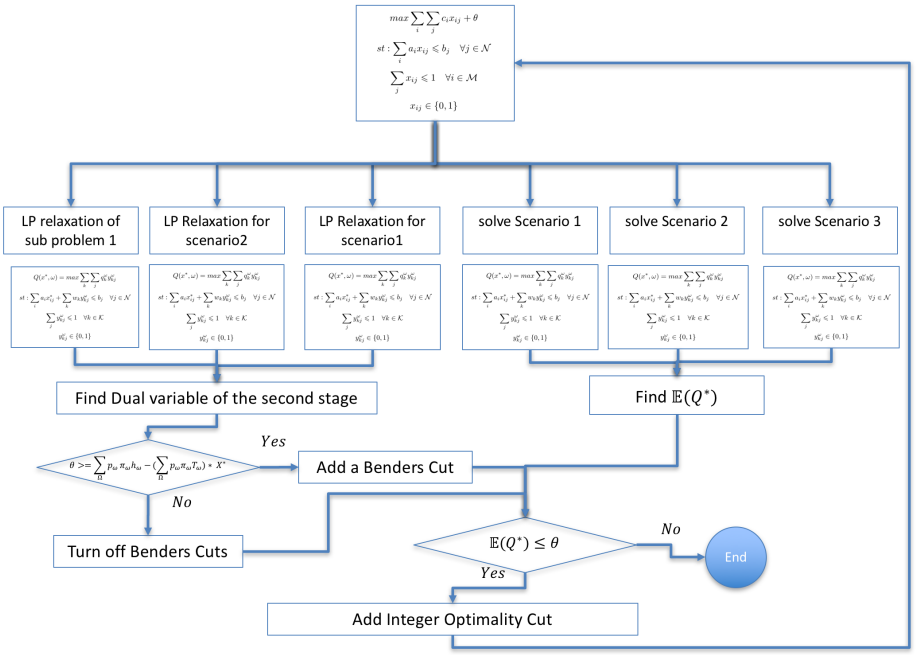


Fig3.Combining Benders and Integer L-shaped algorithms, assuming $\Omega = \{1, 2, 3\}$

According to the given flowchart, in each iteration the master problem is solved. Then for every scenario x^* is plugged in both the subproblem with and without integrality constraint and the solutions are stored, appropriately. Until the Benders check that has been described in the Table 2. is satisfied, or the algorithm is terminated, Benders cut is added to the master problem in each iteration. When the Benders check is returned False value, it means that the optimality condition of the relaxed problem is satisfied. So, the algorithm stop generating this cut. Then, the algorithm continues until the optimal integer solution is found.

4 Result and Discussion

The proposed algorithms are developed in Python and are solved by Gurobi 7.5.1. Then, 13 different instances were randomly generated and solved on a PC with 2.9 GHz Intel Core i5 processor and 8 GB RAM. In this section, first the random parameters are discussed then different numerical examples are introduced. Subsequently, the result and discussion are made.

4.1 Random parameters

Different parameters is randomly generated using uniform distribution as follows:

c : Uniformly distributed between 1 and 100.
a : Uniformly distributed between 10 and 50.
b : Uniformly distributed between 100 and 200.
w : Uniformly distributed between 10 and 100.
q : Uniformly distributed between 1 and 200.

4.2 Generating different instance size:

In this section, 13 different problem instances with different sizes are introduced in order to be solved by the proposed algorithms. Then, a conclusion can be made by comparing the quality of the solutions as well as the solution time.

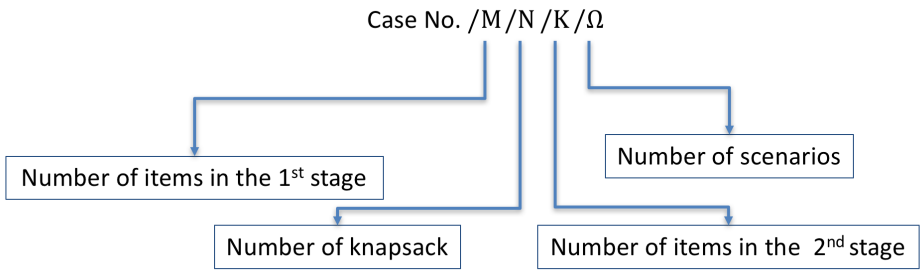


Fig4. Designing numerical instances

4.3 Results

To assess the results, the extensive form model is also developed and the same instances would be solved by this model. The solution time limit of 600 seconds is applied for all the algorithms in order to assess the quality of the algorithms. The results are summarized in the table below:

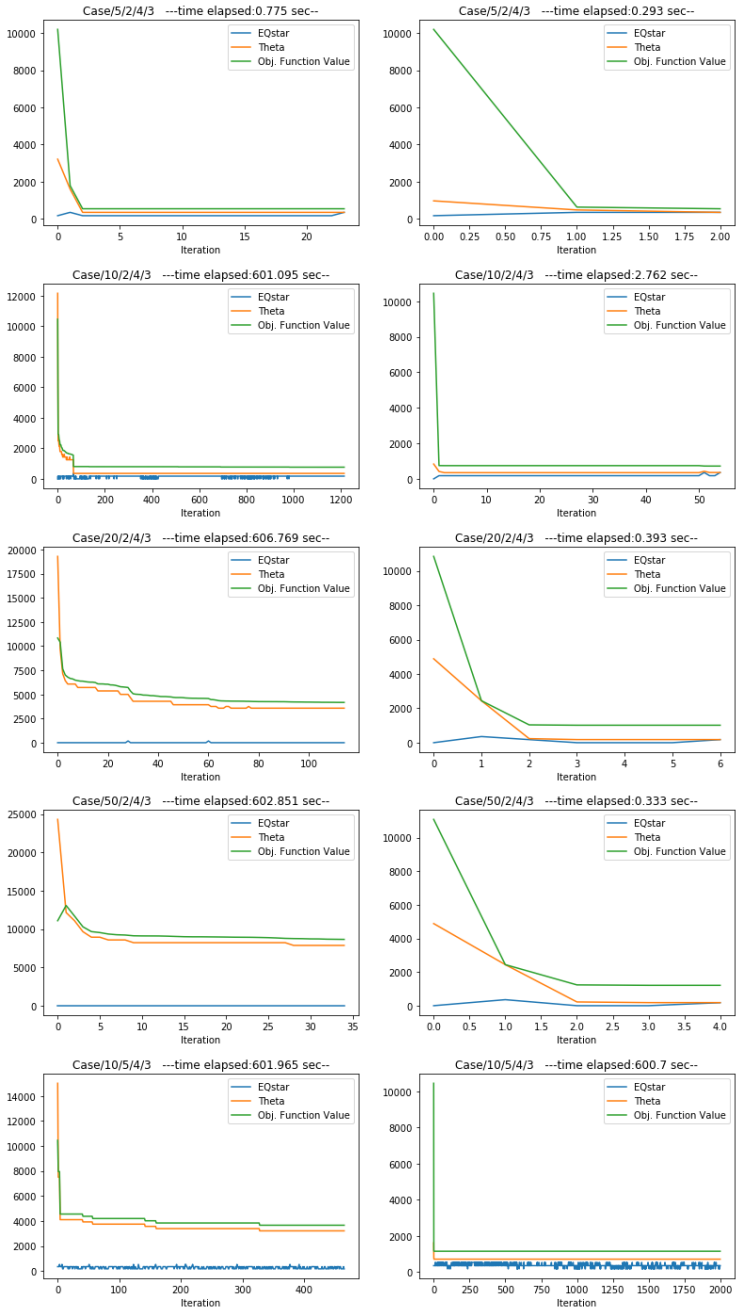
Table3. Overall Results

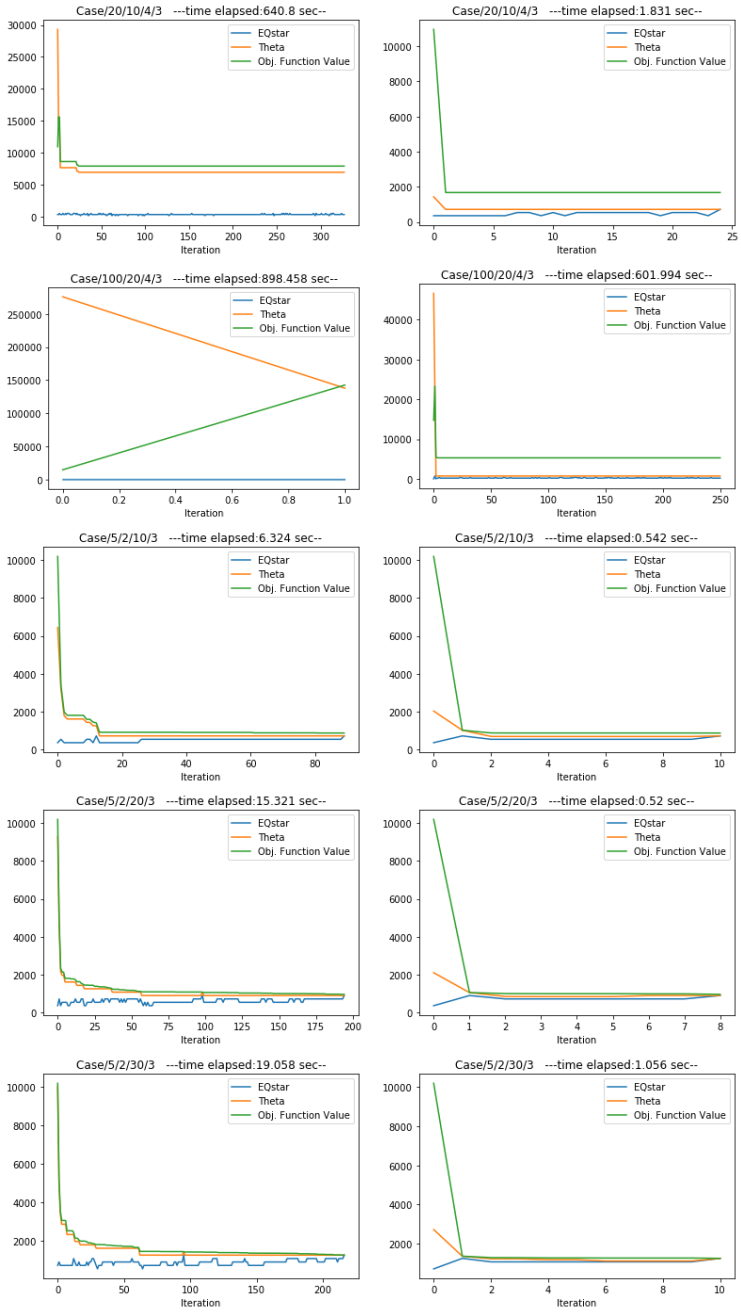
Case No.	Extensive form	L-shaped with only Integer Cut *		L-shaped with Both integer cut and Benders cut*	
	Obj. Value	Obj. Value	% Gap	Obj. Value	% Gap
5/2/4/3	550.33	550.33	0	550.33	0
10/2/4/3	720.33	797.33	10.68	720.33	0
20/2/4/3	1012.66	3851	280.28	1012.66	0
50/2/4/3	1211.17	8646.33	613.88	1211.17	0
10/5/4/3	1153.66	3665	217.68	1153.66	0
20/10/4/3	1675.66	7929	373.18	1675.66	0
100/20/4/3	5279.66	142630.66	2601.51	5280.04	0.0071
5/2/10/3	866.66	866.66	0	866.66	0
5/2/20/3	955.33	955.33	0	955.33	0
5/2/30/3	1250.66	1250.66	0	1250.66	0
5/2/4/10	637	637	0	637	0
5/2/4/20	1024	1024	0	1024	0
5/2/4/100	1024	1024	0	1024	0

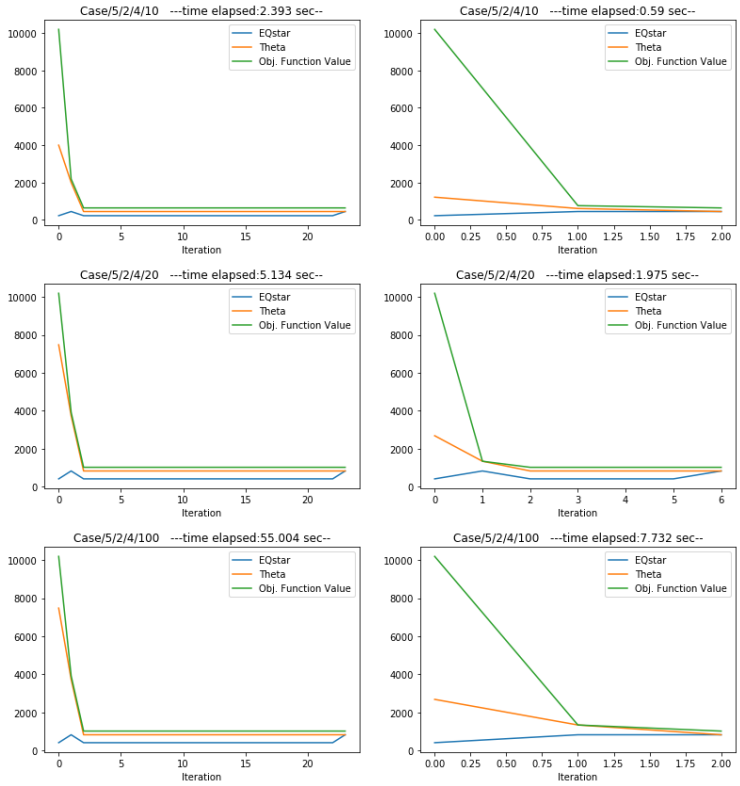
**time limit=600 seconds*

4.4 Discussion

As it is shown in Table 2, the first algorithm with only Integer L-shaped optimality cut is not as effective as the second algorithm with both Integer L-shaped and Benders cuts. For example in the Case 50/2/4/3, there is a 613.88 percent gap between the optimal solution and the best solution that has been found by the first algorithms in the specified time limit. While by adding the Benders cut, the algorithm can converge to the optimal solution in the same time limit. The following figures demonstrate the convergence of both algorithms in the 600 seconds time limit. The left columns is associated with Integer L-shaped algorithm and the second column is depicting the performance of the second algorithm.


Case/5/2/4/3 ---time elapsed:0.775 sec--
Case/5/2/4/3 ---time elapsed:0.293 sec--
Case/10/2/4/3 ---time elapsed:601.095 sec--
Case/10/2/4/3 ---time elapsed:2.762 sec--
Case/20/2/4/3 ---time elapsed:606.769 sec--
Case/20/2/4/3 ---time elapsed:0.393 sec--
Case/50/2/4/3 ---time elapsed:602.851 sec--
Case/50/2/4/3 ---time elapsed:0.333 sec--
Case/10/5/4/3 ---time elapsed:601.965 sec--
Case/10/5/4/3 ---time elapsed:600.7 sec--
EQstar
Theta
Obj. Function Value
Iteration


Case/20/10/4/3 ---time elapsed:640.8 sec--
Case/20/10/4/3 ---time elapsed:1.831 sec--
Case/100/20/4/3 ---time elapsed:898.458 sec--
Case/100/20/4/3 ---time elapsed:601.994 sec--
Case/5/2/10/3 ---time elapsed:6.324 sec--
Case/5/2/10/3 ---time elapsed:0.542 sec--
Case/5/2/20/3 ---time elapsed:15.321 sec--
Case/5/2/20/3 ---time elapsed:0.52 sec--
Case/5/2/30/3 ---time elapsed:19.058 sec--
Case/5/2/30/3 ---time elapsed:1.056 sec--
EQstar
Theta
Obj. Function Value
Iteration



By looking to these graphs, it can be argued that the first algorithm is much more sensitive to the number of items in the first stage and number of knapsacks, comparing to its sensitivity to other parameters. In other words, by increasing in number of items in the first stage and number of knapsacks, the convergence of the algorithm is become slower and slower. But the convergence is reasonable when the number of second stage items and number of scenarios got increased. This result might be expected by looking to the structure of the proposed algorithm, where the master problems need to be solved in each iteration by having the optimality cuts added to the model in each iteration. So the parameters associated with the master problem play a more important role in the convergence speed of the algorithm, comparing the the other parameters. Second algorithm, on the other hand, seems to be very effective in all cases. It could find the optimal solution with zero gap in 12 cases and it ends up with 0.0071 gap in the case 100/20/4/3. Also it should be highlighted that even in that case, the problem got into a very small gap in the first few iterations.

To analyze the performance of the first algorithm, it must be said that this shortcoming is reported in other articles [1,2]. The reason is that as explained in section 3.1 the optimality cut ensures $\theta \leqslant \mathbb{E}(Q(X^r))$, but only for that particular point X^r at which it has been defined. For the other integer points this cut is reduced to a very loose constraint $\theta \leqslant U$ which is trivial.

5 Conclusion

An stochastic multiple binary knapsacks problem is considered in this study. There was an uncertainty in the profit of a group of items. This problem was formulated as a two stage stochastic problem and based on its special structure, two exact algorithms were developed to maximize the total profit. The relative complete recourse property of this problem makes the problem easier as any feasible solution for the first stage problem is also feasible for the second stage problem. In this study, two optimality cuts have been introduced and their effectiveness has been examined by introducing 13 randomly generated numerical instances. Both cuts and their associated computations has been developed in Python 2.7 and solved using Gurobi 7.5.1.

The first optimality cut that has been used in this study was Integer L-shaped cut. The second cut was Benders cut that used the dual information of the LP-relaxation of subproblems in order to be computed.

The result shows that the Integer L-shaped cut is not as effective as Benders cut for this particular problem as in many instances, there was a huge gap between the optimal solution and the best solution that has been found by this algorithm in 600 seconds time limit. By adding Benders cut as well as Integer L-shaped cut, the performance of the algorithm was improved significantly.

Although the proposed algorithms shows a reliable performance, especially the second algorithm, the authors believe that the convergence speed can be improved. Therefore, finding better cuts, clustering of scenarios, and removing unnecessary cuts from the list of already generated cuts might be some direction in which the future researches can be directed.

References:

1- S. Ahmed, S. S. Dey, "Improving the integer L-shaped method", Informs journal on Computing,28 (3) (2016) 483-499.

2- G. Laporte,F.V. Louveaux,"The integer L-shaped method for stochastic integer programs with complete recourse", Operations Research Letters, 13 (1993) 133-142.

3- G. Mavrotas, D. Diakoulaki, A. Kourentzis, "Selection among ranked projects under segmentation, policy and logical constraints", Eur J Oper Res, 187(1) (2008) 177–192.

4- D.C. Vanderster, NJ. Dimopoulos, R. Parra-Hernandez, "Resource allocation on computational grids using a utility model and the knapsack problem", Future Gener Comput Syst, 25(1) (2009) 35–50.

5- F.S. PeetaS Salman, D. Gunnec, "Pre-disaster investment decisions for strengthening a highway network", Comput Oper Res, 37(10) (2010) 1708–1719.

6- J. Lorie, L.J. Savage, "Three problems in capital rationing", Journal of

Business, 28 (1955) 229-239.

7- A.S. Manne, H.M. Markowitz, "On the solution of discrete programming problems", Econometrica, 25 (1957) 84-110.

8- P.C Gilmore, R.E Gomory, "The theory and computation of knapsack functions", Operations Research, 14 (1966) 1045-1075.

9- E. Balas, E. Zemel, "An algorithm for large zero–one knapsack problems", Operations Research, 28 (1980) 1130-1145.

10- D. Fayard, G. Plateau, "Algorithm 47: An algorithm for the solution of the knapsack problem", Computing, 28 (1982) 269-287.

11- S. Martello, P. Toth, "A new algorithm for the 0–1 knapsack problem", Management Sciences, 34 (1988) 633-644.

12- J. Edmonds, "Minimum partition of a matroid into independent subsets", Journal of Research of the National Bureau of Standards, 69B (1965) 67-72.

13- C.C. Petersen, "A capital budgeting heuristic algorithm using exchange operations", AIIE Transactions, 6 (1974) 143-150.

14- R.E. Echols, L. Cooper, "Solution of integer linear programming problems with direct search Journal of the Association for Computational Machinery", 15 (1968) 75-84.

15- B. Gavish, H. Pirkul, "Efficient algorithms for solving multiconstraint zero–one knapsack problems to optimality", Mathematical Programming, 31 (1985) 78-105.

16- P. Chu, J. Beasley, "A genetic algorithm for the multidimensional knapsack problem Journal of Heuristics", 4 (1998) 63-86.

17- R.L. Carraway, R.L. Schmidt, and L.R. Weatherford, "An algorithm for maximizing target achievement in the stochastic knapsack problem with normal returns", Naval Research Logistics, 40 (1993) 161–173.

18- M. Henig, "Risk criteria in a stochastic knapsack problem", Operations Research, 38(5) (1990) 820–825.

19- M. Sniedovich, "Preference order stochastic knapsack problems: methodological issues", The Journal of the Operational Research Society, 31(11) (1980) 1025–1032.

20- J. Murphy, "Benders, Nested Benders and stochastic programming: An intuitive introduction, Cambridge University Engineering Department Technical Report", (2013).

6 Appendix

Python code

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 24 00:17:27 2017

@author: hosseintohidi
"""
import time as time
import sys
sys.path.insert(0, '/Users/hosseintohidi/Desktop/Lshape_code/')
from  knapsack2 import *
import numpy as np
import os
os.getcwd()
from gurobipy import *

import openpyxl
import xlrd
path='/Users/hosseintohidi/Desktop/Workbook1.xlsx'
sheet_name='Sheet8'
wb=openpyxl.load_workbook(path)
wb.get_sheet_names()
sheet=wb.get_sheet_by_name(sheet_name)

book = xlrd.open_workbook(path)
first_sheet=book.sheet_by_name(sheet_name)
starttime=time.clock()
# set parameters
m=100
n=20
k=4
kprime=k
omega=3
M=[i for i in range(m)]
N=[i for i in range(n)]
K=[i for i in range(k)]
Omega=[i for i in range(omega)]
a=[];b=[];c=[];q=[];w=[];teta=0;z=-10000
for i in M:


    a.append(sheet['B'+str(i+2)].value)
    c.append(sheet['A'+str(i+2)].value)
a=map(int,a)
c=map(int,c)
for j in N:
    b.append(sheet['C'+str(j+2)].value)
for s in K:
    w.append(sheet['D'+str(s+2)].value)
q=[]
for kk in K:
    qtemp=[]
    qtemp.append(first_sheet.row_values(2)[4:4+omega])
    q.append(map(int,qtemp[0]))


b=map(int,b)
w=map(int,w)

model2s=Model()

def model2s_solve(k,sol):

        omegatemp=k
        model2s= Model('second-stage problem')

        xx=np.zeros((len(M),len(N)))

        for i in M:
            for j in N:

                st='X['+str(i)+','+str(j)+']'
                xx[i][j]=sol.get(st)

        y = model2s.addVars(K, N,Omega ,name = "Y",vtype = GRB.BINARY)
        #y = model2s.addVars(K, N,Omega ,name = "Y")

        obj2 = model2s.addVars(Omega ,name = "OBJ2")

        model2s.addConstrs((quicksum(a[i]*xx[i][j] for i in M)+quicksum(w[k]*y[k,j,omega] for k in K )<=b[j]\
                            for j in N for omega in Omega if omega==omegatemp) ,"C01")
```

```
            model2s.addConstrs((quicksum(y[k,j,omega] for j in N)<=1 for k in K for omega in Omega if omega==omegatemp) ,"C02")
            model2s.addConstrs((obj2[omega]==quicksum(q[k][omega]*y[k,j,omega] for j in N for k in K)\
                                for omega in Omega if omega==omegatemp) ,"C03")
            # Objective
            obj1=quicksum(c[i]*xx[i][j] for i in M for j in N)
            obj3=quicksum(1.0/len(Omega)*obj2[omega]for omega in Omega if omega==omegatemp)

            model2s.setObjective(obj3, GRB.MAXIMIZE) # maximize profit
            model2s.setParam( 'OutputFlag', False )

            model2s.update()
            #model2s.write("model32.lp")

            model2s.optimize()

            sol2={}
            for v in model2s.getVars():
                sol2.update({v.Varname: v.X})

            return sol2
    #solve the relaxed second stage problem to generate benders cuts
    #create the T matrix
    T=np.zeros([kprime+n,m*n])
    nnn=0
    for jj in N:
        for ii in M:
            T[jj,(ii*n)+jj]=a[ii]
        nnn+=1
    #create H matrix
    H=b+[1]*k

    def Benders_cut(k,sol,T,H):
            omegatemp=k
            modelB= Model('second-stage problem')

            xx=np.zeros((len(M),len(N)))

            for i in M:
                for j in N:




                    st='X['+str(i)+','+str(j)+']'
                    xx[i][j]=sol.get(st)

           y = modelB.addVars(K, N,Omega ,name = "Y",ub=1,lb=0)
           obj2 = modelB.addVars(Omega ,name = "OBJ2")

           modelB.addConstrs((quicksum(a[i]*xx[i][j] for i in M)+quicksum(w[k]*y[k,j,omega] for k in K )<=b[j]\
                              for j in N for omega in Omega if omega==omegatemp) ,"C01")
           modelB.addConstrs((quicksum(y[k,j,omega] for j in N)<=1 for k in K for omega in Omega if omega==omegatemp) ,"C02")
           # Objective
           obj3=quicksum(quicksum(q[k][omega]*y[k,j,omega] for j in N for k in K)for omega in Omega if omega==omegatemp)

           modelB.setObjective(obj3, GRB.MAXIMIZE) # maximize profit
           modelB.setParam( 'OutputFlag', False )

           modelB.update()
           modelB.write("model32.lp")

           modelB.optimize()
           dualB=modelB.getAttr('pi') # get the optimal dual variables' values

           return dualB,modelB.objVal

    #Build model:
    model = Model('Two-stage stochastic multiple binary knapsack problem extensive form')
    x = model.addVars(M, N, name = "X",vtype = GRB.BINARY)
    y = model.addVars(K, N,Omega ,name = "Y",vtype = GRB.BINARY)
    obj2 = model.addVars(Omega ,name = "OBJ2")
    teta = model.addVar(name = "Teta")
    model.addConstrs((quicksum(a[i]*x[i,j] for i in M)<=b[j] for j in N) ,"C1")
    model.addConstrs((quicksum(x[i,j] for j in N)<=1 for i in M) ,"C2")

    # Objective
    obj1=quicksum(c[i]*x[i,j] for i in M for j in N)
    obj3=quicksum(1.0/len(Omega)*obj2[omega]for omega in Omega)

    #find L (here it is an upper bound)
    L=findL(m,n,k,omega,c,a,b,w,q)

    check=True
```

```
BendersCheck=False
vv=0
nn=0
EE=[]
Ef=[]
objlist=[]
teta2=0
EQstar=0

while check:
    nn=nn+1
    if vv==0:
        obj=obj1
        teta2=10000
    else:
        obj=obj1+teta
    model.setParam( 'OutputFlag', False )
    model.setObjective(obj, GRB.MAXIMIZE) # maximize profit
    model.update()
    model.optimize()
    sol={}
    for v in model.getVars():
        sol.update({v.Varname: v.X})
    teta2=sol.get('Teta')
    if vv==0:
        teta2=10000
    #solve model2s for k in K
    sol3={}
    EQstar=0
    sxstar=[]
    xstarindex=[]
    ystarindex=[]

    sx=[]
    yindex=[]
    xindex=[]
    dual=[]
    objB=[]
    for k in Omega:
        sol3={}



        sol3=model2s_solve(k,sol)
        EQstar+=1.0/len(Omega)*sol3.get('OBJ2['+str(k)+']')
        #find benders cut
        dualtemp,objBtemp=Benders_cut(k,sol,T,H)
        dual.append(dualtemp)
        objB.append(objBtemp)


   e=0
   T=np.array(T)
   for k in Omega:
       for kkk in range(n+kprime):
           e+=1.0/len(Omega)*dual[k][kkk]*H[kkk]
   E=np.zeros([omega,m*n])
   for k in Omega:
       for iii in range(m*n):
           for kkk in range(n+kprime):
               E[k,iii]+=1.0/len(Omega)*dual[k][kkk]*T[kkk,iii]

   #check to see if benders cut can be generated:
   E=sum(E)
   xx=np.zeros(m*n)
   nnn=0
   for i in M:
       for j in N:
           st='X['+str(i)+','+str(j)+']'
           xx[nnn]=sol.get(st)
           nnn+=1
   ww=e-E.dot(xx)
   #ww=sum(objB)

   if teta2<= ww :
       BendersCheck=False

   E= np.reshape(E,[m,n])
   EE.append(EQstar)
   Ef.append(teta2)
   if nn==1:
       objlist.append(model.objVal+teta2)
   else:
```

```
    objlist.append(model.objVal)
if abs(EQstar-teta2)<=10: #or time.clock()-starttime>600:
    check = False
else:
    print(nn,'Obj=',round(model.objVal,2),'EQstar',round(EQstar,2),'theta',round(teta2,2),'BendersCheck:',BendersCheck)
    #add optimality cut
    for i in M:
        for j in N:
            st='X['+str(i)+','+str(j)+']'

            if sol.get(st)==1:
                sxstar.append(st)
                xstarindex.append(i)
                ystarindex.append(j)

            else:
                sx.append(st)
                xindex.append(i)
                yindex.append(j)

    cutname='cut'+str(nn)
    model.reset()
    #model.addConstr((teta<=(EQstar-L)*(sum(x[i,j] for i,j in zip(xstarindex,ystarindex))-\
                            #sum(x[i,j] for i,j in zip(xindex,yindex))-len(sxstar))+EQstar),cutname)
    #if  BendersCheck==False:
    model.addConstr((teta<=(EQstar-L)*(sum(x[i,j] for i,j in zip(xstarindex,ystarindex))-sum(x[i,j] for i,j in zip(xindex,yindex))-len(xstarindex)+1)+

    #add benders cut

    bendcut='benderscut'+str(nn)
    #model.reset()
    if BendersCheck:
        model.addConstr(teta <=e-sum(E[i][j]*x[i,j] for i in M for j in N),bendcut)

    model.update()
    #model.write("test22.mps")
    model.write("test22.lp")

vv+=1
```