Stochastic continuous knapsack problem

Nikita Sergeev, BS20-AI

May 2023

1 Qualitative problem formulation

Consider n (divisible) items numbered from 1 up to n and associated with the wights a_i and values c_i, $i = 1, \dots, n$. The weight bound $b > 0$ is not known and it will determined in the first stage such that one has cost $d > 0$ per a weight unit. The weight bound $b > 0$ is affected by some additive stochastic error ξ from a given interval $[-\epsilon, \epsilon], \epsilon > 0$. In the second stage one gets an exact weight bound and the knapsack can be optimally (continuously) filled.

2 Problem formalization in the form of StO problem

Initially we have the following information about the knapsack:

- $x \in [0; 1]^n$ is decision variable corresponds the fraction of the item which will be taken into the knapsack
- $a \in \mathbb{R}^n_+$ is corresponds to the weights of the items
- ξ - stochastic error, which depends on parameter $\omega \sim \mathcal{N}(\mu, \sigma^2)$. $\xi(\omega)$ corresponds to compact set $[-\epsilon, \epsilon]$
- b - weight capacity of the knapsack. It's affected by stochastic error ξ and not known explicitly on the first stage
- $c \in \mathbb{R}^n$ - cost of the items in the knapsack, sum of which we would like to maximize. All other variables are the same as in the first stage.

So, we may formalize this **Stochastic two stage continuous knapsack problem** as follows:

$$\max_{x \in [0;1]^n} \sum_{i=1}^{n} c_i x_i + \mathbb{E}\left[\mathcal{Q}(x, \xi)\right] \tag{1}$$

$$s.t. \ \sum_{i=1}^{n} x_i a_i \leq b + \xi \tag{2}$$

$$\mathcal{Q}(x, \xi) = \max_{y^+, y^- \in [0;1]^n} \sum_{i=1}^{n} \overline{c_i} y_i^+ - d \sum_{i=1}^{n} a_i y_i^- \tag{3}$$

$$s.t \ y_k^+ \leq 1 - x_k, k = 1, \dots, n \tag{4}$$

$$y_k^- \leq x_k, k = 1, \dots, n \tag{5}$$

$$y_k^+ \leq \mathbb{1}_{\mathbb{R}^+}\left(b + \xi - \sum_{i=1}^{n} a_i x_i\right) \cdot y_k^+, k = 1, \dots, n \tag{6}$$

$$y_k^- \leq \mathbb{1}_{\mathbb{R}^+}\left(\sum_{i=1}^{n} a_i x_i - b + \xi\right) \cdot y_k^-, k = 1, \dots, n \tag{7}$$

$$\sum_{i=1}^{n} (x_i + y_i^+ - y_i^-) a_i \leq b + \xi \tag{8}$$

In the given formalization of the problem we have the following parts:

- $\mathcal{Q}(x, \xi)$ - penalization function. It takes into account overload or under load of the knapsack from the first stage and correct the value of the considered functional.
- y^+, y^- are decision vectors, which indicates, what fraction of item $i(y_i^-)$ we will exclude from the knapsack in the case of overload, and and what fraction of item $i(y_i^+)$ we will add to the knapsack in case of not optimally filled knapsack.
- $\mathbb{1}_{\mathbb{R}^+}$ denotes indicator function of the positive real interval. Constraints (6) and (7) assure that items can be added in the second stage only if knapsack is not optimally filled and items can be removed from the knapsack only in the case of overload.
- $\overline{c_i}$ denotes new value of the item, after we will add it in the second stage $\overline{c_i} < c_i$.
- d cost, what we will pay for every weight unit, which overload knapsack. $d \geq \max_{i \in [1,\dots,n]} c_i$

2.1 Properties of the Two stage continuous knapsack problem:

Property 1. The random variable $\mathcal{Q}(x, \xi)$ can rewritten as the linear combination of 2 random variables $\mathcal{Q}_1(x, \xi)$ and $\mathcal{Q}_2(x, \xi)$:

$$
\begin{aligned}
&\mathcal{Q}_1(x, \xi) = \max_{y^+ \in [0,1]^n} \sum_{i=1}^{n} \overline{c_i} y_i^+ \\
&s.t.\ y_k^+ \leq 1 - x_k, k = 1, \dots, n \\
&y_k^+ \leq \mathbb{1}_{\mathbb{R}^+} \left(b + \xi - \sum_{i=1}^{n} a_i x_i \right) \cdot y_k^+, k = 1, \dots, n \\
&\mathbb{1}_{\mathbb{R}^+} \left(b + \xi - \sum_{i=1}^{n} a_i x_i \right) \sum_{i=1}^{n} a_i (x_i + y_i^+) \leq b + \xi
\end{aligned}
$$

and

$$
\begin{aligned}
&\mathcal{Q}_2(x, \xi) = \min_{y^- \in [0,1]^n} \sum_{i=1}^{n} a_i y_i^- \\
&s.t.\ y_k^- \leq x_k, k = 1, \dots, n \\
&y_k^- \leq \mathbb{1}_{\mathbb{R}^+} \left(\sum_{i=1}^{n} a_i x_i - (b + \xi) \right) \cdot y_k^-, k = 1, \dots, n \\
&\mathbb{1}_{\mathbb{R}^+} \left(\sum_{i=1}^{n} a_i x_i - (b + \xi) \right) \sum_{i=1}^{n} a_i (x_i - y_i^-) \leq b + \xi
\end{aligned}
$$

And from it follows:

$$\mathbb{E}[\mathcal{Q}(x, \xi)] = \mathbb{E}[\mathcal{Q}_1(x, \xi)] - d \times \mathbb{E}[\mathcal{Q}_2(x, \xi)]$$

Property 2. let $\tilde{x}$ be a feasible first stage solution. If $\tilde{x}$ lead to an overload, i.e. for a given second stage realization of $\hat{\xi}$ we have that $\sum_{i=1}^{n} x_i c_i \geq b + \hat{\xi}$, it follows that $\mathcal{Q}_1(\tilde{x}, \hat{\xi}) = 0$. And in the case of underload we contrary have $\mathcal{Q}_2(\tilde{x}, \hat{\xi}) = 0$

Property 3. Given first stage decision vector x and realization of the stochastic parameter ξ, solving second stage of the problem means solving a deterministic knapsack problem. In the case of overloaded knapsack we can continuously remove items from the knapsack in deterministic way, and in the case of underload we can optimally fill the knapsack till the true weight capacity.

3 Deterministic solution for the second stage:

In this section we suppose that we already had made a decision about vector $x(\tilde{x})$ and stochastic parameter ξ had revealed($\hat{\xi}$). And after this may be formalized in the one of 2 following ways:

- If knapsack was overloaded in the first stage, then we should optimize $\mathcal{Q}_1(\tilde{x}, \hat{\xi})$
- If knapsack was underloaded in the first stage, then we should optimize $\mathcal{Q}_2(\tilde{x}, \hat{\xi})$

For solving this problem we firstly need to rearrange items in the following way:

$$\frac{c_1}{a_1} \geq \cdots \geq \frac{c_n}{a_n} \geq 0$$

Items can't have negative weights, and in this task we assumed, that cost of all items are non negative. Since we consider problem with infinitely divisible items, it's meaningless to have items with negative cost.

Let L be critical item index from the first stage. That mean what $L-1$ most valuable items was fully added to the knapsack on the first stage. And item with the index L was added partially.

$$x_i = \begin{cases} 1, \text{ if } i < L \\ \frac{(b-b') - \sum_{i=1}^{L-1} a_i}{a_L}, \text{ if }, i = k \\ 0, \text{ if } i > k \end{cases}$$

In the case of underloaded knapsack, when we should add items with the weight b' in total we can get vector y^+ in the following way:

Problem of optimizing $\mathcal{Q}_1(\tilde{x}, \hat{\xi})$ then transforms into simple linear programming problem:

$$\begin{aligned} &\mathcal{Q}_1(x, \xi) = \max_{y^+ \in [0,1]^n} \sum_{i=1}^{n} \overline{c_i} y_i^+ \\ &s.t.\ y_k^+ \leq 1 - x_k, k = 1, \ldots, n \\ &\sum_{i=1}^{n} a_i y_i^+ \leq b' \end{aligned}$$

Which can be solved with numerical method from the gurobi package

Problem of optimizing $\mathcal{Q}_2(\tilde{x}, \hat{\xi})$ can be formulated as following:

$$\begin{aligned} &\mathcal{Q}_2(x, \xi) = \min_{y^- \in [0,1]^n} \sum_{i=1}^{n} a_i y_i^- \\ &s.t.\ y_k^- \leq x_k, k = 1, \ldots, n \\ &\sum_{i=1}^{n} a_i y_i^- \geq b' \end{aligned}$$

As we have continuous version of the knapsack problem, we can choose items subset with any weight. So, in case of the overload in the given task we will always have value $\mathcal{Q}_2(x, \xi) = b'$. So, overall penalty for the b' extra weight units will be db', where $d \geq \max_{i \in \{1,\ldots,n\}} c_i$

4 Approximate solution of the problem

One possible way of finding optimal solution for the given task is using method of **Sampling Average Approximation.** Using this method, problem will take the following form:

$$\max_{x\in[0;1]^n} \sum_{i=1}^{n} c_i x_i + \frac{1}{N}\sum_{i=1}^{N} \mathcal{Q}(x,\xi_i)$$

$$s.t.\ \sum_{i=1}^{n} x_i a_i \le b + \xi_i, \forall i \in [1\dots N]$$

$$\mathcal{Q}(x,\xi) = \max_{y^+,y^-\in[0;1]^n} \sum_{i=1}^{n} \overline{c_i} y_i^+ - d\sum_{i=1}^{n} a_i y_i^-$$

$$s.t\ y_k^+ \le 1 - x_k, k = 1,\dots,n$$

$$y_k^- \le x_k, k = 1,\dots,n$$

$$y_k^+ \le \mathbb{1}_{\mathbb{R}+}\left(b + \xi - \sum_{i=1}^{n} a_i x_i\right)\cdot y_k^+, k = 1,\dots,n$$

$$y_k^- \le \mathbb{1}_{\mathbb{R}+}\left(\sum_{i=1}^{n} a_i x_i - b + \xi\right)\cdot y_k^-, k = 1,\dots,n$$

$$\sum_{i=1}^{n}(x_i + y_i^+ - y_i^-)a_i \le b + \xi_{true}$$

N there is number of scenarios, which we will generate to get an approximate value of the functional. But this approximation directly depends of penalty parameters $(d, \overline{c})$. And results of experiments with different penalty parameters presented in the table bellow.
Initial parameters:

- number of items $n = 10$
- items weights $a = [3, 10, 5, 7, 4, 1, 6, 2, 8, 1]$
- items values $c = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$
- capacity of the knapsack $b = 15$
- parameters of the distribution of ξ - $\mu = 0, \sigma = 10, \epsilon = 7$
- number of scenarios $N = 10000$

d	c	max	min	mean	true val
12	$\overline{c_i} = c_i \div 2, \forall i$	34.30	30.09	32.31	34.30
100	$\overline{c_i} = c_i \div 2, \forall i$	34.980	-69.51	-1.47	34.982
10	$\overline{c_i} = (-c_i) \div 2, \forall i$	44.75	29.0	37.09	44.75

As we can see from the results of experiments that convergence of the method is dependent of the penalization parameters d and $\overline{c}$. So, as we can see from the table above, depending on the penalty parameters, variance of the method may become very big. To deal with it, we can add some improvements to this method.

4.1 Robust optimization:

By this method we will consider 2 possible worst scenarios of the problem

- True value of $\xi = \epsilon$ in the second stage, means what we will underload the knapsack in the most of the cases.
- True value of $\xi = -\epsilon$ in the second stage, means what we will overload the knapsack in the most of the cases.

To implement Robust Optimization for the given task, we will reformulate problem in the following way:

$$\max_{x\in[0;1]^n} \sum_{i=1}^{n} c_i x_i + \frac{1}{N}\sum_{i=1}^{N} \mathcal{Q}(x,\xi_i)$$

$$s.t. \sum_{i=1}^{n} x_i a_i \le b + \xi_i + \mathcal{E}, \forall i \in [1\dots N]$$

$$\mathcal{Q}(x,\xi) = \max_{y^+,y^-\in[0;1]^n} \sum_{i=1}^{n} \overline{c_i} y_i^+ - d\sum_{i=1}^{n} a_i y_i^-$$

$$s.t\ y_k^+ \le 1 - x_k, k = 1,\dots,n$$

$$y_k^- \le x_k, k = 1,\dots,n$$

$$y_k^+ \le \mathbb{1}_{\mathbb{R}+}\left(b + \xi - \sum_{i=1}^{n} a_i x_i\right)\cdot y_k^+, k = 1,\dots,n$$

$$y_k^- \le \mathbb{1}_{\mathbb{R}+}\left(\sum_{i=1}^{n} a_i x_i - b + \xi\right)\cdot y_k^-, k = 1,\dots,n$$

$$\sum_{i=1}^{n}(x_i + y_i^+ - y_i^-)a_i \le b + \xi_{true}$$

$\mathcal{E}$ here may take value of ϵ or $-\epsilon$. To get the value of the $\mathcal{E}$, which will maximize value of the functional we will apply Sample Average Approximation method for both values and choose the best one.
Using the same parameters as in the table above, we will get the next result using Robust Optimization method:

method	d	c	max	min	mean	true val
overload	12	$\overline{c_i} = c_i \div 2, \forall i$	34.04	22.02	30.71	34.04
underload	12	$\overline{c_i} = c_i \div 2, \forall i$	44.83	38.43	42.40	44.83
overload	100	$\overline{c_i} = c_i \div 2, \forall i$	-16.11	-208.24	-104.45	33.15
underload	100	$\overline{c_i} = c_i \div 2, \forall i$	37.73	24.17	33.15	38.34
overload	10	$\overline{c_i} = (-c_i) \div 2, \forall i$	38.66	31.90	36.72	38.66
underload	10	$\overline{c_i} = (-c_i) \div 2, \forall i$	37.12	10.02	27.98	39.26

From the results of experiment it's clear, what consideration of both possible worst cases increases precision of Functional approximation.

5 Numerical method problem solution

For checking result of problem solution I implemented solution, described above in python using **Gurobi Optimization** package.

5.1 Function for optimizing $\mathcal{Q}_1(x,\xi)$

```
def second_stage_solution_underloaded(c_reduced, a, x, b_true):
    n = len(c_reduced)  # Number of variables

    b_underload = b_true - sum(a[i] * x[i] for i in range(n))

    # Create a new model
    model = gp.Model("optimization_problem", env=env)

    # Create decision variables
    y_plus = model.addVars(n, vtype=GRB.CONTINUOUS, lb=0, ub=1, name="y_plus")
```

```
    # Set objective function
    model.setObjective(
        sum(c_reduced[i] * y_plus[i] for i in range(n)), sense=GRB.MAXIMIZE
    )

    # Add constraints
    for k in range(n):
        model.addConstr(y_plus[k] <= 1 - x[k])

    model.addConstr(sum(a[i] * y_plus[i] for i in range(n)) <= b_underload)

    # Optimize the model
    model.optimize()

    # Retrieve the solution
    if model.status == GRB.OPTIMAL:
        optimal_solution = [y_plus[i].x for i in range(n)]
        optimal_objective = model.objVal
        return optimal_solution, optimal_objective
    else:
        return [0] * n, 0
```

5.2 Function for optimizing $\mathcal{Q}_2(x, \xi)$

```
def second_stage_solution_overloaded(d, x, a, b_true):
    n = len(x)  # Number of variables

    b_overload = sum(a[i] * x[i] for i in range(n)) - b_true

    # Create a new model
    model = gp.Model("optimization_problem", env=env)

    # Create decision variables
    y_minus = model.addVars(n, vtype=GRB.CONTINUOUS, lb=0, ub=1, name="y_minus")

    # Set objective function
    model.setObjective(sum(d * y_minus[i] for i in range(n)), sense=GRB.MINIMIZE)

    # Add constraints
    for k in range(n):
        model.addConstr(y_minus[k] <= x[k])

    model.addConstr(sum(a[i] * y_minus[i] for i in range(n)) >= b_overload)

    # Optimize the model
    model.optimize()

    # Retrieve the solution
    if model.status == GRB.OPTIMAL:
        optimal_solution = [y_minus[i].x for i in range(n)]
        optimal_objective = model.objVal
        return optimal_solution, optimal_objective
    else:
        return [0] * n, 0
```

5.3 Function for optimizing whole functional using SAA and Robust Optimization

```
    def sample_average_approximation_robust(
    a, c, b, nu, sigma, epsilon, d, c_redused, N, polarity
):
    first_st_sol = []
    second_st_sol_un = []
    second_st_sol_ov = []
    xis = []

    distribution = norm(loc=nu, scale=sigma)

    true_xi = distribution.rvs()
    while abs(true_xi) > epsilon:
        true_xi = distribution.rvs()

    true_b = b + true_xi

    for i in range(N):
        xi = distribution.rvs()
        while abs(xi) > epsilon:
            xi = distribution.rvs()
        b_stochastic = b + xi

        if polarity == "underload":
            b_stochastic -= epsilon
        elif polarity == "overload":
            b_stochastic += epsilon
        else:
            raise NotImplemented

        x, obj = first_stage_solution(c, a, b_stochastic)
        y_plus, obj_plus = second_stage_solution_underloaded(c_redused, a, x, true_b)
        y_minus, obj_minus = second_stage_solution_overloaded(d, x, a, true_b)

        first_st_sol.append(obj)
        second_st_sol_un.append(obj_plus)
        second_st_sol_ov.append(obj_minus)
        xis.append(xi)
    return first_st_sol, second_st_sol_un, second_st_sol_ov, xis, true_xi
```

Whole code used for problem solution together with the experiments can be found in this GitHub repository.

6 Conclusion

In the given work I considered **two stage stochastic continuous knapsack problem**. For solving this problem I used Sample Average Approximation combined with the Robust Optimization. And as it's visible from the tables with the results, mean approximation of the Functional value converges to the true value. But this approach works only for the continuous version of the problem. If we considering **0-1 knapsack** with indivisible items, implemented method will not give us any optimal solution.

7 Reference List

1. S.Kosuch, A.Lisser, "On two-stage stochastic knapsack problems" Discrete Applied Mathematics, vol.159, September 2011. DOI: 10.1016/j.dam.2010.04.006.

2. Christoph Buchheim, Dorothee Henke, Jannik Irmai, The stochastic bilevel continuous knapsack problem with uncertain follower's objective, http://arxiv.org/abs/2108.12303v1