Multiple Choice Knapsack Problem using Cover Inequalities for Optimizing Armor Selection for Elden Ring.

Simanta Barman

1 Introduction

Elden Ring is an Action Role Playing Game (ARPG) developed by FromSoftware and was released on February 25th, 2022. FromSoftware is famous for making very difficult games and they did not make an exception with this game. A player is assigned many base stats in the beginning of the game which can be upgraded by making progress in the game. One of the most important stats is the equip load. The speed at which the player can move, evade attacks by performing dodge rolls and backsteps depends on this stat. Based on the equip load a player can move in four different speeds. Equip load under $30\%, 70\%$, greater than or equal to 70% and 100% of the maximum weight of the player puts load status of the player to light, medium, heavy and overloaded respectively. Light load allows the player to move the fastest and overloaded restricts all evasive actions and allows only very slow movements. Equip loads depends on the sum of the weight of the items carried by the player. Items include weapons, armors, talismans and armaments. Being the heaviest items and crucial to the protection of the player, armor choice is very important. A player has to select four pieces of armor: 1. Head, 2. Arms, 3. Chest, 4. Legs.

The objective of this project is to find out the best armor pieces from all the armor pieces available in the game so that the sum of the weights of the armor pieces is under a maximum weight threshold. The maximum weight or budget weight from this point in this report will refer to the maximum sum of armor piece's weight only. A brute force solution implementation is already available at github. This project will focus on a solution algorithm based on optimization formulation.

The report is organized in five sections. The first section introduces the objective of the project. The second section describes the dataset using which the problem will be tackled. The third section describes the formulation. The fourth section is divided into four subsections: 1. 0-1 Knapsack using Dynamic Programming, 2. Separation Problem, 3. Lifting Proccedure, 4. Solution Algorithm. The fifth section describes some numerical results using the solution methods discussed in the previous sections. Finally the report concludes with the sixth section where some conclusions are presented.

2 Dataset

The dataset includes the information about the armor pieces. Data for 574 armor pieces are available in the dataset. 4 different classes of armor pieces are present in the dataset:

1. Head piece: $h = 169$ piece's data available.
2. Arms piece: $a = 94$ piece's data available.
3. Chest piece: $c = 204$ piece's data available.
4. Legs piece: $l = 107$ piece's data available.

Each armor piece x has some value $p(x) \in \mathbb{R}_+$ for each element p of the set of attributes, $P = \{$ physical, strike, slash, pierce, magic, fire, lightning, holy, immunity, robustness, focus, poise $\}$

3 Formulation

Let, the set of different classes of armors be denoted by $T = \{H, A, C, L\}$. Here the set of all head, arms, chest and legs pieces are denoted by $H = \{H_i\}_{i=1}^{h}$, $A = \{A_i\}_{i=1}^{a}$, $C = \{C_i\}_{i=1}^{c}$ and $L = \{L_i\}_{i=1}^{l}$ respectively. Here, H_i, A_i, C_i and L_i are binary variables with values 1 if that piece is chosen in the optimal solution otherwise 0. Denote the set of all attributes for an armor piece by P. The weight of item $x \in t$ for all $t \in T$ is denoted by $w(x) \in \mathbb{R}_+$. The normalized values for attribute $p \in P$ for an item $x \in t$ for all $t \in T$ can be accessed by $p(x)$. Let m^p be the multiplier for each attribute which indicates the priority for that attibute of the player. Let $b \in \mathbb{R}_+$ be the maximum weight capacity of the character

1. Choose only 1 piece of armor from each class of armors.

$$\sum_{i=1}^{h} H_i = 1 \qquad \text{and, } H_i \in \{0,1\}\, \forall i \in \{1, \cdots, h\} \tag{1}$$

$$\sum_{i=1}^{a} A_i = 1 \qquad \text{and, } A_i \in \{0,1\}\, \forall i \in \{1, \cdots, a\} \tag{2}$$

$$\sum_{i=1}^{c} C_i = 1 \qquad \text{and, } C_i \in \{0,1\}\, \forall i \in \{1, \cdots, c\} \tag{3}$$

$$\sum_{i=1}^{l} L_i = 1 \qquad \text{and, } L_i \in \{0,1\}\, \forall i \in \{1, \cdots, l\} \tag{4}$$

2. Weight cannot exceed the maximum weight capacity of the character.

$$\sum_{i=1}^{h} w(H_i) \cdot H_i + \sum_{i=1}^{a} w(A_i) \cdot A_i + \sum_{i=1}^{c} w(C_i) \cdot C_i + \sum_{i=1}^{l} w(L_i) \cdot L_i \leq b \tag{5}$$

3. The objective is to maximize the attibute values based on the priorities set on the attibutes.

$$\sum_{i=1}^{h} H_i \left(\sum_{p \in P} p(H_i) \cdot m^p \right) + \sum_{i=1}^{a} A_i \left(\sum_{p \in P} p(A_i) \cdot m^p \right) + \sum_{i=1}^{c} C_i \left(\sum_{p \in P} p(C_i) \cdot m^p \right) + \sum_{i=1}^{l} L_i \left(\sum_{p \in P} p(L_i) \cdot m^p \right) \tag{6}$$

Define, the value for piece $x \in t\ \forall t \in T$ by $v(x) = \sum_{p \in P} p(x) \cdot m^p$.

Then the formulation can be written as the standard Multiple Choice Knapsack Problem (MCKP).

$$\max \quad \sum_{t \in T} \sum_{x \in t} v(x) \cdot x \tag{7}$$

$$\text{s.t.} \quad \sum_{t \in T} \sum_{x \in t} w(x) \cdot x \leq b \tag{8}$$

$$\sum_{x \in t} x = 1 \qquad \forall t \in T \tag{9}$$

$$x \in \{0, 1\} \qquad \forall t \in T,\ \forall x \in t \tag{10}$$

Now, consider the Knapsack inequality given by eqs. (8) and (10). The Knapsack set can be written as

$$X = \left\{ x \in \{0,1\}^n : \sum_{t \in T} \sum_{x \in t} w(x) \cdot x \leq b \right\} \tag{11}$$

where, $n = \sum_{t \in T} |t|$ and $N = \{1, \cdots, n\}$. So, an armor piece variable x_i can be accessed by index $i \in N$ where N is the set of indeces associated to each piece of armor of different classes. The weight function is $w : x \to \mathbb{R}_+$ and budget for weight is $b \in \mathbb{R}_+$. So, the formulation can further be simplified to

$$Z_{\text{mckp}} = \max \left\{ \sum_{i \in N} v(x_i) \cdot x_i : \sum_{i \in N} w(x_i) \cdot x_i \leq b, \sum_{x \in t} x = 1\ \forall t \in T, x \in \{0,1\}^n \right\} \tag{12}$$

4 Methodology

We will solve the MCKP by solving the LP relaxation to get a continuous solution. The continuous solution will be used to generate cover inequalities by solving the separation problem which will be described later. A lifting proccedure will then be used to strengthen the cover inequalities. After adding those inequalities to the LP relaxation of the MCKP, the problem will be solved again to get another continuous solution. Then the process will be repeated until the separation problem results in an objective value that satisfies all cover inequalities.

Definition 1 (Wolsey (2020) Definition 9.6)**.** *A set $\mathcal{C} \subseteq N$ is a cover if $\sum_{j \in \mathcal{C}} a_j > b$. A cover is minimal if $\mathcal{C} \setminus \{j\}$ is not a cover for any $j \in \mathcal{C}$.*

Proposition 1 (Wolsey (2020) Proposition 9.2)**.** *If $\mathcal{C} \subseteq N$ is a cover for X, the cover inequality eq.* (13) *is valid for X*

$$\sum_{j \in \mathcal{C}} x_j \leq |\mathcal{C}| - 1 \tag{13}$$

4.1 0-1 Knapsack with Dynamic Programming

The solution proccedure for $0-1$ knapsack with Dynamic Programming (DP) will be used in the following subsections. The DP recursion to solve the 0-1 knapsack is

$$F[r,h] = \begin{cases} F[r-1,h] & \text{if } a_r^{\text{kp}} > h \\ \max\left\{F[r-1,h], c_r^{\text{kp}} + F[r-1,h]\right\} & \text{otherwise} \end{cases} \tag{14}$$

Here, the first case means that if the current item r is added to the knapsack then total weight will exceed h so the current item cannot be included in the knapsack. The second case means that since current item r can be included in the knapsack, to maximize to the value of the knapsack choose the case where value is maximized.

The DP can be solved using the following algorithm given in class:

Algorithm 1 Maximum objective value to the 0-1 knapsack using DP

Inputs: Item values c^{kp}, weights a^{kp} and budget b^{kp}.
$F[0,h] = 0 \qquad h \in [0, b-1]$
$F[r,0] = 0 \qquad r \in [1, n-1]$
for $r \in [1, n-1]$ **do**
    for $h \in [1, b-1]$ **do**
        if $h \leq a_r^{\text{kp}} - 1$ **then**
            $F[r,h] = F[r-1,h]$
        else
            $\max\left\{F[r-1,h],\ c_r^{\text{kp}} + F[r-1,h-1]\right\}$
        end if
    end for
end for

Algorithm to get the knapsack solution from the table F created during algorithm 1:

Algorithm 2 Getting the solution to the 0-1 knapsack using the DP table

Inputs: $F, a^{\text{kp}}, b^{\text{kp}}$
Initialize solution x.
$h \leftarrow b^{\text{kp}} - 1$
for $r \in [n-1, 0]$ **do**
    if $F[r,h] = F[r-1,h]$ **then**
        $x_r = 0$
    else
        $x_r = 1$
        $h = h - \lfloor a_r^{\text{kp}} \rfloor$
    end if
end for

4.2 Separation Problem

Now the separation problem for the cover inequalities is, given nonintegral x^* with $0 \leq x_j^* \leq 1$ for all $j \in N$ find out whether x^* satisfies all the cover inequalities. Equation (13) can be rewritten as

$$\begin{aligned} & 1 \leq |\mathcal{C}| - \sum_{j \in \mathcal{C}} x_j \\ \implies & 1 \leq \sum_{j \in \mathcal{C}} (1 - x_j) \end{aligned} \tag{15}$$

The separation problem can be written as a decision problem: Does there exist a set $\mathcal{C} \subseteq N$ with $\sum\limits_{j \in \mathcal{C}} w(x_j) > b$ for which $\sum_{j \in \mathcal{C}} (1 - x_j^*) < 1$? Here, the cover $\mathcal{C}$ would contain all the indeces of the armor piece variables where an index indicate an armor piece variable in the flattened list of all armor pieces. This decision problem can be written as an $0-1$ integer program.

$$\zeta = \min \quad \sum_{j \in N} (1 - x_j^*) z_j \tag{16}$$

$$\text{s.t.} \quad \sum_{j \in N} w(x_j) z_j \geq b + 1 \tag{17}$$

$$z_j \in \{0, 1\} \qquad \forall j \in N \tag{18}$$

If $\zeta \geq 1$, x^* satisfies all the cover inequalities (Wolsey Theorem 9.2). Here, $z_j = 1$ if $j \in \mathcal{C}$, otherwise 0. Again, x_j refers to the jth armor piece in the flattened list of all armor pieces.

So, this knapsack problem can be solved to get the cover inequality for which $x*$ is infeasible. Otherwise if x^* is feasible then cover inequality will not be identified.

Here, the values, weights, budget for the knapsack instance would be $c^{\text{kp}} = \big\{(1 - x_j^*)\ \forall j \in N\big\}$, $a^{\text{kp}} = \{w(x_j)\ \forall j \in N\}$, $b^{\text{kp}} = b+1$. Using the DP described in the previous subsection this problem can be solved.

4.3 Lifting Proccedure

After getting a cover by solving the separation problem with x^* obtained from the LP relaxation of the original MCKP the cover inequalities will be strengthened using a lifting proccedure.

1. Generate different permutations of ordering of $N \setminus \mathcal{C}$
2. Let, $j_1, \cdots, j_r$ be an ordering of $N \setminus \mathcal{C}$.
3. Set $t = 1$

4. The valid inequality $\sum_{i=1}^{t-1} \alpha_{j_i} x_{j_i} + \sum_{j \in \mathcal{C}} x_j \leq |\mathcal{C}| - 1$ has been obtained so far. To calculate largest α_{j_t} for which $\alpha_{j_t} x_{j_t} + \sum_{i=1}^{t-1} \alpha_{j_i} x_{j_i} + \sum_{j \in \mathcal{C}} x_j \leq |\mathcal{C}| - 1$ is valid solve the knapsack problem:

$$\zeta_t = \max \sum_{i=1}^{t-1} \alpha_{j_t} x_{j_t} + \sum_{j \in \mathcal{C}} x_j \tag{19}$$

$$\text{s.t.} \sum_{i \in N \setminus \{t\}} w(x_{j_i}) \cdot x_{j_i} \leq b - w(x_{j_t}) \tag{20}$$

$$x_i \in \{0, 1\} \qquad \forall i \in N \setminus t \tag{21}$$

5. Set $\alpha_{j_t} = |\mathcal{C}| - 1 - \zeta_t$
6. Increment t by 1 until $t = r$ and go to step 4.
7. Choose a different ordering $j_1, \cdots, j_r$ of $N \setminus \mathcal{C}$ and go to step 3 if the specified maximum number of orderings to test has not been exceeded yet. Otherwise terminate.

Since, a lot of differnt permutations of the orderings is possible only a specified number of ordering will be tested. This means that all possible cover inequalities may not be identified which may result in not finding the optimal solution to the MCKP.

4.4 Solution Algorithm

Using the separation problem and the lifting proccedure described in the previous section the solution algorithm is given below.

1. Get the LP relaxation of the MCKP.
2. Solve the LP relaxation of MCKP to get x^*.
3. Using the obtained x^*, solve the separation problem to get a cover inequality.
4. Terminate if the objective value from the separation problem, $\zeta \geq 1$. Because that implies all the cover inequalities are satisfied for x^*.
5. Strengthen the cover inequality using the lifting proccedure.
6. Add all the inequalities obtained so far to the LP relaxation of MCKP and get new x^*.
7. Go to step 3.

5 Numerical Results

The solution algorithms were implemented in Python. The solution algorithms were compared with results from the Gurobi solver.

For a maximum budget of $b = 30$ the following results were obtained:

Table 1: Performance from different solution methods for maximum budget, $b = 30$.

	MCKP with Cover Inequalities (DP)	MCKP with Cover Inequalities (Gurobi)	MCKP (Gurobi)
Formulation	Linear Program (LP)	Linear Program (LP)	Integer Program (IP)
Time (seconds)	2.8814997673	22.9754681587	0.020500421524
Objective Value	0.176072650791	0.176072650791	0.180687443257
Total Weight	27.4	27.4	29.8
Inequalities found	11	8	–

MCKP with cover inequalities was implemented twice. In one of the implementations the separation problema and the lifting proccedure was done using the Dynamic Programming (DP) knapsack algorithm. In the other implementation the cover and lifting proccedure was done by solving IP's described in the previous sections directly by Gurobi. Gurobi took a really long time to solve the separation problem and lifting when IP's were solved. The found solutions using the two different implementations did not change. The number of inequalities however changed. The code for both implementation will be attached with this report.

According to table 1, the time taken to find the knapsack cover inequalities, strengthening them using DP and then solving the LP again after adding the inequalities took a lot more time than directly solving the IP with Gurobi. The lifting proccedure being time consuming made the MCKP with cover Inequalities much slower than MCKP with Gurobi. When the cover and lifting proccedure was done using Gurobi only the run time increased even more.

Solving the MCKP LP with cover inequalities (both implementations) resulted in a lower objective value than solving the MCKP with Gurobi's IP solver. The sum of weight of the armor pieces was also closer to the maximum weight budget for MCKP with Gurobi than MCKP with cover inequalities. Because the described lifting proccedure can strengthen cover inequalities in different orderings and all possible orderings were not considered because of very large number of orderings, more stronger inequalities that exist were not added to the MCKP LP. Because of that MCKP LP at the last iteration found x^* which was not the optimal solution and a better x^* would be found if more stronger cover inequalities were added.

A minimum number of iteration is used until which even if $\zeta \geq 1$ from the

separation problem the program would not terminate. Instead x^* at that point would be slightly modified randomly and another attempt at finding a cover will be made. This is done so that more covers can be found with different x^* by solving the separation problem more times.

The cover inequalities eq. (22)–eq. (34) were added to the LP relaxation of the MCKP when DP was used to solve the separation problem.

$$3x_{338} + 4x_{319} + x_{304} + 5x_{232} + 2x_{224} + x_{38} + x_{30} + x_{28} + x_{26} + x_{23} + x_{20} \leq 7 \tag{22}$$

$$3x_{338} + 4x_{319} + x_{304} + 5x_{232} + 2x_{224} + x_{38} + x_{30} + x_{28} + x_{26} + x_{23} + x_{20} + x_{15} \leq 7 \tag{23}$$

$$3x_{338} + 4x_{319} + x_{304} + 5x_{232} + 2x_{224} + x_{38} + x_{30} + x_{28} + x_{26} + x_{23} + x_{20} + x_{15} + x_{12} \leq 7 \tag{24}$$

$$3x_{338} + 4x_{319} + x_{304} + 5x_{232} + 2x_{224} \leq 7 \tag{25}$$

$$3x_{338} + 4x_{319} + x_{304} \leq 7 \tag{26}$$

$$3x_{338} \leq 7 \tag{27}$$

$$3x_{338} + 4x_{319} + x_{304} + 5x_{232} \leq 7 \tag{28}$$

$$3x_{338} + 4x_{319} \leq 7 \tag{29}$$

$$3x_{338} + 4x_{319} + x_{304} + 5x_{232} + 2x_{224} + x_{38} \leq 7 \tag{30}$$

$$3x_{338} + 4x_{319} + x_{304} + 5x_{232} + 2x_{224} + x_{38} + x_{30} \leq 7 \tag{31}$$

$$3x_{338} + 4x_{319} + x_{304} + 5x_{232} + 2x_{224} + x_{38} + x_{30} + x_{28} \leq 7 \tag{32}$$

$$3x_{338} + 4x_{319} + x_{304} + 5x_{232} + 2x_{224} + x_{38} + x_{30} + x_{28} + x_{26} \leq 7 \tag{33}$$

$$3x_{338} + 4x_{319} + x_{304} + 5x_{232} + 2x_{224} + x_{38} + x_{30} + x_{28} + x_{26} + x_{23} \leq 7 \tag{34}$$

The cover inequalities eq. (35)–eq. (42) were found when Gurobi was used to solve the separation problem:

$$x_{544} + x_{519} + x_{500} + x_{414} + x_{342} \leq 3 \tag{35}$$

$$x_{544} + x_{519} + x_{500} \leq 3 \tag{36}$$

$$x_{544} \leq 3 \tag{37}$$

$$x_{544} + x_{519} + x_{500} + x_{414} \leq 3 \tag{38}$$

$$x_{544} + x_{519} \leq 3 \tag{39}$$

$$x_{544} + x_{519} + x_{500} + x_{414} + x_{342} + x_{319} \leq 3 \tag{40}$$

$$x_{544} + x_{519} + x_{500} + x_{414} + x_{342} + x_{319} + x_{232} \leq 3 \tag{41}$$

$$x_{544} + x_{519} + x_{500} + x_{414} + x_{342} + x_{319} + x_{232} + x_{55} \leq 3 \tag{42}$$

The selected armor pieces for maximum budget, $b = 30$ is given in table 2. The MCKP with Cover inequalities using DP and Gurobi both resulted in the same solution.

Table 2: Solution for maximum budget, $b = 30$.

Class	MCKP with Cover Inequalities	MCKP with Gurobi IP
Head	Greathood (weight=5.1)	Greathood (weight=5.1)
Arms	Mushroom Arms (weight=1.7)	Mushroom Arms (weight=1.7)
Chest	Radahn's Lion Armor (weight=17.5)	Radahn's Lion Armor (weight=17.5)
Legs	Mushroom Legs (weight=3.1)	Guardian Greaves (weight=5.5)

The legs armor piece was not optimal with MCKP with Cover Inequalities because a heavier legs piece could have been found like what Gurobi IP found.

6 Conclusions

In this report the problem of selecting the best armor pieces under a maximum weight budget for the Game Elden Ring is considered. First an IP formulation of the problem was given. The LP relaxation of the problem was solved repeatedly until separation problem could not identify any violated cover Inequalities. The separation problem was also discussed. The cover generated by the separation problem were then strengthened using a lifting proccedure which was described as well. After that the entire solution algorithm and the numerical results from the algorithm were discussed. Limitations of the solution algorithm were mentioned in the numerical results section. For a specified maximum budget the solutions from both methods, MCKP with covers and Gurobi's IP solver, selected the same pieces except the leg piece.

Since the problem is not the standard 0-1 Knapsack problem the lifted cover inequalities may not produce the convex hull. So integral solution may not be achieved which is a limitation of the procedure described in this report.

References

L. A. Wolsey. *Integer Programming: 2nd Edition*. J. Wiley, 2020. ISBN 9781119606475. doi: https://doi.org/10.1002/9781119606475.oth1. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119606475.oth1`.

Instructions to run code

Libraries used:

1. Python (version="3.9.4")
2. Numpy (version="1.19.5")
3. Gurobi (version="9.1.1")

The "mckp.py" file can be run directly with Python to see the results.

Data

All the data are available in “data.csv” and github.