2CC3000: Optimisation

CentraleSupélec - 2A

TD3 Optimisation

27/09/2024

CentraleSupélec

Nabil Alami
Edward Lucyszyn
Adam Younsi

Contents

Question 1

On peut dans un premier temps remplir chaque boite successivement avec les objets jusqu'à satisfaire la contrainte $\mathcal{C}_1$, c'est-à-dire, prendre un objet, le mettre dans la dernière boite considérée s'il y a de la place, ou la mettre dans une nouvelle boite dans le cas échéant. Le code MATLAB est en annexe.

On obtient $\boxed{B = 19.}$

Question 2

La contrainte sur $(\mathcal{C}_1)$:

$$\forall 1 \leq b \leq B, \quad \sum_{n=1}^{N} x_{n,b} v_n \leq C y_b.$$

La contrainte sur $(\mathcal{C}_2)$:

$$\forall 1 \leq n \leq N, \quad \sum_{b=1}^{B} x_{n,b} = 1.$$

La minimisation sur $(\mathcal{O}_1)$:

$$\underset{y \in \{0,1\}^B}{\text{minimize}} \sum_{b=1}^{B} y_b.$$

Question 3

Posons,

$$\forall b \in [\![1, B]\!], \quad X_b = \begin{pmatrix} x_{1,b} \\ x_{2,b} \\ \vdots \\ x_{n,b} \end{pmatrix}; \quad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_B \end{pmatrix}; \quad V = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{pmatrix}.$$

Puis, notant 0_B le vecteur colonne rempli de 0 et 1_B le vecteur colonne rempli de 1, on introduit les vecteurs:

$$\forall i \in [\![1, N]\!], Col_i = (\delta_{j,i})_{1 \leq j \leq N} \in \mathbb{R}^N; \quad \forall b \in [\![1, B]\!], Co_b = (\delta_{j,b})_{1 \leq j \leq B} \in \mathbb{R}^B.$$

Enfin, les matrices de notre problème sont:

$$\mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_B \\ Y \end{bmatrix} \in \{0,1\}^{B(N+1)}; \quad \mathbf{c} = \begin{bmatrix} 0_{NB} \\ 1_B \end{bmatrix} = \begin{bmatrix} 0_B \\ 0_B \\ \vdots \\ 0_B \\ 1_B \end{bmatrix} \in \{0,1\}^{B(N+1)};$$

$$\mathbf{L} = \begin{bmatrix} -V^T & 0_N^T & 0_N^T & \dots & 0_N^T & C*Co_1^T \\ 0_N^T & -V^T & 0_N^T & \dots & 0_N^T & C*Co_2^T \\ 0_N^T & 0_N^T & -V^T & \dots & 0_N^T & C*Co_3^T \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ 0_N^T & 0_N^T & 0_N^T & \dots & -V^T & C*Co_B^T \\ Col_1^T & Col_1^T & Col_1^T & \dots & Col_1^T & 0_B^T \\ Col_2^T & Col_2^T & Col_2^T & \dots & Col_2^T & 0_B^T \\ Col_3^T & Col_3^T & Col_3^T & \dots & Col_3^T & 0_B^T \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ Col_N^T & Col_N^T & Col_N^T & \dots & Col_N^T & 0_B^T \\ -Col_1^T & -Col_1^T & -Col_1^T & \dots & -Col_1^T & 0_B^T \\ -Col_2^T & -Col_2^T & -Col_2^T & \dots & -Col_2^T & 0_B^T \\ -Col_3^T & -Col_3^T & -Col_3^T & \dots & -Col_3^T & 0_B^T \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ -Col_N^T & -Col_N^T & -Col_N^T & \dots & -Col_N^T & 0_B^T \end{bmatrix} \in \mathbb{R}^{2N+B,B(N+1)}; \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ -1 \\ -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix} = \begin{bmatrix} 0_B \\ 1_N \\ -1_N \end{bmatrix}.$$

Le problème devient alors:

$$\boxed{\underset{X\in\{0,1\}^{B(N+1)}}{\text{minimize}} \langle \mathbf{c}|\mathbf{X}\rangle \text{ s.t. } \mathbf{LX} \geq \mathbf{b}.}$$

On obtient : Solution optimale = 18, le code MATLAB correspondant se trouve en annexe.

Question 4

Les boîtes ont toutes la même capacité C, leur ordre est donc arbitraire et on peut considérer que les 5 premiers items sont dans les boîtes 1 et 2 sans perte de généralité. Cette contrainte s'exprime:

$$\boxed{\forall 1 \leq i \leq 5, \quad x_{i,1} + x_{i,2} \leq 1.}$$

Le code correspondant, disponible en annexe, donne la valeur : Solution optimale = 18

Question 5

La contrainte de pouvoir mettre dans une même boîte deux objets appartenant à deux groupes différents ne nous permet plus de fixer 3 boîtes quelconques. Par exemple, il se peut que les objets 6 à 10 soient répartis sur 2 boîtes b_1 et b_2, $b_1, b_2 \geq 3$ et sur une des deux premières boîtes, au lieu de les répartir sur 3 boîtes $(b_i)_{i\geq 3}$, ce qui permet de réduire le nombre total de boîtes utilisés d'un point de vue optimisation (on utiliserait alors 2 boîtes pour le Groupe 2 au lieu de 3).

L'expression, $\forall n \in [\![1, N]\!], \quad \sum_{b=1}^{B} b \times x_{n,b}$ représente l'indice de la boite utilisée par l'item d'indice n.

Nous obtenons alors les contraintes $\mathcal{C}_4$ suivante qui assurent que nous n'utiliserons au maximum que 3 boites pour les items du groupe $\mathcal{G}_2$:

$$\boxed{\forall (n,k) \in [\![6, 10]\!]^2, \quad \sum_{b=1}^{B} b \times x_{n,b} - \sum_{b=1}^{B} b \times x_{k,b} \leq 2;}$$

$$\boxed{\forall (n,k) \in [\![6, 10]\!]^2, \quad -\sum_{b=1}^{B} b \times x_{n,b} + \sum_{b=1}^{B} b \times x_{k,b} \leq 2.}$$

En effet, les indices des boites utilisées ne peuvent pas être distants de plus de 2, ce qui assure que nous utiliserons au maximum 3 boites qui se suivent (ce qu'il est possible de faire sans perdre de la qualité de notre optimisation car les boites étant identiques, on peut les permuter sans problèmes).

Question 6

Pour tester si la solution au problème est unique, nous pouvons permuter les éléments de V en respectant les groupes, et regarder si la solution obtenue est la même, modulo permutations, que celle obtenue auparavant. Cela donne $P = (\operatorname{card} \mathcal{G}_1!)(\operatorname{card} \mathcal{G}_2!)[(1 - (\operatorname{card} \mathcal{G}_1 - \operatorname{card} \mathcal{G}_2))!]$ permutations possibles (sans prendre en compte les doublons de volume).

Une autre stratégie demandant moins de calcul est de regarder la solution pour une permutation de V donnée, et de regarder l'espace vide maximale restant dans les boîtes, soit $\max_{1 \leq b \leq B} (C - \sum_{i=1}^{N} x_{i,b} v_i)$. Si cette quantité est plus grande que $\min_{1, \leq n \leq N} v_n$, alors cela signifie que un des objets peut changer de boîte (sous réserve de respecter toujours les conditions $\mathcal{G}_3$ et $\mathcal{G}_4$). Cela donnera donc une autre solution optimale.

Question 7

Notons, $\forall b \in [\![1, B]\!], R_b = C - \sum_{i=1}^{N} v_i x_{i,b}$, la place restante dans la boîte b. On veut maximiser la place restante dans la boite avec le plus de place, i.e.

$$\text{maximize} \operatorname*{argmax}_{1 \leq b \leq B} R_b = \text{maximize} \operatorname*{argmax}_{1 \leq b \leq B} C - \sum_{i=1}^{N} v_i x_{i,b}.$$

Annexe

Listing 1: Code MATLAB pour le remplissage de boîtes

```
% Volume des objets et capacité de la boîte
load VolumeItems50.txt
V=VolumeItems50;  % vecteur des volumes des objets
N=length(V);      % nombre d'objets
C = 2.7;

% Initialisation des variables
remainingItems = 1:length(V); % Indice des éléments non encore placés dans une boîte
currentBoxIndex = 0; % Index de la boîte actuelle
boxes = struct('Items', {}, 'UnusedVolume', {}, 'NumberItems', {}); % Initialisation de la structure de boîtes

% Remplissage des boîtes
while ~isempty(remainingItems)
    currentBoxIndex = currentBoxIndex + 1; % Incrémente l'index de la boîte
    currentBoxVolume = 0; % Initialise le volume actuel de la boîte
    boxes(currentBoxIndex).Items = []; % Initialisation des articles dans la boîte

    % Boucle pour remplir la boîte avec les objets disponibles
    for i = remainingItems
        if currentBoxVolume + V(i) <= C % Si l'ajout de l'objet ne dépasse pas la capacité
            currentBoxVolume = currentBoxVolume + V(i); % Ajoute le volume de l'objet à la boîte
            boxes(currentBoxIndex).Items(end+1) = i; % Ajoute l'indice de l'objet à la boîte
            remainingItems(remainingItems == i) = []; % Retire l'objet de la liste des éléments disponibles
        end
    end

    % Calcul du volume inutilisé de la boîte et le nombre d'objets dans la boîte
    boxes(currentBoxIndex).UnusedVolume = C - currentBoxVolume;
    boxes(currentBoxIndex).NumberItems = length(boxes(currentBoxIndex).Items);
end

% Affichage des résultats
disp('--- Contents of Boxes ---');
```

```
for i = 1:length(boxes)
    disp(['Box ', num2str(i), ':']);
    disp(['  Items: ', num2str(boxes(i).Items)]);
    disp(['  Number of Items: ', num2str(boxes(i).NumberItems)]);
    disp(['  Unused Volume: ', num2str(boxes(i).UnusedVolume)]);
end

% Affichage du nombre total de boîtes utilisées
disp(['Number of boxes : ', num2str(currentBoxIndex)]);
```

Question 3

```
B = 19;
v = V;

f = [zeros(N * B, 1); ones(B, 1)];

A_ineq = zeros(B, N*B + B); % Preallocate A_ineq with the correct size

for b = 1:B
    A_ineq(b, (b-1)*N+1:b*N) = v'; % Assign item volumes to each box
end

% Add the -C factor for y_b variables
for b = 1:B
    A_ineq(b, N*B+b) = -C; % Set capacity constraint for y_b
end
b_ineq = zeros(B, 1); % One entry for each box's capacity constraint
A_eq = [repmat(eye(N), 1, B), zeros(N, B)];
b_eq = ones(N, 1);

% Bounds
lb = zeros(N * B + B, 1);
ub = ones(N * B + B, 1);

intcon = 1:(N * B + B);

[X, fval, exitflag] = intlinprog(f, intcon, A_ineq, b_ineq, A_eq, b_eq, lb, ub);

if exitflag > 0
    disp('Solution optimale :');
    disp(X);
    disp(['Valeur optimale de l''objectif : ', num2str(fval)]);
else
    disp('Le problème n''a pas pu être résolu.');
end
```

question 4 On ajoute cette partie du code pour prendre en compte la contrainte ;

Listing 2: Votre code MATLAB

```
% Création du vecteur v
v = [-1, zeros(1, 49), -1];

% Nombre de lignes et de colonnes de la matrice K
nb_lignes = 5;
nb_colonnes = 51 * 19;

% Initialisation de la matrice K avec des zéros
K = zeros(nb_lignes, nb_colonnes);

% Création des lignes de la matrice K
for i = 1:nb_lignes
    K(i, (i-1)*length(v) + 1 : i*length(v)) = v;
end

A_ineq = [A_ineq ; K];
b_ineq = cat(1, b_ineq, -ones(5, 1));
% Nombre de groupes
nGroups = 5;
\textbf{question 6}
% Nombre de lignes par groupe
nRowsPerGroup = 4;

% Nombre de colonnes par groupe
nColsPerGroup = 19;

% Taille de la matrice
nCols = nGroups * nColsPerGroup + 1;
nRows = nGroups * nRowsPerGroup;

% Création de la matrice
M = zeros(nRows, nCols);

% Remplissage des colonnes
for i = 1:nGroups
    % Indice du groupe positif
    group_pos = i;

    % Indice des lignes à remplir pour le groupe positif
    rows_pos = (i-1)*nRowsPerGroup + 1 : i*nRowsPerGroup;

    % Colonnes du groupe positif
    cols_pos = (group_pos-1)*nColsPerGroup + 1 : group_pos*
        nColsPerGroup;

    % Valeurs du groupe positif
```

```
46      values_pos = repmat((1:nColsPerGroup)', nRowsPerGroup, 1); %
           Repeat the array to make it the correct size
47
48      % Remplissage des colonnes du groupe positif
49      M(rows_pos, cols_pos) = values_pos;
50
51      % Remplissage des colonnes des autres groupes
52      for j = 1:nGroups
53          if j ~= group_pos
54              % Indice des lignes à remplir pour le groupe négatif
55              rows_neg = (j-1)*nRowsPerGroup + 1 : j*nRowsPerGroup;
56
57              % Colonnes du groupe négatif
58              cols_neg = (j-1)*nColsPerGroup + 1 : j*nColsPerGroup;
59
60              % Valeurs du groupe négatif
61              values_neg = zeros(nRowsPerGroup, nColsPerGroup);
62              if mod(j-i, nGroups) == 1
63                  values_neg = -ones(nRowsPerGroup, nColsPerGroup);
64              end
65
66              % Remplissage des colonnes du groupe négatif
67              M(rows_neg, cols_neg) = values_neg;
68          end
69      end
70  end
71
72  % Remplissage de la dernière colonne
73  M(:, end) = 2;
74
75  % Affichage de la matrice
76  disp(M);
```