



USER SCHEDULING IN 5G

INF421 PI

10 février 2024

Yingwei TANG Yuming ZHANG



TABLE DES MATIÈRES

1	Introduction et Formulation du problème	3
1.1	Introduction du contexte	3
1.2	Formulation du problème	3
1.3	Question 1 - ILP Formulation	3
2	Preprocessing	4
2.1	Question2 : Examen de base	4
2.2	Question3 : IP-dominated triplets removing	4
2.3	Question4 : LP-dominated triplets removing	5
2.4	Question5 : Show results of Q2, Q3 and Q4	6
3	Linear program and greedy algorithm	7
3.1	Question6 : Greedy algorithm	7
3.2	Question7 : Comparison between Greedy Algo and LP Solver	8
4	Dynamic programming and branch-and-bound	9
4.1	Dynamic programming	9
4.2	Dynamic programming with power budget	9
4.3	Dynamic programming with rate budget	12
4.4	Branch-and-bound	14
4.5	Résultat	16

1

INTRODUCTION ET FORMULATION DU PROBLÈME

1.1 INTRODUCTION DU CONTEXTE

Dans la 5G, une antenne transmet des paquets de données aux utilisateurs par l'intermédiaire d'un support sans fil, qui est divisé en un ensemble de chaînes de fréquence. Plus la puissance dédiée à un utilisateur est élevée, plus le débit de données qu'il peut atteindre est important. Un programmeur de paquets sans fil est donc chargé d'attribuer des chaînes aux utilisateurs et de répartir le budget total de puissance entre les canaux disponibles de l'antenne. L'objectif de ce projet est de concevoir des programmeurs de paquets optimaux dans ce contexte.

1.2 FORMULATION DU PROBLÈME

Considérons un système avec un ensemble de K clients à servir par un ensemble de N chaînes. Chaque chaîne doit être utilisée pour servir un seul client et ne peut être laissée sans pour des raisons d'efficacité; un client peut être servi par plusieurs chaînes et peut ne pas être servi.

Le programmeur donne une puissance d'émission $p_{n,k}$ pour servir le client k par la chaîne n . Nous définissons toutes les puissances des chaînes pour tous les clients. Si un client n'est pas servi par une chaîne, la puissance correspondante est 0. Le débit de données acquis par le client k sur la chaîne n est $r_{k,n} = u_{k,n}(p_{k,n})$ où la fonction u est la fonction d'utilité de débit de k sur n . Toutes les fonctions de ce genre sont des fonctions en escalier croissantes qui ont une finité de valeurs non négatives, on fixe le nombre de valeurs M pour tout n et k . La notation est la suivante :

$$u_{k,n}(p_{k,n}) = \begin{cases} 0 & p_{k,n} < p_{k,1,n} \\ r_{k,1,n} & p_{k,1,n} \leq p_{k,n} < p_{k,2,n} \\ \dots & \dots \\ r_{k,M,n} & p_{k,M,n} < p_{k,n} \end{cases} \quad (1)$$

Le but de ce problème est de répartir le budget de puissance aux chaînes et aux clients pour maximiser le débit de données total. Il est à noter qu'une chaîne ne peut pas servir plusieurs clients, le budget et les valeurs de r, p sont tous des entiers.

Ce problème peut être vu comme un Integer Linear Program (ILP) problème. En bref, le projet va chercher progressivement sur des traitements préliminaires des instances de problème, les solutions utilisant divers algorithmes et optimiser les solutions.

1.3 QUESTION 1 - ILP FORMULATION

La variable binaire $x_{k,m,n}$ représente le fait que le client k est servi par la chaîne n au débit $r_{k,m,n}$ ou pas. Les restrictions et les variables sont des entiers, donc c'est une formulation de ILP. On utilise le produit de la variable binaire et la valeur correspondante de la fonction d'utilité de débit pour calculer le vrai débit. Voici la formulation :

$$\max_{x_{k,m,n}} \sum_{k,m,n} x_{k,m,n} \cdot u_{k,n}(p_{k,n}) \quad (2)$$

$$\text{s.t. } k \in [1..K], m \in [1..M], n \in [1..N], x_{k,m,n} \in \{1, 0\}, p_{k,n} \in \mathbb{R}^+ \quad (3)$$

$$\forall n \in [1..N], \sum_{m,k} x_{k,m,n} = 1 \quad (4)$$

$$\sum_{k,n} p_{k,n} \leq p, p \in \mathbb{N} \quad (5)$$

$$u_{k,n}(p_{k,n}) = \begin{cases} 0 & p_{k,n} < p_{k,1,n} \\ r_{k,1,n} & p_{k,1,n} \leq p_{k,n} < p_{k,2,n} \\ \dots & \dots \\ r_{k,M,n} & p_{k,M,n} < p_{k,n} \end{cases} \quad (6)$$

(3) représente le cadre des paramètres, (4) représente la restriction qu'une chaîne doit et peut servir exactement un client, (5) représente la restriction du budget de puissance totale, (6) représente la fonction d'utilité de débit.

2 PREPROCESSING

2.1 QUESTION2 : EXAMEN DE BASE

Considérons la formulation formée de la Question 1, un exemple de problème n'a pas de solution quand ses paramètres contiennent des contradictions avec les restrictions.

Si la puissance la plus faible d'une chaîne est supérieure au budget total, alors cette chaîne ne peut jamais servir de clients. Cela contredit la restriction (4), donc il n'y a pas de solution acceptable.

On implémente le preprocessing avec code Python par des fonctions readData et prePro. readData d'abord lit les données, puis prePro faire des comparaisons et obtenir une valeur booléenne pour indiquer si les données forment un problème acceptable.

2.2 QUESTION3 : IP-DOMINATED TRIPLETS REMOVING

Il convient de mentionner ici que nous conserve les données sous la forme d'une classe Channel définie par nous, qui représente une chaîne. Elle a ses attributs et une liste users de User, qui est une autre classe qui représente une ligne de la matrice, c'est-à-dire des données d'un client de cette chaîne. Dans une User, elle a ses attributs et une liste powers de Power, qui représente un point d'une ligne de la matrice, c'est-à-dire une valeur d'ordre m d'un client k de cette chaîne. Une Power contient ses valeurs p,r et son index dans la matrice N*K*M.

Selon le lemme 1, si les ordres de puissance et de débit de deux triplets sont inverses, alors le triplet dont la plus haute puissance doit être exclu de la solution. Donc l'algorithme doit chercher des paires d'inversion et exclure le point avec une puissance haute.

Input :

Two 2-dimension sets, one contains all p values of the channel, another contains all r values respectively. Their size is K*M.

Preparation :

We set a empty 1-dimension array, put all points of the channel in it and sort this array by value p.

Idea :

We compare all points in order. To avoid neglecting any inverse pair, we should store the first point, compare the next points with it and update it when we meet a point whose rate is higher, and so on.

Pseudo-code :

rem is the stock which stores the points to remove, res the one which stores acceptable points. A the 1-dimension array of points of this channel.

res[0] = A[0]

For i = 1 to KM-1 do

Add A[i] to res ;

If res[-1].r ≤ res[-2].r then res pops the point and remove it from origin matrix, put this point in rem

Output : return rem and res

Correctness : "The last point stored in res has always the highest rate" and "Values of p are sorted increasingly" are the loop invariants.

Complexity : We use the built-in sorting methods of Python to sort the K*M array by p value, the complexity is $O(KM \log(KM))$ for one channel, and the part of iteration is linear, its complexity is $O(KM)$, so the total complexity of all channels is $T(N, K, M) = N \cdot (O(KM \log(KM)) + O(KM)) = O(NKM \log(KM))$.

2.3 QUESTION4 : LP-DOMINATED TRIPLETS REMOVING

Le lemme 2 indique qu'on veut avoir une "fonction" concave à la fin du traitement des données dans le but d'enlever des intervalles avec un ratio r/p faible qui sont des choix non optimaux.

Comme la question3, nous devons d'abord ordonner des points par l'ordre ascendant de p, puis utiliser encore l'idée de garder-renouveler pour itérer les données d'une chaîne. Par exemple, on veut garder les points p1 et p4 mais enlever les points p2 et p3.

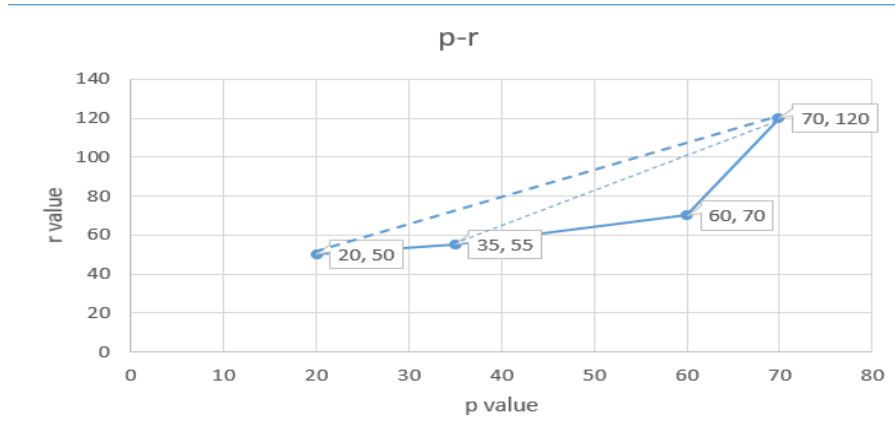


FIGURE 1 – Exemple

Input & Preparation : Two 2-dimension sets, one contains all p values of the channel, another contains all r values respectively. Their size is K*M. We set a empty 1-dimension array, put all points of the channel in it and sort this array by value p.

Idea : As every point will be checked, put in and eventually pop out once, the data structure stack is suitable for the points stored. We keep the two first points in a stack S : S[0], S[1]. From the third one x, we compare the ratio of difference written in lemma 2 of this point and the two last points (S[-1], S[-2]) in S. If the last point

$S[1]$ in S should be removed, then remove it from S and add the new point x in S ; if not, then simply add the new point x in S (From now on the last two points of S is changed).

Pseudo-code :

rem is the stock which stores the points to remove, S the stack where we can push and pop elements which stores acceptable points. A the 1-dimension array of points of this channel.

Set $d = (r_1 - r_0)/(p_1 - p_0)$, push $A[0]$ and $A[1]$ into S .

Forall $i \in [2..KM-1]$ do

 Forall $s \in S$ (in reverse order) do

 If $(r_i - r_s)/(p_i - p_s) \geq d$, then pop s from S and add it into rem.

 Push r_i into S

Output : return rem and S

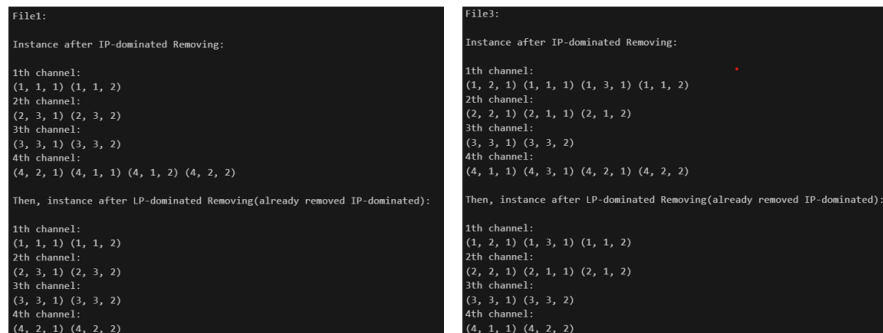
Correctness : "The points added into S are compared to all points in S and keep their validity for lemma2 if we keep them in S .", "One point will be checked only if the points behind it in S are all checked and popped" are the loop invariants.

Complexity : We use the built-in sorting methods of Python to sort the $K \times M$ array by p value, the complexity is $O(KM \log(KM))$ for one channel. The comparisons in the iteration is of linear time complexity, because each points can only be checked in polynomial time. So the total complexity of all channels is $T(N, K, M) = N \cdot (O(KM \log(KM)) + O(KM)) = O(NKM \log(KM))$.

Remarque : Dans notre code de Python, on utilise la liste de Python qui a naturellement les fonctions de la structure STAC ; Pour simplifier la marche du code pour les questions suivantes, il y a IP-dominated Removing fonction en tête de la fonction pour LP-dominated Removing fonction, donc l'ordre est directement ascendant par valeur p et les points inacceptables de question 3 sont déjà enlevés.

2.4 QUESTION5 : SHOW RESULTS OF Q2, Q3 AND Q4

La fonction "qfive" dans le fichier "Presentation.py" imprime les résultats des traitements Preprocessing, IP-dominated Removing et LP-dominated Removing pour les cinq fichiers de test. Voici les résultats (parties pour testfile4 et testfile5) :

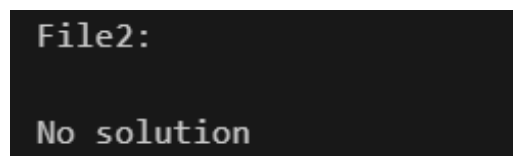


```

File1:
Instance after IP-dominated Removing:
1th channel:
(1, 1, 1) (1, 1, 2)
2th channel:
(2, 3, 1) (2, 3, 2)
3th channel:
(3, 3, 1) (3, 3, 2)
4th channel:
(4, 2, 1) (4, 1, 1) (4, 1, 2) (4, 2, 2)
Then, instance after LP-dominated Removing(already removed IP-dominated):
1th channel:
(1, 1, 1) (1, 1, 2)
2th channel:
(2, 3, 1) (2, 3, 2)
3th channel:
(3, 3, 1) (3, 3, 2)
4th channel:
(4, 2, 1) (4, 2, 2)

File3:
Instance after IP-dominated Removing:
1th channel:
(1, 2, 1) (1, 1, 1) (1, 3, 1) (1, 1, 2)
2th channel:
(2, 2, 1) (2, 1, 1) (2, 1, 2)
3th channel:
(3, 3, 1) (3, 3, 2)
4th channel:
(4, 1, 1) (4, 3, 1) (4, 2, 1) (4, 2, 2)
Then, instance after LP-dominated Removing(already removed IP-dominated):
1th channel:
(1, 2, 1) (1, 3, 1) (1, 1, 2)
2th channel:
(2, 2, 1) (2, 1, 1) (2, 1, 2)
3th channel:
(3, 3, 1) (3, 3, 2)
4th channel:
(4, 1, 1) (4, 2, 2)
    
```

FIGURE 2 – Les résultats de testfile1 et testfile3



```

File2:

No solution
    
```

FIGURE 3 – Le résultat de testfile2

```

File4:

Instance after IP-dominated Removing:

1th channel:
(1, 4, 1) (1, 4, 3) (1, 11, 1) (1, 11, 2) (1, 11, 3) (1, 11, 4) (1, 22, 4) (1, 46, 6) (1, 46, 7) (1, 46, 8)
(1, 46, 9) (1, 46, 10) (1, 46, 12) (1, 18, 15) (1, 17, 14) (1, 54, 15) (1, 2, 15) (1, 33, 16)
2th channel:
(2, 17, 1) (2, 59, 2) (2, 38, 2) (2, 38, 3) (2, 38, 4) (2, 12, 5) (2, 5, 4) (2, 38, 6) (2, 38, 7) (2, 4, 8)
(2, 8, 7) (2, 4, 9) (2, 58, 9) (2, 58, 10) (2, 58, 12) (2, 58, 13) (2, 29, 13) (2, 58, 14) (2, 60, 13) (2,
53, 14) (2, 8, 15)

Then, instance after LP-dominated Removing(already removed IP-dominated):

1th channel:
(1, 4, 1) (1, 11, 1) (1, 11, 2) (1, 11, 3) (1, 11, 4) (1, 22, 4) (1, 46, 6) (1, 46, 7) (1, 46, 8) (1, 46, 9)
(1, 46, 10) (1, 46, 12) (1, 18, 15) (1, 17, 14) (1, 54, 15) (1, 2, 15) (1, 33, 16)
2th channel:
(2, 17, 1) (2, 59, 2) (2, 38, 2) (2, 38, 3) (2, 38, 4) (2, 12, 5) (2, 5, 4) (2, 38, 6) (2, 8, 7) (2, 4, 9)
(2, 58, 9) (2, 58, 10) (2, 58, 12) (2, 58, 13) (2, 29, 13) (2, 58, 14) (2, 60, 13) (2, 53, 14) (2, 8, 15)
3th channel:
(3, 41, 1) (3, 2, 1) (3, 3, 1) (3, 5, 2) (3, 13, 3) (3, 33, 4) (3, 33, 5) (3, 33, 6) (3, 3, 3) (3, 39, 6)
(3, 15, 9) (3, 39, 8) (3, 15, 10) (3, 39, 9) (3, 39, 10) (3, 58, 13) (3, 52, 12) (3, 52, 13) (3, 47, 14) (3,
47, 15) (3, 17, 14) (3, 15, 14) (3, 52, 15) (3, 32, 16) (3, 18, 16)

```

FIGURE 4 – La partie du résultat de testfile4

```

File5:

Instance after IP-dominated Removing:

1th channel:
(1, 2, 1) (1, 13, 1) (1, 7, 2) (1, 13, 2) (1, 7, 3) (1, 2, 3) (1, 6, 4)
2th channel:
(2, 3, 1) (2, 2, 1) (2, 2, 2) (2, 5, 3) (2, 5, 4) (2, 2, 3) (2, 2, 4)
3th channel:
(3, 7, 1) (3, 13, 1) (3, 7, 2) (3, 4, 1) (3, 5, 2) (3, 4, 2) (3, 14, 3) (3, 10, 3) (3, 14, 4)
4th channel:
(4, 9, 1) (4, 9, 2) (4, 1, 1) (4, 12, 2) (4, 1, 2) (4, 1, 3) (4, 3, 4) (4, 10, 4)

Then, instance after LP-dominated Removing(already removed IP-dominated):

1th channel:
(1, 2, 1) (1, 13, 2) (1, 7, 3) (1, 2, 3) (1, 6, 4)
2th channel:
(2, 3, 1) (2, 2, 1) (2, 2, 2) (2, 5, 3) (2, 5, 4) (2, 2, 3) (2, 2, 4)
3th channel:
(3, 7, 1) (3, 4, 1) (3, 5, 2) (3, 4, 2) (3, 14, 3) (3, 10, 3) (3, 14, 4)
4th channel:
(4, 9, 1) (4, 9, 2) (4, 1, 1) (4, 12, 2) (4, 1, 2) (4, 1, 3) (4, 3, 4) (4, 10, 4)

```

FIGURE 5 – La partie du résultat de testfile5

3

LINEAR PROGRAM AND GREEDY ALGORITHM

3.1 QUESTION6 : GREEDY ALGORITHM

Ici nous considérons le sens d'efficacité incrémentielle, en fait elle représente le rendement de choisir l'échelon suivant au lieu de choisir cet échelon-là face d'un reste de budget. Selon la conception de "Greedy Algorithm", on cherche à chaque pas le choix avec le meilleure efficacité incrémentielle afin d'assurer le débit total maximal. Le point important est que toutes les chaînes doivent travailler, donc on fait des comparaisons entre toutes les chaînes à chaque pas.

Dans cette question, les données sont déjà filtrées par les traitements de Question 2,3,4, donc l'ordre des données est préparé. Et le remarque que il y a au plus deux variables fractionnels (obligatoirement dans la même chaîne) veut dire que dans le cas où l'échelon i n'utilise pas tout le budget mais celui $i+1$ le dépasse est accepté en prenant deux coefficients dont la somme est 1 et la somme des produits de coefficients avec les puissances respecte exactement le budget :

$$p_{autreschaînes} + x_i \cdot p_i + x_{i+1} \cdot p_{i+1} = p_{budget}$$

$$x_i + x_{i+1} = 1$$

Input & Preparation : The input is the filtered data of a whole file. The parameters are N channels, K users and M levels of rate, and the total budget is p .

Idea : As we have already done the filter of question 3 and 4, the data of every channel has now increasing order by p and the pairs of neighbor points have decreasing (Non increasing) order by incremental efficiency e . We use a array RES of size N to keep the element chosen of each channel, another array TMP of size N to do the comparison of value e . At every step we choose the point with the highest e in TMP and put it in the corresponding position of RES and update the TMP by replace the point by the next point of the same channel. At last, we do a test for the case where we have fractional variables.

Pseudo-code :

Two empty array of size N : RES and TMP. Initiate RES with the first point of every channel, initiate TMP with the e calculated of the first two points of every channel.

While $\text{sum}(p \text{ value of RES}) < P$ do

m_k the point with $e_{m_k} = \max(\text{TMP})$

 replace RES[k-1] by m_k

 replace TMP[k-1] by e value calculated between m_k and the next point in channel k

If $\text{sum}(p \text{ value of RES}) > p$ do

 Calculate the correct relative weight of the last point and his predecessor in the same channel to make the used power equal the budget. Put these points with their weight in RES.

Output : return RES, $\text{sum}(p \text{ value of RES})$

Complexity : We do calculate and comparison $O(KM)$ times for an array of size N , so the time complexity is $O(KMN)$.

Remarque : Il convient de mentionner ici que pour baisser la complexité due aux répétitions de comparaisons dans TMP, c'est possible d'avoir une alternative similaire de la structure Max-Heap, qui accélérerait la démarche de chercher la valeur maximale dans TMP et celle d'insérer la nouvelle valeur du point suivant. La complexité en utilisant une telle structure serait peut-être $O(KM \log(N))$.

3.2 QUESTION 7 : COMPARISON BETWEEN GREEDY ALGO AND LP SOLVER

Nous utilisons le LP-Solver GLOP de OR-Tools qui est développé par Google : <https://developers.google.com/optimization>
Voici les résultats des cinq fichiers de test en utilisant notre Greedy Algorithm et le LP-Solver :

Testfile	Number of Variables	Greedy Algorithm		LP-Solver	
		Rate	Power	Rate	Power
1	24	365.00	78.00	365.00	78.00
2	/	/	/	/	/
3	24	372.15	100.00	372.15	100.00
4	614400	9642.10	16000.00	9870.32	16000.00
5	2400	1559.00	1000.00	1637.00	1000.00

FIGURE 6 – La comparaison

On peut voir que dans le cas où il permet seulement deux valeurs fractionnelles dans une seule chaîne, les résultats de notre Greedy Algorithm sont assez proche de la solution optimale donnée par le LP-Solver. Surtout pour les données de petites tailles, les résultats sont pareils.

Ensuite, on utilise le module time de Python pour comparer les temps de marche des deux façons sur les fichiers de test 4 et 5 :

Time(second)	Greedy Algorithm	LP-Solver
Testfile 4	1.242890	5.531292
Testfile 5	0.005443	0.012601

FIGURE 7 – Le temps de marche

D'abord, on peut voir que les relations entre les temps et les tailles de fichier sont en accord (614400/2400 et 1.2429/0.0054). Puis, notre Greedy Algorithm a un temps de marche presque 4-5 fois moins long que le LP-Solver sur le fichier 4 qui est de la plus grande taille, et une moitié de temps sur le fichier 5 qui est d'une taille plus petite. On peut conclure que le Greedy Algorithm possède un avantage sur la complexité de temps en fonction de la taille des données.

4

DYNAMIC PROGRAMMING AND BRANCH-AND-BOUND

4.1 DYNAMIC PROGRAMMING

Comme indiqué précédemment, nos solutions de relaxation ne sont pas directement applicables aux problèmes de IP car un canal pourrait être attribué à deux utilisateurs. De même, les algorithmes greedy ne conviennent pas aux ILP, comme nous l'avons rencontré avec le problème du sac à dos 0-1. Dans cette étude, nous explorons une approche de résolution directe des ILP à l'aide de la programmation dynamique. Bien que l'utilisation d'une approche récursive soit pratique du point de vue de l'implémentation, elle présente des problèmes de complexité dans l'espace, comme nous le verrons plus tard dans notre analyse.

4.2 DYNAMIC PROGRAMMING WITH POWER BUDGET

$$\begin{aligned}
 DP(i, p) &= \max_{\substack{1 \leq j \leq K \\ 1 \leq m \leq M \\ p_{j,m} \leq p}} \{ \text{rate}_j + DP(i+1, p - p_{j,m}) \}, & \text{ si } 0 \leq i \leq n \\
 DP(i, p) &= 0, & \text{ si } i > n
 \end{aligned} \tag{2}$$

Ceci représente l'équation de récurrence pour l'algorithme de programmation dynamique avec le budget défini par la puissance. $DP(i, p)$ représente le meilleur taux de tous les canaux de i à n avec un budget de puissance p . Le principe fondamental est que toute sous-partie d'une solution est également optimale, ce qui signifie que le taux cible est optimal sous la contrainte du budget donné. Cette structure de sous-optimisation est utilisée pour formuler l'équation de récurrence. En termes d'implémentation, nous utilisons une approche récursive, où une fonction principale appelle une fonction récursive pour calculer les résultats. Voici le pseudo-code correspondant :

Algorithm 1 Dp_by_power**Function** Dp_by_power(N , $target$, $*channels$) :

```

1   $result \leftarrow \{\}$  // Initialiser un dictionnaire de résultats
2  Appeler Dp_by_power_rec(0,  $result$ ,  $target$ ,  $*channels$ ) // Appelez une autre fonction pour la
   récurssion
3   $choices \leftarrow []$  // Initialiser une liste de choix des niveau
4   $i \leftarrow 1$  // Initialiser l'indice à 1 correspondant au premier canal
5   $used\_power \leftarrow 0$ 
   // Compteur de puissances utilisées
6  while  $target > 0$  do // Reconstruire le chemin par des infomations stockées
   Ajouter  $result[(i, target)][\text{"choice"}]$  à la liste de choix
    $used\_power \leftarrow used\_power + result[(i, target)][\text{"used"}]$ 
    $target \leftarrow target - result[(i, target)][\text{"used"}]$  // Itérer à l'inverse
7   $i \leftarrow i + 1$ 

```

Algorithm 2 Dp_by_power_rec**Function** Dp_by_power_rec(n , $result$, p , $*channels$) :

```

8  if  $(n+1, p)$  est une clé de  $result$  then // Si le résultat est stocké, return directement
    $\text{return } result[(n+1, p)][\text{"rate"}]$ 
9   $max \leftarrow 0$  // Initialisation de la rate maximale
10  $k \leftarrow 0$   $m \leftarrow 0$   $used \leftarrow 0$  // Initialisation de la puissance utilisée par chaque choix
11 if  $n \geq \text{le nombre des channels}$  then
    $\text{return } 0$  // Retourner 0 si n est plus grand que la longueur des canaux
12 for  $user$  dans  $channels[n].users$  do
13   for  $power$  dans  $user.powers$  do // Itérer les choix pour tout K et M
14     if  $power.p > p$  then
15        $\text{continue}$  // Passer à la puissance suivante si elle dépasse le budget
16      $temp \leftarrow power.r + Dp\_by\_power\_rec(n+1, result, p-power.p, *channels)$  // récursion avec
       l'équation
17     if  $temp > max$  then // dans le cas du merilleur choix
18        $max \leftarrow temp$  // Mettre à jour la valeur maximale
19        $k \leftarrow user.index[1]$  // Mettre à jour l'index utilisateur
20        $m \leftarrow power.index[2]$  // Mettre à jour l'index de puissance
21        $used \leftarrow power.p$  // Mettre à jour la puissance utilisée
22    $result[(n+1, p)] \leftarrow \{\text{"choice"} : (n+1, k+1, m+1), \text{"rate"} : max, \text{"used"} : used\}$  // Sauvegarder le
       résultat
23 return  $result[(n+1, p)][\text{"rate"}]$  // Retourner la valeur maximale

```

COMPLEXITÉ DE L'ALGORITHME

Pour prouver la complexité de l'algorithme, nous devons analyser le nombre d'opérations effectuées en fonction de la taille de l'entrée. Dans ce cas, la taille de l'entrée est déterminée par les paramètres N , target et le nombre de canaux.

La complexité de l'algorithme dépend principalement de la fonction récursive `Dp_by_power_rec`. Considérons d'abord cette fonction :

- Dans la fonction `Dp_by_power_rec`, nous itérons sur chaque utilisateur et chaque puissance disponible dans chaque canal. Cette itération a une complexité de $O(K \times M)$, où K est le nombre maximal d'utilisateurs dans un canal et M est le nombre maximal de puissances disponibles pour un utilisateur.
- À chaque itération, nous appelons récursivement la fonction `Dp_by_power_rec` pour $n + 1$ canaux et $p - p_{j,m}$ comme nouveau budget. Comme nous itérons sur N canaux, cette récursion se produit N fois.
- La fonction récursive est appelée pour chaque combinaison possible de canal, utilisateur et puissance, ce qui peut potentiellement conduire à un grand nombre d'appels récursifs. Cependant, grâce à la mémoire des résultats déjà calculés, de nombreux appels récursifs sont évités, ce qui réduit la complexité effective.

En prenant tout cela en considération, la complexité de l'algorithme est souvent dominée par le nombre d'appels récursifs nécessaires pour calculer les valeurs de $DP(i, p)$ pour toutes les combinaisons de i et p . Dans le pire des cas, chaque appel récursif nécessite $O(K \times M)$ opérations. Étant donné qu'il y a N canaux et que p peut prendre jusqu'à target valeurs différentes, la complexité totale de l'algorithme est souvent de l'ordre de $O(N \times p \times K \times M)$.

Pour analyser la complexité d'espace de l'algorithme, nous devons examiner l'espace utilisé par les structures de données et les variables de l'algorithme.

Espace utilisé par les structures de données : L'algorithme utilise principalement un dictionnaire (*result*) pour stocker les résultats calculés. La taille de ce dictionnaire dépend du nombre de sous-problèmes résolus, c'est-à-dire du nombre de combinaisons possibles de i et p . Dans le pire des cas, chaque combinaison de i et p est unique, ce qui signifie que le dictionnaire peut contenir jusqu'à $n \times \text{target}$ éléments.

Espace utilisé par les variables : En plus du dictionnaire, l'algorithme utilise quelques variables supplémentaires pour le stockage temporaire des choix, des indices et des valeurs maximales. Ces variables ont une taille fixe et ne dépendent pas de la taille de l'entrée.

Complexité d'espace totale : En combinant l'espace utilisé par les structures de données et les variables, la complexité d'espace totale de l'algorithme est $O(n \times \text{target})$, où n est le nombre de canaux et target est la valeur cible de puissance.

PREUVE DE LA CORRECTION DE L'ALGORITHME

Pour prouver la correction de l'algorithme, nous devons démontrer deux choses : d'abord, que l'algorithme termine et ne boucle pas indéfiniment, et ensuite, que l'algorithme produit le résultat correct.

• TERMINAISON DE L'ALGORITHME

L'algorithme termine lorsque toutes les valeurs de $DP(i, p)$ sont calculées pour chaque combinaison de i et p . Comme il y a un nombre fini de canaux N , un nombre fini de valeurs possibles pour target , et chaque appel récursif réduit p d'une quantité finie à chaque fois, la récursion finit par atteindre une base où la valeur de p est nulle ou où tous les canaux ont été parcourus. Par conséquent, l'algorithme termine toujours.

• CORRECTION DE L'ALGORITHME

Pour prouver que l'algorithme produit le résultat correct, nous devons démontrer que la valeur de $DP(i, p)$ calculée est bien la valeur maximale pour le sous-problème donné. Cela peut être démontré par induction sur i .

- **Cas de base :** Lorsque $i > n$, la valeur de $DP(i, p)$ est toujours nulle, ce qui est correct car aucun canal supplémentaire ne peut être utilisé.
- **Hypothèse d'induction :** Supposons que l'algorithme produise le bon résultat pour tous les sous-problèmes de $i + 1$ à n .
- **Cas inductif :** Pour i entre 0 et n , l'algorithme choisit la meilleure combinaison possible de canal, utilisateur et puissance disponible pour maximiser le taux. En récurant sur les canaux suivants avec le nouveau budget $p - p_{j,m}$, l'algorithme s'assure qu'il considère toutes les possibilités pour atteindre le taux maximal. Ainsi, la valeur de $DP(i, p)$ calculée par l'algorithme est en effet la valeur maximale pour le sous-problème donné.

En combinant la terminaison de l'algorithme avec sa correction, nous concluons que l'algorithme produit le bon résultat dans tous les cas.

4.3 DYNAMIC PROGRAMMING WITH RATE BUDGET

$$DP(i, r) = \min_{\substack{1 \leq j \leq K \\ 1 \leq m \leq M \\ r_j \leq r}} \{p_{j,m} + DP(i + 1, r - r_j)\}, \quad \text{si } 0 \leq i \leq n$$

$$DP(i, r) = 0, \quad \text{si } i > n$$

L'algorithme de programmation dynamique alternative basé sur une puissance minimale donnée un taux supérieur fixé. Il possède également une structure de sous-problèmes optimaux, comme nous l'avons démontré précédemment. Cependant, il est unique en ce sens qu'il nécessite une borne supérieure pour exécuter l'algorithme. Nous devons avoir une idée approximative du résultat à l'avance, ce qui nous rappelle l'algorithme greedy. En tant que méthode de résolution de programme linéaire, l'algorithme greedy donne toujours une borne supérieure U comme valeur initiale du taux.

ANALYSE DE LA COMPLEXITÉ

Comme ce qui précède, la complexité temporelle totale de l'algorithme est donc de l'ordre de $O(N \times K \times M \times U)$.

La complexité spatiale totale de l'algorithme est donc de l'ordre de $O(N \times U)$.

Algorithm 3 Dp_by_rate

Function Dp_by_rate(*target*, **channels*) :

```

    result ← {} // Initialiser un dictionnaire de résultats
24 Appeler Dp_by_rate_rec(0, result, target, *channels) // Appelez une autre fonction pour la
    récurSION
25 choices ← [] // Initialiser une liste de choix des niveau
26 i ← 1 // Initialiser l'indice à 1 correspondant au premier canal
27 Afficher "La dépense minimale est ", result[(i, target)]["power"] // Afficher la puissance minimale
28 while target > 0 do
    // Reconstruire les choix par des informations stockées
29     Ajouter result[(i, target)]["choice"] à la liste de choix i ← i + 1
    target ← target - result[(i, target)]["used"] // Itérer à l'inverse
30 Afficher "Les choix effectués sont ", choices

```

Algorithm 4 Dp_by_rate_rec**Function** Dp_by_rate_rec(n , $result$, r , $*channels$) :

```

  if ( $n+1$ ,  $r$ ) est une clé de  $result$  then
    return  $result[(n+1, r)]["power"]$  // Si le résultat est stocké, return directement
31
   $min \leftarrow \infty$  // Initialisation de la puissance minimale
32  $k \leftarrow 0$   $m \leftarrow 0$   $used \leftarrow 0$  // Initialisation de la puissance utilisée par chaque choix
33 if  $n \geq$  le nombre des channels then
34   return 0 // Retourner 0 si n est plus grand que la longueur des canaux
35 for user dans  $channels[n].users$  do
36   for power dans  $user.powers$  do
    // Itérer les choix pour tout K et M
37     if  $power.r > r$  then
38       continue // Passer à la puissance suivante si elle dépasse le budget
39      $temp \leftarrow power.p + Dp\_by\_rate\_rec(n+1, result, r - power.r, *channels)$  // récursion avec
    l'équation
40     if  $temp < min$  then
    // dans le cas du meilleur choix
41        $min \leftarrow temp$  // Mettre à jour la puissance minimale
42        $k \leftarrow user.index$  // Mettre à jour l'index utilisateur
43        $m \leftarrow power.index$  // Mettre à jour l'index de puissance
44        $used \leftarrow power.r$  // Mettre à jour la puissance utilisée
45    $result[(n+1, r)] \leftarrow \{ "choice" : (n+1, k+1, m+1), "power" : min, "used" : used \}$  // Sauvegarder le
    résultat
46   return  $result[(n+1, r)]["power"]$  // Retourner la puissance minimale

```

4.4 BRANCH-AND-BOUND

La méthode de branchement et de borne consiste à diviser progressivement le problème en différents sous-problèmes, et lors du traitement de ces sous-problèmes, il est possible de faire des coupes pour réduire le nombre de problèmes à traiter. Dans cette approche, nous étudions comment transformer un problème LP en un problème ILP.

Ainsi, pour chaque sous-problème considéré, nous calculons sa solution de relaxation. Si cette solution est inférieure à la solution existante, il n'est pas nécessaire de poursuivre la considération de cette branche du sous-problème. De plus, si cette solution est suffisamment grande et est une solution entière, nous pouvons mettre à jour la meilleure solution actuelle.

En ce qui concerne le branchement, il est nécessaire de prendre en compte la possibilité de faire des coupes. Certaines branches peuvent être plus propices à une convergence rapide vers la bonne réponse. Nous avons envisagé de subdiviser les branches en limitant la puissance utilisée par chaque canal, ce qui transformerait le problème en un problème binaire. Cela devrait conduire à une convergence plus rapide. Cependant, la mise en œuvre de cette approche est compliquée. Finalement, nous avons opté pour une approche plus naïve : examiner séquentiellement chaque canal et considérer tous ses utilisateurs. Ainsi, chaque bifurcation générera K nœuds, ce qui est évidemment une perte de puissance de calcul et nécessite une grande capacité de stockage, ce qui a conduit à une implémentation qui n'a pas donné les résultats escomptés.

CHOIX DE LA STRUCTURE DE DONNÉES

Pour choisir une structure de données, il est important de considérer le stockage des données et la stratégie de parcours. La méthode de recherche en largeur consiste à explorer tous les sous-problèmes à une distance fixe de la racine avant de passer à des profondeurs supérieures. Cela garantit la découverte de la solution optimale la plus proche de la racine. Cependant, si les solutions optimales sont souvent situées à des profondeurs plus importantes, cette approche n'est pas la plus adaptée pour la technique de branch and bound. La recherche en profondeur consomme moins de mémoire, mais il est possible de ne jamais rencontrer certaines solutions optimales si elles sont proches de la racine. Nous avons donc opté pour l'utilisation d'une file de priorité maximale afin de nous assurer d'examiner en priorité les nœuds les plus prometteurs. Pour maintenir cette file de priorité maximale, nous avons défini une classe `Node`, où sont stockées les données, la stratégie de branchement vers ce nœud, sa borne, la puissance utilisée par cette stratégie, et sa profondeur. En outre, nous avons également défini la méthode `__lt__()` pour comparer les bornes de chaque nœud. Pour l'algorithme de relaxation de la solution, nous avons choisi l'algorithme greedy étudié précédemment.

L'ALGORITHME

L'algorithme utilise une classe personnalisée `Node` pour maintenir un tas maximal et faciliter le processus de coupe. Cependant, le problème réside dans le stockage des données représentées par chaque branche. Chaque fois qu'une branche est considérée, une copie profonde des données initiales est effectuée, puis elles sont converties en fonction de la situation représentée par la branche. Cela entraîne clairement une complexité temporelle élevée. Bien que nous utilisions une liste simple de longueur n pour stocker les situations de branche, ces informations ne sont pas bien traduites en données lors de la conversion, ce qui pourrait également expliquer pourquoi notre implémentation ne parvient pas à obtenir de résultats.

COMPLEXITÉ TEMPORELLE DE L'ALGORITHME

Pour analyser la complexité temporelle de l'algorithme de branch and bound, nous examinons le coût des opérations effectuées à chaque étape de l'algorithme.

Algorithm 5 Branch and Bound**Function** `branch_and_bound(p, *channels)` :Initialiser un node avec bound -inf *Initialiser une PriorityQueue***while** *q non vide* **do**

Extraire un node

if *node.bound < current_feasible_bound* **then**

// Il n'y a plus d'autres solutions possibles

break

Data ← Copier profondement des channels

Incrémenter level

Transformer Data à la branche actuelle d'après new_branche

for *i ← 1 à K* **do**

new_branch.append(i) // Itérer les clients du canal suivant

Calculer la résultat relaxée par greedy fonction // On choisit ce client pour notre branche

Créer ce nouveau node

if *node.bound ≤ current_feasible_bound* **then**

// Si le résultat est inférieur que la borne actuelle

continue **if** *La solution est entière* **then** Mise à jour de borne, solution, puissance utilisé // On n'a plus besoin de considérer
 cette branche **continue**

q.put(node)

return *current_feasible_solution, current_feasible_bound, used_power*

- **Opérations dans la boucle principale :** La boucle principale de l'algorithme utilise une file de priorité pour maintenir les nœuds à explorer. Les opérations d'insertion et de suppression dans une file de priorité ont une complexité de $O(\log n)$, où n est le nombre de nœuds dans la file de priorité, dans notre cas c'est finalement $N \times K$. Après chaque suppression d'un nœud de la file de priorité, plusieurs opérations sont effectuées, notamment la vérification de la poursuite de la recherche, la copie de la branche, la copie profonde des données et le calcul de la relaxation du problème. Supposons que le coût de ces opérations soit $O(N \times K \times M)$, où $N \times K \times M$ est la taille des données. Ainsi, la complexité temporelle de la boucle principale est approximativement $O((\log N \times K) \times N \times K \times M)$.
- **Calcul de la relaxation du problème :** Après chaque suppression d'un nœud de la file de priorité, nous devons calculer la relaxation du problème. Dans l'algorithme donné, une méthode greedy est utilisée pour résoudre ce problème dont le coût est $O(N \times K \times M)$. Ainsi, la complexité du calcul de la relaxation du problème est approximativement $O(N^2 \times K^2 \times M)$.

En résumé, la complexité temporelle totale de l'algorithme de branch and bound peut être approximativement exprimée comme $O(N^2 \times K^2 \times M)$ qui vient des calculs de relaxation.

4.5 RÉSULTAT

DP_by_power	runtime	max_rate	used_power
test1	0.000334	365	78
test2			
test3	0.000343	350	68
test4			
test5	0.696245	1637	1000

La performance de la programmation dynamique sur test4 n'est pas satisfaisante, probablement en raison de l'utilisation de l'approche récursive, ce qui entraîne une consommation élevée de la mémoire. Une amélioration consisterait à utiliser une approche itérative, mais nous n'avons pas fourni de mise en œuvre correspondante. Il est également possible que l'algorithme IP soit trop complexe. Cependant, dans les autres fichiers, l'algorithme donne d'excellents résultats. De même, la méthode de branchement et de bornage n'a pas non plus donné de résultats.