

**Question 1.**

For  $k \in \mathcal{K}, n \in \mathcal{N}, m = 1, \dots, M$ , we define the binary variables  $x_{k,m,n} \in \{0, 1\}$  such that  $x_{k,m,n} = 1$  if and only if  $p_{k,m,n} \leq p_{k,n} < p_{k,m+1,n}$ , with  $p_{k,M+1,n}$  interpreted as  $+\infty$ . Thus, the constraint by the total power budget is

$$\sum_{\substack{k \in \mathcal{K} \\ n \in \mathcal{N} \\ m=1, \dots, M}} p_{k,m,n} x_{k,m,n} \leq p$$

the constraint that each channel serves one and only one user is

$$\sum_{\substack{k \in \mathcal{K} \\ m=1, \dots, M}} x_{k,m,n} = 1, \forall n \in \mathcal{N}$$

and the target function is

$$U := \sum_{\substack{k \in \mathcal{K} \\ n \in \mathcal{N} \\ m=1, \dots, M}} r_{k,m,n} x_{k,m,n}$$

In all, we have the ILP below

$$x_{k,m,n} \in \{0, 1\} \tag{1}$$

$$\sum_{k,m,n} p_{k,m,n} x_{k,m,n} \leq p \tag{2}$$

$$\sum_{k,m} x_{k,m,n} = 1, \forall n \in \mathcal{N} \tag{3}$$

with target function

$$U := \sum_{k,m,n} r_{k,m,n} x_{k,m,n}$$

the corresponding LP is obtained by replacing (1) with  $x_{k,m,n} \in [0, 1]$

**Question 2.**

**Proof of Lemma 1**

Suppose  $p_{k,m,n} \leq p_{k',m',n}$  and  $r_{k,m,n} \geq r_{k',m',n}$ . Given an optimal solution to the ILP, if  $x_{k',m',n} \neq 0$ , then  $x_{k',m',n} = 1$  by (1). If we replace  $x_{k,m,n}$  by 1 and  $x_{k',m',n}$  by 0, (1) and (3) are clearly satisfied, (2) is also satisfied since  $p_{k,m,n} \leq p_{k',m',n}$ , and  $U$  will not decrease since  $r_{k,m,n} \geq r_{k',m',n}$ , so we get an optimal where  $x_{k',m',n} = 0$

### Question 3.

REMOVE-IP-DOMINATED( $n$ )

- 1 sort the pairs  $(p_{k,m,n}, r_{k,m,n})$  in increasing order of  $p_{k,m,n}$  into an array A. If several pairs have the same  $p$ , leave only the one with the greatest  $r$
- 2  $cm = A[0].r$
- 3 **for**  $i = 0$  to  $A.length-1$
- 4     **if**  $A[i].r \geq cm$
- 5          $cm = A[i].r$
- 6     **else** remove  $A[i].p$  and  $A[i].r$  from the original data

sorting  $p_{k,m,n}$  takes time  $O(KM \log(KM))$ , the loop from line 3 takes time  $O(KM)$ . We will run REMOVE-IP-DOMINATED for each  $n \in \mathcal{N}$ , so in all it takes time  $O(NKM \log(KM))$  to remove IP-dominated terms.

N.B. Accounting for that  $p_{k,m,n}$  is increasing according to  $m$  for  $k, n$  fixed, we may sort quicker in line 1 and achieve a complexity of  $O(NKM \log(K))$ .

### Question 4.

Proof of **Lemma 2**

Suppose  $p_{k,m,n} \leq p_{k',m',n} \leq p_{k'',m'',n}$  and

$$\frac{r_{k'',m'',n} - r_{k',m',n}}{p_{k'',m'',n} - p_{k',m',n}} \geq \frac{r_{k',m',n} - r_{k,m,n}}{p_{k',m',n} - p_{k,m,n}} \quad (4)$$

note that we have

$$p_{k',m',n} = p_{k,m,n} \frac{p_{k'',m'',n} - p_{k',m',n}}{p_{k'',m'',n} - p_{k,m,n}} + p_{k'',m'',n} \frac{p_{k',m',n} - p_{k,m,n}}{p_{k'',m'',n} - p_{k,m,n}} \quad (5)$$

and from (4) we can deduce that

$$r_{k',m',n} \leq r_{k,m,n} \frac{p_{k'',m'',n} - p_{k',m',n}}{p_{k'',m'',n} - p_{k,m,n}} + r_{k'',m'',n} \frac{p_{k',m',n} - p_{k,m,n}}{p_{k'',m'',n} - p_{k,m,n}} \quad (6)$$

Given an optimal solution to the LP, we can construct another solution with

$$\begin{aligned} x'_{k,m,n} &= x_{k,m,n} + x_{k',m',n} \frac{p_{k'',m'',n} - p_{k',m',n}}{p_{k'',m'',n} - p_{k,m,n}} \\ x'_{k',m',n} &= 0 \\ x'_{k'',m'',n} &= x_{k'',m'',n} + x_{k',m',n} \frac{p_{k',m',n} - p_{k,m,n}}{p_{k'',m'',n} - p_{k,m,n}} \end{aligned}$$

(3) is satisfied since  $x'_{k,m,n} + x'_{k',m',n} + x'_{k'',m'',n} = x_{k,m,n} + x_{k',m',n} + x_{k'',m'',n}$ , and so is (2) since  $x'_{k,m,n}, x'_{k',m',n}, x'_{k'',m'',n} \geq 0$  and none of them can surpass 1 or else one of the  $x'_{\cdot,\cdot,n}$  would be negative.

(2) is satisfied owing to (5), and  $U' \geq U$  owing to (6). Thus we get an optimal solution where  $x_{k',m',n} = 0$

In the pseudo-code below, we consider the input as points in the plane with coordinates  $(p_{k,m,n}, r_{k,m,n})$ . For simplicity, for a stack  $S$ , we note  $S[0]$  the top element and  $S[1]$  the second top one; for two points  $A, B$  in the plane, we note  $L(A, B)$  the slope of the line formed between them; we say  $B$  is *dominated* by  $A$  and  $C$  if and only if  $A.p \leq B.p \leq C.p$  and  $L(A, B) \leq L(B, C)$ , which is just another interpretation of (4)

REMOVE-LP-DOMINATED( $n$ )

```

1  sort the pairs  $(p_{k,m,n}, r_{k,m,n})$  in increasing order of  $p_{k,m,n}$  into an array  $A$ . If several pairs
   have the same  $p$ , leave only the one with the greatest  $r$ 
2  let  $S$  be a stack
3  PUSH( $A[0]$ ,  $S$ )
4  PUSH( $A[1]$ ,  $S$ )
5  for  $i = 2$  to  $A.length-1$ 
6      while  $L(A[i], S[0]) \geq L(S[0], S[1])$ 
7          POP( $S$ )
8      PUSH( $A[i]$ ,  $S$ )
9  return  $S$ 
```

**Proof of correctness** We use the loop invariant that after each iteration of line 5,  $S$  contains exactly all the points that are not dominated by points from  $A[0]$  to  $A[i]$ , the line segments form a convex curve.

The invariant trivially holds before line 5. Suppose it holds before the  $i$ th iteration. During the  $i$ th iteration, the points removed by line 7 are clearly dominated by  $A[i]$  and  $S[1]$ . After the **while** loop terminates, we can be sure that all points in  $A$  between  $S[0]$  and  $A[i]$  are dominated by these two points and thus cannot be added to  $S$ , nor can the points between  $A[0]$  and  $S[0]$  by the loop invariant.

Note that the points in  $S$  form a convex curve, so that by  $L(A[i], S[0]) < L(S[0], S[1])$  we have that after line 8 the points in  $S$  still form a convex curve and so no points in  $S$  are dominated by points from  $A[0]$  to  $A[i]$ , and by the arguments above, these are exactly all the points that have this property.

**Time complexity** Line 1 takes time  $O(KM \log(KM))$ . Regarding the **for** loop from line 5 to 8, note that each point can only be pushed or popped only once, so in all the **for** loop takes time  $O(KM)$ . We run the algorithm for each  $n \in \mathcal{N}$ , so altogether it takes time  $O(NKM \log(KM))$  to remove all LP-dominated terms.

**Question 5.**

file	original	simple preprocess	IP preprocess	LP preprocess
1	24	24	10	8
2	24	4	4	4
3	24	24	13	9
4	614400	614400	14688	4974
5	2400	2400	329	193

Table 1: Preprocess results

In the table above are the numbers of data for each test file after each phase of preprocessing. To better illustrate how the data are reduced, we present the graphs of the first 6 channels of testfile5 below. We see that after three types of preprocessing, the rate-power curve is increasing and convex.

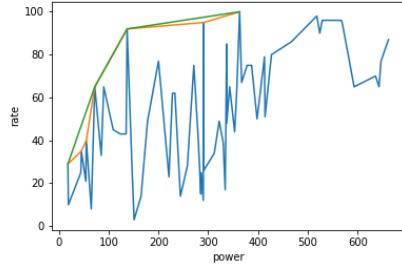


Figure 1: channel 1

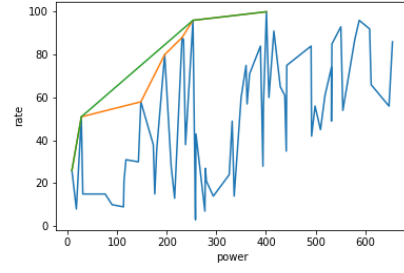


Figure 2: channel 2

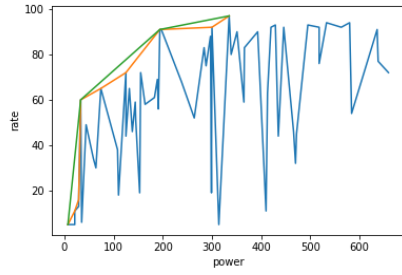


Figure 3: channel 3

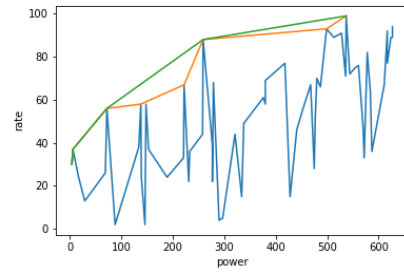


Figure 4: channel 4

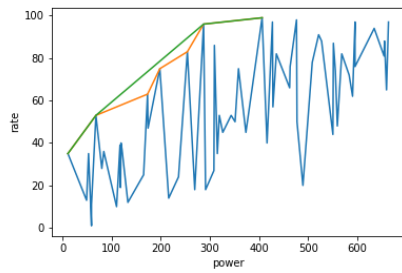


Figure 5: channel 5

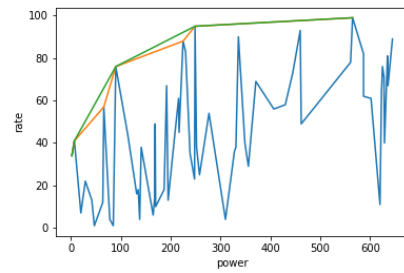


Figure 6: channel 6

### Question 6.

**Greedy Algorithm** We consider establishing a greedy solution. As indicated in the question stem, for each  $n$ , we sort the  $(p_{l,n}, r_{l,n})$  in the ascending order of  $p_{l,n}$ . We add one more convention  $\forall n, p_{0,n} = 0, r_{0,n} = 0$ . Initially, we allocate to each channel no user, which means  $(p_{0,n}, r_{0,n})$ . Thus the initial utility is 0.

To increase the total utility, every loop, we choose one channel to allocate more power. The criteria is the  $e_{l,n} = \frac{r_{l+1,n} - r_{l,n}}{p_{l+1,n} - p_{l,n}}$ , which shows the average payoff we can get by increase  $l$  to  $l+1$ . So each time we choose the channel with the largest  $e_{l,n}$  and change its user from  $l$  to  $l+1$ . The loop ends, when all channels have been allocated with the users with the largest power or  $p \leq p_{current}$

When  $p < p_{current}$ , the current allocation is not feasible. So we allocate to the last channel a linear combination of its last two allocation. We have

$$\sum_{k \neq n} p_{l_k, k} + p_{l-1, n} < p \quad (7)$$

$$\sum_{k \neq n} p_{l_k, k} + p_{l, n} = p_{current} > p \quad (8)$$

We search for a  $\epsilon$  that

$$\sum_{k \neq n} p_{l_k, k} + \epsilon p_{l-1, n} + (1 - \epsilon) p_{l, n} = p \quad (9)$$

Combining (8) and (9), we get  $\epsilon$

$$\epsilon = \frac{p_{current} - p}{p_{l, n} - p_{l-1, n}}$$

GREEDY-ALGORITHM-SOLUTION( $p, r, N, M, K$ )

```

1   $p_{current} = 0$ 
2  for  $n = 1$  to  $N$ 
3       $l[n] = 0$ 
4  while  $p_{current} < p$ 
5      find the  $n \in 1, \dots, N$  with the largest  $e_{l[n], n} = \frac{r_{l[n]+1, n} - r_{l[n], n}}{p_{l[n]+1, n} - p_{l[n], n}}$ 
6       $p_{current} += p_{(l[n_m]+1), n_m} - p_{l[n_m], n_m}$ 
7       $l[n] += 1$ ;
8       $n_m = n$ 
9  if  $p_{current} == p$ 
10     for  $n = 1$  to  $N$ 
```

```

11          $x_{l[n],n} = 1$ 
12     else
13         for n = 1 to N not  $n_m$ 
14              $x_{l[n],n} = 1$ 
15              $x_{l[n_m]-1,n_m} = \epsilon := \frac{p_{current} - p}{p_{l[n_m],n_m} - p_{l[n_m]-1,n_m}}$ 
16              $x_{l[n_m],n_m} = 1 - \epsilon$ 
17     return  $x$ 

```

**Time Complexity** Line 2 to 3 take  $O(N)$ . Considering the **while** loop from line 4 to line 7, every loop we have to check every channel to get the max payoff, so each loop takes  $O(N)$ . In the worst case, the final allocation could be  $p_{L,n}, r_{L,n}$  for every  $n < N$ , and it takes  $NL$  loops to get that. The line 8 to the end takes constant time. So the complexity for the entire algorithm is  $O(N^2L)$

TO DO: If we use a priority queue to store the  $l[n]$ . Every loop takes constant time. Then the complexity of the algorithm becomes  $O(NL)$

#### Question 7.

The result of the Greedy Solution and LP Solution is presented in the table below

file	Optimal Rate by Greedy Solution	Optimal Rate by LP Solution
1	365	365
2	0	0
3	372.15	372.15
4	9215.97	9970.25
5	1573.11	1657.82

Table 2: Comparison between Two Solutions

We observe that the greedy solution is not the optimal solution;  
The cpu time of the *testfile 4* and *testfile 5* are in the table below.

file	CPU Time of Greedy Solution (ms)
4	125
5	12.96

Table 3: CPU run time

### Question 8.

**DP Solution** To find a DP Solution of the whole problem, we consider the subproblem: *find the best utility we can get using only the first  $n$  channels with total power limit  $p'$*  . Let  $DP_{n,p}$  be the matrix who stores these values. We have the relation below:

$$\begin{aligned} DP(0, p') &= 0 \quad \forall p' \in \{0 \dots p\} \\ DP(n, 0) &= 0 \quad \forall n \in \{0 \dots N\} \\ DP(n, p') &= \max_{l \in \{0, \dots, L\}} \{DP(n-1, p' - p_{l,n}) + r_{l,n}\} \end{aligned}$$

By filling iteratively the matrix, we get the optimal allocation

DP-SOLUTION-MAXIMUM-UTILITY(p,r,N,M,K)

```

1  DP = ZEROS[N][p]
2  for q = 1 to p
3      for n = 1 to N
4          for l = 1 to L
5              if q - pl,n ≥ 0:
6                  DP[n][q] = MAX(DP[n-1][q], DP[n-1][q - pl,n] + rl,n)
7  return DP[N][P]
```

But this algorithm only return the maximum utility but not the optimal allocation. We need a way to store the optimal allocation and we decide to use another matrix of dimension  $(N, p)$  to store the optimal allocation for channel  $n$  in the sub problem  $(n, q)$

DP-SOLUTION(p,r,N,M,K)

```

1  DP = ZEROS[N][p]
2  LastTask = ZEROS[N][p]
3  for q = 1 to p
4      for n = 1 to N
5          for l = 1 to L
6              lM = 0
7              if q - pl,n ≥ 0 and DP[n-1][q - pl,n] + rl,n ≥ DP[n][q]
8                  DP[n][q] = DP[n-1][q - pl,n] + rl,n
9                  lM = l
10         LastTask[n][q] = lM
```



```

11   $q = p$ 
12  for  $n = N$  to 1
13       $l_M = \text{LastTask}[n][q]$ 
14       $x_{l_M, n} = 1$ 
15       $q -= p_{l_M, n}$ 
16  return  $x$ 

```

**Time Complexity** The loop from line 3 to line 10 takes  $O(NLp)$ . The loop after line 12 takes  $O(N)$ . Thus the time complexity of the entire algorithm is  $O(NLp)$

**Space Requirement** The matrices  $DP$  and  $\text{LastTask}$  take  $O(Np)$

### Question 9.

**DP Solution** Right now we consider the subproblem: *find the minimum power we need to reach total utility  $r$  using only first  $n$  channels.* Here  $r$  ranges from 0 to  $U := \sum_{k=1}^n r_{L,k}$ .  $U$  is the highest utility we can get from these  $n$  channels. Let  $DP(n, r)$  be the matrix who stores these values. We have the relations below:

$$\begin{aligned}
 DP(0, r) &= \infty \quad \forall r \in \{0 \dots U\} \\
 DP(n, r^-) &= 0 \quad \forall n \in \{0 \dots N\} \quad \forall r^- \leq 0 \\
 DP(n, r) &= \min_{l \in \{0, \dots, L\}} \{DP(n-1, r - r_{l,n}) + p_{l,n}\}
 \end{aligned}$$

By filling iteratively the matrix, we get the optimal allocation when  $DP(N, r)$  reach  $p$ . If all values in  $DP[N][:]$  are less than  $p$ , we can reach the maximum utility  $U$  by assigning the maximum power.

DP-SOLUTION( $p, r, N, M, K$ )

```

1   $U = 0$ 
2  for  $n = 1$  to  $N$ 
3       $U += r_{L,n}$ 
4   $DP = \text{INFITY}[N][r]$ 
5  for  $n = 1$  to  $N$ 
6       $DP[n][0] = 0$ 
7   $\text{LastTask} = \text{ZEROS}[N][p]$ 
8  for  $r = 1$  to  $U$ 

```

```

9      for  $n = 1$  to  $N$ 
10         for  $l = 1$  to  $L$ 
11             $l_M = 0$ 
12            if  $(r - r_{l,n}) \geq 0$  and  $(DP[n-1][r - r_{l,n}] + p_{l,n}) \leq DP[n][r]$ 
13                $DP[n][r] = DP[n-1][r - r_{l,n}] + p_{l,n}$ 
14                $l_M = l$ 
15            elif  $(r - r_{l,n}) \leq 0$  and  $p_{l,n} \leq DP[n][r]$ 
16                $DP[n][r] = p_{l,n}$ 
17                $l_M = l$ 
18             $LastTask[n][q] = l_M$ 
19         if  $DP[N][r] > p$ 
20             $r_M = r - 1$ 
21            break
22  $r = r_M$ 
23 for  $n = N$  to  $1$ 
24     $l_M = LastTask[n][r_M]$ 
25     $x_{l_M,n} = 1$ 
26     $r -= r_{l_M,n}$ 
27    if  $r \leq 0$ 
28        break
29 return  $x$ 

```

**Question 10.**

**Branch-and-Bound** We set each node as a subproblem defined as *The highest utility we can get with the constraint :  $[1 \leq l_1 \leq L_1, \dots, 1 \leq l_N \leq L_N]$  .*

BB-SOLUTION( $p, r, N, M, K$ )

```

1   $r_{max} = 0$ ;
2  Let  $Q$  be a queue
3  PUSH( $Q, [L, \dots, L]$ )
4  while  $Q$  is not empty
5       $[L_1, \dots, L_N] = DEQUEUE(Q)$ 

```

```

6      if  $\sum_{n=1}^N p_{l_n,n} \leq p$ 
7          if  $\sum_{n=1}^N r_{l_n,n} \geq r_{max}$ 
8               $r_{max} = \sum_{n=1}^N r_{l_n,n}$ 
9               $A_{best} = [L_1, \dots, L_N]$ 
10         break
11     for  $n = 1$  to  $N$ 
12         if  $L_n > 1$ 
13             PUSH(  $Q, [L_1, \dots, L_n - 1, \dots, L_N]$  )
14 for  $n = 1$  to  $N$ 
15      $x_{A_{best}[n],n} = 1$ 
16 return  $x$ 

```

In the worst case, the algorithm will have to examine all possible nodes, which is in fact a brute-force algorithm, and the time complexity is  $O(L^N)$

#### Question 11.

The result of DP Solution and BB Solution are presented in the tables below.

file	( Optimal Rate / power ) by DP Solution - 1	DP Solution -2	BB Solution
1	365 / 78	365 / 78	365 / 78
2	0 / 0	0 / 0	0 / 0
3	350 / 68	350 / 68	350 / 68
4	9970 / 16000	9970 / 15999	
5	1655 / 1000	1655 / 1000	

Table 4: Comparison between Two Solutions

file	( CPU Time ) DP Solution - 1 (ms)	DP Solution - 2 (ms)	BB Solution (ms)
4	178951.33	182053.49	$\geq 2h$
5	480.71	1231.72	$\geq 2h$

Table 5: CPU run time

For test files 4 and 5, the run time is so long that we did not get the result.

### Question 12.

Inspired by the greedy Solution, we consider to establish a value *payoff* to justify our choice. For example, in the greedy solution, we set the  $payoff = \frac{\partial rate}{\partial power}$ . We can also simply set the  $payoff = rate$ , which means every time we receive an user, we offer him the channel which can give the most rate.

However, ususally we have more users than channels available. We cannot let the users arriving earlier get what they want and let the others starve. So we want to set a threshold for the payoff. Every time a new user arrives, if its maximum *payoff* is smaller than the threshold or its channel preferred is already occupied , we reject it. Else we give it the channel which maximize its payoff.

We should always keep an eye on the power we use. If the maximum *payoff* will exceed the power limit, we choose the second biggest *payoff*.

ONLINE-SCHEDULING(p, r, M, N, K, threshold)

```
1  result = [ ]
2  channelUsed = [ ]
3  totalRate = 0
4  totalPower = 0
5  for k = 1 to K:
6      payoffMax = 0
7      for n = 1 to N, m = 1 to M:
8          payoff[n, m] = PAYOFF(r, p)
9      payoffSorted = SORT(FLATTEN(payoff))
10     for i = N × M to 1:
11         (m, n) = payoff.INDEX( payoffSorted[i] )
12         if totalPower +  $p_{m,n,k} \leq p$ :
13             payoffMax =  $\frac{r_{m,n,k}}{p_{m,n,k}}$ 
14             # find the biggest payoff which does not exceed the power limit
15             break
16     if payoffMax ≥ threshold:
17         if n not in channelUsed:
```

```

18         PUSH(channelUsed, n)
19         PUSH(res, [m,n])
20     else
21         PUSH(res, [-1,-1]) # user rejected
22 else
23     PUSH(res, [-1,-1]) # user rejected
24 # res holds the channel and power level for every user
25 return res

```

**Question 13.**