User scheduling in 5G

Pedro Luiz Silva and Sabrina Lomelino Sartori

February 11, 2024

Contents

Remarks

- A list of size n is indexed from 1 to n (not from 0 to $n-1$)
- The complete source code can be found here
- The results of the algorithms implemented in this project can be found in the directory *notebooks*

1 Problem definition

Consider $x_{k,m,n} = 1$ if the chanel n serves user k and $p_{k,m,n} \leq p_{k,n} \leq p_{k,m+1,n}$, and $x_{k,m,n} = 0$ otherwise. The ILP is given by the following:

- Maximize $\sum_{k \in \kappa} \sum_{n \in N} \sum_{m=1}^{M} r_{k,m,n} \times x_{k,m,n}$
- Constraint of power budget: $\sum_{k \in \kappa} \sum_{n \in N} \sum_{m=1}^{M} p_{k,m,n} \times x_{k,m,n} \leq p$
- Constraint of one user per channel: $\sum_{k \in \kappa} \sum_{m=1}^{M} x_{k,m,n} = 1$ for all $n \in N$
- Constraint of boolean variable: $x_{k,m,n} \in \{0, 1\}$

2 Preprocessing

2.1 Removing trivial unfeasible triplets

If $p_{k,n} > p$ then the instance is not a solution. We will then set $x_{k,m,n} = 0$ for all triplets (k, m, n) such that $p_{k,m,n} > p$.

Since every user channel must be used by exactly one user, we must have that

$$p_{k,m,n} + \sum_{n' \neq n} p_{min,n'} \leq p \tag{1}$$

Where $p_{min,n'} = \min_{k \in \mathcal{K}} p_{k,n'}$.

2.2 Removing IP dominated triplets

For each one of the N channels, we should first sort the elements $(p_{k,m,n}, r_{k,m,n}, k, m)$ by increasing order of power. Then we should store the maximum value of rate r_{max} while we go through the elements of the sorted list. If the current element has a bigger rate than r_{max}, we should update this value.

Algorithm 1: Remove IP-dominated triplets

```
input  : List $u_{k,n}$ $\forall(k,n) \in \mathcal{K} \times \mathcal{N}$
output: not IP dominated triplets $(k,m,n)$
1  for $n \in \mathcal{N}$ do
2      Create list $L = [(p_{k,m,n}, r_{k,m,n}, k, m)]$
3      $L_{sorted} \leftarrow$ Sort $L$ by increasing order of $p_{k,m,n}$
4      $r_{max} \leftarrow L_{sorted}[1,2]$
5      for $e \in L_{sorted}$ do
6          $r \leftarrow e[2]$
7          $k, m \leftarrow e[3], e[4]$
8          if $r \leq r_{max}$ then
9              Remove $e$ from $L_{sorted}$
10         else
11             $r_{max} \leftarrow r$
12 return $L_{sorted}$
```

The code complexity for sorting the list for a given channel is $O(KMlog(KM))$, and the complexity to visit every element of the sorted list is $O(KM)$. We can conclude that the total complexity for the algorithm is $O(NKMlog(KM))$.

2.3 Removing LP Dominated triplets

We will take into consideration the Graham Scan algorithm, which finds the convex hull of a finite set $\mathcal{S}$ of points in the plane. In each iteration, this algorithm removes concavities. *i.e.* points (x_2, y_2) that satisfy the inequality

$$\frac{y_1 - y_2}{x_1 - x_2} \leq \frac{y_2 - y_3}{x_2 - x_3} \tag{2}$$

For points $(x_1, y_1), (x_3, y_3) \in \mathcal{S}$. The algorithm is detailed below.

Algorithm 2: Graham Scan for convex hulls

```
   input  : List $P = [(a_1, b_1), \ldots, (a_n, b_n)]$
   output: Points of $P$ on the convex hull
1  $x_1, y_1 \leftarrow$ Point with lowest $y$ coordinate of $P$
2  $convexHull \leftarrow [(x_1, y_1)]$
3  $P_{sorted} \leftarrow$ Sort $P$ by increasing order of $\frac{y_k - y_1}{x_k - x_1}$
4  for $(x_i, y_i) \in P_{sorted}$ do
5      while true do
6          $(x_j, y_j) \leftarrow$ Last element of $convexHull$
7          $(x_{j-1}, y_{j-1}) \leftarrow$ Penultimate element of $convexHull$
8          if $\frac{y_j - y_i}{x_j - x_i} > \frac{y_{j+1} - y_j}{x_{j+1} - x_j}$ then
9              Push $(x_i, y_i)$ to $convexHull$
10             break
11         Pop $(x_j, y_j)$ from $convexHull$
12 return $convexHull$
```

In a similar way to the Graham Scan algorithm, we will sort our list by increasing power and keep all elements that belong to the convex hull (*i.e.* that satisfy the inequality given) in a stack. In each iteration, we must not only check if the element should be added to the stack, but also verify if the last added element still should be on the stack. In the case it shouldn't, then we need to pop it from the stack.

Algorithm 3: Remove LP-dominated triplets

```
   input  : List $u_{k,n}$ $\forall (k, n) \in \mathcal{K} \times \mathcal{N}$
   output: Non LP dominated triplets $(k,m,n)$
1  for $n \in \mathcal{N}$ do
2      Create list $L = [(p_{k,m,n}, r_{k,m,n}, k, m)]$
3      $l_1 = (k_1, m_1, n) \leftarrow$ Element with lowest power $p_{k,m,n}$ of $L$
4      $convexHull \leftarrow [l_1]$
5      $L_{sorted} \leftarrow$ Sort $L$ by increasing order of $\frac{r_{k,m,n} - r_{k_1,m_1,n}}{p_{k,m,n} - p_{k_1,m_1,n}}$
6      for $(p_{k,m,n}, r_{k,m,n}, k, m) \in L_{sorted}$ do
7          $l \leftarrow (k, m, n)$
8          while true do
9              $l_{-1} = (k_{-1}, m_{-1}, n) \leftarrow$ Last element of $convexHull$
10             $l_{-2} = (k_{-2}, m_{-2}, n) \leftarrow$ Penultimate element of $convexHull$
11             if $\frac{r_{k_{-1},m_{-1},n} - r_{k,m,n}}{p_{k_{-1},m_{-1},n} - p_{k,m,n}} > \frac{r_{k_{-2},m_{-2},n} - r_{k_{-1},m_{-1},n}}{p_{k_{-2},m_{-2},n} - p_{k_{-1},m_{-1},n}}$ then
12                 Push $l$ to $convexHull$
13                 break
14             Pop $l_{-1}$ from $convexHull$
15 return $convexHull$
```

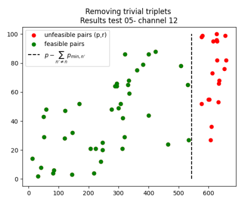
For each one of the N channels, sorting the list L has a complexity of

$O(KMlog(KM))$. Since every element can be added and removed from the stack at most once, the code complexity for the loop (lines 6 to 14) is $O(2KM)$. So the final complexity of the algorithm is $O(NKMlog(KM))$.

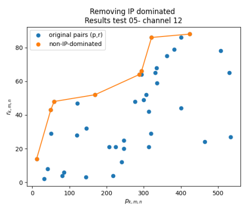
2.4 Preprocessing results

	test1	**test2**	**test3**	**test4**	**test5**
Initial size	24	24	24	614400	2400
Removing trivial triplets	10	0	24	614400	1954
Removing IP dominated triplets	10	0	13	14688	300
Removing LP dominated triplets	8	0	9	4974	179

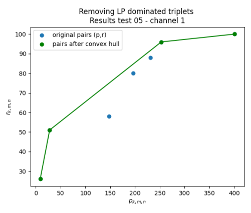
Table 1: Number of elements that were not removed during preprocessing



(a) Removing trivial triplets result on test 05. Note that we removed every pair $(p_{l,n}, r_{l,n})$ such that $p_{l,n} > p - \sum_{n' \neq n} p_{min,n'}$



(b) Removing IP dominated triplets result on test 05. Note that we removed every pair $(p_{l,n}, r_{l,n})$ such that there exists $p_i \leq p_{l,n},\ r_i \geq r_{l,n}$.



(c) Removing IP dominated triplets result on test 05. Note that we removed every pair $(p_{l,n}, r_{l,n})$ inside the convex hull.

Figure 1: Elements removed during preprocessing

3 Greedy algorithm

First of all, let's define some variables and notations that we will use in our approach:

- *solution* $= [(k^\star_1, m^\star_1), \dots (k^\star_n, m^\star_n)]$ the list of pairs such that $x_{k^\star_i, m^\star_i, i} = 1$
- The variables $p_{l,n}$ and $r_{l,n}$ are sorted by increasing order of $p_{l,n}$. This means that $p_{1,n} < p_{2,n} < \dots < p_{L,n}, \forall n \in \mathcal{N}$

Then let's describe our approach. Since we have the constraint that all channels must be allocated, for all channel n, we will start setting n to the least power-consuming user for this channel, given by (k_n, m_n, n). Initially, for all $n \in \mathcal{N}$:

$$\begin{aligned} solution[n] &= (k_n, m_n) \\ k_n, m_n &= \underset{k,m}{\arg\min}\ p_{k,m,n} \end{aligned} \tag{3}$$

This leads us to a remaining power budget of $p' = p - \sum_{n \in \mathcal{N}} p_{k_n,m_n,n}$ and a starting data rate of $r' = \sum_{n \in \mathcal{N}} r_{k_n,m_n,n}$.

We will then create $e = [(l_1, n_1, e_{l_1,n_1}),\ \dots]$, a list sorted by decreasing order of $e_{l,n}$. At each step, we will try to update *solution* with the first element of e (the element with maximal increasing efficiency).

Suppose that l_1 is called for the pair (k^1, m^1). Also suppose that at this stage of the execution $solution[n_1] = (k, m)$. Then we will try to allocate the channel n_1 to the pair (k^1, m^1) if we have enough power, *i.e.* .

$$\begin{aligned} &solution[n_1] = (k^1, m^1) \\ &\text{if } p_{k^1,m^1,n_1} - p_{k,m,n_1} \leq p' \end{aligned} \tag{4}$$

When we run out of power, we will start using fractional allocation for the triplets (k, m, n_1) and (k^1, m^1, n_1) in order to use the complete power budget p. We will then solve our problem with LP relaxation.

The complete pseudo-code is described below.

Algorithm 4: Greedy approach on LP

```
input  : Preprocessed lists $p_{l,n}$ and $r_{l,n}$ $\forall(l,n) \in \mathcal{L} \times \mathcal{N}$, power budget $p$
output: Maximal data rate $r$
1  $solution = [\arg\min_{k,m} p_{k,m,n}\ \forall n \in \mathcal{N}]$
2  $r = 0$
3  // update the data rate and power budget
4  for $n \in \mathcal{N}$ do
5      $k, m \leftarrow solution[n]$
6      $p \leftarrow p - p_{k,m,n}$
7      $r \leftarrow r + r_{k,m,n}$
8  $e$ = list of pairs sorted by decreasing order of incremental efficiency
9  for $(k^1, m^1, n) \in e$ do
10     if $p_{k^1,m^1,n} - p_{k,m,n} <= p$ then
11         $solution[n] \leftarrow (k, m)$
12         $p \leftarrow p - p_{k^1,m^1,n} - p_{k,m,n}$
13         $r \leftarrow r + r_{k^1,m^1,n} - r_{k,m,n}$
14     else
15         break
16 if $p == 0$ then
17     // We have an integer solution
18     return $r$
19 else
20     // Fractional allocation
21     $\lambda = \frac{p}{p_{k^1,m^1,n} - p_{k,m,n}}$
22     $r \leftarrow r + \lambda r_{k^1,m^1,n} + (\lambda - 1) r_{k,m,n}$
23     return $r$
```

Sorting the pairs in decreasing order of incremental efficiency has a time complexity of $O(NKMlog(NKM))$, the first loop has complexity $O(N)$ and the second loop $O(KMN)$. We conclude that the total complexity of the greedy algorithm is $O(NKMlog(NKM))$.

3.1 Greedy algorithm results

In order to compare our greedy solution with the exact one, we have used PuLP, which is an open-source Python library used for Linear Programming. Its documentation can be found here.

The table below summarizes the results found.

	test1	**test2**	**test3**	**test4**	**test5**
r (Greedy)	365	-	372.15	9870.32	1637
r (PuLP)	365	-	372.15	9870.32	1637
Runtime (Greedy)	13	-	17	1156	145
Runtime (PuLP)	5	-	6	113	13

Table 2: Results of runtime and data rate achieved by the greedy (4) and PuLP algorithms. The runtimes are on ms.

4 Dynamic programming algorithm

Let's solve the ILP by a dynamic programming approach. Lets call $R^\star(n,p)$ the solution of our problem given n channels and power budget p. We want to write $R^\star(n,p)$ in terms of solutions of $n-1$ channels.

Given that we have the optimal solution of the problem with $n-1$ channels, we can expand it to n with the following equation

$$R^\star(n,p) = \max_{l\in\mathcal{L}, p_{l,n}\le p} \{R^\star(n-1, p-p_{l,n}) + r_{l,n}\} \tag{5}$$

We will start by $n=1$ and store all possible values of $(p_{l,1}, r_{l,1})$ in a hash map. Then for each channel $n\in\mathcal{N}$ we will iterate over every triple (k,m,n) and complete the hash map with the attainable values of power with its respective data rates. This gives us a time complexity of $O(pKMN)$

To lower memory consumption, when we look at the channel n, we only store the results for the channel $n-1$, which gives us a space complexity of $O(p)$

Algorithm 5: DP approach on ILP

```
input  : Lists $p_{l,n}$ and $r_{l,n}$ $\forall(l,n)\in\mathcal{L}\times\mathcal{N}$, power budget $p$
output: Maximum data rate value
1 $R$ = HashMap() // Here we will store the optimal solutions for the step $n-1$
2 // initial step
3 $R(1, p_{l,1}) = r_{l,1}$ $\forall l\in\mathcal{L}$ s.t. $p_{l,1}\le p$
4 for $n\in\{2,\dots,N\}$ do
5     $R'$ = HashMap()
6     for every key $(n-1, p_i)\in R$ do
7         Set $R'(n,p') = \max_{i,l}\{R(n-1,p_i) + r_{l,n}|$ s.t. $p' = p_i + p_{l,n}\}$
8     $R = R'$
9 return $\max_{p_i\le p} R(N, p_i)$
```

In this second approach, we will try to minimize the power given a certain rate r and n channels. We will approach this with a $P^\star(n,r)$ function, which can be computed with the following DP equation.

$$P^\star(n,U) = \min_{l\in\mathcal{L}, r_{l,n}\le U} \{P^\star(n-1, U-r_{l,n}) + p_{l,n}\} \tag{6}$$

Algorithm 6: Another DP approach on ILP

```
input  : Lists $p_{l,n}$ and $r_{l,n}$ $\forall(l,n) \in \mathcal{L} \times \mathcal{N}$, maximum data rate $U$,
         power budget $p$
output: Maximum data rate value
1  $P = HashMap()$ // Here we will store optimal solutions for the step
     $n-1$
2  // Initial step
3  $P(1, r_{l,1}) = p_{l,1}$ $\forall l \in \mathcal{L}$ s.t. $r_{l,1} \leq U$
4  for $n \in \{2, \ldots, N\}$ do
5      $P' =$ HashMap()
6      for every key $(n-1, r_i) \in P$ do
7          Set $P'(n, r') = \min_{i,l}\{P(n-1, r_i) + p_{l,n}|$ s.t. $r' = r_i + r_{l,n}$ $\}$
8      $P = P'$
9  return $\max r$ s.t. $P(N, r) \leq p$
```

5 Branch and Bound approach

In this approach, we attempt to solve the ILP problem iteratively by employing LP relaxation. In each iteration, we recursively expand the problem to cover all combinations of $x_{l,n}$ while having the constraint $\sum_l x_{l,n} = 1$. Consequently, the worst-case computational complexity becomes $O(L^N) = O((KM)^N)$.

Algorithm 7: Branch and Bound algorithm for ILP

```
input  : Partial solution from channels 1 to $n_0$, $n_0$ and the remaining
         power budget
output: No output, but updates the partial solution
1  Global variable $bestSolution$ = empty solution
2  Function BB(solution, $n_0$, remainingPowerBudget):
3      if $n_0 = N$ then
4          if solution is better than bestSolution (in terms of data rate)
             then
5              update $bestSolution$ ←solution
6      $newSolution$ = try to solve relaxed sub LP
7      if No solution found then
8          return
9      if All variables $x_{k,m,n}$ of newSolution are integer then
10         $bestSolution$ ←newSolution
11         return
12     for $k, m \in$ list of pairs sorted by increasing power do
13         set $x_{k,m,n_0+1} = 1$
14         BB(newSolution, $n_0 + 1$, remainingPowerBudget $- p_{k,m,n_0}$)
15 By the end of execution $bestSolution$ stores the solution of the ILP
```

In the figure 4 we explain how this approach works. Whenever we find an integer solution to the LP, we store the data rates. The algorithm returns the higher data rate corresponding to an integer solution to the LP.

The table below shows the results both for the dynamic programming and for the branch and bound approaches.

	test1	**test2**	**test3**	**test4**	**test5**
r (DP1)	365	-	350	9850	1637
r (DP2)	365	-	350	9850	1637
r (B&B)	365	-	350	-	1637
Runtime (DP1)	1	-	2	81193	171
Runtime (DP2)	1	-	3	4679	220
Runtime (B&B)	8	-	49	∞	45

Table 3: Results of runtime and data rate achieved by dynamic programming and branch and bound algorithms. The runtimes are on ms.

6 Online algorithm

Firstly, we should consider what happens in each iteration, i.e., each time a new user arrives. Let's call this new user k. It is clear that, in this iteration, each channel n should choose whether to allocate user k to one of the possible pairs $(p_{k,m,n}, r_{k,m,n})$, where $1 \leq m \leq M$, or not to allocate this user and wait for the next ones to come.

Hence we have two decisions to make, given a user k and a channel n: the first one is to determine the best possible point to allocate user k to channel n (which we will call *if allocated* scenario) and the second one is if this best allocation should happen or not (which we will call *should allocate or not* scenario).

To deal with the first decision, we must compare the rate efficiency of all feasible points $(r_{k,m,n}, p_{k,m,n})$ for given k and n. In other words, the *if allocated* solution is the triplet (k, m^*, n) with maximal ratio $\frac{p_{k,m,n}}{r_{k,m,n}}$ among all $1 \leq m \leq M$ subject to the constraint that $p_{k,m^*,n}$ is lower than or equal to the remaining power to allocate.

For the second decision, there should be a criteria to check if the above solution $p_{k,m^*,n}$ is better than not allocating channel n to user k. An intuitive way to decide if the allocation should take place is comparing its the rate efficiency with the expected value of $\frac{r}{p}$, given by:

$$\mathbb{E}\left[\frac{r}{p}\right] = \frac{1}{p_{max} r_{max}} \sum_{p=1}^{p_{max}} \sum_{r=1}^{r_{max}} \frac{r}{p} \tag{7}$$

Our criteria for *should allocate or not* will be:

- If $\frac{r_{k,m^*,n}}{p_{k,m^*,n}} < \mathbb{E}\left[\frac{r}{p}\right]$, then we will not allocate user k to channel n

- If $\frac{r_{k,m^*,n}}{p_{k,m^*,n}} \geq \mathbb{E}\left[\frac{r}{p}\right]$, then we will allocate user k to channel n with the point $(p_{k,m^*,n}, r_{k,m^*,n})$

The final Online algorithm as described above is detailed below.

Algorithm 8: Online algorithm for user scheduling

```
input  : $K, M, N, p, p_{max}, r_{max}$
output: Data rate and total power used
1  allocated_channels = HashSet()
2  data_rate = 0
3  total_power = 0
4  ev = $\frac{1}{p_{max} r_{max}} \sum_{p=1}^{p_{max}} \sum_{r=1}^{r_{max}} \frac{r}{p}$
5  for $k \in \{1, \ldots, K\}$ do
6      generate pairs $(k, m, n, p_{k,m,n}, r_{k,m,n})$ for all channels
7      for $n \in \{1, \ldots, N\}$ and $n \notin$ allocated_channels do
8          if $\nexists (p_{k,m,n}, r_{k,m,n})$ s.t. total_power $+ p_{k,m,n} \leq p$ then
9              continue
10         Let $(k, m^*, n, p_{k,m^*,n}, r_{k,m^*,n})$ be the feasible pair with maximal
            efficiency
11         if $k = K - 1$ then
12             Add $n$ to allocated_channels
13             data_rate $\leftarrow$ data_rate $+ r_{k,m^*,n}$
14             total_power $\leftarrow$ total_power $+ p_{k,m^*,n}$
15         else
16             if should_allocate$(p_{k,m^*,n}, r_{k,m^*,n}, ev)$ then
17                 Add $n$ to allocated_channels
18                 data_rate $\leftarrow$ data_rate $+ r_{k,m^*,n}$
19                 total_power $\leftarrow$ total_power $+ p_{k,m^*,n}$
20 return data_rate, total_power
```

The function *should_allocate* used in line 16 of the pseudocode is given by:

Algorithm 9: Function *should_allocate* for Online Scheduling

```
input  : $p, r, \mathbb{E}\left[\frac{r}{p}\right]$
output: True if $\frac{r}{p} \geq \mathbb{E}\left[\frac{r}{p}\right]$ and False if not
1  return $\frac{r}{p} \geq \mathbb{E}\left[\frac{r}{p}\right]$
```

6.1 Results of the online algorithm

We can compare the online to the offline optimal solution. In the online solution we allocate users when they arrive and we can not change previous allocations. However, in the optimal solution, we consider the pairs generated in the online scheduling algorithm as an input and allocate every channel to a user.

In 2 we see the results for the values $p = 100$, $p_{max} = 50$, $r_{max} = 100, M = 2, N = 4$ and $K = 10$ ran over 100 epochs. It is clear that the online scheduling performs worse than the offline solution, and the results are:

- Average competitive ratio: 0.724, with standard deviation of 0.11
- Average power budget of online algorithm: 28.9
- Average power budget of offline optimal algorithm: 81.5

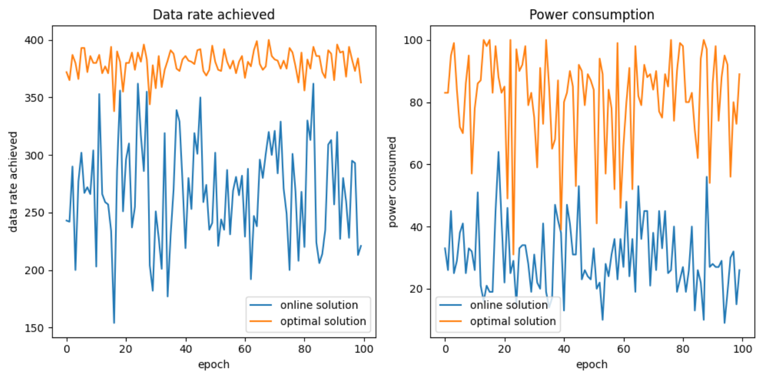


Figure 2: Data rates and power consumptions obtained in online scheduling and offline optimal solutions in 100 epochs

7 Appendix

7.1 Class diagrams of solutions

The code was object-oriented built and, in order to store the points $(k, m, n, p_{k,m,n}, r_{k,m,n})$ used in our algorithms, we have used a HashMap. Its keys were the channels' numbers $n \in 1, ..., N$, while the values were DataFrames whose columns were $k, m, n, p_{k,m,n}, r_{k,m,n}$. There is a column n because in later steps we have to sort all our points in increasing order of efficiency, and it was useful to keep track of the channel number when merging all DataFrames. This choice of data architecture was made in order to have our code as clear and clean as possible.

The details of our solution's architecture are given in figure 3.

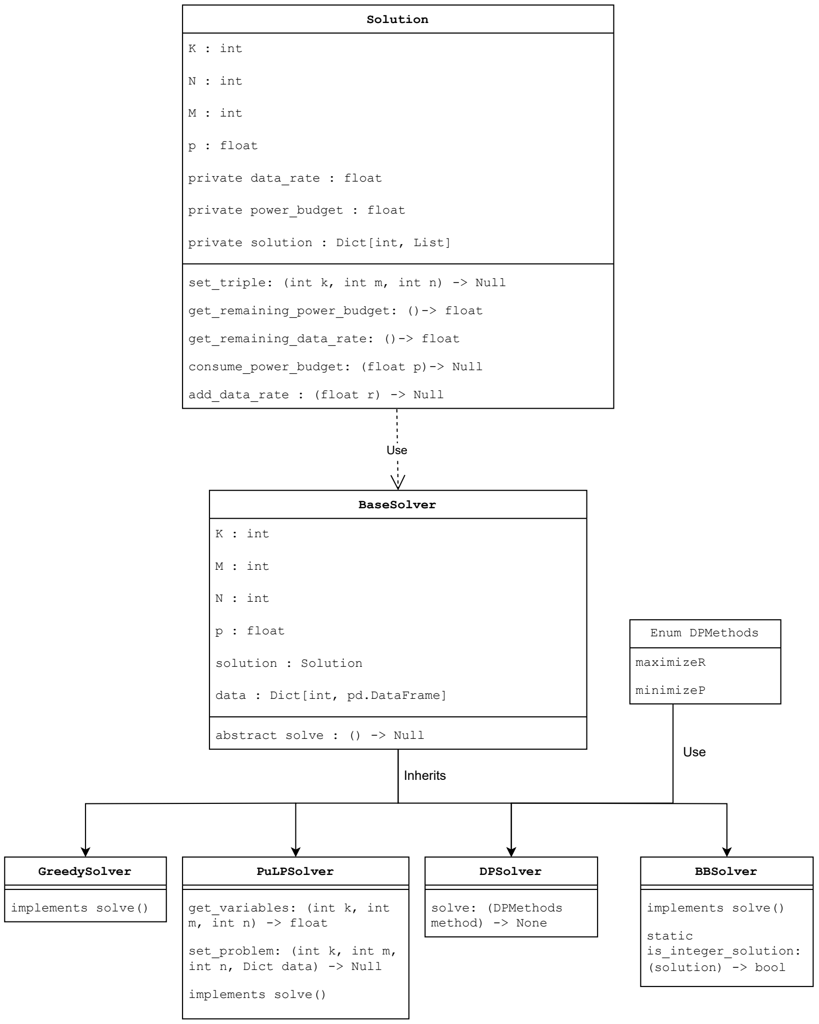

Solution
K : int
N : int
M : int
p : float
private data_rate : float
private power_budget : float
private solution : Dict[int, List]
set_triple: (int k, int m, int n) -> Null
get_remaining_power_budget: ()-> float
get_remaining_data_rate: ()-> float
consume_power_budget: (float p)-> Null
add_data_rate : (float r) -> Null
Use
BaseSolver
K : int
M : int
N : int
p : float
solution : Solution
data : Dict[int, pd.DataFrame]
abstract solve : () -> Null
Enum DPMethods
maximizeR
minimizeP
Use
Inherits
GreedySolver
implements solve()
PuLPSolver
get_variables: (int k, int m, int n) -> float
set_problem: (int k, int m, int n, Dict data) -> Null
implements solve()
DPSolver
solve: (DPMethods method) -> None
BBSolver
implements solve()
static is_integer_solution: (solution) -> bool

Figure 3: Class diagram of the solution. All the different solvers inherits from an abstract class BaseSolver

7.2 Branch and Bound flow chart

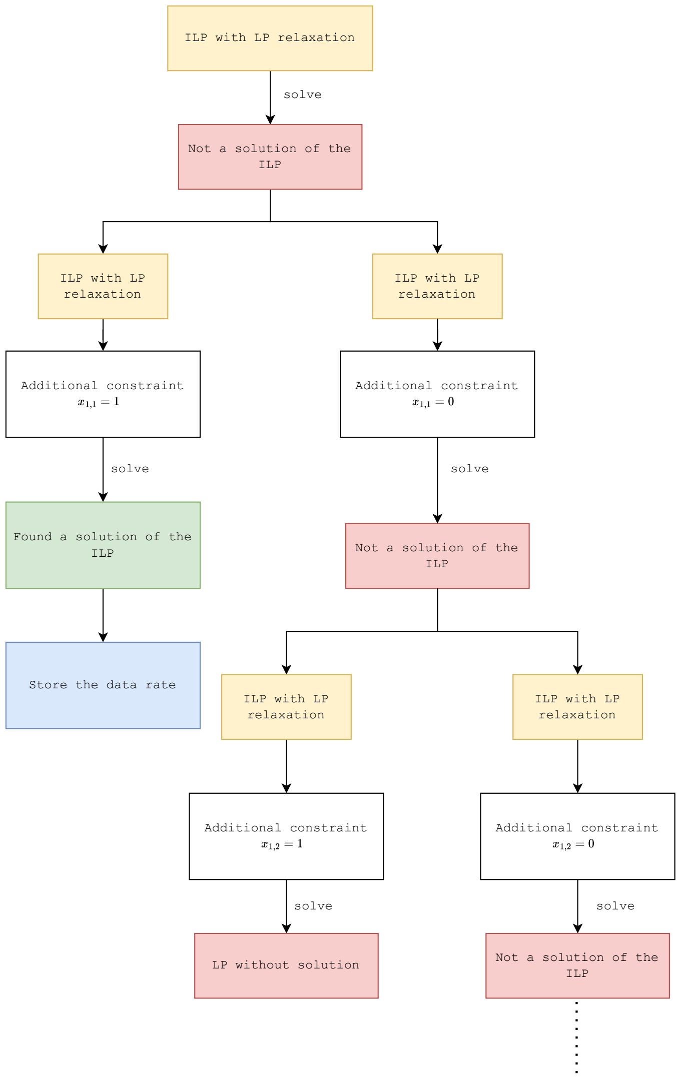

ILP with LP relaxation
solve
Not a solution of the ILP
ILP with LP relaxation
ILP with LP relaxation
Additional constraint $x_{1,1} = 1$
Additional constraint $x_{1,1} = 0$
solve
solve
Found a solution of the ILP
Not a solution of the ILP
Store the data rate
ILP with LP relaxation
ILP with LP relaxation
Additional constraint $x_{1,2} = 1$
Additional constraint $x_{1,2} = 0$
solve
solve
LP without solution
Not a solution of the ILP

Figure 4: Branch and Bound method flow chart for solving Integer Linear Programs (ILP).