Report of project INF421: User scheduling in 5G

Zhicheng HUI Zhengya NING

January 2024

1 Problem formulation

1.1 Question 1

The power transmitted by channel $n \in \mathcal{N}$ to user $k \in \mathcal{K}$ is denoted by $p_{k,n}$, and the corresponding data rate is given as $r_{k,n} = u_{k,n}(p_{k,n})$. The rate utility function $u_{k,n}$ is a non-decreasing step function, implying that the data rate remains constant when the power is in the interval $[p_{k,m,n}, p_{k,m+1,n}[$. In the following, we note K and N the carnality of the set $\mathcal{K}$ and $\mathcal{N}$ respectively.

Given an instance of the user scheduling problem, one solution will still remain feasible if we increase the total transmit power budget. Hence, for any user $k \in \mathcal{K}$ served on channel $n \in \mathcal{N}$, the solution will not become worse if we decrease the transmit power from $p_{k,n}$ to the biggest $p_{k,m,n}$ smaller than it, for it saves us $p_{k,n} - p_{k,m,n}$ units of power. Consequently, in the following we will only consider solutions such that:

$$\forall k \in \mathcal{K},\ n \in \mathcal{N},\ p_{k,n} \in \{0\} \cup \{p_{k,m,n}, m \in [1, M]\}$$

.

We introduce binary variables $x_{k,m,n}$ to indicate whether channel n serves user k with a transmit power of $p_{k,m,n}$. If channel n serves user k with transmit power $p_{k,m,n}$, then $x_{k,m,n}$ is set to 1; otherwise, it's set to 0. By formulating the problem in this way, we are actually choosing triplets among $\{(k, m, n), k \in \mathcal{K}, m \in [1, m], n \in \mathcal{N}\}$.

Two key constraints define the problem. Firstly, each channel must serve a single user, and all the channels must be allocated. This constraint can be expressed as:

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

.

Secondly, the total transmit power must not exceed the budget p. This constraint can be expressed as:

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

.

Our objective is to maximize the sum of data rates. Therefore, the problem can be formulated as the following Integer Linear Programming (ILP) problem:

$$\begin{aligned} \text{Maximize} \quad & \sum_{k,m,n} x_{k,m,n} \times u_{k,n}(p_{k,m,n}) \\ \text{Subject to} \quad & \sum_{k,m} x_{k,m,n} = 1 \quad \forall n \in \mathcal{N} \\ & \sum_{k,m,n} x_{k,m,n} \times p_{k,m,n} \leq p \end{aligned}$$

where the binary variables $x_{k,m,n} \in \{0, 1\}$.

2 Pre-processing

2.1 Question 2

Consider an instance of the user scheduling problem, it doesn't have solution if we are too tight on budget. That is to say, the sum of the minimum transmit power on each channel must not exceed the total transmit power budget:

$$\sum_{n} \min_{k,m} p_{k,m,n} \leq p$$

.

If this condition is not satisfied, then the problem instance has obviously no solution.

We can not choose a triplet (k, m, n) if it costs too much — whose presence leads to a situation where, even if all other channels have their minimum power, the total transmit power would still exceed the budget. Formally speaking, for a given triplet (k', m', n'), we verify:

$$p_{k',m',n'} + \sum_{n,n \neq n'} \min_{k,m} p_{k,m,n} \leq p$$

If this condition is not satisfied, then the triplet should be removed from the problem instance.

2.2 Question 3

We call a triplet IP-dominated if there exists another triplet of the same channel with faster data rate and less cost. By replacing the former with the latter we can always ameliorate the solution.

In order to remove all the IP-dominated triplets for a given channel $n \in \mathcal{N}$, we can sort the triplets in ascending order of transmit power $p_{k,m,n}$, we then expect their data rate to be equally ascending. We can do this by linearly scanning the array of sorted triplets and only keep those with strictly higher data rate $r_{k,m,n}$ than previous ones.

Let us assume that channels are numbered from 1 to N. For a given channel $n \in \mathcal{N}$, its triplets are stored in an array L_n. The pseudo-code is presented below:

Algorithm 1: removeInversion(L)

Input: An array of triplets L sorted in ascending order of transmit power
Output: The array L with IP-dominated terms removed

```
lastRate ← 0
foreach (k, m, n) ∈ L do
    if r_{k,m,n} <= lastRate then
        delete (k, m, n) from L
    else
        lastRate ← r_{k,m,n}
    end
end
return L
```

Algorithm 2: removeIPDominated($N, L_1, ..., L_N$)

Input: The number of channels N and their triplets in arrays $L_1, ..., L_N$
Output: Arrays of triplets with IP-dominated terms removed

```
for i ← 1 to N do
    Sort the array L_i of triplets (k, m, n) by increasing order of p_{k,m,n}
    L_i ← removeInversion(L_i)
end
return L_1, ..., L_N
```

For each channel $n \in \mathcal{N}$, its array of triplets have size $O(KM)$. By using an efficient sorting algorithm, we can sort the array in $O(KM \log(KM))$ time. We remove the IP-dominated terms through a single traversal of the whole array, suppose the time taken to delete an element of a array to be constant (this can be simulated by appending non IP-dominated triplets to another array initially set empty and return it), the time complexity of processing a single channel is $O(KM \log(KM))$. Consequently, a total time complexity of $O(NKM \log(KM))$ can be achieved.

2.3 Question 4

If we remove the constraints of $x_{k,m,n}$ being integer, we can obtain a Linear Programming (LP) problem, on which further optimizations on the input may become applicable.

From now on, the decision made by one channel may be a linear combination of its choice, as long as the sum of their coefficient equals one. Consequently, if for a given channel $n \in \mathcal{N}$, we have three choices $(p_{k,m,n}, r_{k,m,n}), (p_{k',m',n}, r_{k',m',n}), (p_{k'',m'',n}, r_{k'',m'',n})$ of which the middle one is by definition LP-dominated such that:

$$\begin{cases} p_{k,m,n} < p_{k',m',n} < p_{k'',m'',n} \\ r_{k,m,n} < r_{k',m',n} < r_{k'',m'',n} \\ \dfrac{r_{k',m',n} - r_{k,m,n}}{p_{k',m',n} - p_{k,m,n}} \leq \dfrac{r_{k'',m'',n} - r_{k',m',n}}{p_{k'',m'',n} - p_{k',m',n}} \end{cases}$$

Then, we actually have no reason to consider the choice $(p_{k',m',n}, r_{k',m',n})$. If we have a solution such that $x_{k',m',n} \neq 0$, we can replace $x_{k',m',n}$ by a linear combination of $x_{k,m,n}$ and $x_{k'',m'',n}$:

$$\begin{cases} x_{k,m,n} = \dfrac{p_{k'',m'',n} - p_{k',m',n}}{p_{k'',m'',n} - p_{k,m,n}} \times x_{k',m',n} \\ x_{k'',m'',n} = \dfrac{p_{k',m',n} - p_{k,m,n}}{p_{k'',m'',n} - p_{k,m,n}} \times x_{k',m',n} \end{cases}$$

The cost will remain unchanged while we can obtain a possibly higher date rate:

$$\begin{cases} x_{k,m,n} \times p_{k,m,n} + x_{k'',m'',n} \times p_{k'',m'',n} = x_{k',m',n} \times p_{k',m',n} \\ x_{k,m,n} \times r_{k,m,n} + x_{k'',m'',n} \times r_{k'',m'',n} \geq x_{k',m',n} \times r_{k',m',n} \end{cases}$$

That is to say, we want to keep an array of choices such that the ratio $\frac{r_{k',m',n} - r_{k,m,n}}{p_{k',m',n} - p_{k,m,n}}$ is strictly decreasing.

In order to remove all the LP-dominated triplets for a given channel $n \in \mathcal{N}$, we can still sort the triplets is ascending order of transmit power $p_{k,m,n}$ (which will by default be done if we perform the pre-processing in question 3). During the algorithm, we plan to store the temporary array of strictly decreasing ratio $\frac{r_{k',m',n} - r_{k,m,n}}{p_{k',m',n} - p_{k,m,n}}$ in a queue. Each time when we meet a new triplet $(p_{k'',m'',n}, r_{k'',m'',n})$, suppose the two last triplets stored in the queue to be $(p_{k',m',n}, r_{k',m',n})$ and $(p_{k,m,n}, r_{k,m,n})$ (when the queue only contains one element, we simply add the new triplet to the queue), we iteratively compare the two ratio $\frac{r_{k'',m'',n} - r_{k',m',n}}{p_{k'',m'',n} - p_{k',m',n}}$ and $\frac{r_{k',m',n} - r_{k,m,n}}{p_{k',m',n} - p_{k,m,n}}$. If the condition is not satisfied, we remove the first element in the queue and redo the process, until that either $\frac{r_{k'',m'',n} - r_{k',m',n}}{p_{k'',m'',n} - p_{k',m',n}} < \frac{r_{k',m',n} - r_{k,m,n}}{p_{k',m',n} - p_{k,m,n}}$ or there is no enough element in the queue, we then add the triplet in the queue.

Let us assume that channels are numbered from 1 to N. For a given channel $n \in \mathcal{N}$, its triplets are stored in an array L_n. The pseudo-code is presented below, where we note the last and the second last triplet in the queue (if they exist) as $(p_{k',m',n}, r_{k',m',n})$ and $(p_{k,m,n}, r_{k,m,n})$:

Algorithm 3: removeIncreasingRatio(L)

Input: An array of triplets L sorted in ascending order of transmit power
Output: The array L with LP-dominated terms removed

```
tempQueue ← EmptyQueue
foreach (k'', m'', n) ∈ L do
    while tempQueue.size() ≥ 2 and (r_{k'',m'',n} − r_{k',m',n}) / (p_{k'',m'',n} − p_{k',m',n}) ≥ (r_{k',m',n} − r_{k,m,n}) / (p_{k',m',n} − p_{k,m,n}) do
        Remove the last triplet from the queue
    end
    Add the current triplet to the queue
end
Remove all the triplets in L
Put the triplets in the queue to L
return L
```

Algorithm 4: removeLPDominated($N, L_1, ..., L_N$)

Input: The number of channels N and their triplets in arrays $L_1, ..., L_N$
Output: Arrays of triplets with LP-dominated terms removed

```
for i ← 1 to N do
    Sort the array L_i of triplets (k, m, n) by increasing order of p_{k,m,n}
    L_i ← removeIncreasingRatio(L_i)
end
return L_1, ..., L_N
```

Similar as the complexity analysis of question 3, for each channel $n \in \mathcal{N}$, its array of triplets have size $O(KM)$. During a single traversal of the array, each element will be added and deleted by at most once, of which the time complexity will be $O(KM)$. Consequently, a total time complexity of $O(NKM \log(KM))$ can be achieved if we take into account the step of sorting.

2.4 Question 5

For each channel $n \in \mathcal{N}$, there are originally KM triplets associated with it. We note the initial pre-processing, the removal of IP-dominated and of LP-dominated terms as step 1, 2 and 3, respectively. And we measure the size of a problem instance by the total number of triplets associated with each channel. The table below shows how the size of problem instances are reduced after each of the three steps:

	Original size	Size after step 1	Size after step 2	Size after step 3
test1.txt	24	24	10	5
test2.txt	24	0	0	0
test3.txt	24	24	13	9
test4.txt	614400	614400	14732	4991
text5.txt	2400	1954	301	180

After the pre-processing, removing IP-dominated and LP-dominated terms, we can observe a significant reduction in the size of all problem instances. The effect is especially significant for larger instances, for example, in 'text4.txt', the size has been diminished by more than a hundred times. Particularly, we can see substantial decreases in the size of the problem instances after step 2 and step 3, indicating that removing IP-dominated and LP-dominated terms are efficient ways to reduce instance sizes. In conclusion, these steps are crucial for optimizing the problem-solving process.

3 Linear program and greedy algorithm

3.1 Question 6

We are now proposing a greedy algorithm to find a solution to the LP problem. The main idea is to always select the pair $(p_{k,m,n}, r_{k,m,n})$ with the highest incremental efficiency from the set of all possible pairs. The intuitive reason behind this approach is that higher incremental efficiency implies more data rate can be gained with a unit increase in power.

After the pre-processing steps, we have reduced the input instance size and sorted all possible pairs of each channel in ascending order of transmit power $p_{k,m,n}$, but also in descending order of their incremental efficiency, according to Question 4. We start by allocating each channel its minimum possible transmit power to ensure that no channel remains unallocated. After the procedure described in Question 2, such initial allocation will not exceed the total transmit power budget.

Next, we ameliorate our solution incrementally in a greedy way. In a given state of our algorithm, we have for all channels the current selected pair, and each channel has the possibility to increase its investment by moving to the next pair with immediate higher transmit power. Since we aim at a sum data rate as high as possible, we will always choose the current best price–performance ratio, which is, the pair with highest incremental efficiency.

We track the current next pair of each channel by using a priority queue, where pairs are sorted descendingly by their incremental efficiency so that the top element represents the pair with the highest incremental efficiency across all channels. We continue selecting the best pair and adding it to the solution as long as we haven't exceeded the power budget. If, after adding the last pair, there is exactly no remaining power, we have obtained an integral solution.

However, if the remaining power is insufficient to accommodate the currently best pair, we will introduce fractional variables. That is, we drain the remaining budget by a linear combination of the two pairs in order to obtain the highest sum data rate possible.

We note the pair with highest incremental efficiency and the current pair of the corresponding channel respectively as $(p_{k',m',n}, r_{k',m',n})$ and $(p_{k,m,n}, r_{k,m,n})$, we also note the remaining budget as R. The fractional variables can then be calculated as follows, we note that there are at most two fractional variables of the same channel n:

$$\begin{cases} x_{k,m,n} + x_{k',m',n} = 1 \\ x_{k,m,n} \times p_{k,m,n} + x_{k',m',n} \times p_{k',m',n} = p_{k,m,n} + R \end{cases}$$

The pseudo-code is presented below, we note the first element of an array L by $L[0]$, and we refer to the transmit power and data rate of a pair by *pair.power* and *pair.rate*:

Algorithm 5: greedySolveLP($N, L_1, ..., L_N$)

Input: The number of channels N and their pairs in sorted arrays $L_1, ..., L_N$
Output: Feasible greedy solution of the LP Problem, allowing at most two variables of the same channel to be fractional

```
remainPower ← p
nextPairs ← EmptyPriorityQueue
Solution ← EmptySolution
for i ← 1 to N do
    remainPower ← remainPower − L_i[0].power
    nextPairs.push(L_i[0])
    Update Solution by L_i[0]
    Remove L_i[0]
end
while not nextPairs.empty() do
    nextPair ← nextPairs.top()
    nextPairs.pop()
    i ← the channel number corresponding to nextPair
    if remainPower ≥ nextPair.power then
        Update Solution by nextPair
        if remainPower = 0 then
            break                                   // Power budget fully utilized
        end
        if L_i is not empty then
            remainPower ← remainPower − nextPair.power
            nextPairs.push(L_i[0])
            Remove L_i[0]
        end
    end
    else
        Update Solution by fractional variables     // Power budget insufficient
        break
    end
end
return Solution
```

Each channel $n \in \mathcal{N}$ has a number of $O(KM)$ pairs, so the algorithm will terminate with at most $O(NKM)$ pair selection. In a single selection, there are at most $O(N)$ pairs stored in the priority queue, and we will both delete the pair on top and add a new pair. By using a reasonable implementation of the priority queue, such as a heap, each operation will take time $O(\log(N))$. So the overall time complexity of this greedy algorithm is $O(NKM \log(N))$.

3.2 Question 7

We test our greedy algorithm on the test files and compare the result to that of a LP-solver. Not that there are cases where we actually get an integer solution, and the gap between used power and total power budget becomes bigger when the problem size grows:

	Optimal rate	Used power	Total power
test1.txt	386.1538	96.0000	100.0000
test2.txt	No solution	No solution	No solution
test3.txt	368.0000	94.0000	100.0000
test4.txt	8543.0973	11644.0000	16000.000000
text5.txt	1390.0000	751.0000	1000.0000

	CPU runtime (ms)	Optimal rate	Used power
Greedy algorithm	29	1390.0000	751.0000
LP solver	1689	1637.0000	1000.0000

For "test5.txt", the comparison is shown as above, note that the LP solver manages to use up all the power budget.

4 Algorithms for solving the ILP

4.1 Question 8

The idea of Dynamic Programming (DP) is to break a complex problem down into simpler sub-problems recursively. Let us assume that we plan to calculate the maximum sum data rate for the first n channels under a total transmit power budget p. If we allocate the n^{th} channel with $(p_{k,m,n}, r_{k,m,n})$, the remaining $n-1$ channels must be allocated optimally within the remaining budget of $p - p_{k,m,n}$. We denote $f_{n,p}$ as the maximum sum data rate achievable with the first n channels under the total transmit power budget p. The dynamic programming equation for this problem can be formulated as follows:

$$f_{n,p} = \max_{k,m}\{f_{n-1,p-p_{k,m,n}} + r_{k,m,n}\}$$

Here, $f_{n-1,p-p_{k,m,n}}$ represents the maximum sum data rate achievable with the remaining $n-1$ channels after reducing the budget by $p_{k,m,n}$. We can seek the optimal allocation that maximizes the total transmit data rate by considering all possible pairs of user $k \in \mathcal{K}$ and power level $m \in [1, M]$. By doing this way, we have broken the problem of the first n channels into sub-problems of the first $n-1$ channels. Below shows the pseudo-code of our algorithm:

Algorithm 6: DPSolveILP($N, L_1, ..., L_N$)

Input: The number of channels N and their pairs in sorted arrays $L_1, ..., L_N$
Output: Optimal DP solution of the ILP Problem

```
Initialize the whole matrix f_{n,p} to zero
foreach (k, m, n) ∈ L_1 do
    for i ← p_{k,m,1} to p do
        f_{1,i} ← r_{k,m,1}
    end
end
for i ← 2 to N do
    for j ← 1 to p do
        foreach (k, m, i) ∈ L_i do
            if j − p_{k,m,i} ≥ 0 then
                f_{i,j} ← max(f_{i−1,j−p_{k,m,i}} + r_{k,m,i}, f_{i,j})
            end
        end
    end
end
Calculate Solution by revisiting the matrix f_{n,p}
return Solution
```

Initially, we fill the first row of the matrix $f_{n,p}$, corresponding to the scenario where only the first channel is available. In this specific case, the highest data rate achievable within budget p is the data rate of the pair with immediate smaller transmit power than p in the array L_1. The entirety of all necessary information can be extracted from a single traversal of the sorted array L_1.

In order to fill in one element of the matrix $f_{n,p}$ of size $N \times p$, we should traverse an array of size $O(KM)$, the calculation is thus of time complexity $O(NpKM)$. Since the initialization and the calculation of solution do not exceed that complexity, the overall time complexity is also $O(NpKM)$. The matrix $f_{n,p}$ takes up $\Theta(Np)$ space, but we can also improve that to $\Theta(p)$ by using a rolling array. Since the input is already of size $\Theta(NKM)$, this improve may not appear to be so much significant.

4.2 Question 9

Another alternative DP approach involves considering subproblems of finding minimal power allocations providing a given sum data rate r. We denote the minimum power output achieving the sum data rate r with the first n channels by $f_{n,r}$. The dynamic programming equation for this problem can be formulated as:

$$f_{n,r} = \min_{k,m}\{f_{n-1,r-r_{k,m,n}} + p_{k,m,n}\}$$

Similar to Question 8, we should fill in a matrix of size $N \times U$, where U represents the upper bound of data rate that can be achieved. Such an upper bound can be calculated by adding up the maximum possible data rate of each channel. Below shows the pseudo-code of our algorithm:

Algorithm 7: anotherDPSolveILP($N, L_1, ..., L_N$)

Input: The number of channels N and their pairs in sorted arrays $L_1, ..., L_N$
Output: Optimal DP solution of the ILP Problem

```
U ← 0
for i ← 1 to N do
    U ← U+ the maximal data rate in L_i
end
Initialize the whole matrix f_{n,r} to zero
foreach (k, m, n) ∈ reverse(L_1) do
    for i ← 1 to r_{k,m,1} do
        f_{1,i} ← p_{k,m,1}
    end
end
for i ← 1 to N do
    for j ← 1 to U do
        foreach (k, m, i) ∈ L_i do
            if j − r_{k,m,i} ≥ 0 then
                f_{i,j} = min(f_{i−1,j−r_{k,m,i}} + p_{k,m,i}, f_{i,j})
            end
        end
    end
end
Calculate Solution by revisiting the matrix f_{n,r}
return Solution
```

Similarly, the time complexity is $O(NUKM)$, because the calculation of U, the initialization and the calculation of solution should not take up time more than $O(NUKM)$. The space taken up by the matrix is $\Theta(NU)$, and can be improved to $\Theta(U)$. The input space remains $\Theta(NKM)$.

4.3 Question 10

The root of the tree corresponds to the initial ILP problem, which involves allocating a budget p among N channels. Each node v in the tree is a subproblem characterized by (N_v, p_v), which means we consider the first N_v channels under the budget p_v. Every time we select a new node for branching, we make a choice on one channel, consequently each node v has at most KM child nodes, where a child node $(N_v - 1, p_v - p_{k,m,N_v})$ of v represents the situation where we have allocated the user k with power level m on the channel N_v, so the subproblem is to consider the first $N_v - 1$ channels under the budget $p_v - p_{k,m,N_v}$.

In order to have an estimate of the upper bound of the optimal value can be attained by the current branch, we relax the integer restriction and run the LP problem on our LP solver to get an optimal value, denoted by $\overline{z}$. Here we have 3 cases : (i) $\overline{z} \leq currentFeasibleSolution$. The integer solution of this subproblem is smaller than $\overline{z}$, indicating that it cannot be better than the $currentFeasibleSolution$ we have found so far. In this case, we simply cut this branch. (ii) $\overline{z}$ is associated an integer solution, so the upper bound $\overline{z}$ can actually be reached and it represents a better feasible solution according to (i). In this case, we update $currentFeasibleSolution$ with this solution and stop searching on this branch. (iii) $\overline{z} > currentFeasibleSolution$. We cannot

determine if the current branch has a better solution than *currentFeasibleSolution*. In this case, we continue our search in order to collect more clues on the ILP problem.

Note that we will always need to compare $\overline{z}$ with the current best feasible solution, in order to get a feasible solution as quick as possible, we adopt a Depth-First-Search (DFS) strategy in this framework. That is, nodes are stored in a stack. Below shows the pseudo-code for our Branch-and-Bound algorithm, where *currentFeasibleSolution* is defined globally:

Algorithm 8: branchAndBound($N, L_1, ..., L_N$)

Input: The number of channels N and their pairs in sorted arrays $L_1, ..., L_N$
Output: Feasible BB solution of the ILP Problem

```
1  if N = 0 then
2  |  Update currentFeasibleSolution with the current solution
3  |  return
4  end
5  LPResult ← LPSolver()
6  if LPResult <= currentFeasibleSolution then
7  |  return
8  end
9  if LPResult is an integer solution then
10 |  Update currentFeasibleSolution with the current solution
11 |  return
12 end
13 foreach possible choice of channel N do
14 |  branchAndBound(N − 1, L_1, ..., L_{N−1})
15 end
16 return Solution
```

As we get no guarantee that pruning will ever happen (or not so much), our Branch-and-Bound algorithm can be viewed in worst case as a brute-force one, thus have a complexity of $O((KM)^N f(K, M, N))$, where f represents the time complexity of our LP solver. By Question 11, we can see that the actual performance appears much better than the exponential complexity.

4.4 Question 11

We test and compare our Dynamic Programming and Branch and Bound algorithm on the test files:

For "test5.txt", the comparison is shown as above.

5 Stochastic online scheduling

5.1 Question 12

In this scenario, users arrive sequentially in the system, and each channel must be allocated to a user. The objective is to maximize the total data rate under a total power budget p. Inspired by the Greedy algorithm, we aim to select pairs with the highest ratio of $\frac{r_{k,m,n}}{p_{k,m,n}}$. At the same time, we have to ensure that the total power budget is not exceeded, and all channels are allocated by the end of the process.

As the distribution set of powers and rates are known, we can conclude that the expectation of the ratio $\mathbb{E}(\frac{r_{k,m,n}}{p_{k,m,n}}) = \frac{r^{max}}{p^{max}}$. Here's a detailed description of the algorithm: (i) When a user $k < K-1$ arrive and when there are more than 1 channels available, we calculate the ratio $\frac{r_{k,m,n}}{p_{k,m,n}}$ for all pairs with $p < remainPower$ in available channels. For the pair $(p_{k,m_k,n_k}, r_{k,m_k,n_k})$ with the highest ratio, if this ratio is smaller than the expectation, we leave this user not served. If this ratio is greater than the expectation, the channel n_k provides p_{k,m_k,n_k} power to the user k and the channel n_k is occupied. (ii) When user K arrived and there are still several unallocated channels, we iteratively allocate pairs with power smaller than the remaining power and the highest ratio to the corresponding channels. (iii) When only one channel is left unallocated, we allocate all the remaining power to this channel.

5.2 Question 13

We generate 100 ILP problem instances with the given parameter and distribution and run our online and offline algorithms. The average power budgets used by the two algorithms are shown as below: