USER SCHEDULING OVER MULTIPLE CHANNELS

Mohamed Malhou, Benbaki Riade
Supervisor: Marceau Coupechoux

January 2020

1 Introduction

An antenna transmits data packets to smartphones (or users) through a wireless medium, which is divided into a set of frequency channels. Figure 1 is an example of simultaneous transmission towards three users using three channels.

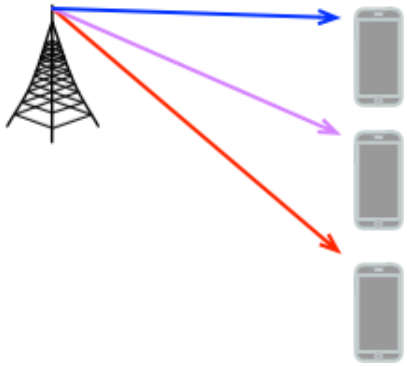

Figure 1: Wireless packet scheduler transmitting data simultaneously to three users over three (frequency) channels.

The higher the power dedicated to a user, the higher the data rate it can experience. The exact dependence between power and data rate is however user and channel specific. With the same transmit power, a user close to the antenna will enjoy for example a higher data rate than a user far away. A wireless packet scheduler is thus responsible to allocated channels to users and to divide the total power budget of the antenna among the available channels. The goal of this project is to design optimal packet schedulers in this context.

2 Problem formulation

We consider a system with a set $\mathcal{K}$ of K users to be served over a set $\mathcal{N}$ of N channels. Every channel shall be used to serve a single user and cannot be left unallocated for efficiency reasons; a user can be served using several channels; however some users may not be served. The scheduler chooses a transmit power $p_{k,n}, k \in K, n \in N$ to serve user k on channel n. If user k is not served on channel n, we have $p_{k,n} = 0$. When user k is served over channel n with power $p_{k,n}$, its data rate is $r_{k,n} = u_{k,n}(p_{k,n})$, where the function $u_{k,n}$ is called the rate utility function of user k on channel n. This utility function is assumed to be known by the scheduler. In practical systems, $u_{k,n}$ is a non-decreasing step function that takes a finite number of non-zero values, say M for all k and n and such that $u_{k,n}(0) = 0$ for all k and n. To fix notations, we thus define:

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

The task of the scheduler is to allocate channels to users and transmit powers to users so as to maximise the sum data rate of the system under the constraint of a total transmit power budget p and the constraint of having exactly one user served per channel. For simplicity, we assume that all coefficients $r_{k,m,n}, p_{k,m,n}$ and p are non-negative integers.

To formulate the problem as an integer linear program (ILP), we aim to maximise $\sum_{k,m,n} x_{k,m,n} r_{k,m,n}$ subject to

1. $\sum_{k,m,n} x_{k,m,n} p_{k,m,n} \leq p$
2. $\sum_{k,m} x_{k,m,n} = 1$
3. $x_{k,m,n} \in \mathbb{N}$

for all $0 \leq k \leq K, 0 \leq m \leq M$ and $0 \leq n \leq N$.

This is a multiple-choice knapsack problem (MCKP) which is a generalization of the ordinary knapsack problem, where the set of items is partitioned into classes. The binary choice of taking an item is replaced by the selection of exactly one item out of each class of items[1].
This ILP is known to be NP hard. By relaxing the integrality constraint on $x_{k,m,n}$, we obtain a linear program (LP), which provides an upper bound for our problem. If the solution to the LP is integer, then we have a solution for the ILP. In the following sections, we first make some preprocessing to reduce problem instance size, then solve the LP and the IP, and at last consider an online version of the problem.

3 Preprocessing

3.1 First preprocessing step

We want to make a quick preprocessing to check if an instance has obviously no solution and to remove triplets (k, m, n) that, if chosen, obviously prevent any solution to be feasible.
A triplet (k,m,n) is obviously not a part from any solution if $p_{k,m,n} + \sum_{n' \neq n} p_{min,n} > P$, $p_{min,n}$ being the minimum power among all terms in channel n. If after this preprocessing there is a channel with no terms remaining, then the problem has no solution.

3.2 Removing IP-dominated terms

Lemma 1 *For a given channel n, if $p_{k,m,n} \leq p_{k',m',n}$ and $r_{k,m,n} \geq r_{k',m',n}$ then there is an optimal solution of the ILP such that $x_{k',m',n} = 0$. We say that (k', m', n) is IP-dominated*

Based on Lemma 1, we provide an algorithm to remove IP-dominated terms of an instance of the IP problem.
We sort the terms by their power values for each channel (and by decreasing rate in case of equal power), then as we go through the sorted array, we compare the rate to the maximum rate of previous terms, and we update this maximum.

Algorithm 1 Remove IP-dominated terms

```
for n = 0 to N − 1 do
  channels[n].sortbypower()
  maxrate = channels[n][0].rate
  for i = 0 to channels[n].size() do
    currterm = channels[n][i]
    if currterm.rate <= maxrate then
      remove(currterm)
    end if
    maxrate = max(maxrate, currterm.rate)
  end for
end for
```

For each channel, sorting costs $O(KMlog(KM))$, then we go through the terms of each channel, which takes linear time. This gives a total complexity of $O(NKMlog(KM))$

3.3 Removing LP-dominated terms

Lemma 2 *For a given channel n, if* $p_{k,m,n} < p_{k',m',n} < p_{k",m",n}$ *and* $r_{k,m,n} < r_{k',m',n} < r_{k",m",n}$ *satisfy:*

$$\frac{r_{k",m",n} - r_{k',m',n}}{p_{k",m",n} - p_{k',m',n}} \geq \frac{r_{k',m',n} - r_{k,m,n}}{p_{k',m',n} - p_{k,m,n}} \tag{2}$$

then there is an optimal solution of the LP such that $x_{k',m',n} = 0$. *We say that* $(k^{'}, m^{'}, n)$ *is LP-dominated*

LP dominated terms are in this case terms that don't belong to the convex hull of the set of all points. In the following algorithm, we determine this convex hull by going through all the terms by order of power, and keeping a stack that contains, after each iteration of the while loop, the convex hull of the already processed points. When processing a new point, we know that this point belongs to the new convex hull, but we first need to remove points that will no longer be in it. These points are necessarily on top of the stack, so we keep testing the top of the stack until all these points are removed, then we add the new point.

Algorithm 2 Remove LP-dominated terms

```
for n = 0 to N − 1 do
  channels[n].sortbypower()
  convexhull = [ channels[0] ]
  for i = 1 to channels[n].size() do
    currterm = channels[n][i]
    while currterm is at the right of the line (convexhull[-2],convexhull[-1]) do
      convexhull.pop()
    end while
    convexhull.push(currterm)
  end for
  channels[n] = convexhull
end for
```

For each channel, each term is pushed in the stack once, and poped out at most one, so the total complexity, for each channel, is $O(KMLog(KM) + 2KM) = O(KMLog(KM))$, therefore the total complexity is $O(NKMLog(KM))$ or $O(NKM)$ if already sorted by power in the previous procedures.

3.4 Applying the preprocessing

We apply the three steps pre-processing on instances of test files 'test1.txt', 'test2.txt' and 'test3.txt'.

Preprocessing step/TestfFile	1	2	3	4	5
Before preprocessing	24	24	24	614400	2400
After removing impossible terms	24	0	24	614400	2400
After removing IP-dominated terms	10	N.A	13	14687	329
After removing LP-dominated terms	8	N.A	9	4982	193

Figure 2: Total number of pairs remaining in the problem after each step of the preprocessing

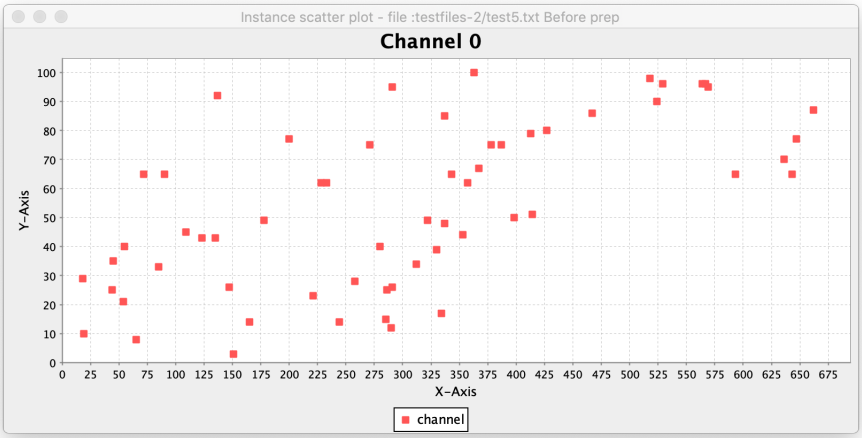

Figure 3: Pairs in channel 0 of file 5 before preprocessing

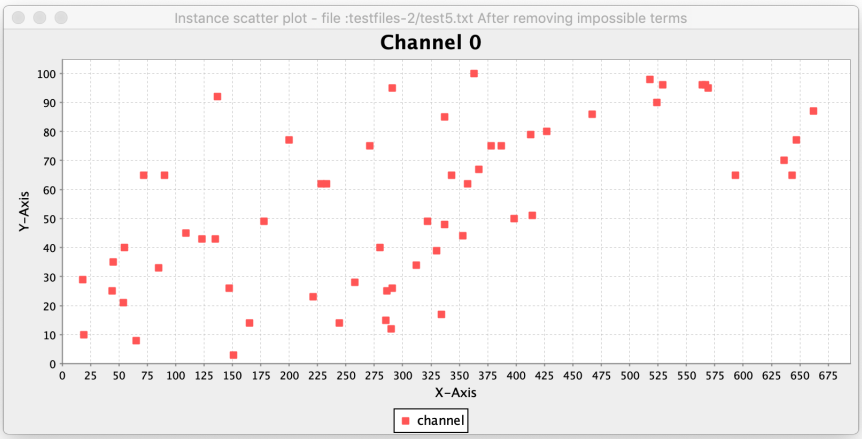

Figure 4: Pairs in the first channel of file 5 after removing impossible terms

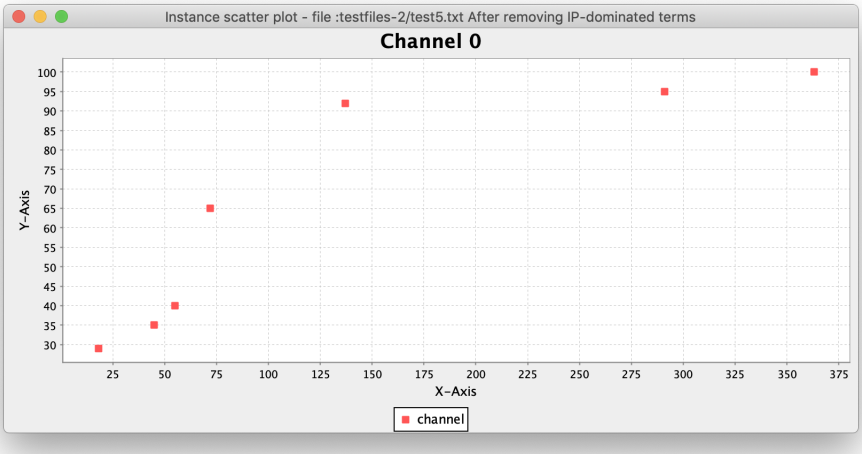

Figure 5: Pairs in the first channel of file 5 after removing IP-dominated terms

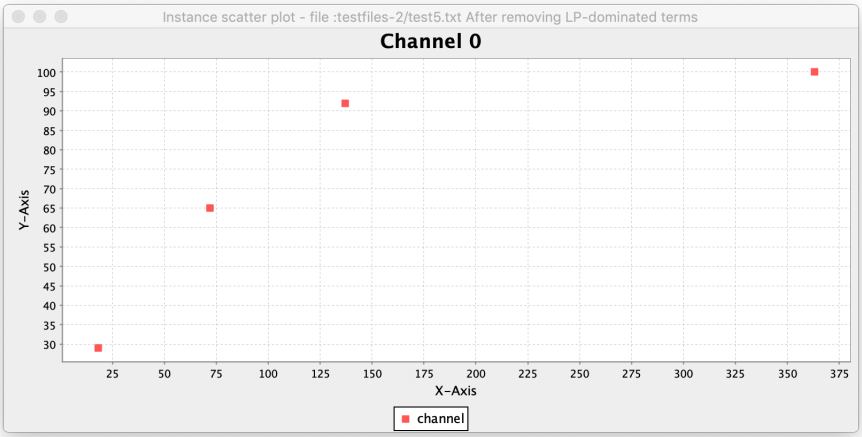

Figure 6: Pairs in the first channel of file 5 after removing LP-dominated terms

4 Linear Program and Greedy Algorithm

Now that the instance size has been reduced, we study a greedy algorithm for the LP problem. For a given channel n, all possible pairs $(p_{k,m,n}, r_{k,m,n})$ are sorted in ascending order of $p_{k,m,n}$ and reindexed with the set $\mathcal{L} = \{1, ..., L\}, L = KM$. We define the incremental efficiency of choosing pair $l > 1$ instead of pair $l1$ for the transmission on channel n as follows:

$$e_{ln} = \frac{r_{l,n} - r_{l-1,n}}{p_{l,n} - p_{l-1,n}} \tag{3}$$

Based on this notion, we propose a greedy algorithm to provide a solution to the LP problem,

4.1 The pseudo-code of the greedy algorithm

Algorithm 3 Greedy Algorithm to solve the LP problem

Input List of all pairs, preprocessed and with access to indexes of each pair
Output Maximum rate achievable for the LP problem instance
sortedbyeff = SORT ALL PAIRS BY INC EFF DESC
power budget = P - $\sum_{i=1}^{n} p_{1,i}$ //budget remaining to fill
Set $x_{1,i} = 1 \forall i \in \{1, ..., n\}$ //we use the same indexing for x variables as for pairs
$rate = \sum_{i=1}^{n} r_{1,i}$
i, j =sortedbyeff.pop().indexes
while power budget >= $p_{i,j} - p_{i,j-1}$ **do**
  $x_{i,j} = 1, x_{i,j-1} = 0$
  power budget $-= p_{i,j} - p_{i,j-1}$
  $rate+ = r_{i,j} - r_{i,j-1}$
  i, j = sortedbyeff.pop().indexes
end while
if power budget == 0 **then**
  return $rate$ //We have an integral solution !
else
  $x = \frac{powerbudget}{p_{i,j} - p_{i,j-1}}$
  $x_{i,j} = x, x_{i,j-1} = 1 - x$
  rate $+= (r_{i,j} - r_{i,j-1}) * x$
  power budget $-= (p_{i,j} - p_{i,j-1}) * x$
  return $rate$
end if

Sorting and the beginnig of the algorithm takes $O(KMNlog(KMN))$, after that, we perform a linear traversal of the list, each loop iteration having a constant cost. The overall complexity is therefore $O(KMNlog(KMN))$

4.2 Results of the greedy algorithm

TestfFile	1	2	3	4	5
Greedy	365.0	N.A	372.15384615384613	9870.32183908046	1637.0
LPsolver	365.0	N.A	372.15384615384613	9870.32183908046	1637.0

Figure 7: Maximum rates achieved

TestfFile	4	5
Preprocessing	168.98987497	0.62586537
Greedy (on preprocessed instance)	2.08267759	0.03906349
LPsolver (on preprocessed instance)	386.13193925	5.14502407
LPsolver (without preprocessing,over 1 execution)	35256	29

Figure 8: CPU runtime in ms, averaged over 100 execution

5 Algorithms for solving the ILP

As the relaxed solution cannot be implemented in practice (this would require the possibility to schedule two users on the same channel), we tackle in this section the ILP problem (when the power budget is an integer). Unfortunately, the greedy algorithm can be arbitrarily bad in this case. Although there is an improved greedy algorithm for this problem, this is only a 1/2-approximation. We are looking here for an optimal solution by first relying on Dynamic Programming (DP).

5.1 Dynamic Programming Solution

Let's tackle a DP algorithm based on sub-problems with power budget less than p.
The equation we are using is given as follows:

$$R(n,p) = \max_{\substack{pair \in channel_n \\ pair.p \leq p}} R(n-1, p-pair.p) + pair.r \tag{4}$$

Time complexity : it takes at most $O(KM)$ to find the maximum of the equation (1), so, regarding the two main loops of our algorithm, we have a time complexity of $O(PNKM)$.
space complexity : $O(P)$ as we use two arrays of length P.

Algorithm 4 Dynamic Programming algorithm to solve IP

Input p power budget, data, an array such that data[n] is an array of all pairs of channel n
Output maximum rate value $\sum_{k,m,n} x_{k,m,n} r_{k,m,n}$
1 channel case :
Let L an array of length p
for $i = 1$ to p **do**
  L[i-1] = max(pair.r such $pair.p \leq i$ and pair $\in$ data[0])
end for
for $n = 1$ to N **do**
  Let currentChannel = data[n]
  Let aux an auxiliary list of length p
  for $power = 1$ to p **do**
    aux[power-1] = max(L[power− pair.p −1] + pair.r, with $pair.p \leq$ power ,L[power−$pair.p - 1$]> 0) for pair $\in$ currentChannel
  end for
  L = aux
end for
return L[-1]

5.2 A second dynamic programming approach

An alternative DP approach is to consider sub-problems of finding minimal power allocations providing a given sum data rate r less than some upper bound U for the objective function. The equation of finding minimal power allocations is :

$$P(n, U) = \min_{\substack{pair \in channel_n \\ pair.r \leq U}} P(n-1, U - pair.r) + pair.p \tag{5}$$

Algorithm 5 Another DP algorithm to solve IP, based on finding minimal power allocations

Input U upper bound for rate, data.
Output maximum rate value $\sum_{k,m,n} x_{k,m,n} r_{k,m,n}$
reversing the function u_1 :
Let L an array of length U
for $i = 1$ to U **do**
  L[i-1] = min(pair.p such $pair.r = i$ and pair $\in$ data[0]) #0 otherwise
end for
for $n = 1$ to N **do**
  Let currentChannel = data[n]
  Let aux an auxiliary list of length U
  for $rate = 1$ to U **do**
    aux[rate-1] = min(L[rate− pair.r −1] + pair.p, with $pair.r \leq$ rate ,L[rate−$pair.r - 1$]> 0) for pair $\in$ currentChannel
  end for
  L = aux
end for
return maximum index r such that L[r-1] <= p & L[r-1] > 0

time complexity : O(NUKM) and space complexity : O(U)

5.3 Branch And Bound Solution

Another classical way of solving IPs is called Branch-and-Bound (BB). The principle of BB is as follows. We construct a tree of sub-problems, whose root corresponds to the initial problem. Each vertex v corresponds to a sub-problem, which is generated from its parent in the tree by adding an additional constraint. At node v (a branch of the tree), the relaxed sub-problem is solved in order to get an upper

bound $\bar{z}_v$ of the optimal value for the sub-problem. This branch is not further explored if (1) $\bar{z}_v$ is less than a current feasible solution we have already or (2) $\bar{z}_v$ is associated to an integer solution, in which case we can update the current feasible solution or (3) the relaxed sub-problem is infeasible.

Each level of the tree represents a channel, and each node of each level represents a feasible pair choice in that channel.So each path from the source to a node in level k represents a possible configuration where channels from 1 to k are assigned to a pair. At each node, we use the relaxed problem to find an upper bound for the problem, using the greedy algorithm. The greedy algorithm also gives us a feasible solution, so we update the lower bound each time this feasible solution is better than the one we already have. The worst case complexity is $O(KM * (KM)^N)$ because it's possible to go through all the nodes of the tree, but in most cases, several branches will be eliminated, thus giving a lower amortised complexity. The fact that we use a stack or a queue in the algorithm changes the traversal order of the tree : DFS using a stack, BFS using a queue. In this case, the later runtime results show that using a DFS is faster.

Algorithm 6 Auxiliary function GreedyBound() returning upper and lower bounds of a branch in the Branch and Bound algorithm

Input Current channel, left power budget and a sorted list of all pairs by INC EFF DESC.
Output Lower bound and upper bound performed by the greedy algorithm
Let $Rate = 0$
Let index $i = 0$ and current pair $currPair$
while $i <$ listOfPairs.size() and listOfPairs.get(i).$p <$ PowerBud **do**
  currPair = listOfPairs.get(i)
  if currPair.n >= currChannel **then**
    PowerBud $-=$ currPair.incPower;
    Rate $+=$ currPair.incRate;
  end if
  i++
end while
Let LowerBound = Rate
if PowerBud > 0 and $i <$ listOfPairs.size() **then**
  currPair = listOfPairs.get(i)
  Let x = PowerBud/currPair.incPower;
  Rate $+= x\times$currPair.incRate
  PowerBud $-= x\times$ currPair.incPower
end if
return Rate and LowerBound

Algorithm 7 Branch and Bound Algorithm

```
sortedInc = SORT ALL PAIRS BY INC EFF DESC
Let Stack a stack< vertex >
Stack.push(new vertex(0,0,0)) // args : current Channel, left power and achieved rate
currBounds = GreedyBound(0,0,sortedInc)
while Stack.size()> 0 do
  vertex = Stack.get()
  for all pair ∈ data[vertex.currChannel] do
    if pair.p + vertex.usedPower > powerBud then
      continue
    end if
    if vertex.currChannel < N-1 then
      Let BranchBound = GreedyBound(vertex.currChannel+1, powerBud − vertex.usedPower −
      pair.p,sortedInc)
      if BranchBound.UBound + vertex.rateAchieved + pair.r > currBounds.LBound then
        Stack.push(new vertex(vertex.currChannel+1,vertex.usedPower + pair.p, ver-
        tex.rateAchieved + pair.r))
      end if
      currBounds.LBound = max(currBounds.LBound, vertex.rateAchieved + pair.r + Branch-
      Bound.UBound)
    else
      currBounds.LBound = max(currBounds.LBound, vertex.rateAchieved + pair.r )
    end if
  end for
end while
return currBounds.LBound
```

5.4 Results of IP algorithms

Test File	1	2	3	4	5
DP1	365	N.A	350	9870	1637
DP2	365	N.A	350	9870	1637
BB	365	N.A	350	TLE	1637

Figure 9: Maximum rates achieved

Test File	1	2	3	4	5
DP1	1.493625	N.A	0.506143	5722.798771	9.421062
DP2	0.038489999999999996	N.A	0.042031	4539.203344	7.3717749999999995
BB (doing a DFS)	0.004246	N.A	0.012026	TLE	0.07812899999999999
BB (doing a BFS)	0.21529399999999999	N.A	0.271411	TLE	0.442135

Figure 10: CPU Runtime in ms averaged over 100 execution

6 Stochastic Online Scheduling

In this section, users arrive sequentially in the system. The scheduler is no longer aware of the whole instance before taking a decision and has to take a decision each time a new user is coming. To be more precise, we assume that the scheduler is aware of the number of users K that will arrive in the system. At time $t = k$, user k arrives in the system providing to the scheduler all the pairs $(p_{k,m,n}, r_{k,m,n})$, $m = 1, ..., M, n = 1, ..., N$. All pairs indexed by $k' > k$ are unknown. At this time instant, the scheduler must assign the variables $x_{k,m,n}$, $m = 1, ..., M, n = 1, ..., N$ without being able to modify them in the sequel, i.e., at $t > k$. We assume that powers are independent and identically distributed with uniform discrete distribution on the set $\{1, 2, ..., p^{max}\}$; rates are independent and identically distributed with

uniform discrete distribution on the set $\{1, 2, ..., r^{max}\}$, where p^{max} and r^{max} are positive integers. These distributions are supposed to be known by the scheduler.

Due to lack of knowledge, optimum solution seems to be unapproachable. Thus we aim at proposing an approximation of a greedy algorithm that only looks at maximum efficiencies. The first idea that comes to mind is that in each iteration, we select all pairs with efficiency exceeding our expectation of the last pair to be chosen among the left pairs not generated yet.
In a formal way, let Th_k be the threshold of the k-th iteration and N_k the residual channels, we have :

$$Th_k = \mathbb{E}(X_{N_k:(K-k)NM}|N_k) \tag{6}$$

where $X_t = \frac{R_t}{P_t}$, $(R_t)_{t\in\mathbb{N}}$ and $(P_t)_{t\in\mathbb{N}}$ uniformly distributed random variables.
With the convention $X_{1:n} >= ... >= X_{n:n}$
The constraint is that there is no easy formula for calculating these expectations, and simulating the variables in order to use LLN to find an approximation would require a lot of processing. we would rather use a deterministic way trying to be fair to all the users, that is, we give each user a fair power budget : $\frac{p}{K}$ to use efficiently without restricting the number of channels to use. we get Algorithm 8

Algorithm 8 Online algorithm

```
Input pmax, rmax, p, M, N, K
Output achieved rate
Let L a list[N] such that L[i] == 1 if channel i is taken
Let powerPerUser p/K
Let Rate = 0
for all k = 0 to K do
    data = generate(NM pairs )
    sort data by efficiency ratio r/p
    Let up = 0 be the used power by user k
    Let i = 0
    for all pair ∈ data do
        if L[pair.n] == 0 and up +pair.p <=PowerPerUser then
            L[pair.n] = 1
            Rate + = pair.r
            p − = pair.p
            up + = pair.p
        end if
    end for
    up = 0
end for
return Rate
```

Using the parameters $p = 100, p^{max} = 50, r^{max} = 100, M = 2, N = 4$ and $K = 10$, over 10000 samples, we get a an average ratio of 0.48 and used power budget of 78.23 versus 82.8 for optimum solution.

References

[1] Ulrich Pferschy Hans Kellerer and David Pisinger. *Knapsack Problems*. Springer-Verlag Berlin Heidelberg, 2004. ISBN: 978-3-642-07311-3.