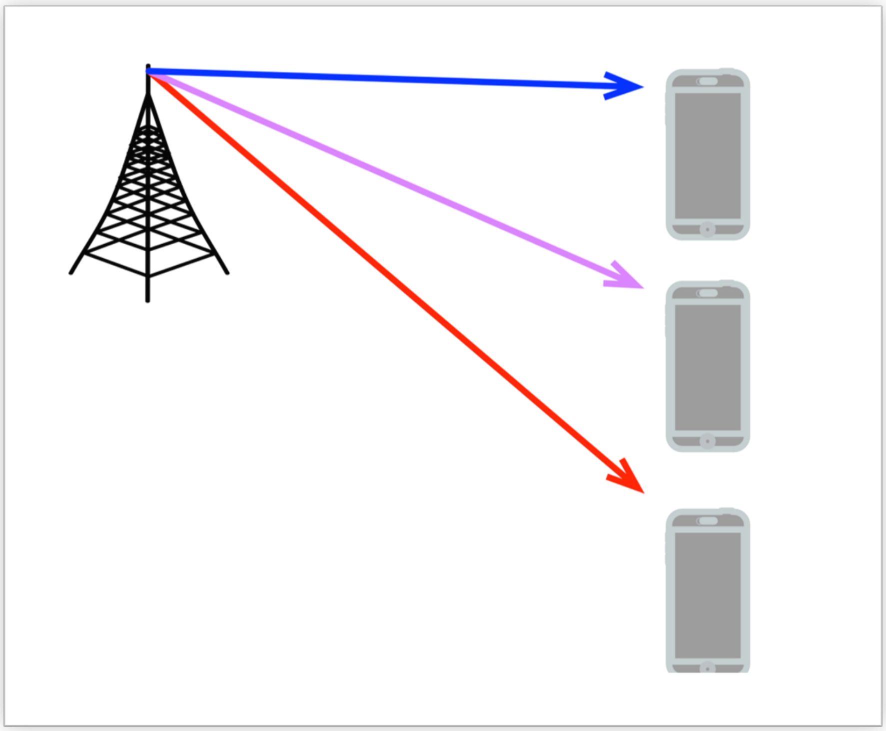
USER SCHEDULING IN 5G

Subject proposed by Marceau Coupechoux



Direct link to the subject and test files:
https://marceaucoupechoux.wp.imt.fr/enseignement/english-inf421-pi/

Théo Barré & Virgile Foussereau | INF421 | 1er février 2022 | École Polytechnique

Formulating the problem as an integer linear program

We want to solve the following problem:

$$\underset{x_{k,m,n} \in \{0,1\}^{KMN}}{\text{Max}} \sum_{\substack{1\le k\le K \\ 1\le m\le M \\ 1\le n\le N}} x_{k,m,n} r_{k,m,n}$$

$$\text{Such that : } \forall\, n, \sum_{\substack{1\le k\le K \\ 1\le m\le M}} x_{k,m,n} = 1$$

$$\text{and } \sum_{\substack{1\le k\le K \\ 1\le m\le M \\ 1\le n\le N}} x_{k,m,n} p_{k,m,n} \le p$$

Preprocessing

Quick preprocessing

A triplet (k,m,n) is unfeasible if $p_{k,m,n} + \sum_{n' \neq n} \min_{k,m} p_{k,m,n} > p$, so we compute the terms $\min_{k,m} p_{k,m,n}$ for each $n \in [\![1, N]\!]$, and then go through all triplets (k,m,n) to check if they are feasible. Those which aren't are deleted of the dataset.

Algorithm 1 – Unfeasible triplets

Input: (N,K,M,pmax, PR) where PR[n] = [($p_{k,m,n}$, $r_{k,m,n}$, k,m) for (k,m) in $[\![1, K]\!] \times [\![1, M]\!]$]
Output: (N,K,M,pmax, PR) with PR[n] cleared from infeasible (k,m,n) for each n

- Lpnmin = [$\min_{k,m} p_{k,m,n}$ for n in range(N)]
- Smin = $\sum_{1\le n\le N} \min_{k,m} p_{k,m,n}$
- For n in range(N) :
  - For (k,m) in $[\![1, K]\!] \times [\![1, M]\!]$:
    - If $p_{k,m,n} + Smin - Lpnmin[n] > pmax$: delete (n,k,m) from PR
- Return PR

Removing IP-dominated terms

A term (k,m,n) is IP-dominated is its rate is inferior to the highest rate amongst terms with power inferior to $p_{k,m,n}$. Therefore we just have to go across a channel sorted by power, comparing the rate with the maximum rate encountered until now.

Algorithm 2 – Remove IP-dominated terms

Input: (N,K,M,pmax, PR) where PR[n] = [($p_{k,m,n}$, $r_{k,m,n}$, k,m) for (k,m) in $[\![1, K]\!] \times [\![1, M]\!]$ (except unfeasible triplets)]
Output: (N,K,M,pmax, PR) with PR[n] cleared from IP-dominated (k,m,n) for each n

- For n in range(N) :
  - Sort PR[n] by ascending $p_{k,m,n}$
  - rmax = PR[n][1]
  - For u in PR[n][1:] :
    - if u[1] ≤ rmax :
      - delete u from PR[n]
    - else :
      - rmax = u[1]
- return PR

We sort each channel in O(KMlog(KM)) and then look at each element in O(KM).
Therefore the total complexity of the algorithm is O(NKMlog(KM)).

Removing the LP-dominated terms

An LP-dominated triplet is simply a triplet that prevents the utility function from being concave.

Algorithm 3 – Remove LP-dominated terms

Input: An instance PR containing all the triplets for each channel and pmax
Output: Triplets that are not LP-dominated

- for n=0 to N-1:
  - Sort channel n by power
  - Let be L containing only the first element of the sorted channel n
  - for each triplet t in channel n after the first element :
    - while length(L)>1 and $\frac{\text{L}[-1].\text{rate}-\text{L}[-2].\text{rate}}{\text{L}[-1].\text{power}-\text{L}[-2].\text{power}} \leq \frac{t.\text{rate}-\text{L}[-1].\text{rate}}{\text{t.power}-\text{L}[-1].\text{power}}$:
      - L.pop() #we remove former terms that are now LP-dominated
    - L.append(t) #the last triplet has to be in the hull
  - Replace channel n by L in PR
- return PR

For each channel we sort and then a term is added and deleted at most once. This gives a complexity of O(KMlog(KM) + 2KM)= O(KMlog(KM)). Therefore the total complexity is O(NKMlog(KM)).
If algorithms 1, 2 and 3 are always used one after another we can sort each channel only at the beginning of the preprocessing. This would give a complexity of O(NKM) to the algorithm 3.

Results of preprocessing

The following table gives the total number of triplets after each step of preprocessing:

	”test1.txt”	”test2.txt”	”test3.txt”	”test4.txt”	“test5.txt”
Initially	24	24	24	614400	2400
Quick Preprocessing	24	0	24	614400	1954
Remove IP dominated	10	0	13	13242	280
Remove LP dominated	8	0	9	4846	175

Greedy Algorithm to solve the ILP

We start with the configuration where for each n, we give power to the user requiring the minimal power over all k,m. This solution is feasible since the correspondent used power is $\sum_{1\leq n\leq N} \min_{l} p_{l,n} \leq p$. Then, while our used power is inferior to p, and when we are in configuration using $p_{l(n),n}$ for each n, we take the n with the highest ratio $e_{l(n)+1,n}$ compatible with budget p and use $p_{l(n)+1,n}$ instead of $p_{l(n),n}$. When there is no more transition $p_{l(n),n}$ to $p_{l(n)+1,n}$ compatible with a total budget of p, if we achieved budget p exactly then we are done, and we have $x_{l(n),n} = 1$ for all n, and the other $x_{l,n}$= 0. Otherwise, we take the highest ratio $e_{l(n)+1,n}$ (over all n), and share the weight 1 between $x_{l(n),n}$ and $x_{l(n)+1,n}$ with affine combination so that we reach a total budget of p.

Algorithm 3– Greedy algorithm

Input: A list L such that L[n] contains the ($p_{l,n}$, $r_{l,n}$, l) sorted by increasing $p_{l,n}$ (and $p_{l,n}$ increases with l) and a power budget pmax
Output: Greedy rate for the given power budget, allocation $x_{l,n}$

- for n =1 to N :
  - let $x_{1,n}$ = 1 and $x_{l,n}$= 0 for $l \neq 1$
- E = [$e_{2,n}$ for n in range(N)] sorted by increasing $e_{2,n}$
- p = $\sum_{1 \leq n \leq N} p_{1,n}$ the budget associated
- e = E[-1], l,n the indexes associated (e = $e_{l,n}$)
- While p < pmax - $p_{l,n}$:
  - p = p - $p_{l-1,n} + p_{l,n}$
  - $x_{l,n} = 1$, $x_{l-1,n} = 0$
  - Delete e from E, add $e_{l+1,n}$ to E, maintaining the list sorted
  - e = E[-1], l,n the indexes associated
- If p < pmax :
  - $x = \frac{pmax-p}{p_{l,n} - p_{l-1,n}}$
  - $x_{l,n} = x$, $x_{l-1,n} = 1 - x$
- Datarate = $\sum_{\substack{1 \leq k \leq K \\ 1 \leq m \leq M \\ 1 \leq n \leq N}} x_{k,m,n} p_{k,m,n}$
- Return($x_{l,n}$ $for\ 1 \leq l \leq L\ and\ 1 \leq n \leq N,\ datarate$)

Each $e_{l,n}$ is examined at most once and added to an ordered list of size N in complexity O(N), therefore the complexity of our algorithm is $O(N^2L) = O(N^2KM)$

Results for the LP problem

greedyLP()	"test1.txt"	"test2.txt"	"test3.txt"	"test4.txt"	test5.txt"
Optimal rate	365	/	372.15	9642	1637
Used power	78	/	100	16000	1000
runtime	3.5×10^{-5}s	/	4.4×10^{-5}s	0.03s	4.8×10^{-4}s

LP_solver()	"test1.txt"	"test2.txt"	"test3.txt"	"test4.txt"	test5.txt"
Optimal rate	365	/	372.15	9642	1637
Used power	78	/	100	16000	1000
runtime	7.6×10^{-3}s	/	6.6×10^{-3}s	353s	0.048s

The optimal rate and used power are exactly the same for our greedy algorithm and the LP solver, as expected for an optimal solution. However, we can see that our greedy algorithm is a lot faster than the generic LP solver, especially on big data sets.

Dynamic programming to solve the ILP

Let be R(n, pmax) the best rate with n channels and a power budget of pmax.
We can write that R(n,pmax) = $\max_{\substack{(r,p)\in channel\ n, \\ p<pmax}} (R(n-1, pmax-p)+r)$.

Therefore we can use a DP algorithm based on this equation:

Algorithm 5 – Dynamic Programming algorithm to solve the ILP

Input: An instance PR containing all the triplets for each channel and a power budget pmax
Output: Optimal rate for the given power budget

- for P=0 to pmax:
  - let be R(1,P)= $\max_{\substack{(r,p)\in channel\ 1, \\ p\leq P}} r$
- for n=2 to N the number of channels:
  - for P=0 to pmax:
    - Compute R(n, P) = $\max_{\substack{(r,p)\in channel\ n, \\ p\leq P}} (R(n-1, P-p)+r)$.
- return R(N,pmax)

Computing the maximum over a channel for a given P is done in complexity O(KM). Therefore, the total complexity is O(NKMpmax). The space requirement is O(Npmax) if we keep all the values of R. However, we can only keep the values for the current n and the previous one, lowering the space requirement to O(2pmax) = O(pmax).

We can keep track of the power used but for the sake of simplicity it is not specified in the pseudocode.

An alternative DP 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. Assuming that the upper bound U is provided we can write the following equation with P(n,U) the minimal power allocation with n channels : p

$$P(n,U) = \min_{\substack{(r,p)\in channel\ n, \\ r<U}} (P(n-1, U-r) + p)$$

Therefore, we can use a DP algorithm based on this equation:

Algorithm 6 – Alternative Dynamic Programming algorithm to solve the ILP

Input: An instance PR containing all the triplets for each channel, a power budget pmax and an upper bound U for rate
Output: Optimal rate

- for u=0 to U:
  - let be P(1,u)= $\min_{\substack{(r,p)\in channel\ 1, \\ r=u}} p$
- for n=2 to N the number of channels:
  - for u=0 to U:
    - Compute P(n, u) = $\min_{\substack{(r,p)\in channel\ n, \\ r\le u}} (P(n-1, u-r) + r)$.
- return maximum u such that $P(N,u) \le pmax$ and the corresponding P(N,u)

For each n we have a loop of length U, computing a minimum in O(KM). Therefore, the total complexity is O(NKMU). As for the algorithm 5, we can use only two arrays at a time, giving a space requirement of O(U).

Branch-and-Bound approach

We now try a branch-and-bound approach to solve the ILP. Each level of the tree represents a channel, and each node of each level represents a feasible pair choice in that channel. Therefore, a path from from the source to a leaf is a solution to our problem. At a node k we use the greedy algorithm to get both an upper bound (relaxed problem) and a lower bound, to choose if the branch is further explored or not. The complexity is O(KM^{N+1})because in the worst case we go through all the nodes. However, in practice several branches are not explored, lowering the average complexity.

First we need an adapted greedy algorithm to give us the bounds of sub-problems :

Algorithm 7 – GreedyBounds

Input: current channel n, left power budget P, E list of all pairs sorted by efficiency in reversed order, C list of current used pairs
Output: Upper and lower bound for the optimal rate

- Let be R=0 and i=0
- While i<length(E) :
    - t = E[i]
    - if t.n ≥ n :
        - if t.powerInc ≤ P :
            - P-= t.powerInc
            - R+= t.rateInc
        - else:
            - break
    - i++
- min=R
- if P>0 and i<length(E):
    - t = E[i]
    - x = P/t.powerInc
    - R+=x*t.rateInc
- return R, min

Then we can use the BB algorithm:

Algorithm 8 – Branch-and-Bound algorithm to solve the ILP

Input: A list L containing all the pairs for each channel and a power budget pmax
Output: Optimal rate

- Let be E the list of all pairs sorted by efficiency
- Let be S a stack
- Let be P=pmax
- S.push((0,0,0)) #current channel, used power and achieved rate
- R, min = GreedyBounds(n, P, E)
- While S in not empty:
  - vertex = S.pop()
  - for pair in L[vertex[0]] :
    - if pair.power +v[1]>pmax:
      - break
    - if vertex[0]<N-1:
      - R1, min1 = GreedyBounds(n+1, P-v[1]-pair.power, E)
      - if R1+ vertex[2] + pair.rate > min:
        - S.push((v[0]+1, pair.power +v[1], vertex[2] + pair.rate))
        - min= R1+ vertex[2] + pair.rate
    - else:
      - min=max(min, vertex[2] + pair.rate)
- return min

Results for the ILP problem

The following table gives the results for the dynamic programming algorithms:

DP()	”test1.txt”	”test2.txt”	”test3.txt”	”test4.txt”	test5.txt”
Optimal rate	365	/	350	9642	1637
Used power	78	/	68	16000	1000
runtime	0.0014s	/	0.001s	18.9s	0.07s

DP2()	”test1.txt”	”test2.txt”	”test3.txt”	”test4.txt”	test5.txt”
Optimal rate	365	/	350	9642	1637
Used power	78	/	68	16000	1000
runtime	0.0015s	/	0.001s	8.9s	0.08s

We can see that the alternative DP algorithm is a lot faster on big data sets. However, we need to provide him a good upper bound (by using the LP algorithm for example), otherwise it will take much more time (65s for test4 with 100 000 as an upper bound for example).

Stochastic Online Scheduling

To simulate the problem, we make a function data(pmax, rmax, M, N) which gives a list of NM pairs of power less than pmax and rate less than rmax, using discrete uniform distribution. The list is returned sorted on the ratio rate/power. Each pair contain the values of power, rate and n (channel number).

When the user k arrives, we call data() to gives us a list of pairs. Then we use a greedy approach: we try every pair by decreasing ratio rate/power until we find one which complies with two constraints:

- The corresponding channel is not already used

- The pair power + the already used power is less than k*pmax/K.

The first one is self-explanatory. For the second one, we first though of just verifying that pair.power was less than pmax/K. However, this would mean that if a previous user did not used all the power allowed to him, some power will be wasted unnecessarily.

Algorithm 9 – Online algorithm

Input: pmax and rmax upper bounds for power and rate of the uniform distribution, p the power budget, M, N and K
Output: online rate

- Let be channels an array with channels[0]=1 if the channel is used, 0 if not.
- Let be pcurrent=0
- Let r=0 the current achieved rate
- for k=0 to K-1:
    - for pair in data(pmax, rmax, M, N)
        - if channel[pair.n]==0 and pair.power+pcurrent ≤(k+1)*p/K:
            - channel[pair.n]==1
            - pcurrent+= pair.power
- return r, pcurrent

After implementing this algorithm, we tried another version with a little modification: we compute the average ratio rate/power for a uniform distribution with the given parameters pmax and rmax. Then we only keep pairs which have a ratio rate/power of at least the average. It is especially efficient if the number of users is larger than the number of channels, which is the case in the studied example.

Algorithm 10 – Online algorithm with average

Input: pmax and rmax upper bounds for power and rate of the uniform distribution, p the power budget, M, N and K
Output: online rate

- Let be E the average of rate/power for a uniform distribution of power and rate with parameters pmax and rmax
- Let be channels an array with channels[0]=1 if the channel is used, 0 if not.
- Let be pcurrent=0
- Let r=0 the current achieved rate
- for k=0 to K-1:
  - for pair in data(pmax, rmax, M, N)
    - if channel[pair.n]==0 and pair.power+pcurrent ≤(k+1)*p/K and pair.rate/pair.power ≥ E:
      - channel[pair.n]==1
      - pcurrent+= pair.power
- return r, pcurrent

Results for the online problem

For p=100, pmax =50, rmax =100, M=2, N=4 and K= 10 we compute the average competitive ratio, rate achieved and used power of our algorithms on a large number of problem instances:

	online()	online2()	DP()
Competitive ratio	0.52	0.58	1
Average rate achieved	180	203	348
Average used power	34.7	34.8	41.7

We can see that both online algorithms have a competitive ratio just above ½. The version 2 has a better ratio with 0.06 more, which is non-negligeable as we can see on the average rate achieved.

The two version are really close in terms of average used power, slightly less than the offline optimal algorithm.