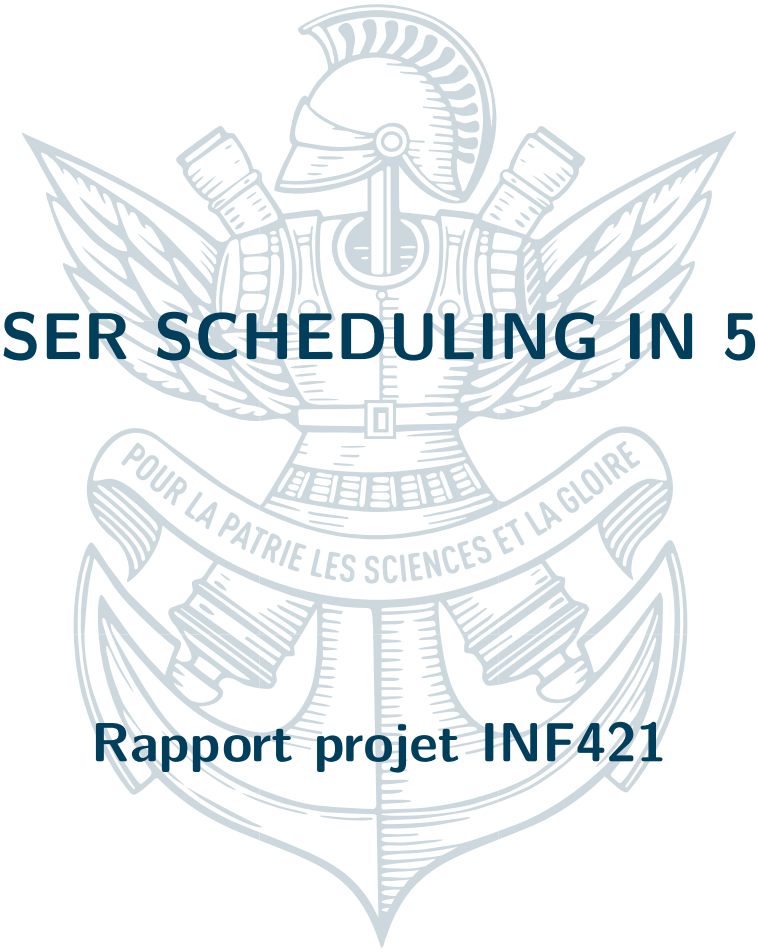

USER SCHEDULING IN 5G

Rapport projet INF421

3 février 2020

Youssef A. Sariah AS.

CHAPITRE 1

FORMULATION DU PROBLÈME

Question 1 :

Données : Ensemble des canaux $\mathcal{N}$ de cardinal N, ensemble des utilisateurs $\mathcal{K}$ de cardinal K, entier M, budget de puissance P, tableaux des puissances $\{p_{n,k,m}\}$ et tableaux des débits $\{r_{n,k,m}\}$ tels que $n \in \mathcal{N}$, $k \in \mathcal{K}$, $m \leq M$.

Sortie : Tableau $\{x_{n,k,m}\}$, avec $x_{n,k,m} \in \{0,1\}$, tel que ce tableau réalise

$$\max \sum_{n,k,m} x_{n,k,m} r_{n,k,m} \tag{1}$$

sous les contraintes

$$\forall n \quad \sum_{k,m} x_{n,k,m} = 1 \tag{2}$$

$$\sum_{n,k,m} x_{n,k,m} p_{n,k,m} \leq P \tag{3}$$

PRÉTRAITEMENT

Question 2 :

Dans un premier temps, on vérifie que

$$\sum_{n} \min_{k} p_{n,k,0} \leq P \tag{4}$$

Si ce n'est pas le cas, alors l'instance donnée en entrée ne présente aucune solution qui puisse satisfaire la contrainte (2) et la contrainte (3) simultanément.

Ensuite, on élimine dans notre tableau les triplets (n,k,m) tels que $p_{n,k,m} \geq P$, car ces triplets ne sont pas admissibles dans la solution finale. Si un de ces triplets est choisi, alors de façon évidente le tableau retourné en résultat ne respecte pas les contraintes du problème.

Question 3 :

Algorithme : On propose l'algorithme suivant pour éliminer les termes IP-dominés.
Pour chaque canal n, rassembler dans une liste de taille $L, L \leq= K * M$ l'ensemble des couples (k, m). Trier cette liste dans l'ordre des $p_{k,m}$ croissants. Parcourir les indices dans l'ordre obtenu et éliminer les couples (k, m) qui brisent la croissance des $r_{k,m}$ en parcourant la liste.
En particulier, pour deux valeurs de $p_{k,m}$ identiques, on élimine le couple (k, m) pour lequel $r_{k,m}$ est inférieur .

Pseudo-Code : On fournit le pseudo-code suivant, où l'on suppose que les données sont valides après application de l'algorithme de la question 2. On remarque que l'élimination de quelques termes grâce à la question 2 nous empêche d'établir l'égalité $L = K*M$ dans la ligne 4.

Algorithm 1: IP Processing

```
1  for n ∈ 𝒩 do
2      Créer la liste des couples (p_{k,m}, r_{k,m}) ;
3      Trier la liste par ordre croissant de p_{k,m};
4      Réindexer cette liste sous la forme (p_l, r_l), l ≤ L;
5      l = 0 ;
6      i = 1 ;
7      while l + i < L do
8          if r_{l+i} ≤ r_l then
9              affecter à x_{k,m} associé à l'indice l + i la valeur ELIM;
10             i = i + 1
11         else
12             l = l + i;
13             i = 1
```

Pour plus de clarté on ne mentionne pas dans ce pseudo-code le cas où $r_{l'} \neq r_l$ avec $p_{l'} = p_l$. On élimine dans ce cas l'indice l tel que $r_l \leq r_{l'}$.

Analyse de la complexité : Pour chaque canal n, on crée une liste de taille KM en $O(KM)$ dans la cas où aucun couple (k, m) n'a été éliminé par prétraitement, ce qui correspond à notre pire des cas. On trie cette liste en temps $O(KM \log(KM))$, puis on parcourt cette liste en effectuant à chaque itération des comparaisons en temps constant, ce qui se fait en temps $O(KM)$. On répète cette opération pour chaque canal, donc N fois. D'où la complexité finale :

$$O(NKM \log(KM)) \tag{5}$$

Question 4 :

Algorithme : Le prétraitement LP consiste à retrouver la partie supérieure de l'enveloppe convexe de l'ensemble des points $\{(p_{n,k,m}, r_{n,k,m})\}$ pour un canal $n \in N$ donné. On choisit d'appliquer l'algorithme de Graham, qui utilise la structure de piles. On suppose que l'on a déjà éliminé les termes IP-dominés.

Pseudo-Code : On fournit le pseudo-code suivant, où l'on suppose que les données sont valides après application de l'algorithme de la question 2, et que l'on a déjà appliqué l'algorithme de la question 3.

Algorithm 2: LP Processing

```
1  for n ∈ 𝒩 do
2      Créer la liste des couples (p_{k,m}, r_{k,m}) ;
3      Trier la liste par ordre croissant de p_{k,m};
4      Réindexer cette liste sous la forme (p_l, r_l), l ≤ L;
5      i = 1 ;
6      enveloppe une pile vide;
7      enveloppe.push((p_0, r_0));
8      enveloppe.push((p_1, r_1));
9      l = 2 ;
10     while l < L do
11         enveloppe.push((p_l, r_l));
12         booléen valide = true while not valide and enveloppe.size > 2 do
13             point3 = enveloppe.pop;
14             point2 = enveloppe.pop;
15             pivot = enveloppe.pop;
16             if point2 à droite du segment [pivot, point3] then
17                 enveloppe.push(pivot);
18                 enveloppe.push(point3);
19                 Eliminer la variable associée à point2;
20             else
21                 enveloppe.push(pivot);
22                 enveloppe.push(point2);
23                 enveloppe.push(point3);
24                 valide= true;
25             l++;
```

Analyse de la complexité : Pour chaque canal n, on crée une liste de taille KM en temps $O(KM)$, on la trie en temps $O(KM \log(KM))$, puis on applique l'agorithme de Graham dont

la compléxité en temps est $O(KM\log(KM))$ pour une liste de taille KM. D'où la complexité en temps finale :

$$O(NKM\log(KM)) \tag{6}$$

Question 5 :

La méthode "Display()" permet d'afficher le tableau $\{x_{n,k,m}\}$. Pour plus de lisibilité, on indique le nombre de variables éliminées après l'application de ces algorithmes de prétraitements.

Fichier test1 :

— Variables éliminées après :
  Prétraitement rapide : 0
  Traitement IP : 14
  Traitement LP : 16

Fichier test2 :

— Variables éliminées après :
  Prétraitement rapide : Ce problème n'a pas de solution.

Fichier test3 :

— Variables éliminées après :
  Prétraitement rapide : 0
  Traitement IP : 11
  Traitement LP : 15

Fichier test4 :

— Variables éliminées après :
  Prétraitement rapide : 0
  Traitement IP : 599713
  Traitement LP : 609418

Fichier test5 :

— Variables éliminées après :
  Prétraitement rapide : 0
  Traitement IP : 2071
  Traitement LP : 2207

Le nombre de variables éliminées présenté est le nombre cumulé après application des algorithmes de prétraitement successivement.

PROGRAMME LINÉAIRE ET ALGORITHME GLOUTON

Question 6 :

Algorithme :

Pseudo-Code : On fournit le pseudo-code suivant. On suppose que les algorithmes de prétraitement on été appliqué au préalable.

Algorithm 3: Greedy

Pour chaque $n \in \mathcal{N}$, $keyList[n]$ est la liste des indices du canal n triés par ordre croissant de puissance.
Pour chaque $n \in \mathcal{N}$, $Candidat[n]$ est l'indice du canal n ayant la plus grande efficacité incrémentale.
$target := 0$ // indice du candidat retenu à chaque étape de la boucle while.
$budget := 0$ // budget épuisé
while $budget \leq P$ **do**
    $target = Candidat[n], n \in \mathcal{N}$ *celui qui a la plus grande efficacité incrémentale*
    $Candidat[target] += 1$
    $budget += Puissance[Candidat[target]] - Puissance[Candidat[target] - 1]$ // *actualisation du budget épuisé après avoir changé la puissance potentiellement retenue*
Calculer le budget épuisé dans la totalité des canaux et le rate atteint
if $budget \geq P$ **then**
    Répartir la puissance disponible avant la dernière itération de la boucle while sur les deux derniers candidats du canal en question pour que $budget = P$.

Analyse de la complexité : On crée N tableaux de taille L en temps $O(L)$. On initialise chaque tableau avec tri en temps $O(L\log(L))$. La complexité de l'initialisation coûte $O(NL\log(L))$ au total.
Chaque itération de la boucle while coûte $O(N)$ car nécessite de parcourir chaque canal. L'élément ayant la plus grande efficacité incrémentale dans chaque canal se trouve en temps constant grâce au lemme 2, c'est en fait le premier élément de $keyList[n]$. La boucle while contient au plus P itérations, la complexité de la boucle est donc en $O(NP)$
La dernière opération de l'algorithme sur le budget s'effectue en temps constant.

La complexité en temps de l'algorithme est donc en $O(NP + NL\log(L))$. La complexité en espace est en $O(NL)$.

Question 7 :

Fichier test1 :

- Résultats :
  - Budget épuisé : 78/100
  - Rate atteint : 365
- Temps (ms) :
  - Greedy : 34
  - LP Solver : 16

Fichier test2 : Il n'y a pas de solution pour cette instance.

Fichier test3 :

- Résultats :
  - Budget épuisé : 100/100
  - Rate atteint : 3.7215385e+02
- Temps (ms) :
  - Greedy : 36
  - LP Solver : 17

Fichier test4 :

- Résultats :
  - Budget épuisé : 16000/16000
  - Rate atteint : 9.8703218e+03
- Temps (ms) :
  - Greedy : 715
  - LP Solver : 4073

Fichier test5 :

- Résultats :
  - Budget épuisé : 1000/1000
  - Rate atteint : 1637
- Temps (ms) :
  - Greedy : 64
  - LP Solver : 50

Remarques

— Les deux algorithmes donnent les mêmes résultats sur tous les tests. Il est à noter que l'algorithme LP Solver introduit des légères perturbations négatives parfois dans sa résolution.

— Nous remarquons que l'algorithme LP Solver est plus rapide en règle générale, sauf sur le test 4 à cause du grand nombre de variables de décision (au nombre de 6927, comparé à 207 pour le test 5).

Question 8 :

Algorithme : On propose le premier algorithme de programmation dynamique suivant, qui retourne la valeur maximale de débit pouvant être distribué sous la contrainte de budget considérée P.
On crée un nouveau tableau $\{w_{n,p}\}$ de N lignes et de P colonnes. La valeur $\{w_{n,p}\}$ correspond au débit maximal pouvant être distribué avec $n \leq N$ canaux étant donné une contrainte de budget $p \leq P$.
On initialise la première ligne de ce tableau de la façon suivante :

$$\forall p \leq P, w_{0,p} = \max_k u_{k,0}(p) \tag{7}$$

avec $u_{k,n}$ les fonctions utilitaires définies dans l'énoncé.
Ensuite, on remplit le tableau ligne par ligne avec les équations DP suivantes :

$$\forall 1 \leq n \leq N, \forall p \leq P, w_{n,p} = \max_{q \leq P} \{w_{n-1,q} + \max_k u_{k,n}(p-q)\} \tag{8}$$

La valeur recherchée à la fin est la valeur $w_{N,P}$, qui donnera la valeur maximale de débit pouvant être obtenue avec cette approche dynamique.

Pseudo-Code : On fournit le pseudo-code suivant. On suppose que les algorithmes de prétraitement on été appliqué au préalable.

Algorithm 4: DPPower

```
u[1..N][1..P] nouveau tableau;
w[1..N][1..P] nouveau tableau;
for n ∈ 𝒩 do
    Créer la liste des couples couples = (p_{k,m}, r_{k,m}) ;
    Trier la liste par ordre croissant de p_{k,m};
    Initialiser la liste u[n][p], 1 ≤ p ≤ P;
Initialiser la liste w[1][p], 1 ≤ p ≤ P;
for 2 ≤ n ≤ N do
    for 1 ≤ p ≤ P do
        max = 0;
        w = 0 ;
        for 1 ≤ q ≤ p do
            r = u[n][p − q];
            x = w[n − 1][q] + r;
            if max ≤ x then
                max = x;
        w[n][p] = max;
return w[N][P];
```

Analyse de la complexité : On crée deux tableaux de taille $N * P$ en temps $O(NP)$. On initialise un premier tableau en temps $O(NP + NL\log(L))$, puis une liste en temps $O(P)$. Ensuite, on remplit le deuxième tableau de taille $N * P$, et à chaque itération on recherche le maximum d'une liste ayant au plus P éléments, ce qui se fait en temps $O(NP^2)$. La complexité en temps est donc de $O(NP^2 + NL\log(L))$. Sachant que $L \leq P$ car la suite $\{p_l\}$ est strictement croissante après prétraitement, on obtient une complexité finale en temps

$$O(NP^2) \tag{9}$$

La complexité en espace est $O(NP)$.

Question 9 :

Algorithme : On propose un deuxième algorithme de programmation dynamique, qui retourne la valeur minimale de puissance nécessaire pour délivrer une valeur de débit $r \leq U$, U étant une borne de débit donnée.
On crée un nouveau tableau $\{v_{n,r}\}$ de N lignes et de U colonnes. La valeur $\{v_{n,r}\}$ correspond à la puissance minimale nécessaire pour délivrer un débit $r \leq U$ avec $n \leq N$ canaux.

On initialise la première ligne de ce tableau de la façon suivante :

$$\forall r \leq U, v_{0,r} = \min_k u_{k,0}^{-1}(r) \tag{10}$$

avec $u_{k,n}^{-1}(r)$ la fonction inverse de la fonction utilitaire définie dans l'énoncé, donnant la valeur de puissance *minimale* nécessaire pour fournir un débit r à l'utilisateur k grâce au canal n. Cette fonction peut prendre des valeurs infinies dans le cas où le débit r donné ne peut pas être livré du tout à l'utilisateur k par le canal n.
Ensuite, on remplit le tableau ligne par ligne avec les équations DP suivantes :

$$\forall 1 \leq n \leq N, \forall r \leq U, v_{n,r} = \min_{q \leq r}\{v_{n-1,q} + \min_k u_{k,n}^{-1}(r-q)\} \tag{11}$$

La valeur recherchée à la fin est la valeur

$$R_0 = \arg\max_{r \leq U}\{v_{N,r}; v_{N,r} \neq \infty\} \tag{12}$$

qui donnera la valeur maximale de débit pouvant être obtenue avec cette approche dynamique. La puissance utilisée sera alors v_{N,R_0}.

Pseudo-Code : On fournit le pseudo-code suivant. On suppose que les algorithmes de pré-traitement on été appliqué au préalable.

Algorithm 5: DPRate

```
u^{-1}[1..N][1..U] nouveau tableau;
v[1..N][1..U] nouveau tableau;
for n ∈ 𝒩 do
    Créer la liste des couples couples = (p_{k,m}, r_{k,m}) ;
    Trier la liste par ordre croissant de p_{k,m};
    Initialiser la liste u^{-1}[n][r], 1 ≤ r ≤ U;
Initialiser la liste v[1][r], 1 ≤ r ≤ U;
for 2 ≤ n ≤ N do
    for 1 ≤ r ≤ U do
        min = 0;
        v = 0 ;
        for 1 ≤ s ≤ r do
            p = u^{-1}[n][r − s];
            x = v[n − 1][s] + p;
            if min ≥ x then
                min = x;
        v[n][r] = min;
R = U;
while u^{-1}[N][R] = ∞ do
    R = R − 1;
return v[N][R];
```

Analyse de la complexité : On crée deux tableaux de taille $N * U$ en temps $O(NU)$. On initialise un premier tableau en temps $O(NU + NL\log(L))$, puis une liste en temps $O(U)$. Ensuite, on remplit le deuxième tableau de taille $N * U$, et à chaque itération on recherche le maximum d'une liste ayant au plus U éléments, ce qui se fait en temps $O(NU^2)$. La complexité en temps est donc de $O(NU^2 + NL\log(L))$. Sachant que $L \leq U$ car la suite $\{r_l\}$ est strictement croissante après prétraitement, on obtient une complexité finale en temps

$$O(NU^2) \tag{13}$$

La complexité en espace est $O(NU)$.

Question 10 :

Cet algorithme repose sur la structure d'arbre dont la racine est le problème LP, donc sans obliger que la solution soit entière. A chaque niveau de l'arbre, si la solution n'est pas entière, on crée un nouveau niveau composé de neouds fils où chaque noeud est le même problème que le père en imposant la valzeur d'une des variables de décision à 0 ou 1. Nous explorons les noeuds au fur et à mesure et décidons d'arrêter cette exploration si les noeuds ne sont pas intéressants.

Algorithm 6: BB

```
1  Initialiser arbre tree avec racine z la solution du LP Solver pour le problème LP.
2  z_v := z
3  z_opt := 0
4  Candidat la liste des valeurs des variables de décision du problème
5  while there is a branch z_v left do
6      if z_v not feasible then
7          Passer au noeud suivant
8      else
9          if z_v is an integer then
10             if z ≤ z_opt then
11                 Passer au noeud suivant
12             else
13                 z_opt = z_v
14                 Candidat reçoit les nouvelles variables de décison
15             for every x_{n,k,m} do
16                 Ajouter noeud fils avec la valeur de la solution LP pour le problème du
                    prédecesseur en ajoutant la contrainte x_{n,k,m} = 0
17                 Ajouter noeud fils avec la valeur de la solution LP pour le problème du
                    prédecesseur en ajoutant la contrainte x_{n,k,m} = 1
18         else
19 return Candidat
```

Question 11 :

Fichier test1 :

- Résultats DPPower :
  Budget épuisé : 78/100
  Rate atteint : 365
- Résultats DPRate :
  Budget épuisé : 78/100
  Rate atteint : 365

Fichier test2 : Il n'y a pas de solution pour cette instance.

Fichier test3 :

- Résultats DPPower :
  Budget épuisé : 68/100
  Rate atteint : 350
- Résultats DPRate :
  Budget épuisé : 68/100
  Rate atteint : 350

Fichier test4 :

- Résultats DPPower :
  Budget épuisé : 16000/16000
  Rate atteint : 9870
- Résultats DPRate :
  Budget épuisé : 15999/16000
  Rate atteint : 9870
- Temps (ms) :
  DPPower : 218654 (env. 4min)
  DPRate : 110346 (env. 2min)

Fichier test5 :

- Résultats DPPower :
  Budget épuisé : 1000/1000
  Rate atteint : 1637
- Résultats DPRate :
  Budget épuisé : 995/1000
  Rate atteint : 1630

— Temps (ms) :

DPPower : 195

DPRate : 270

Question 12 :

Nous proposons pour cette question trois différents algorithmes pour approcher différemment le compromis entre performance et nombre de cas où une contrainte n'est pas respectée par l'algorithme.

Première approche Le premier algorithme dispose de la meilleure performance en taux transmis, il a cependant le plus grand nombre de violations de contraintes. Cet algorithme,

quand il le fait, viole uniquement la contrainte sur la limite du budget.

Algorithm 7: Online

```
1  Activated := [] //Activated[n]=1 si le canal n a été activé
2  n_activ := 0 //nombre de canaux activés
3  for k ∈ 𝒦 do
4      indices := []
5      efficacite := -1
6      for n ∈ 𝒩 do
7          for m ≤ M do
8              p_{k,m} = entier aléatoire entre 1 et p_max
9              r_{k,m} = entier aléatoire entre 1 et r_max
10         if n_activ ≤Net(k≥K-(N - n_activ) ou worth(k, p_max, r_max))) then
11             for n ∈ 𝒩 do
12                 for m ≤ M do
13                     candidat := p_{k,m,n}/r_{k,m,n}
14                     if k ≥ K − (N − n_activ) et activated[n] ≠ 1 et p_{k,m,n} ≤puissance du
                        dernier candidat then
15                         efficacite = candidat
16                         indices[0] = n
17                         indices[1] = m
18                     else
19                         if activated[n] ≠ 1 et p ≤ Puissance disponible et r_{n,k,m} >= 0.75r_max
                            et efficacite <= candidate then
20                             efficacite = candidat
21                             indices[0] = n
22                             indices[1] = m
23             if k ≥K-(N - n_activ) ou efficacite ≠ −1 then
24                 n := indices[0]
25                 m := indices[1]
26                 x_{n,k,m} = 1
27                 activated[n] = 1
28                 n_activ + +
```

La fonction worth (ligne 10) prend en argument k, p_{max} et r_{max} et retourne true si la plus grande efficacité des couples du nouvel utilisateur k est supérieure à l'espérance de la variable aléatoire efficacité $= \frac{r}{p}$. Elle concrétise l'intuition selon laquelle pour certains utilisateurs, il peut ne pas valoir le coup de dépenser de la puissance.

Deuxième approche Le deuxième algorithme dispose d'une moins bonne performance que le premier en taux transmis, mais il a moins de violations de contraintes. Cet algorithme, quand

il le fait, viole uniquement la contrainte sur le nombre de canaux à activer (n'en active pas suffisamment).

Algorithm 8: Online2

```
1  Activated := [] //Activated[n]=1 si le canal n a été activé
2  n_activ := 0 //nombre de canaux activés
3  for k ∈ 𝒦 do
4      indices := []
5      for n ∈ 𝒩 do
6          for m ≤ M do                                              height 0.4
7              p_{k,m} = entier aléatoire entre 1 et p_max
8              r_{k,m} = entier aléatoire entre 1 et r_max
9          if n_activ ≤Net(k≥K-(N - n_activ) ou worth(k, p_max, r_max))) then
10             t := 0
11             for t ≤ N − n_activ do
12                 Parcourir par ordre décroissant les r_{n,k,m} jusqu'à réaliser la condition
                   r_{n,k,m}/p_{n,k,m} ≥ r/p où r et p sont tirés aléatoirement dans les lois uniformes discrètes
                   de l'énoncé.
13                 if k ≥K-(N - n_activ) ou r_{n,k,m}/p_{n,k,m} ≥ r/p then
14                     n := indices[0]
15                     m := indices[1]
16                     x_{n,k,m} = 1
17                     activated[n] = 1
18                     n_activ + +
```

Troisième approche Le deuxième algorithme dispose d'une moins bonne performance que le deuxième en taux transmis, mais il n'a quasiment jamais de violation de contrainte en revanche.

Algorithm 9: Online3

```
Activated := [] //Activated[n]=1 si le canal n a été activé
n_activ := 0 //nombre de canaux activés
for k ∈ K do
    indices := []
    efficacite := -1
    for n ∈ N do
        for m ≤ M do
            p_{k,m} = entier aléatoire entre 1 et p_max
            r_{k,m} = entier aléatoire entre 1 et r_max
        if n_activ ≤N then
            Parcourir les couples jusqu'à trouver celui dont l'efficacité est la plus grande sous height 0.4pt d
             la contrainte que p_{n,k,m} ≤ Puissancedisponible
            if efficacite ≠ −1 then
                n := indices[0]
                m := indices[1]
                x_{n,k,m} = 1
                activated[n] = 1
                n_activ + +
```

Question 13 :

Les calculs de ratios moyens sont effectués sur 100000 itérations.

Première approche

- Ratio moyen de puissance : 78%
- Ratio moyen de rate : 86%
- Pourcentage de dépassement de contrainte : 5.293%

Deuxième approche

- Ratio moyen de puissance : 83%
- Ratio moyen de rate : 78%
- Pourcentage de dépassement de contrainte : 2.771%

Troisième approche

— Ratio moyen de puissance : 61%

— Ratio moyen de rate : 69%

— Pourcentage de dépassement de contrainte : 0.067%