Mini-Projet d'algorithmique

Confitures

3I003

Compte-rendu du projet

MOREIRA GUERRA Véronique
et
LEVESQUE Paul-Antoine

Table des matières

Première partie

Partie théorique

Algorithme I : Recherche exhaustive

Question 1

On veut montrer par récurrence forte sur s que l'algorithme de recherche exhaustive est valide et se termine.
Initialisation :
Pour $s < 0$ le programme se termine et retourne $+\infty$. Le programme est valide car il est impossible de réaliser une quantité négative.
Pour $s = 0$ le programme se termine et retourne 0. C'est bien le résultat attendu pour 0dg de confiture, le programme est donc valide.
Induction :
Supposons par récurrence forte sur s que l'algorithme est valide et se termine pour $n \leq s$, avec $n \in \mathbb{N}$, et montrons qu'il en est de même pour $n = s + 1$.
À l'éxuction de *RechercheExhaustive(k, V, n)* le programme appelle recursivement *RechercheExhaustive(k, v, n - V[i])*, qui se termine et est valide d'après l'hypothèse de récurrence, car $n - V[i] \leq s$. Alors l'appel avec $n = s + 1$ se termine aussi et est valide.
Conclusion :
L'initialisation est vraie, de plus on a montrer que si l'hypothèse de recurrence est vérifiée pour $n \leq s$ alors elle l'est aussi pour $n = s + 1$. Donc l'algorithme de recherche exhaustive est valide et se termine pour tout s.

Question 2

On suppose que les seuls bocaux disponibles sont des bocaux de capacités 1dg et 2dg. On note $a(s)$ le nombre d'appels récursifs effectués par RechercheExhaustive(2,[1,2],s).

a) Expression de a(s) :

$$a(s) = \begin{cases} 0 & si\ s = 0 \\ 1 & si\ s = 1 \\ a(s-1) + a(s-2) + 2 & si\ s \geq 2 \end{cases}$$

b) Soit b(s) la suite définie par

$$b(s) = \begin{cases} 0 & si\ s = 0 \\ 2 & si\ s = 1 \\ 2b(s-2) + 2 & si\ s \geq 2 \end{cases}$$

Soit c(s) la suite définie par

$$c(s) = \begin{cases} 0 & si\ s = 0 \\ 2c(s-1) + 2 & si\ s \geq 1 \end{cases}$$

Montrons par récurrence que $b(s) \leq a(s) \leq c(s)$ pour tout $s \geq 0$.
Initialisation :
Pour $s = 0$, on a $a(s) = 0$, $b(s) = 0$ et $c(s) = 0$. La propriété est donc vraie.
Pour $s = 1$, on a $a(s) = 2$, $b(s) = 2$, et $c(s) = 2$. L'hypothèse est encore vraie ici.
Induction :
Supposons par récurrence forte que l'hypothèse est vraie pour $n \leq s$ et montrons qu'elle est vraie pour $n = s + 1$ avec $n \in \mathbb{N}$.

Notons :

$$\begin{cases} b(s) \leq a(s) & (1) \\ a(s) \leq c(s) & (2). \end{cases}$$

Montrons d'abord (1) :
Comme $b(s-1) \leq b(s)$ et que par hypothèse de récurrence on a, $b(s) \leq a(s)$, on peut dire que :

$$\begin{aligned} & b(s-1) \leq a(s) \\ & b(s-1) + b(s-1) \leq a(s) + a(s-1) \\ & 2b(s-1) \leq a(s) + a(s-1) \\ & 2b(s-1) + 2 \leq a(s) + a(s-1) + 2 \\ & b(s+1) \leq a(s+1). \end{aligned} \tag{1}$$

L'inéquation (1) est vérifiée.

Montrons maintenant (2) sur le même principe :
Comme $a(s-1) \leq a(s)$ et que par hypothèse de récurrence on a, $a(s) \leq c(s)$, on peut dire que :

$$\begin{aligned} & a(s-1) \leq c(s) \\ & a(s) + a(s-1) \leq c(s) + c(s) \\ & a(s) + a(s-1) \leq 2c(s) \\ & a(s) + a(s-1) + 2 \leq 2c(s) + 2 \\ & a(s+1) \leq c(s+1). \end{aligned} \tag{2}$$

L'inéquation (2) est vérifiée.
On a alors : $b(s+1) \leq a(s+1) \leq c(s+1)$
Conclusion :
L'initialisation est vraie, de plus on a montrer que $b(s) \leq a(s) \leq c(s)$ pour tout $s \geq 0$. Donc l'hypothèse est vraie.

c) Montrons que le terme général de c(s) est : $c_g(s)=2^{s+1} - 2$ pour tout $s \in \mathbb{N}$. Avec $c_g(0) = 0$.
Initialisation :
Pour $s = 0$, $c_g(0) = c(0) = 0$.
Pour $s = 1$, $c_g(1) = c(1) = 2$. L'initialisation est vraie.

Induction :
Supposons que $c_g(s)$est vraie pour $s-1$ et montrons qu'elle est vraie pour s, avec $s \in \mathbb{N}$.
D'après la définition de la suite : $c(s) = 2c(s-1) + 2$ On a :

$$\begin{aligned} c(s) &= 2(2^s - 2) + 2 && \text{(par hypothèse de récurrence)} \\ &= 2^{s+1} - 4 + 2 \\ &= 2^{s+1} - 2 \\ &= c_g(s). \end{aligned}$$

Conclusion :
On a montré que $c_g(s)$ est bien le terme général de la suite $c(s)$ pour tout $s \in \mathbb{N}$.

d) Montrons que $b(s) = c(\lceil \frac{s}{2} \rceil)$
Initialisation :
Pour $s = 0$, $b(0) = c(\frac{0}{2}) = c(0) = 0$.
Pour $s = 1$, $b(1) = c(\frac{1}{2}) = c(1) = 2$. L'initialisation est vraie.
Induction :
Supposons par récurrence forte sur s que l'hypothèse de récurrence est vraie pour $k \leq s-1$, avec $n \in \mathbb{N}$, et montrons qu'il en est de même pour $k = s$.

$$\begin{aligned} c(\lceil \frac{s}{2} \rceil) &= 2c(\frac{s}{2} - 1) + 2 \\ &= 2c(\frac{s-2}{2}) + 2 \\ &= 2b(s-2) + 2 && \text{(par hypothèse de récurrence)} \\ &= b(s). \end{aligned}$$

Conclusion :
Les étapes d'initialisation et d'induction sont vraies, on a donc bien $b(s) = c(\lceil \frac{s}{2} \rceil)$ pour tout $s \in \mathbb{N}$.

e) D'après les questions c) et d) on peut déterminer le terme général de de b(s).

$$b_g(s) = 2^{\lceil \frac{s}{2} \rceil + 1} - 2 \tag{3}$$

Et d'après a) on peut donc encadrer a(s) :
$2^{\lceil \frac{s}{2} \rceil + 1} - 2 \leq a(s) \leq 2^{s+1} - 2$
On en déduit que la complexité temporelle de l'algorithme *RechercheExhaustive* est en $\mathcal{O}(2^s)$

Algorithme II : Programmation dynamique

Question 3

a)
Valeur de $m(S)$ en fonction de $m(s,i)$

$$m(S) = \begin{cases} +\infty & si\ le\ tableau\ V\ est\ vide \\ +\infty & si\ S < 0 \\ 0 & si\ S = 0 \\ m(S,k) & si\ S \geq 1. \end{cases}$$

b)
Montrons par que pour tout $i \in \{1, ..., k\}$

$$m(s,i) = \begin{cases} 0 & si\ s = 0 \\ min\{m(s,i-1), m(s-V[i],i)+1\} & sinon. \end{cases}$$

- Si $s = 0$:
Alors on n'a pas de confiture donc quelque soit i, $m(s,i) = 0$ puisqu'il n'est pas possible de remplir un bocal.
- Si $s \neq 0$:
Alors on a deux cas :

Cas 1 : $m(s - V[i], i) + 1$
Dans ce cas, la solution comprend un bocal de taille $V[i]$. On le soustrait à s, ce qui compte pour un bocal rempli d'où le $+1$, puis on calcule le nombre minimal de bocaux pour $s - V[i]$ sur les i bocaux.

Cas 2 : m(s, i - 1)
Dans ce cas, la solution ne comprend pas de bocal de taille $V[i]$. On cherche donc une solution sur les $i - 1$ bocaux restants.
Conclusion : Une fois le cas 1 et 2 traité, on prend le minimum des deux pour avoir le nombre minimum de bocaux correspondant à la solution optimale. La relation de récurrence est donc bien vérifiée.

Question 4

a) On remplit les cases en commençant par gérer les cas où $s = 0$, dans ce cas la case de la matrice vaut 0. On gère ensuite les cas ou $i = 0$ en mettant les cases correspondantes à $+\infty$.
Puis on remplit le reste de la matrice grâce à la formule de récurrence, en s'assurant que $j - V[i] \geq 0$ pour que $M[s - V[i]][i]$ ne cherche pas la valeur d'une case d'indice négatif. De même on s'assure que $i \geq 1$ pour que $M[s][i-1]$ ne cherche pas la valeur d'une case d'indice négatif.

b)

Algorithm 1 AlgoProgDyn(k : entiers, V : tableau de k entiers, s : entier) : entier

```
l, j, v1, v2 : entiers
M[s+1][i+1] : tableau double d'entiers avec s ∈ {0, ..., S} et i ∈ {0, ...., k}
for l :=0 to k do
    for j := 0 to s do
        if j = 0 then
            M[j][l] ← 0
        else if l = 0 then
            M[j][l] ← +∞
        else
            if j − V[l] ≥ 0 then
                v1 ← 1 + M[j − V[l]][l]
            end
            if l ≥ 1 then
                v2 ← M[j][l − 1]
            end
            M[j][l] ← min(v1, v2)
        end
    end
end
return M[s][k]
```

c) La complexité spatiale de l'algorithme est en $\mathcal{O}(k \times S)$, c'est-à-dire la taille du tableau doublement indicé. Et sa complexité temporelle est aussi en $\mathcal{O}(k \times S)$, car une opération par case pour calculer sa valeur.

Question 5

a) Dans chaque case du tableau M doublement indicé, en plus d'affecter la valeur de la case d'indice $[i][j]$, il faut aussi ajouter à cette case un tableau A de taille i qui correspond aux bocaux $V[0]...V[i]$ pris pour une quantité s. Il faut ajouter dans l'algorithme une boucle qui va déterminer dans le tableau A le nombre de bocaux utilisés, à l'aide d'un compteur, pour la quantité s courante. On n'oublie pas de réinitialiser correctement le compteur à la fin de la boucle. Au fur et à mesure que l'algorithme tourne, une fois arrivé à la dernière case $M[s][k]$, on aura dans cette case le tableau A qui correspond au nombre de bocaux utilisés $V[0]...V[k]$.

b) On rajoute à l'algorithme de la question 4b) :

```
Bocaux[k] : tableau d'entiers pour
for i :=0 to k do
  | Bocaux[i] ← 0
end
i = k
j = s
int cpt = 1
while j > 0 do
  | if j = 1 then
  |   | j ← 0
  |   | Bocaux[0] ← Bocaux[0] + 1
  | end
  | else if j = V[j − 1] then
  |   | j ← 0
  |   | Bocaux[i − 1] ← Bocaux[i] + 1
  | else if j − V[i] ≥ 0 and M[j][i] = 1 + M[j − V[i]][i] then
  |   | j = j − V[i]
  |   | Bocaux[i] = cpt + +
  |   |
  | else
  |   | i = i − 1
  |   | cpt = 1
  | end
end
return Bocaux
```

Cette boucle se termine car a chaque itération soit j est décrémentée, soit i est décrémentée. On en conclut que la compléxité temporelle de cette boucle est en $\mathcal{O}(k+s)$. Cette boucle n'affecte donc pas la compléxité temporelle car la complexité $\mathcal{O}(k \times S)$ domine.
Pour sa complexité spatiale on rajoute un tableau de taille k donc $\mathcal{O}(k + k \times S) = \mathcal{O}(k \times S)$.
On en conclut que ni la complexité temporelle ni la complexité spatiale ne change.

Question 6

Dans cet algorithme backward, on remonte progressivement vers le haut dans la matrice. On remarqueque $M[s, i]$ est égal soit à $M[s - V[i]][i + 1]$, soit $M[s, i - 1]$. Si j est égal à 0 ou à la dernière case du tableau, on est en $\mathcal{O}(1)$ et l'algorithme se termine (on a mis toute la confiture dans un seul bocal ou alors il n'y a pas de confiture). Si j n'est pas égal à 0 et si j n'est pas égal à la dernière case du tableau, on décrémente l'indice i du tableau des bocaux et on vérifie dans la matrice si on peut mettre une quantité j dans le bocal d'indice i, et ainsi de suite. On est donc dans le pire cas où il n'y aurait que des bocaux $V[1] = 1$ en $\mathcal{O}(j) - > \mathcal{O}(n)$. On a donc bien une complexité polynomiale.

Algorithme III : Cas particulier et algorithme glouton

Question 7

Algorithm 2 AlgoGlouton(k : entiers, V : tableau de k entiers, s : entier) : entier

```
i, NbCont, cpt : entiers
NbCont ← s
cpt ← 0
while NbCont > 0 do
    if V[i] ≤ s then
        NbCont ← NbCont − V[i]
        cpt ← cpt + 1
    else
        i ← i − 1
    end
end
return cpt
```

Dans le pire des cas l'algorithme itère la boucle s fois donc la complexité temporelle de *AlgoGlouton* est $\mathcal{O}(s)$.

Question 8

On cherche à montrer qu'il existe des systèmes qui ne sont pas glouton-compatibles. Le but est d'avoir le minimum de bocaux de bocaux de confiture à stocker. On suppose qu'on a un système de capacité tel que $V[1] = 1$, $V[2] = 101$, $V[3] = 200$, et qu'on a une quantité $s = 202$. Dans ce cas, l'algorithme glouton va choisir $V[3]$ en premier, puis deux fois $V[1]$. On a donc trois bocaux de pris. Or, ici, la solution optimale aurait été de prendre deux bocaux de 101 qui correspondent à $V[2]$. Il existe donc des systèmes de capacité qui ne sont pas glouton compatibles.

Question 9

Dans cette question on veut montrer par l'absurde que le système de capacité Expo est glouton-compatible.
Pour cela on considère $g = (g_1, g_2, ..., g_k)$, une solution produite par l'algorithme glouton pour le système Expo, et une solution optimale $o = (o_1, o_2, ..., o_k)$ que l'on suppose différente de g. On veut montrer que les deux solutions sont identiques.
Par l'absurde supposons qu'elles sont différentes.

a) Montrons qu'il existe un plus grand indice $j \in \{1, ..., k\}$ tel que $g_j > o_j$.

Soit j le plus grand entier appartenant à $\{1, ..., k\}$ tel que $g_j \neq o_j$.
Comme o est une solution optimale, elle utilise moins de bocaux que g, donc si un tel j existe et que $g_j \neq o_j$ alors forcement $g_j > o_j$.

b) Montrons que $\sum_{i=1}^{j} V[i]g_i = \sum_{i=1}^{j} V[i]o_i$

On a supposé que les deux solutions étaient différentes mais on sait qu'elles stockent la même quantité de confiture alors on a : $S = V[1]g_1 + V[2]g_2 + ... + V[k]g_k = V[1]o_1 + V[2]o_2 + ... + V[k]o_k$.
Donc $\sum_{i=1}^{k} V[i]g_i = \sum_{i=1}^{k} V[i]o_i$
Comme il existe un $j \in \{1, ..., k\}$, et que l'algorithme glouton utilise le maximum de bocaux de capacité i et que o est une solution optimale, alors pour tout i tel que $j + 1 < i < k$, on a $g_i = o_i$.

On a alors $\sum_{i=j+1}^{k} V[i]g_i = \sum_{i=j+1}^{k} V[i]o_i$.

Et donc $\sum_{i=1}^{j} V[i]g_i = \sum_{i=1}^{j} V[i]o_i$.

c)
$$\begin{aligned}
&\sum_{i=1}^{j-1} V[i]g_i + V[j]g_j = \sum_{i=1}^{j-1} V[i]o_i + V[j] + o_j \\
&\sum_{i=1}^{j-1} V[i]g_i = \sum_{i=1}^{j-1} V[i]o_i + V[j] + o_j - V[j]g_j \\
&\sum_{i=1}^{j-1} V[i]g_i = \sum_{i=1}^{j-1} V[i]o_i + V[j](g_j - o_j) \qquad (1)
\end{aligned}$$

Alors : $(1) \geq V[j](g_j - o_j) \geq V[j]$.
Et on a bien $\sum_{i=1}^{j} V[i]g_i = \sum_{i=1}^{j} V[i]o_i \geq V[j]$.

d) On suppose dans cette question que $o_i \leq d-1$ pour tout $i \in \{1, ..., j-1\}$.

$$\begin{aligned}
&o_i \leq (d-1) \\
&o_i V[i] \leq (d-1)V[i] \\
&\sum_{i=1}^{j-1} o_i V[i] \leq \sum_{i=1}^{j-1} (d-1)V[i] \\
&\qquad \leq (d-1)\sum_{i=1}^{j-1} V[i] \\
&\qquad \leq (d-1)\sum_{i=1}^{j-1} d^{i-1} \\
&\qquad \leq (d-1)\frac{d^{j-1}-1}{d-1} \\
&\qquad \leq d^{j-1} - 1 \\
&\qquad \leq V[j] - 1 \\
&\qquad < V[j]
\end{aligned}$$

On viens de montrer que $\sum_{i=1}^{j-1} o_i V[i] < V[j]$ alors que l'on a montré à la question *c)* que $\sum_{i=1}^{j-1} o_i V[i] \geq V[j]$. Contradiction.

Question 10

Montrons par l'absurde que tout système de capacité V avec $k = 2$ est glouton-compatible.
Supposons qu'il existe un système dont la solution optimale ne soit pas retournée par l'algorithme glouton.
On a alors $g = (g_1, g_2)$ une solution du nombre de bocaux de taille $V[1]$ et $V[2]$ rendue par l'algorithme glouton.
Et $o = (o_1, o_2)$ une solution optimale.
L'algorithme glouton remplit les bocaux les plus grands tant qu'il le peut, et comme $o \neq g$ car les deux algorithmes sont différents, on a $g_2 > o_2$.
Or si $o_2 < g_2$, alors $o_1 = g_1 + (g_2 - o_2)V[2]$.
Posons $h = (g_2 - o_2)$V[2].
On remplit les bocaux de taille $V[2]$ qui n'ont pas été remplit par la solution optimale par des bocaux de taille 1.
Or $g_2 - o_2 \geq 1$ et $V[2] > 1$
On en déduit que $h > 1$ et donc que l'on a $g_1 + g_2 < o_1 + o_2$
On a alors une contradiction, donc $g = o$ et il n'existe donc pas de solution meilleure. Alors un système de capacité V avec $k = 2$ est glouton-compatible.

Question 11

Dans le meilleur des cas, on a un $k < 3$ et dans ce cas on retourne directement vrai. On est donc en $\mathcal{O}(1)$.
Si $k > 3$, on va faire varier S dans la boucle principale. De plus, on va appeler $2 \times k$ l'algorithme glouton sur la valeur courante de S. On rappelle que la complexité de l'algorithme glouton est en $\mathcal{O}(n)$. On a donc $\mathcal{O}(2 \times k \times n) \times \mathcal{O}(n) = \mathcal{O}(n^2)$. On a donc bien une complexité polynomiale.

Deuxième partie

Mise en œuvre

Implémentation

Analyse de complexité expérimentale

Question 12

Dans le cas de l'algorithme de recherche, on a vu que la complexité était $\mathcal{O}(2^n)$. En réalisant différents tests en faisant varier s, en laissant le système de capacité défini au début du projet, on constate que la complexité est bien exponentielle.

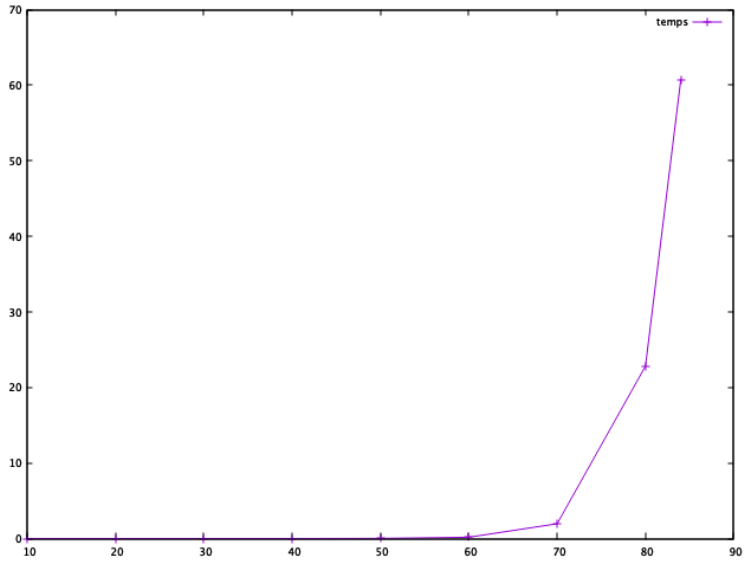

FIGURE 1 – Variation de s uniquement

Lorsqu'on utilise le système de capacité avec $d = 2$, $d = 3$ ou $d = 4$, en augmentant k jusqu'à 13 par exemple, on constate que la complexité est toujours exponentielle.

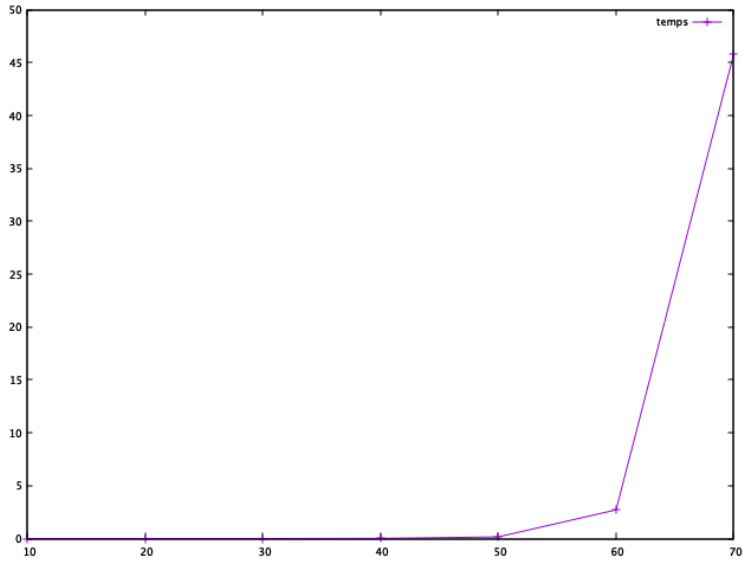

FIGURE 2 – Graphe pour $d = 3$

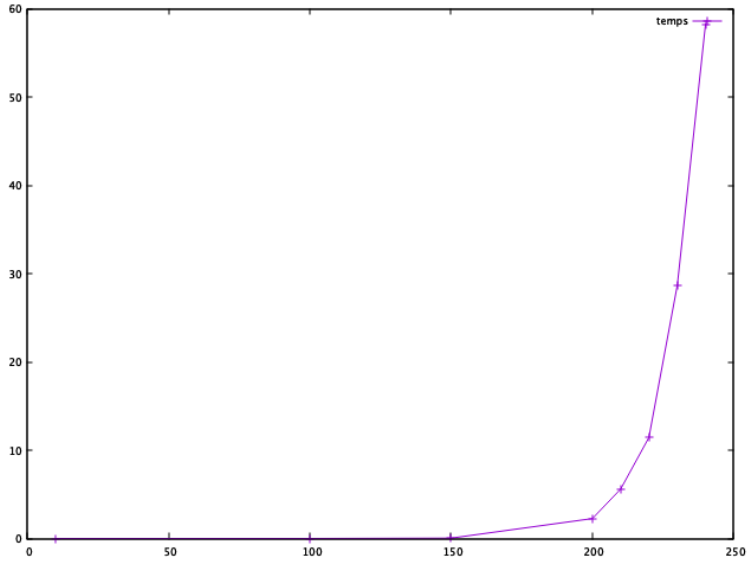

FIGURE 3 – Graphe pour $d = 4$

Dans ces deux cas, on ne dépasse pas $s = 70$ environ car le temps est ensuite beaucoup trop élevé. Lorsqu'on fixe s à par exemple 100 et qu'on utilise les systèmes de capacité $d = 2$, $d = 3$ et $d = 4$, on voit que l'on met beaucoup moins de temps à faire tourner cet algortithme et qu'il est possible d'augmenter encore s et que le temps d'exécution soit encore acceptable.
On en déduit que la taille k influe sur la durée d'exécution, ainsi que la capacité des bocaux, étant elle aussi exponentielle.

En revanche, pour un s fixé à 25, si on fait varier k, on obtient une courbe de cette allure.

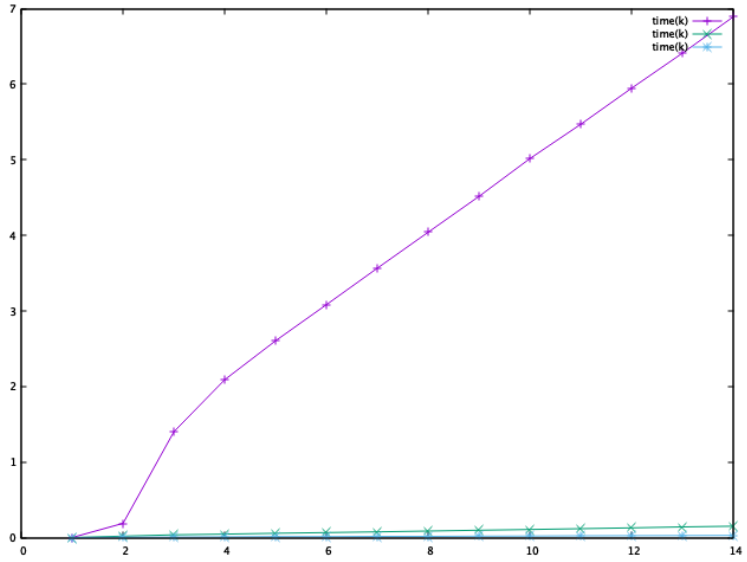

FIGURE 4 – s fixé et k varie

Ici, on observe qu'on a un temps linéaire pour les trois courbes (il faudrait zoomer davantage et tester pour des k beaucoup plus grands pour d = 3 et d = 4).
Dans le cas de l'algorithme dynamique (II), on a une complexité en $\mathcal{O}(n)$. Même en faisant varier les capacités des bocaux d'après le système $d = 2$, $d = 3$, $d = 4$, on a toujours la même forme

de courbe. La capacité des bocaux ou la quantité de confiture ne change pas le comportement de l'algorithme.

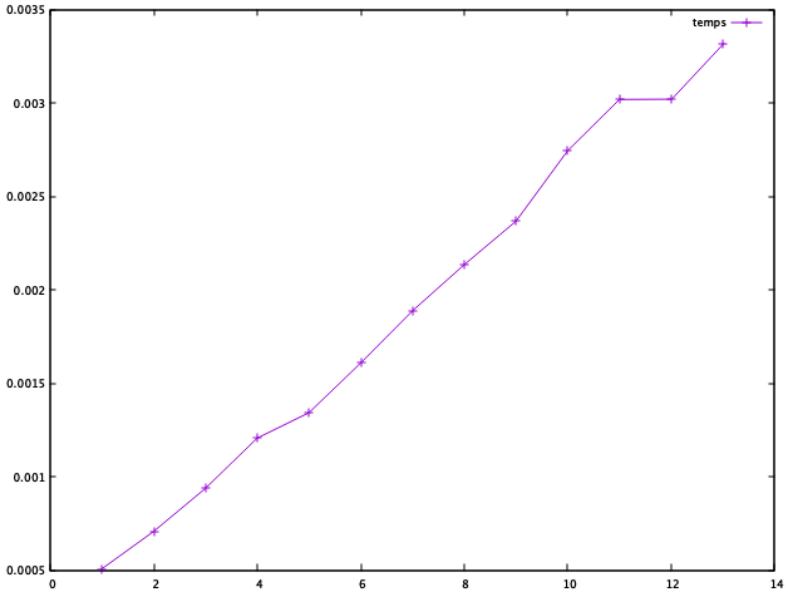

FIGURE 5 – s fixé et k varie pour $d = 2$

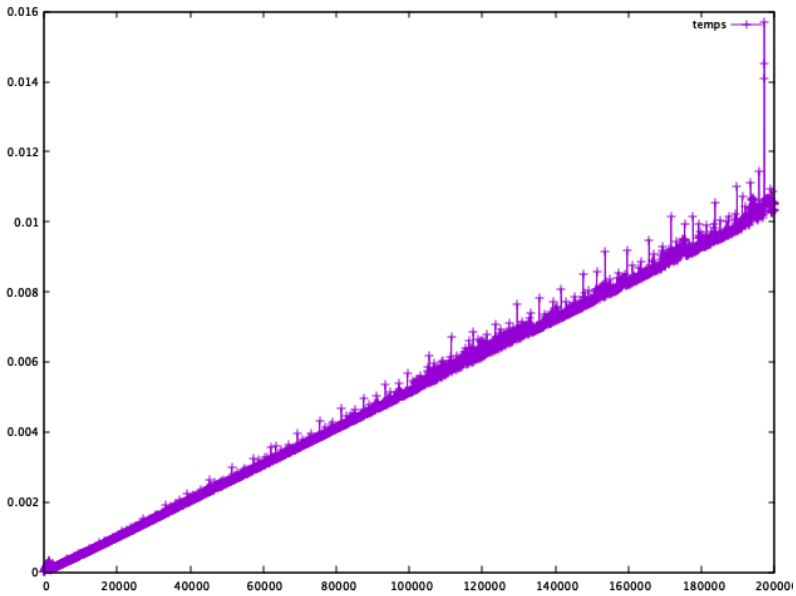

FIGURE 6 – k fixé et s varie pour $d = 2$

Dans le cas de l'algorithme glouton (III), on a également une complexité en $\mathcal{O}(n)$. Lorsqu'on lui met un k petit, avec peu de valeurs, par exemple $k = 3$, que ce soit pour $d = 2$, $d = 3$ ou $d = 4$, on peut voir cette courbe On en déduit que quand on donne à cet algorithme un système de capacité avec le plus grand bocal qui est largement inférieur à s, il met un temps linéaire à s'exécuter de plus en plus important.

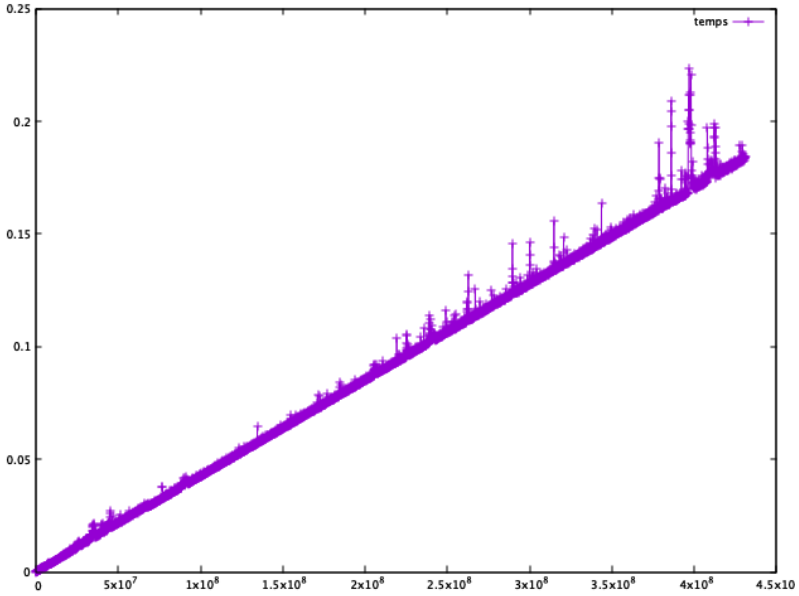

FIGURE 7 – Graphe pour $k = 3$ et $d = 3$

En revanche, lorsqu'on lui donne un grand k, avec un très grand plus grand bocal ($k = 15$) et qu'on fait varier s, le temps est quasiment constant et beaucoup plus court que lorsqu'on met un système de capacité avec des grandes valeurs.

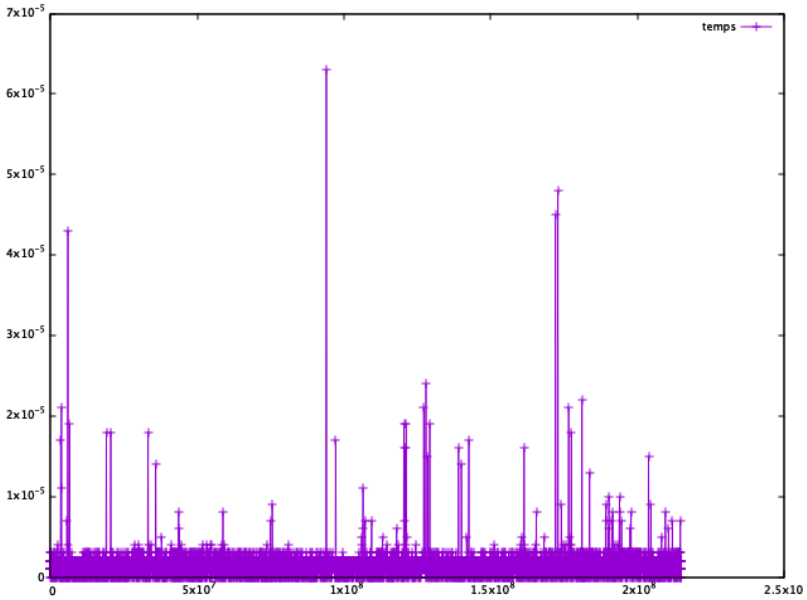

FIGURE 8 – Variation de s pour $k = 15$ et $d = 4$

Utilisation de l'algorithme glouton

Question 13

Fixons $k = 8$ la taille du tableau V. On s'assure que $V[0] = 1$ et que le tableau soit correctement trié avant d'effectuer le test.
Peu importe qu'on ait des doublons, étant donné que le tableau est trié, l'algorithme glouton passera au bocal suivant et ne modifiera pas la valeur du nombre minimal de bocaux, ce qui permet bien d'avoir une solution valide même en cas de doublons dans le tableau (dans le cas où le système serait effectivement glouton-compatible).
Choisissons par exemple une taille maximale de bocaux égale à 10. On va donc tirer aléatoirement des bocaux entre 2 et 10. Dans ce cas là, pour 100000 tirages, on a une fréquence d'apparition de systèmes glouton-compatibles d'environ $0,09$.
Faisons varier la taille des bocaux en laissant toujours $k = 8$ et 100000 tirages.
Avec une taille maximale de bocaux égale à 15, on a une fréquence d'apparition de systèmes glouton-compatibles d'environ $0,025$. On constate déjà que la fréquence a beaucoup chuté alors que l'on a peu augmenté la capacité maximale des bocaux.
Avec une taille maximale de bocaux égale à 20, on a une fréquence d'apparition de systèmes glouton-compatibles d'environ $0,01$.
Avec une taille maximale de bocaux égale à 30, on a une fréquence d'apparition de systèmes glouton-compatibles d'environ $0,002$.
Avec une taille maximale de bocaux égale à 40, on a une fréquence d'apparition de systèmes glouton-compatibles d'environ $0,0008$.

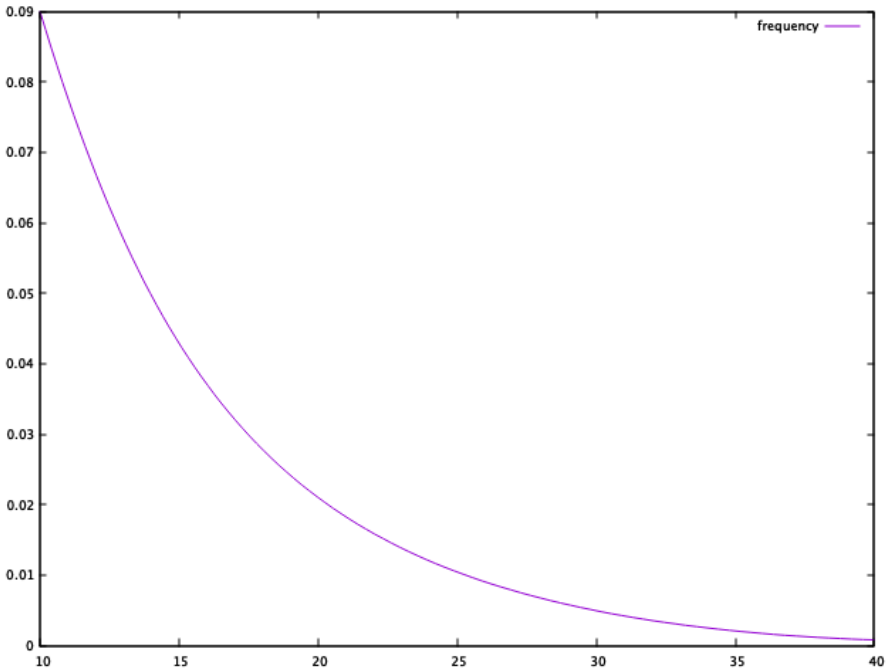

FIGURE 9 – Fréquence d'apparition de systèmes glouton-compatible en fonction de la taille maximale des bocaux

On constate qu'avec un k moyen et une capacité maximale faible, la fréquence des systèmes glouton-compatible décroit très vite.

Faisons maintenant varier k et fixons la capacité maximale des bocaux à 20.
On a démontré précédement que tout système avec $k = 2$ était glouton-compatible. On va donc commencer avec $k = 3$.
Avec une taille $k = 3$, on a une fréquence d'apparition de systèmes glouton-compatibles d'environ $0,44$. Première remarque, avec un k petit et une taille de bocaux maximale petite, on a quasiment deux chances sur cinq voire trois chances sur cinq d'obtenir un système glouton-compatible.
Avec une taille $k = 4$, on a une fréquence d'apparition de systèmes glouton-compatibles d'environ $0,18$.
Avec une taille $k = 5$, on a une fréquence d'apparition de systèmes glouton-compatibles d'environ $0,078$.
Avec une taille $k = 6$, on a une fréquence d'apparition de systèmes glouton-compatibles d'environ $0,03$.
Plus on augmente le k, plus la fréquence d'apparition des systèmes glouton-compatibles est faible avec une taille maximale de bocaux fixée. A partir d'une taille maximale grande (100 environ), la fréquence approche beaucoup de 0 et l'ordinateur n'affiche pas assez de décimales.

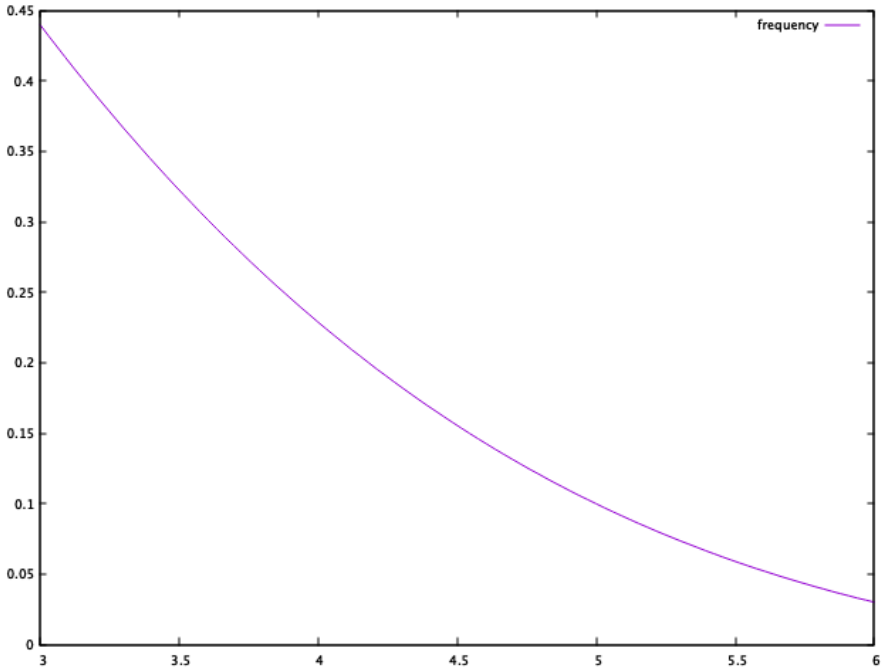

FIGURE 10 – Fréquence d'apparition de systèmes glouton-compatible en fonction de k

Si on choisit un grand k et une grande taille maximale de bocaux, là encore il y a très peu de chances de trouver un système glouton-compatible même sur un grand nombre de tirages.
Autre fait observé : fixons à 15 par exemple la taille maximale des bocaux, et faisons varier k, de 1 à 100 par exemple. On peut constater que comme dit plus haut, la fréquence diminue énormément et très vite jusqu'à s'approcher de 0. Or, lorsqu'on commence à s'approcher de $k = 50$, la fréquence remonte beaucoup jusqu'à s'approcher des 100%. Cela s'explique par le fait que, plus le tableau est grand, la taille maximale des bocaux étant fixe, on a plusieurs fois la même valeur dans le tableau et dans ce cas, le système est forcément glouton-compatible puisqu'on a tiré toutes les valeurs ou presque comprises entre 2 et la taille maximale des bocaux.
Réduisons maintenant à 8 la taille maximale des bocaux, et faisons de nouveau varier k de 1 à 100. On constate que la fréquence remonte beaucoup plus vite et que cette fois on atteint vraiment 100% de temps en temps.

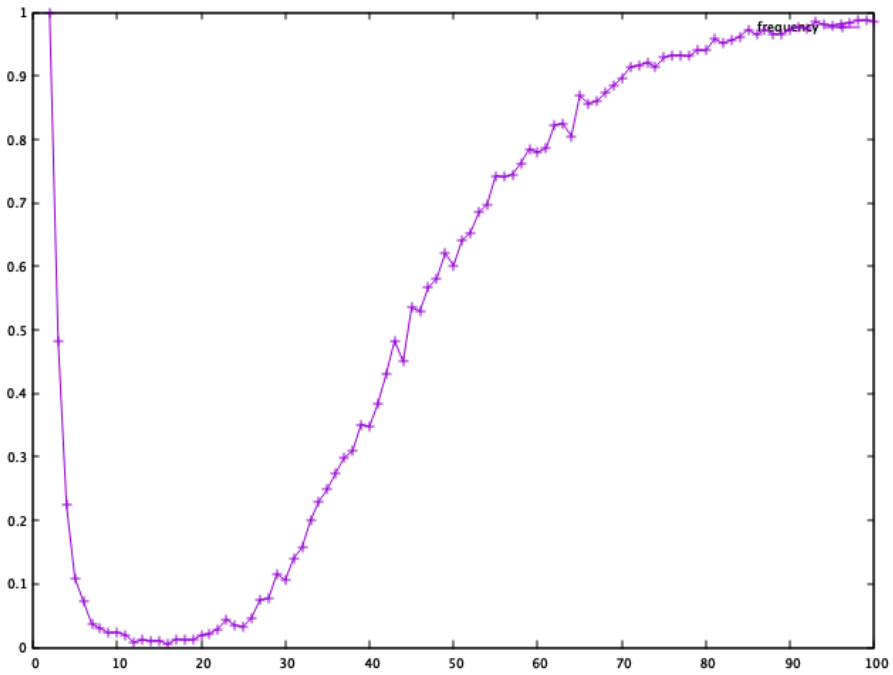

FIGURE 11 – Fréquence d'apparition de systèmes glouton-compatible en fonction de k pour une taille de bocal maximale égale à 15

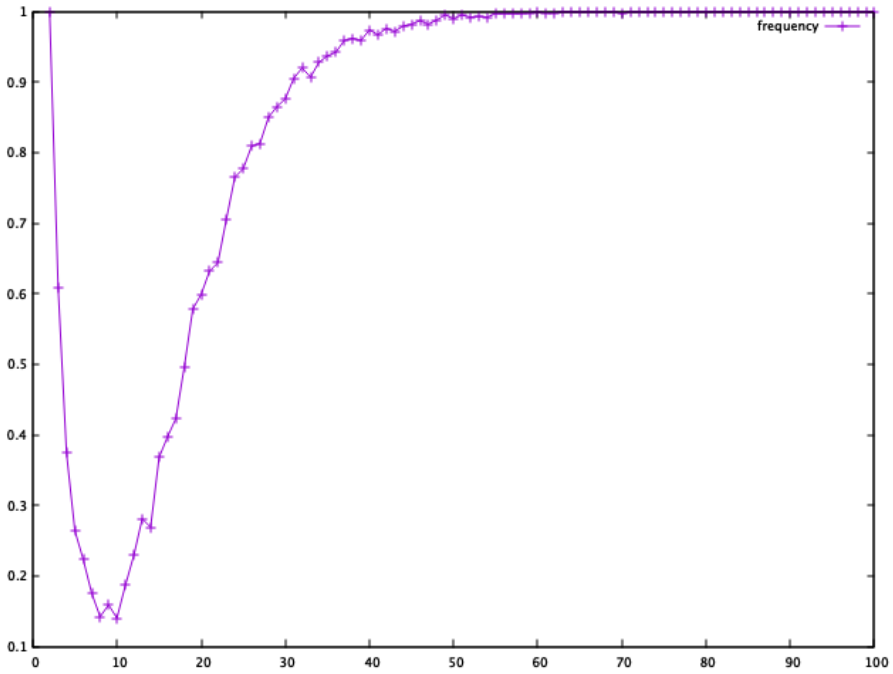

FIGURE 12 – Fréquence d'apparition de systèmes glouton-compatible en fonction de k pour une taille de bocal maximale égale à 8

Question 14

Pour cette question, on décide de fixer $pmax$ à 5 d'abord, puis $f = 5$. On cherche à calculer le pire écart obtenu entre l'algorithme dynamique et l'algorithme glouton. On fait varier $pmax = 10$, puis $pmax = 15$ (on ne touche pas à f). On obtient la courbe suivante.

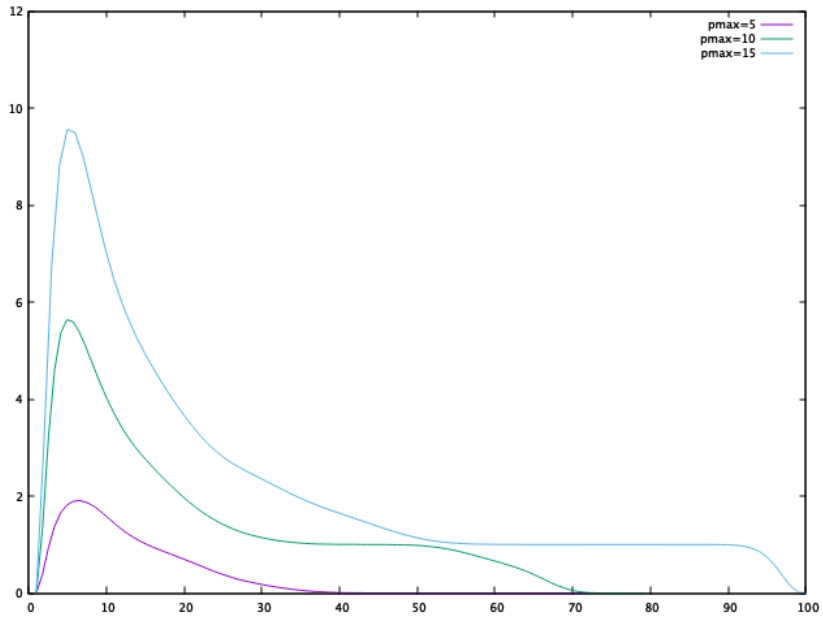

FIGURE 13 – Ecart moyen entre la solution optimale et celle renvoyée par l'algorithme glouton

On peut voir en analysant cette courbe que, lorsqu'on a beaucoup de systèmes qui ne sont pas glouton-compatibles, la solution optimale est relativement éloignée de celle renvoyée par l'algorithme glouton, ce qui fait qu'on a un grand écart moyen. En revanche, comme vu dans la question précédente, quand la fréquence de systèmes glouton-compatibles augmente beaucoup, on a un écart moyen qui diminue beaucoup et rapidement, car la solution optimale est exactement celle de l'algorithme glouton ou très proche, ce qui est logique.
Pour la moyenne des écarts moyens, la courbe a à peu près la même allure, et ce pour les mêmes raisons que précédemment.

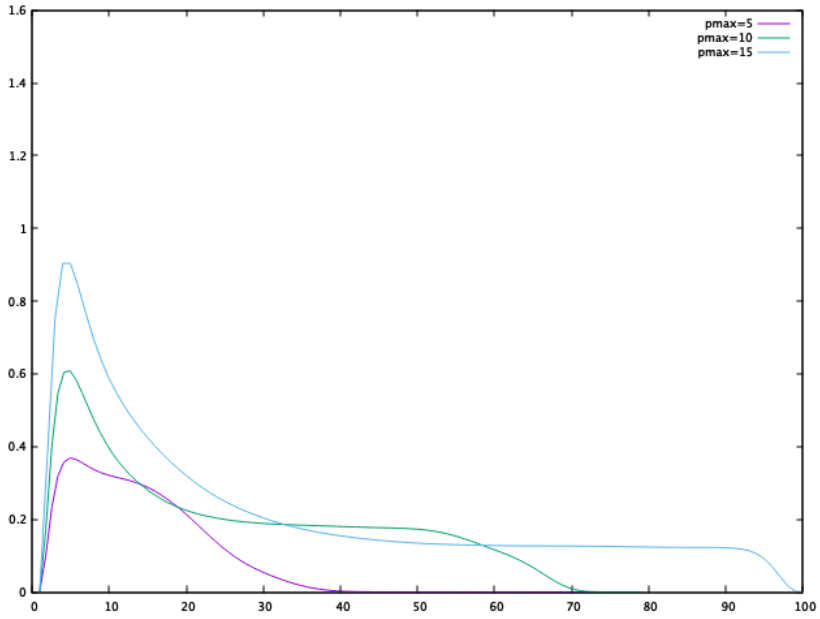

FIGURE 14 – Moyenne des écarts moyens entre la solution optimale et celle renvoyée par l'algorithme glouton

Troisième partie

Annexe : Code des fonctions principales

```
int rechercheExhaustive(int k, int V[], int s){
    cpt++;
    if(cpt == 2147483647){
        cpt = 0;
        cpt2++;
    }
    if(cpt%10000000== 0){
        printf("%d %d\n", cpt/10000000, cpt2);
    }
    int i = 0;
    int NbCont = 0;
    int x = 0;

    if(s < 0){
        return INF;
    }
    else{
        if(s == 0){
            return 0;
        }
        else{
            NbCont = s;
            for(i = 0; i<k; i++){
                x = rechercheExhaustive(k, V, s-V[i]);
                if(x+1 < NbCont){
                    NbCont = x+1;
                }
            }
            return NbCont;
        }
    }
}
```

```
    int AlgoProgDyn(int k, int V[], int s, int Bocaux[]) {
        int i, j;
        int M[s][k];
        for(i = 1; i <= s; i++){
            for(j = 1; j <= k; j++){
                int v1 = 0;
                int v2 = 0;
                int v3 = 0;
                if(j-1 == 0){
                    v1 = INF;
                }
                else{
                    v1 = M[i-1][j-2];
                }
                int quantite = i - V[j-1];
                if(quantite <= 0){
                    if(quantite == 0){
                        v3 = 0;
                    }else{
                        v3 = INF;
                    }
                }else{
                    v3 = M[quantite-1][j-1];
                }
                v2 = v3 + 1;
                M[i-1][j-1] = min(v1,v2);
            }
        }
        for(i = 0; i < k; i++){
            Bocaux[i] = 0;
        }
        i = k;
        j = s;
        int cpt = 1;

        while(j > 0){
            if(j == 1){
                j = 0;
                Bocaux[0]++;
            }else if(j == V[i-1]){
                j = 0;
                Bocaux[i-1]++;
            }else if(j - V[i-1] >= 0 && M[j-1][i-1] == (1 + M[j - V[i-1]-1][i-1])){
                j = j - V[i-1];
                Bocaux[i-1] = cpt++;
            }
            else{
                i = i-1;
                cpt = 1;      /* on remet cpt 1 */
            }
    }
    return M[s-1][k-1];
}
```

```
int AlgoGlouton(int s, int V[], int k){
    int liste_bocaux[k];
    int j;
    int cpt = 0;
    for(j = 0; j < k; j++){
        liste_bocaux[j] = 0;
    }
    int i = k - 1;
    while(s > 0){
        if(V[i] <= s){
            s -= V[i];
            liste_bocaux[i]++;
            cpt++;
        }
        else{
            i--;
        }
    }
    printf("(%d, [", cpt);
    for(int index = 0; index < k; index ++){
        if(index == k - 1){
            printf("%d", liste_bocaux[index]);
        }
        else printf("%d, ", liste_bocaux[index]);
    }
    printf("])");
    return cpt;
}
```