Projet d'algorithmique

CONFITURES

LU3IN003

Compte-rendu du projet

UNG Thierry

Table des matières

Première partie

Partie théorique

Algorithme I

Question 1

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 que pour tout $i \in \{1, \ldots, k\}$

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

- Si $s = 0$:

Alors on n'a pas de confiture donc quelque soit i, $m(s, i) = 0$

- Si $s = 0$:

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 2

```
Algorithm 1(k : entier, V : tableau de k entiers, s : entier) : entier
i, Cst, x : entiers
if s<0 then
        return+∞
else
        if s=0 then
                return 0
        else
                Cst< - s
                for i de 1 à k do
                   x ← Algorithm 1(k, V, s - V[i])
                   if x + 1 < Cst then
                      Cst← x+1
                   end if
                end for
                return Cst
        end if
end if
```

Algorithme II

Question 5

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 2 AlgoOptimise(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 $O(k \times S)$, c'est-à-dire la taille du tableau doublement indicé. Et sa complexité temporelle est aussi en $O(k \times S)$, car une opération par case pour calculer sa valeur.

Question 6

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 2b) :

```
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
```

c) 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 $O(k + s)$. Cette boucle n'affecte donc pas la compléxité temporelle car la complexité $O(k \times S)$ domine. Pour sa complexité spatiale on rajoute un tableau de taille k donc $O(k + k \times S) = O(k \times S)$.
On en conclut que ni la complexité temporelle ni la complexité spatiale ne change.

Question 7

Dans cet algorithme backward, on remonte progressivement vers le haut dans la matrice. On remarque que $M[s, i]$ 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 (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 $(j) > (n)$. On a donc bien une complexité polynomiale.

Algorithme III : Cas particulier et algorithme glouton

Question 8

Algorithm 3 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 $O(s)$.

Question 9

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 $V[1] = 25$, $V[2] = 100$, $V[3] = 150$, et qu'on a une quantité $s = 200$. 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 100 qui correspondent à $V[2]$. Il existe donc des systèmes de capacité qui ne sont pas glouton compatibles.

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 $o = g$ 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

L'algorithme TestGloutonCompatible est composé de deux boucles imbriquées, on remarque que la première est bornée par $V[k]$ donc la première boucle est en $O(V[k])$ La deuxième itère de 1 à k donc elle est en $\Theta(k)$, à l'intérieur de cette boucle, on effectue un test dans lequel il y'a deux appels à AlgoGlouton, qui est en $O(S)$, donc on effectue k fois 2 appels à AlgoGlouton, la complexité de la deuxième boucle «for» est donc en $O(k*S)$. On en déduit que l'algorithme TestGloutonCompatible est en $O(V[k]*k*S)$.

Cependant, on ne peut pas dire que l'algorithme est polynomial, en effet, il est pseudo-polynomial car la complexité d'un algorithme est calculée en fonction de la taille de codage de l'entrée et qui est logarithmique en la valeur de cette entrée. De plus, rien ne nous empêche d'avoir $V[k] \gg k$. Donc l'algorithme TestGloutonCompatible est pseudo-polynomial.

Deuxième partie

Mise en œuvre

Implémentation

Analyse de complexité expérimentale

Question 12

Dans le cas de l'algorithme II, on a une complexité en (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 del'algorithme.

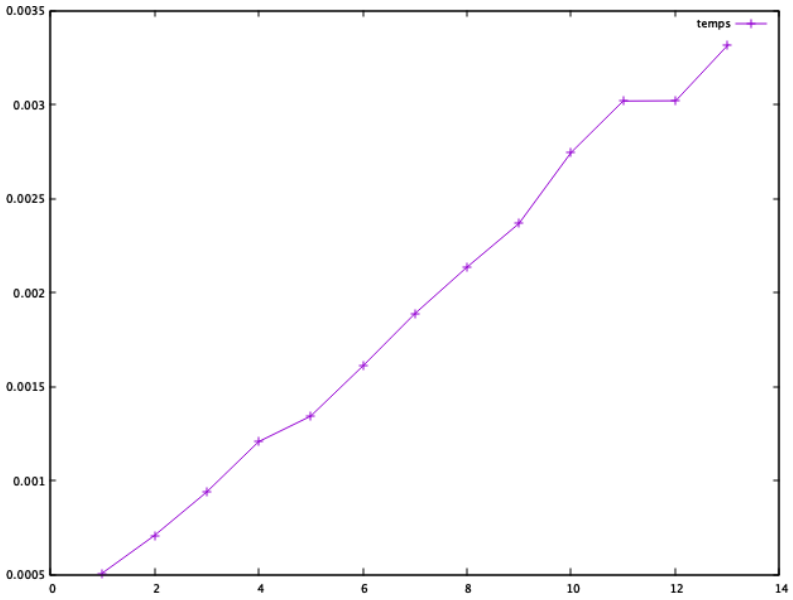

FIGURE 1 – s fixé et k varie pour $d = 2$

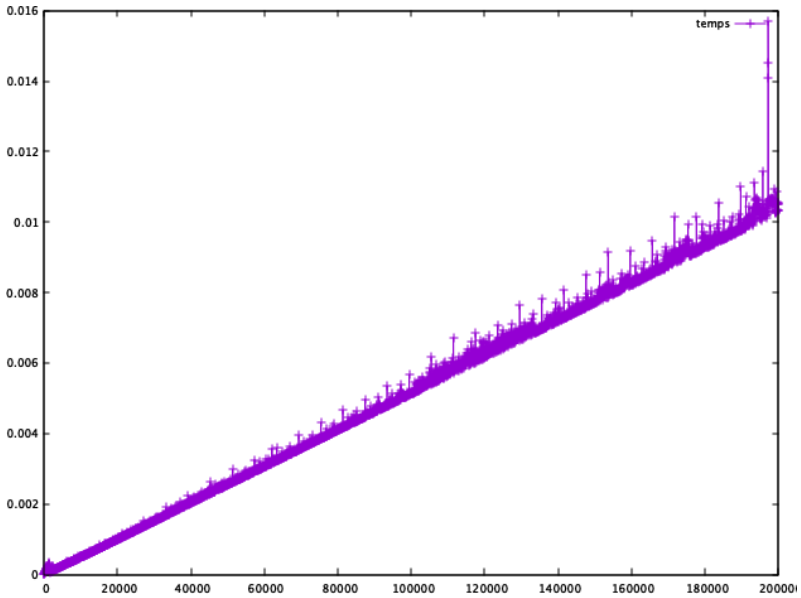

FIGURE 2 – k fixé et s varie pour $d = 2$

Dans le cas de l'algorithme glouton (III), on a également une complexité en 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.

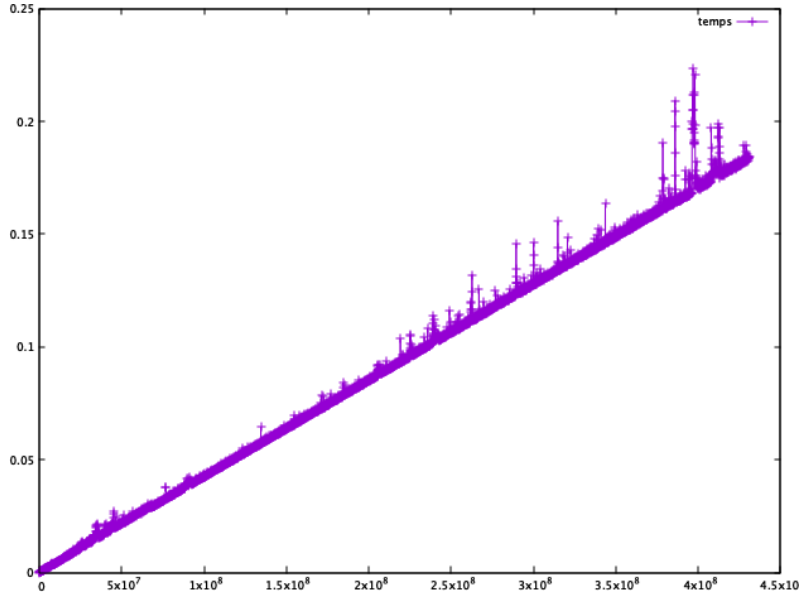

FIGURE 3 – Graphe pour $k = 3$ et $d = 3$

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.

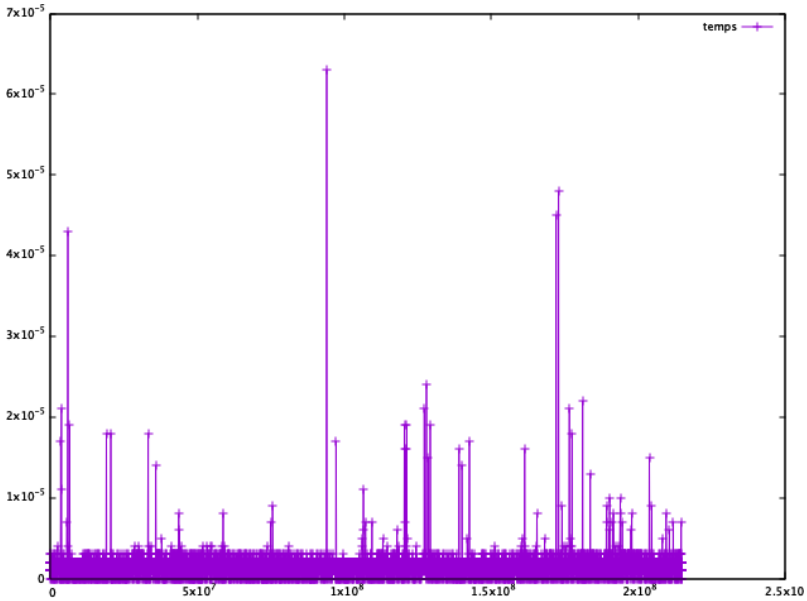

FIGURE 4 – Variation de s pour $k = 15$ et $d = 4$

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 $0,0008$.

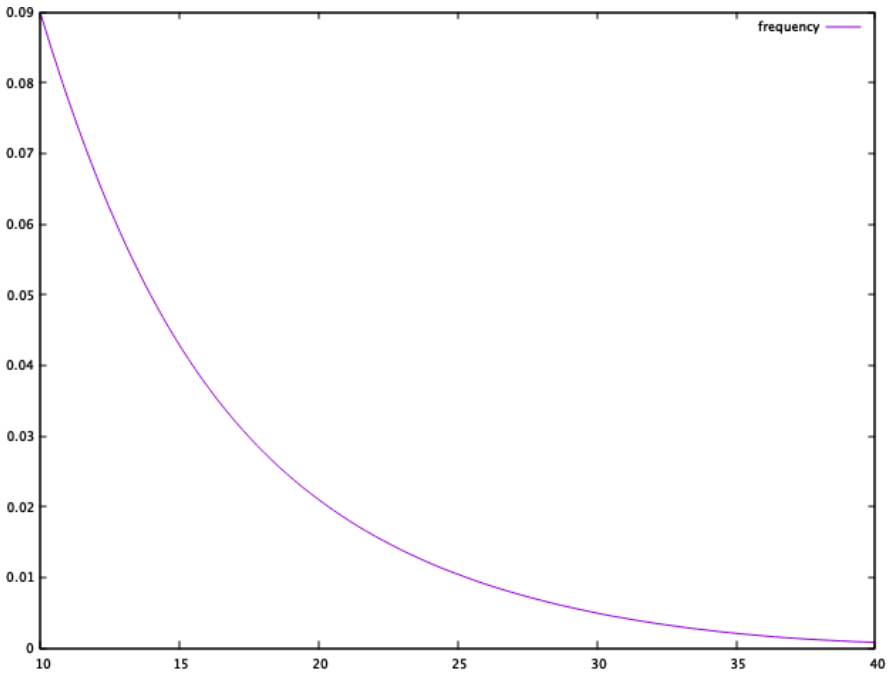

FIGURE 5 – Fréquence d'apparition de systèmes glouton-compatible en fonction de la taille maximale des bocaux

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.

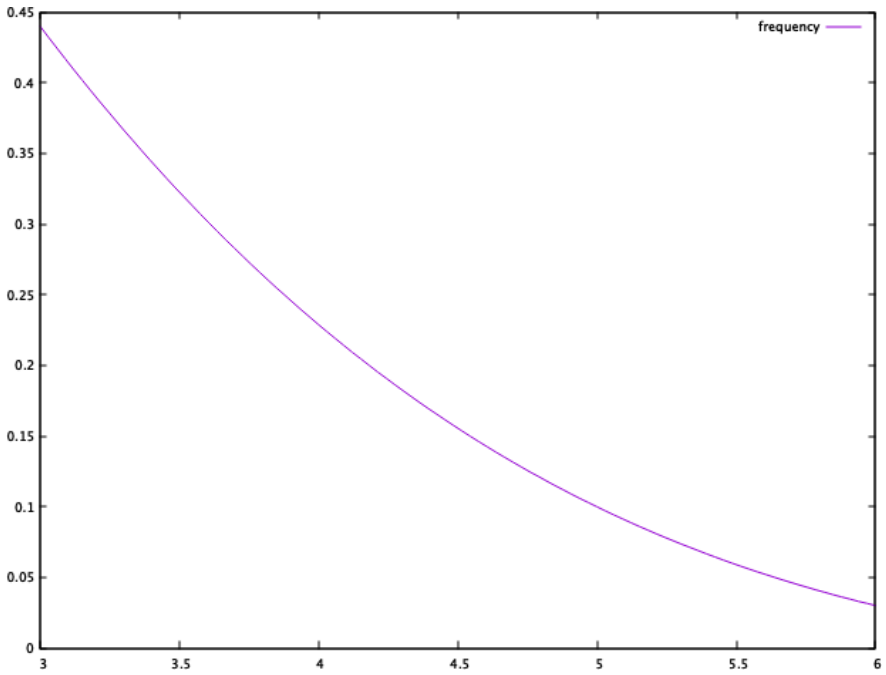

FIGURE 6 – Fréquence d'apparition de systèmes glouton-compatible en fonction de k

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.

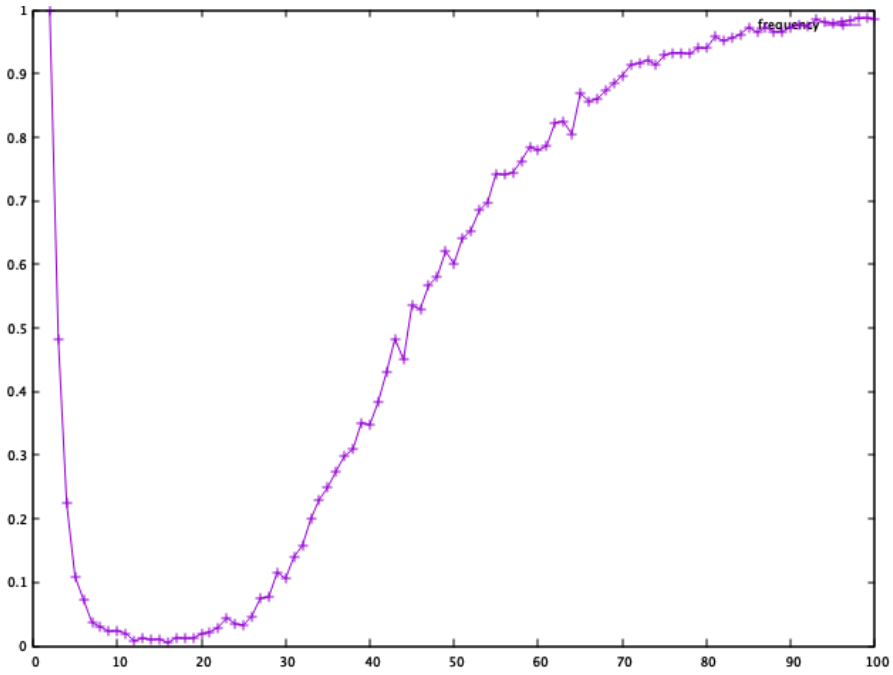

FIGURE 7 – Fréquence d'apparition de systèmes glouton-compatible en fonction de k pour une taille de bocal maximale égale à 15

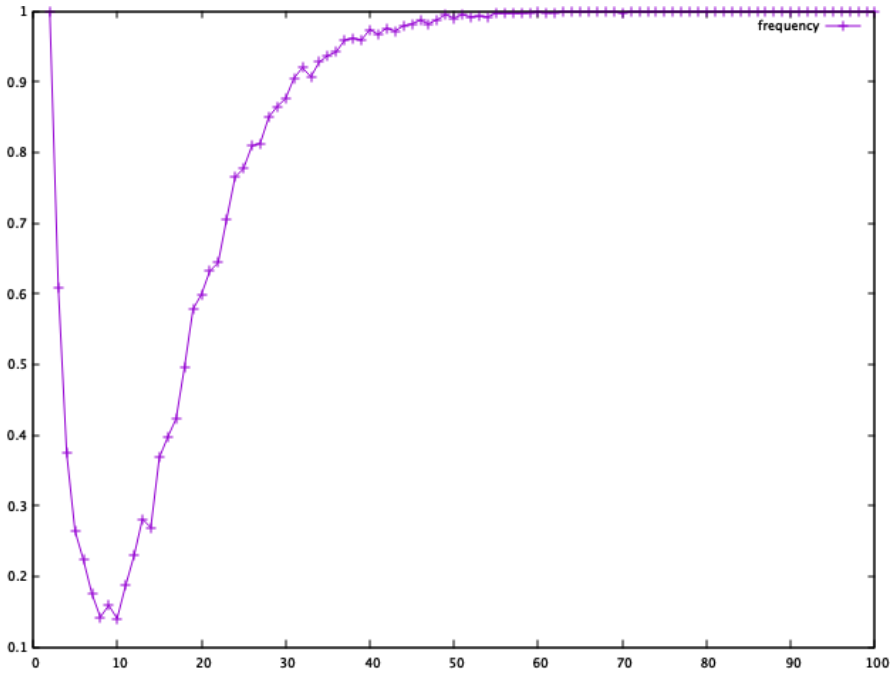

FIGURE 8 – Fréquence d'apparition de systèmes glouton-compatible en fonction de k pour une taille de bocal maximale égale à 8

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.

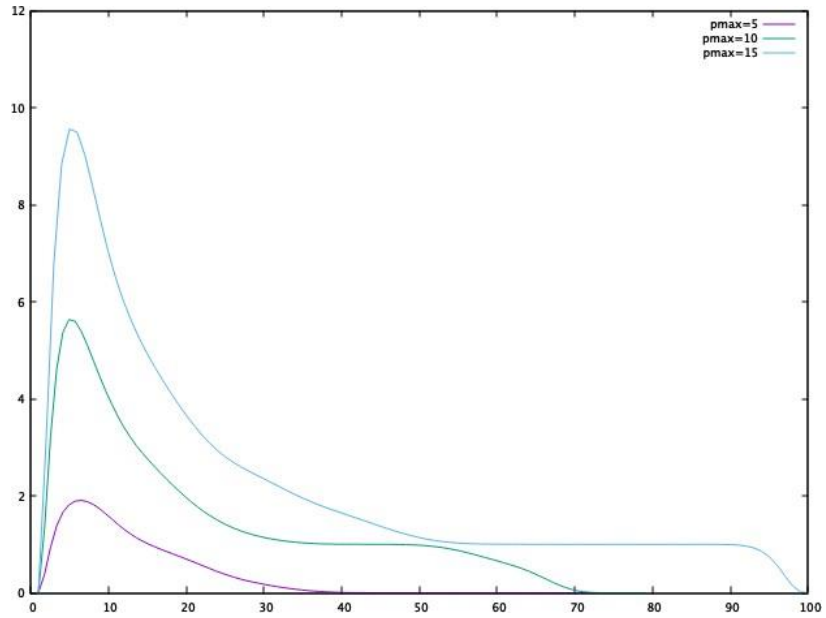

FIGURE 9 – Ecart moyen entre la solution optimale et celle renvoyée par l'algorithme glouton

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.

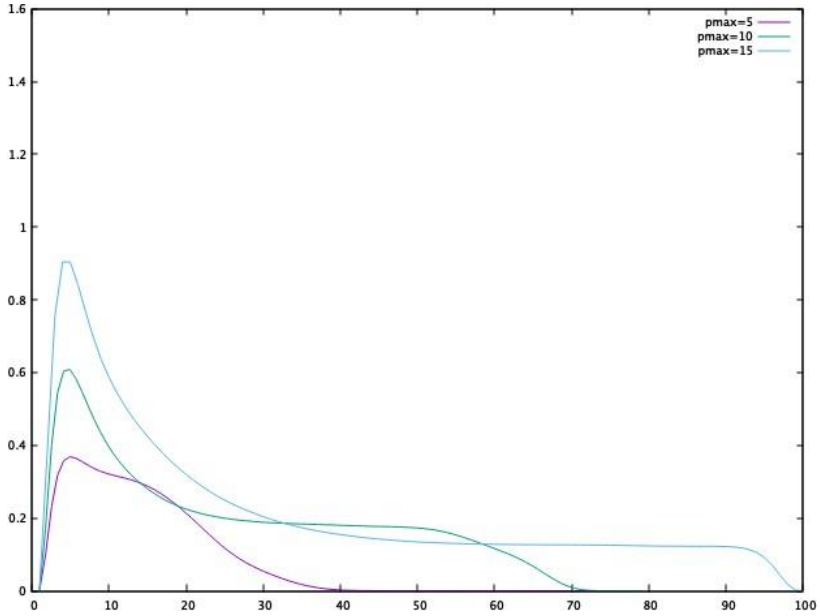

FIGURE 10 – Moyenne des écarts moyens entre la solution optimale et celle renvoyée par l'algorithmeglouton