TP Méthode approchées UMIN215

Bruno Y., Chlöé T., Julien D. et Rémi F.

14 avril 2015

1 Partie Théorique

1.1 Programmation linéaire en nombres entiers

Exercice 1 - Sur le problème de la couverture sommet minimale : trois approches différentes

1. a) On a :

$$x_i = \begin{cases} 0 \text{ si } x_i \notin S \\ 1 \text{ si } x_i \in S \end{cases}$$

On peut donc déduire que le (PL) donne bien une solution optimale. z compte bien le nombre de sommets dans S. Si $(v_r, v_s) \in E$, on a x_r et x_s qui valent 1 : il faut qu'il y ait au moins un sommet couvre l'arête.

b) On ne peut pas avoir $x_s + x_r = 1$ car v_r et v_s peuvent tout deux appartenir à S.

c) Soit une solution optimale, alors elle respecte les contraintes du (PL) et correspond à un z minimum : elle correspond donc à une solution du (PL).

d) Supposons qu'il existe $(v_r, v_s) \in E$ tel que $x_r < \frac{1}{2}$ et $x_s < \frac{1}{2}$, alors on aurait $x_r + x_s < 1$, ce qui viole une contrainte. Donc $x_r \geq \frac{1}{2}$ ou $x_s \geq \frac{1}{2}$.

e) Soit I une instance quelconque, alors on a $\rho = \frac{max(A(I))}{Opt(I)}$. Par ailleurs, on sait que :

$$Opt(I) = S_{ILP}(I) \tag{1}$$

$$S_{LP}(I) \leq S_{ILP}(I) \tag{2}$$

On peut déduire de (2) que $\frac{1}{S_{LP}(I)} \geq \frac{1}{S_{ILP}(I)}$. Donc on arrive à l'inéquation suivante : $\frac{A(I)}{S_{LP}(I)} \geq \frac{A(I)}{Opt(I)}$. Notons que $A(I)$ est la solution obtenue en arrondissant la solution de $S_{LP}(I)$.
On peut remarquer que puisqu'il existe i tel que $x_i \geq \frac{1}{2}$, donc $2x_i \geq 1$.
On peut donc écrire :

$$A(I) = \sum_{i, x_i \geq \frac{1}{2}} 1 \leq \sum_{i \in S^*} 2x_i \leq 2 \sum_{i=1}^{n} x_i = 2Opt(I). \tag{3}$$

On déduit finalement que $\frac{A(I)}{Opt(I)} = 2$ et $\rho = \frac{max(A(I))}{Opt(I)} \leq 2$ Montrons que $\rho = 2$ en considérons le cas suivant :

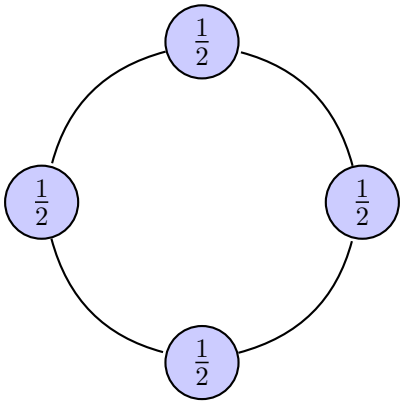


FIGURE 1 – Résultat donné par l'Algorithme 1 pour le problème Vertex cover.

On peut remarquer qu'ici, $A(I) = 4$ alors que $Opt(I) = 2$.

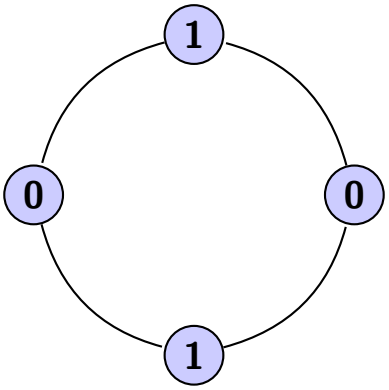


FIGURE 2 – Résultat optimal pour le problème Vertex cover.

On peut donc finalement conclure que $\rho = \frac{max(A(I))}{OptI} = 2$.

f) i) Pour obtenir notre nouvel programme linéaire (PL'), il suffit simplement d'introduire les poids $w_i \in \mathbf{N}^*, i \in \{1, \ldots, n\}$ et de poser $z = \sum_{j=1}^{n} w_i x_i$.

$$PL' \begin{cases} \min z = \sum_{j=1}^{n} w_i x_i \\ x_r + x_s \geq 1, \forall \{v_r, v_s\} \in E \\ x_j \in \{0, 1\}, j \in \{1, \ldots, n\} \end{cases}$$

ii) On peut remplacer le programme linéaire en nombre entiers (PL') par un autre utilisant une matrice A. Pour un graphe $G = (V, E)$, la matrice A utilisé dans le programme linéaire en nombres entiers possède les caractéristiques suivantes :
— A à $2 * |E|$ lignes et $|V|$ colonnes.
— Chaque ligne apparaît deux fois dans la matrice.
— La matrice A est constituée uniquement de 1 et il n'y a que deux 1 par ligne.

$$PL' \begin{cases} \min z = \sum_{j=1}^{n} w_i x_i \\ A * \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \geq \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}. \\ x_j \in \{0, 1\}, j \in \{1, \ldots, n\} \end{cases}$$

iii) A terminer..

2. a) Dans les pires des cas, les deux sommets de chaque arêtes sont pris : $A(I) \geq 2 * m$ avec $m = |E|$. De plus, il y a toujours, au moins un sommet par arête dans la solution optimale : $Opt(I) \geq n$. On peut donc écrire que $\frac{1}{Opt(I)} \leq n$ et donc que $\frac{A(I)}{Opt(I)} \leq 2$.

b) Considérons le graphe suivant :

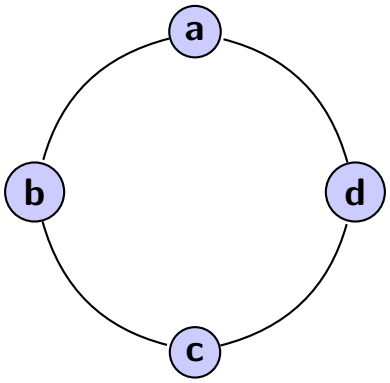


FIGURE 3 – Exemple d'instance pour lequel l'Algorithme 2 atteint la borne de deux.

On a donc $E' = (\{a, b\}, \{a, d\}, \{b, c\}, \{c, d\})$.

étape	E'	arête sélectionné	C
1	{ {a,b}, {a,d}, {b,c}, {c,d} }	{a,b}	{ {a,b} }
2	{ {c,d} }	{c,d}	{ {a,b}, {c,d} }
3	$\emptyset$	$\emptyset$	{ {a,b}, {c,d} }

On à ici $A(I) = 4$ alors que $Opt(I) = 2$ donc $\rho = \frac{max(A(I))}{Opt(I)} = 2$.

c) On commence par prendre les $\frac{k!}{k}$ sommets de degrés k (les triangles), puis les $\frac{k!}{k-1}$ sommets de degrés $k-1$, etc.. On va donc prendre en tout $k! \sum_{i=0}^{k} \frac{1}{k-i} = k! \sum_{i=1}^{k} \frac{1}{i} = k! * H_k$, avec H_k la k-ième somme partielle de la série harmonique. L'optimal est de prendre que les $k!$ sommets de degrés k (les carrés). On a donc pour un k fixé, $\frac{A(I)}{Opt(I)} = H_k$. Or si $k \to \infty$, H_k diverge et tend vers ∞.

Exercice 2 - Sur le problème du couplage maximum de poids maximum : un début d'étude polyédrale sur le problème de couplage

Dans cet exercice, nous travaillerons sur un graphe $G = (V, E)$ avec $|V| = n$ et $|E| = m$. Chaque arête $(i, j) \in E$ possède un poids que l'on not $w_{i,j}$.

1. On peut considérer le (PL) suivant afin de modéliser le problème :

$$PL \begin{cases} \max\ z = \sum_{(i,j)\in E} w_{i,j} x_{i,j} \\ \forall i \in \{1, \ldots, n\}, \sum_{j \in V(i)} x_{i,j} \leq 1 \\ x_j \in \{0, 1\} \end{cases}$$

2. Considérons que le graphe de la figure 2 est le suivant :

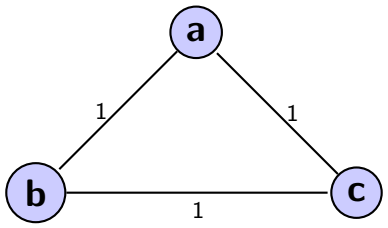


FIGURE 4 – Graphe représentant la figure 2.

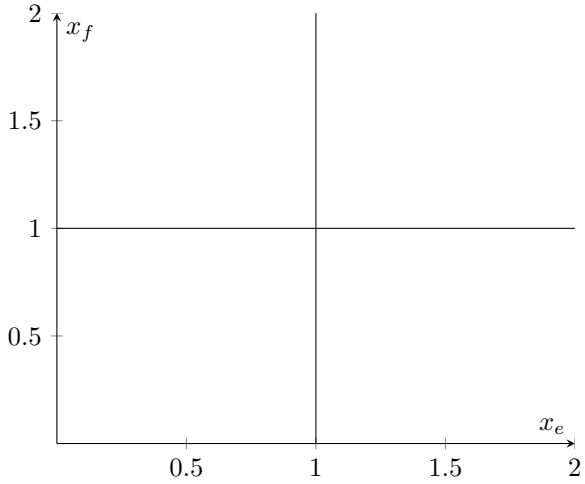
Les équations des contraintes sont donc les suivantes :

$$\begin{cases} x_{a,b} + x_{a,c} \leq 1 \\ x_{a,c} + x_{b,c} \leq 1 \\ x_{a,b} + x_{b,c} \leq 1 \end{cases}$$

3. Une solution optimale du programme linéaire en nombre entiers est $z = 1$ avec $x_{a,b} = 1$, $x_{b,c} = 0$ et $x_{a,c} = 0$.
4. Une solution pour le même programme relaxé est $z = \frac{3}{2}$ avec $x_{a,b} = \frac{1}{2}$, $x_{b,c} = \frac{1}{2}$ et $x_{a,c} = \frac{1}{2}$.
5. La formulation est mauvaise puisqu'avec une solution de LP, on ne peut pas retrouver une solution de ILP.

6. Pour la figure 2, $\frac{|S|-1}{2} = 1$. On rajoute donc la contrainte suivante : $\sum_{(i,j)\in E} x_{i,j} \leq 1$. On remarque qu'avec cette contrainte, $z(ILP) = z(LP) = \frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 1$. Cette solution ne peut être le résultat du PL car ce n'est pas un point extrême, puisqu'il est combinaison linéaire des solutions $(1,0,0), (0,1,0)$ et $(0,0,1)$.

7. a) Pour la première figure, les solutions admissibles sont $x_e = x_f = 1$; $x_e = 1$ et $x_f = 0$; $x_e = 0$ et $x_f = 1$; $x_e = x_f = 0$



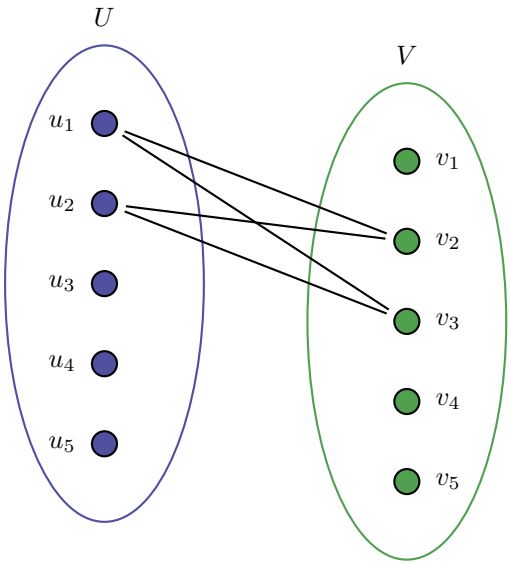
Le polytope associé est défini par les points $(0,0)$, $(0,1)$, $(1,0)$ et $(1,1)$.

Pour la deuxième figure, les différentes combinaisons possibles sont : $x_e = 0, x_f = 0, x_g = 0$; $x_e = 0, x_f = 0, x_g = 1$; $x_e = 0, x_f = 1, x_g = 0$; $x_e = 1, x_f = 1, x_g = 0$ et $x_e = 1, x_f = 0, x_g = 0$. Le polytope associé est une pyramide définie par une base carré ($(0,0,0)$, $(0,1,0)$, $(1,0,0)$ et $(1,1,0)$.) et un sommet $((0,0,1))$.

Le cas du triangle les combinaisons possibles sont : $x_e = 0, x_f = 0, x_g = 0$; $x_e = 0, x_f = 0, x_g = 1$; $x_e = 0, x_f = 1, x_g = 0$ et $x_e = 1, x_f = 0, x_g = 0$. Le polytope associé est une pyramide définie par une base triangulaire ($(0,0,0)$, $(0,1,0)$ et $(1,0,0)$) et un sommet $((0,0,1))$.

b) Si $\lambda_1 M_1 + \lambda_2 M_2 + \lambda_3 M_3 = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$
$\lambda_1(1,0,0) + \lambda_2(0,1,0) + \lambda_3(0,0,1) = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$
alors $\lambda_1 = \lambda_2 = \lambda_3 = \frac{1}{2}$ et donc $\sum_{i=1}^{3} \lambda_i = \frac{3}{2} > 1$. Le point n'appartient donc pas au polytope.

c) Soit un point x du polytope fractionnaire FM, alors $\exists \lambda_1, \ldots, \lambda m, \exists M_1, \ldots M_m$ tel que $\sum_{i=1}^{m} \lambda_i = 1$ et $\sum_{i=1}^{m} \lambda_i M_i = x$.
Un graphe biparti n'est composé que de cycles pairs. Soit une solution optimale de LP.



Considérons le cycle $(u_1, v_2, u_2, v_3, u_1)$.
On a forcément $x_{u_1,v_2} + x_{u_2,v_3} = x_{u_1,v_3} + x_{u_2,v_2}$ puisque l'on prend une arête sur deux le long du cycle. Donc on peut prendre $x'_{u_1,v_2} = x_{u_1,v_2} + x_{u_1,v_3}$, $x'_{u_2,v_3} = x_{u_2,v_3} + x_{u_2,v_2}$ et $x'_{u_2,v_2} = x'_{u_1,v_3} = 0$.
On répétant cette opération, on "casse" tous les cycles tout en gardant la même valeur pour la fonction objectif.

1.2 Problèmes appartenant à la classe APX

Exercice 3 - Sur le problème de la coupe maximum

Dans cet exercice, nous travaillerons avec un graphe $G = (V, E)$ avec $|V| = n$ et $|E| = m$.

1. La boucle *while* ne peut s'effectuer plus de n fois puisqu'à chaque tour de boucle, nous déplaçons un sommet de l'autre côté de la coupe. Puis pour trouver un sommet qui augmente la valeur de la coupe, on parcours potentiellement tout les sommet $O(n)$ puis l'on doit calculer le nombre d'arcs dans la coupe et hors de la coupe $O(n^2)$. On obtient en tout du $O(n^3)$.
2. Soit (Y_1, Y_2) la coupe rendue par l'algorithme. Montrons par l'absurde que $\forall v \in Y_1, |Voisins(v) \cap Y_1| \leq |Voisins(v) \cap Y_2|$. Supposons qu'il existe un sommet $v \in Y_1, |Voisins(v) \cap Y_1| > |Voisins(v) \cap Y_2|$, alors on pourrait déplacer ce sommet de l'autre côté de la coupe et augmenter la valeur de celle-ci de $|Voisins(v) \cap Y_1| - |Voisins(v) \cap Y_2| > 0$. Ce qui contredit le fait que (Y_1, Y_2) soit le résultat de l'algorithme.

   Il nous reste à montrer que l'Algorithme est 2-approché. Posons C la coupe

optimale et (Y_1, Y_2) la coupe rendue par l'algorithme. On peut donc écrire que :

$|C| = |\{\text{Arêtes de}Y_1 \text{ dans } Y1 \in C\}| + |\{\text{Arêtes de}Y_1 \text{ dans } Y2 \in C\}| + |\{\text{Arêtes de}Y_2 \text{ dans } Y2 \in C\}|$
$\leq ...$

3. Considérons le graphe suivant :

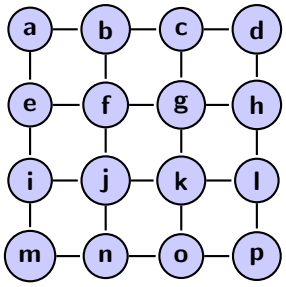


FIGURE 5 – Exemple pour lequel l'algorithme atteint sa borne.

Déroulement de l'algorithme :

étape	S	V - S	Taille de la coupe (S, V-S)
1	$\emptyset$	{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	0
2	{a }	{b,c,d,e,f,g,h,i,j,k,l,m,n,o,p}	2
3	{a, b }	{c,d,e,f,g,h,i,j,k,l,m,n,o,p}	3
4	{a,b,c }	{d,e,f,g,h,i,j,k,l,m,n,o,p}	4
5	{a,b,c,e }	{d,f,g,h,i,j,k,l,m,n,o,p}	5
6	{a,b,c,e,i }	{d,f,g,h,j,k,l,m,n,o,p}	6
7	{a,b,c,e,i,p }	{d,f,g,h,j,k,l,m,n,o}	8
8	{a,b,c,e,i,p,l }	{d,f,g,h,j,k,m,n,o}	9
9	{a,b,c,e,i,p,l,h }	{d,f,g,j,k,m,n,o}	10
10	{a,b,c,e,i,p,l,h,o }	{d,f,g,j,k,m,n}	11
11	{a,b,c,e,i,p,l,h,o,n }	{d,f,g,j,k,m}	12

L'algorithme s'arrête et renvoi donc cette coupe. Cependant on remarque que si l'on prend la coupe $(S, V - S)$ avec $S = \{a, c, f, h, i, k, n, p\}$, on obtient une coupe de taille 24.

1.3 Constructions de PTAS

Exercice 4 - Sur le problème de Partition

Exercice 5 - Sur le problème du sac à dos simple

Soit $w_1, \ldots, w_n$ les poids de n objets, b la capacité du sac à dos. On cherche à trouver $T \subseteq \{1, \ldots, n\}$ tel que $\sum_{i \in T} w_i$ soit maximum.

1. a) Trier les $w_i, i \in \{1, \ldots, n\}$ peut se faire en $O(nlog(n))$. On répète la boucle n fois et les éléments à l'intérieur s'effectuent en $O(1)$. On peut donc conclure que l'algorithme s'effectue en $O(nlog(n))$.

   b) On peut distinguer deux cas :
   — Si $\sum_{i=1}^{n} w_i \leq b$, alors on peut mettre tous les objets dans le sac à dos. Donc $T = \{1, \ldots, n\}$ et le cas se résout trivialement.
   — Si par contre $\sum_{i=1}^{n} w_i > b$, on peut démontrer par l'absurde qu'il existe j tel que $cost(T) + w_{j+1} > b$. Supposons qu'un tel élément n'existe pas, alors à chaque étape de la boucle, l'élément i est ajouté à T. En particulier si $i = n$, on rajoute l'élément n à T puisque $cost(T) + w_n = \sum_{i=1}^{n} w_i \leq b$, ce qui est absurde.

   Montrons maintenant que dans le deuxième cas, on a forcément $cost(T) \geq \frac{b}{2}$. Etudions les trois cas possibles :
   — On ne peut pas avoir $j = 0$ puisque $b \geq w_1$.
   — Si $j = 1$, alors on a $w_1 + w_2 > b$ or on sait que $w_1 \geq w_2$ et donc $2w_1 \geq w_1 + w_2 > b$, d'où $cost(T) = w_1 \geq \frac{b}{2}$.
   — Si $j \geq 2$, on a alors $T = \{1, \ldots, j\}$ d'après la démonstration précédente. $cost(T) = \sum_{i \in T} w_i \leq b$ et $cost(T) + w_{j+1} > b$. Supposons par l'absurde que $cost(T) < \frac{b}{2}$ alors on aurait :

$$w_{j+1} + \frac{b}{2} > cost(T) + w_{j+1} > b$$

   On conclut que $w_{j+1} > \frac{b}{2}$, et puisque $j \geq 2$ alors $cost(T) \geq w_1 + w_2 \geq 2w_{j+1} > b$. C'est une contradiction.

   c) Montrons que l'algorithme admet une performance relative de deux. Soit I une instance quelconque du problème du sac à dos. Alors $Opt(I) \leq b$. Notre algorithme nous assure une solution $A(I) \geq \frac{b}{2}$ d'après la question précédente. On a alors :

$$\frac{A(I)}{Opt(I)} \geq \frac{b}{2} * \frac{1}{b} = \frac{1}{2}$$

2. a) Il y a $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ ensembles S de taille k. On peut donc calculer le nombre total d'ensembles S :

$$N = \sum_{i=1}^{k} \binom{n}{i} = n!(\frac{1}{(n-1)!} + \cdots + \frac{1}{k!(n-k)!}) = n + \frac{n(n-1)}{2!} + \cdots + \frac{n(n-1)\ldots(n-k+1)}{k!}$$

   Il y 'en a donc N $\approx O(n^k)$. Mais puisque pour chacun de ces ensembles, on utilise un algorithme glouton en $O(nlog(n))$. On arrive à une complexité en $O(n^{k+1})$.

   b) i. Si $p \leq k$, cela signifie que l'on va trouver la meilleure solution S possible et donc que $Sol_{PTAS}(I) = Opt(I)$.

ii. Supposons maintenant que $p > k$. On note $P = \{i_1, \ldots, i_k\}$ les indices associés à l'ensemble S. $w_{i_1}, \ldots, w_{i_k}$ sont les points associés. Et on note P^* les indices des objets de l'ensemble retourné par notre algorithme S^*.

A. Si $P^* = M$ alors on a $Sol_{PTAS}(I) = Opt(I)$.

B. Supposons maintenant que $P^* \neq M$. On sait déjà que $b \geq cost(M)$. Soit S un ensemble de taille k composé des k objets dont les poids sont maximaux et appliquons notre algorithme glouton qui va ajouter des objets. D'après la question 1.a), il existe i_q tel que $cost(P^*)+w_{i_q} > b$ (sinon on ajouterais tout les objets). Soit j le premier objet ajouté par l'algorithme glouton, alors $j > i_k$ sinon on aurait pris cet objet à la place. On a donc $i_k < j < i_q$. Si on ajoute pas d'objets, on a quand même $i_k < i_q$, c'est à dire $w_{i_q} < w_{i_k}$ car sinon $cost(S) > b$.

C. On a $w_{i_q} < w_{i_k}$ donc $(k+1)w_{i_q} < kw_{i_k}+w_{i_k} \leq cost(P^*)+w_{i_q} \leq cost(M)$ et donc on peut déduire que $w_{i_q} \leq \frac{cost(M)}{k+1}$

D. $cost(P^*)+w_{i_q} \geq cost(M)$ d'après la question B. On peut donc écrire que :

$$\frac{cost(M)}{k+1} \geq w_{i_q} \geq cost(M) - cost(P^*)$$

$$\frac{cost(M)}{cost(P^*)} \geq (k+1)\frac{cost(M)}{cost(P^*)} - (k+1)$$

$$\frac{cost(M)}{cost(P^*)} \leq \frac{k+1}{k} = 1 + \frac{1}{k} = 1 + \epsilon$$

Il s'agit donc bien d'un algorithme PTAS.

1.4 Construction d'un FPTAS : le problème de la somme d'un sous-ensemble

1.4.1 Un algorithme de complexité exponentielle

1. Si les deux liste L et L' sont triés alors la procédure $FUSIONNER-LISTES(L, L')$ s'effectue en $O(n)$ où $n = max(taille(L), taille(L'))$. En effet, il suffit de parcourir les deux listes en prenant à chaque fois le plus petit élément.
2. La trace de l'algorithme pour $S = \{1, 4, 5\}$ et $n = 3$ est effectué dans le tableau suivant :

i	L_{i-1}	x_i	L_i
1	{0}	1	{0,1}
2	{0,1}	4	{0, 1 , 4 , 5 }
3	{0, 1 , 4 , 5 }	5	{0,1,4,5,6,9,10}

L'algorithme retourne 10.

3. On note P_i l'ensemble des valeurs pouvant être obtenues en faisant la somme de tous les éléments d'un sous ensemble de $\{x_1, \ldots, x_i\}$. Il est évident de voir que $P_i = P_{i-1} \cup (P_{i-1} + x_i)$. En effet si l'on note S_i l'ensemble des éléments de P_i utilisant l'élément x_i et S'_i les éléments de P_i n'utilisant pas x_i alors on a $P_i = S_i \cup S'_i$. Et on remarque facilement que $S_i = P_{i-1} + x_i$ et que $S'_i = P_{i-1}$.
   Montrons par récurrence sur i que la liste L_i est une liste tirée contenant tous les éléments de P_i dont la valeur n'est pas supérieure à t.
   — Pour $i = 0$, il n'y a qu'un élément dans L_0, donc cette liste est triée.
   — Supposons que $L_{i\in\{1,\ldots,n-1\}}$ soit triée, montrons que la liste L_{i+1} reste triée. Il est évident que $L_i + x_{i+1}$ est triée, on peut donc appeler la procédure $FUSIONNER-LISTES(L_i, L_i+x_{i+1})$ qui nous rend une liste triée. On supprime ensuite tout les éléments supérieurs à t, ce qui nous garantie que la liste L_{i+1} est triée et ne contient pas d'éléments supérieurs à t.

   La longueur maximale de la liste L_i est 2^i. En effet, dans le pire des cas, $L_{i-1} \cap (L_{i-1} + x_i) = \emptyset$ pour tout $i \in \{1, \ldots, n\}$, et donc la taille de L_i est doublée à chaque itération.
   On peut donc conclure sur la complexité exponentielle de l'algorithme, en effet, pour fusionner l'ensemble des listes on a une complexité en $\sum_1^n i*2^i \approx n * 2^{n+1}$ et donc $O(2^n)$.

1.4.2 Un schéma d'approximation en temps entièrement polynomial

1. Le déroulement des calculs est présenté dans le tableau suivant :

y_i	Existe t'il z tel que $\frac{y-z}{y} \leq \delta$	L'
10	non	[10]
11	oui	[10]
12	non	[10 , 12]
15	non	[10 , 12, 15]
20	non	[10 , 12, 15, 20]
21	oui	[10 , 12, 15, 20]
22	oui	[10 , 12, 15, 20]
23	non	[10 , 12, 15, 20, 23]
24	oui	[10 , 12, 15, 20, 23]
29	non	[10 , 12, 15, 20, 23, 29]

   L'algorithme renvoie $L' = [10, 12, 15, 20, 23, 29]$.

2. Question ?

3. On applique l'algorithme sur une liste $L = [104, 102, 201, 101]$ avec $t = 308$ et $\epsilon = 0.2$. La trace de l'algorithme est donné dans le tableau suivant :

L_{i-1}	x_i	L_i	L_i après SEUILLER$(L_i, 0.05)$
[0]	104	[0 , 104]	[0 , 104]
[0 , 104]	102	[0 , 102 , 104 , 206]	[0 , 102, 206]
[0 , 102, 206]	201	[0 , 102, 201, 206, 303, 407]	[0, 102, 201, 303]
[0, 102, 201, 303]	101	[0, 101, 102, 201, 203, 302, 303, 404]	[0, 101, 201, 302]

L'algorithme rend 302.

4. Soit C^* la valeur d'une solution optimale et C la valeur de la solution donnée par notre algorithme.
   a) Montrons par récurrence sur i que $\forall y \in P_i, y \leq t, \exists z \in L_i$ tel que $(1-\frac{\epsilon}{n})^i y^* \leq z \leq y^*$.
      — Pour $i = 1$, $\forall y \exists z$ tel que $(1-\delta)y \leq z \leq y$ par définition du seuillage.
      — Supposons que $\forall y \exists z, (1-\frac{\epsilon}{n})^i y \leq z \leq y$. Alors soit :
         — $y \in P_i$. Si le z précédent à été supprimé, alors $\exists z', (1-\frac{\epsilon}{n})z \leq z' \leq z$. Donc $(1-\frac{\epsilon}{n})^{i+1} y \leq z(1-\frac{\epsilon}{n}) \leq z' \leq z \leq y$ sinon $(1-\frac{\epsilon}{n})^{i+1} y \leq (1-\frac{\epsilon}{n})^i y = z \leq y$.Cela reste vrai.
         — $y \notin P_i$. $y = y' + x_{i+1}$ avec $y' \in P_i$ et $\exists z', (1-\frac{\epsilon}{n})^i y' \leq z' \leq y'$.
            — Si $z' + x_{i+1} \in P_{i+1}$,

$$(1-\frac{\epsilon}{n})^i y' + x_{i+1} \leq z' + x_{i+1} \leq y' + x_{i+1}$$

$$(1-\frac{\epsilon}{n})^i (y' + x_{i+1}) \leq (1-\frac{\epsilon}{n})^i y' + x_{i+1} \leq z' + x_{i+1}$$

$$(1-\frac{\epsilon}{n})^{i+1} \leq (1-\frac{\epsilon}{n})^i y \leq z \leq y$$

            — Sinon :

$$\exists z, (1-\frac{\epsilon}{n})(z' + x_{i+1}) \leq z \leq z' + x_{i+1}$$

$$(1-\frac{\epsilon}{n})^i y' + x_{i+1} \leq z' + x_{i+1} \leq y' + x_{i+1}$$

$$(1-\frac{\epsilon}{n})^{i+1}(y'+x_{i+1}) \leq (1-\frac{\epsilon}{n})^{i+1} y' + (1-\frac{\epsilon}{n}) x_{i+1} \leq (1-\frac{\epsilon}{n}) z' + x_{i+1} \leq z \leq z' + x_{i+1} \leq y' + x_{i+1}$$

$$(1-\frac{\epsilon}{n})^{i+1} y \leq z \leq y$$

   En particulier, $(1-\frac{\epsilon}{n})^n y^* \leq z \leq y^*$ donc $1 \geq \frac{z}{y^*} \geq (1-\frac{\epsilon}{n})^n$

5. a) Si on a $\frac{z}{z'} \leq \frac{1}{1-\frac{\epsilon}{n}}$, alors $z(1-\frac{\epsilon}{n}) \leq z$, on aurait donc supprimé z.
   b) Supposons que le 1^{er} élément est 1 et le dernier élément t.

$$\frac{z_1}{z_2} * \frac{z_2}{z_3} * \cdots * \frac{z_{n-1}}{z_n} = t$$

$$nlog(\frac{1}{1-\frac{\epsilon}{n}}) \leq log(\frac{z_1}{z_2}) + \cdots + log(\frac{z_{n-1}}{z_n}) = log(t)$$

$$n \leq \frac{log(t)}{-log(1-\frac{\epsilon}{n})} \text{ donc } n \leq \frac{ln(t)}{-ln(1-\frac{\epsilon}{n})}$$

Exercice 6 - Programmation dynamique

Exercice 7 - Sur le produit matriciel

Exercice 8 - Résolution numérique

Exercice 9 - Seuil d'approximation pour le problème Bin Packing

Exercice 10 - Seuil d'approximation pour le problème de la coloration de sommets (reps. d'arêtes)

Exercice 11 - Comparaisons branch and bound and branch and cut

1. On peut tracer les droites correspondantes aux contraintes de PL_0 :
   — $y_1 = 5 - \frac{3}{2}x$
   — $y_2 = \frac{17}{2} - \frac{2}{5}x$

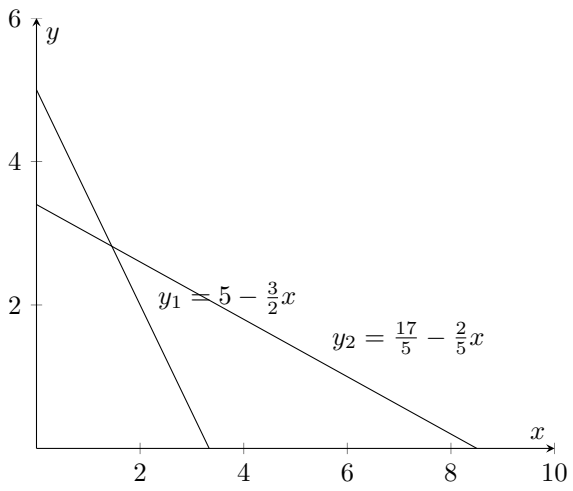
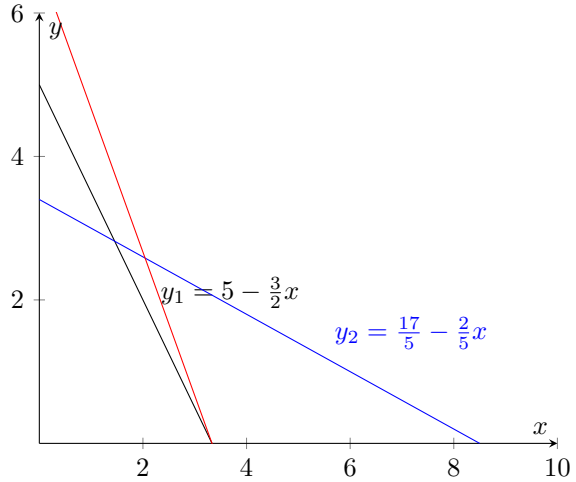


FIGURE 6 – Représentation des équations de PL_0.

Le polytope associé aux équations de PL_0 est donc celui formé par les points : $P_0(0,0)$, $P_1(0, \frac{17}{5})$, $P_2(\frac{16}{11}, \frac{31}{11})$ et $P_3(\frac{10}{3}, 0)$

2. On peut tracer la fonction objective :
   — $y = -2x + k, k \in \mathbf{R}$ (rouge)



La solution optimale pour PL_0 est donc $x_1 = \frac{10}{3}$ et $x_2 = 0$ avec $z = \frac{20}{3}$.

3. On commence par reprendre le programme linéaire donné :

$$PL_0 \begin{cases} \max\ z(x_1, x_2) = 2x_1 + x_2 \\ 2x_1 + 5x_2 \leq 17 \\ 3x_1 + 2x_2 \leq 10 \\ x_1, x_2 \geq 0 \end{cases}$$

On ajoute les variables d'écarts x_3 et x_4 pour obtenir PL_1 :

$$PL_1 \begin{cases} \max\ z(x_1, x_2) = 2x_1 + x_2 \\ 2x_1 + 5x_2 + x_3 = 17 \\ 3x_1 + 2x_2 + x_4 = 10 \\ x_1, x_2 \geq 0 \end{cases}$$

On obtient donc le tableau initial :

			2	1	0	0
			x_1	x_2	x_3	x_4
0	x_3	17	2	5	1	0
0	x_4	10	3	2	0	1
	z	0	-2	-1	0	0

La variable entrante est x_1 et la variable sortante est x_4. Le tableau final est donc :

			2	1	0	0
			x_1	x_2	x_3	x_4
0	x_3	$\frac{31}{3}$	0	$\frac{11}{3}$	1	$\frac{-2}{3}$
2	x_1	$\frac{10}{3}$	1	$\frac{2}{3}$	0	$\frac{1}{3}$
	z	$\frac{20}{3}$	0	$\frac{1}{3}$	0	$\frac{2}{3}$

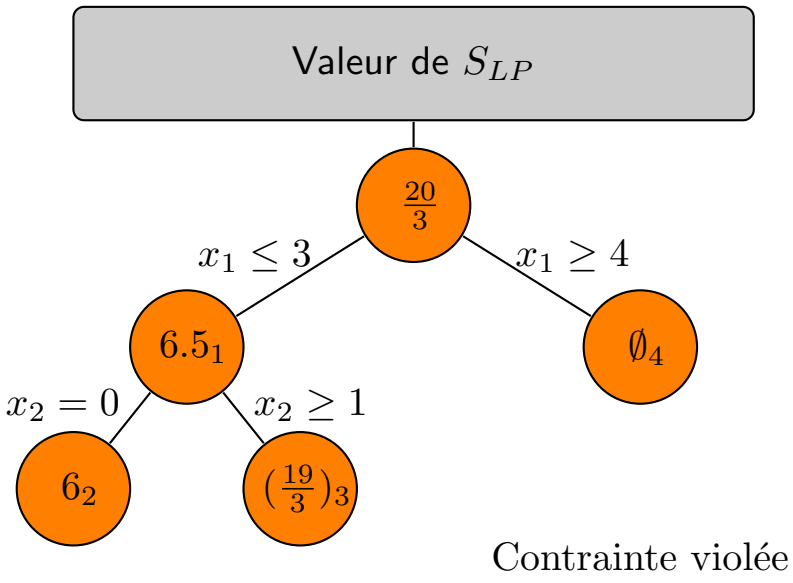
La solution optimale est donc $x_1 = \frac{10}{3}$ et $x_2 = 0$ avec $z = \frac{20}{3}$.

4. a) On effectue notre Branch and Bound dans l'ordre : x_1 puis x_2. Puisque dans la solution optimale du (PL), $x_1 = \frac{20}{3}$, on a $x_1 \leq 3$ ou que $x_1 \geq 4$.
   — Si l'on considère que $x_1 \leq 3$, la solution optimale devient $z = \frac{13}{2}$ avec $x_1 = 3$ et $x_2 = \frac{1}{2}$. On a ensuite, $x_2 = 0$ ou $x_2 > 0$.

      — Si l'on considère que $x_2 = 0$, alors la solution optimale devient $z = 6$ avec $x_1 = 3$ et $x_2 = 0$.
      — Si l'on considère que $x_2 \geq 1$, alors la solution optimale devient $z = \frac{19}{3} \approx 6.33$ avec $x_1 = \frac{8}{3}$ et $x_2 = 1$. On ne développe pas plus cette branche puisque l'on à déjà trouvé une solution avec $z = 6$.

   — Si l'on considère que $x_1 \geq 4$, la deuxième contrainte est violée.

   L'arbre du Branch and bound récapitulatif est donné plus bas.



b) Puisque que l'on a $\frac{31}{3} = \frac{11}{3}x_2 + x_3 - \frac{2}{3}x_4$, on peut déduire que : $\frac{11}{3}x_2 - \frac{2}{3}x_4 \geq \frac{1}{3}$.
Puisque l'on a $\frac{10}{3} = x_1 + \frac{2}{3}x_2 + \frac{1}{3}x_4$, on peut donc déduire que $x_4 = 10 - 3x_1 - 2x_2$.
On obtient finalement la contrainte : $2x_1 + 5x_2 \geq 7$.
Le tableau final donne :

			2	1	0	0	0
			x_1	x_2	x_3	x_4	x_5
0	x_3	10	0	0	1	0	1
2	x_1	$\frac{36}{11}$	1	0	0	$\frac{5}{11}$	$\frac{2}{11}$
1	x_2	$\frac{1}{11}$	0	1	0	$\frac{-2}{11}$	$\frac{-3}{11}$
	z	$\frac{73}{11}$	0	0	0	$\frac{8}{11}$	$\frac{1}{11}$

On choisi l'équation $\frac{36}{11} = x_1 + \frac{5}{11}x_4 + \frac{2}{11}x_5$, on déduit donc $\frac{5}{11}x_4 + \frac{2}{11}x_5 \geq \frac{3}{11}$. Ce qui revient à écrire par l'équation précédente que $\frac{36}{11} - x_1 \geq \frac{3}{11}$ et donc que $x_1 \leq 3$.

			2	1	0	0	0	0
			x_1	x_2	x_3	x_4	x_5	x_6
0	x_3	$\frac{17}{2}$	0	0	1	$\frac{-5}{2}$	0	$\frac{11}{2}$
0	x_4	$\frac{3}{2}$	0	0	0	$\frac{5}{2}$	1	$\frac{-11}{2}$
1	x_2	$\frac{1}{2}$	0	1	0	$\frac{1}{2}$	0	$\frac{-3}{2}$
2	x_1	3	1	0	0	0	0	1
	z	$\frac{13}{2}$	0	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$

On sélectionne l'équation $\frac{1}{2} = x_2 + \frac{1}{2}x_4 - \frac{3}{2}x_6$. Donc on déduit que $\frac{1}{2} \leq \frac{-3}{2}x_6$

2 Partie Pratique

2.1 Programmation dynamique

2.2 Branch and Bound

2.3 Comparaisons entre un algorithme de complexité exponentielle et un FP-TAS