HAI722I — DM

Ivan Lejeune

12 décembre 2025

Table des matières

Instructions

Ce devoir est à rendre avant le 12 décembre 2025 à 12h, soit par mail à l'adresse : rodolphe.giroudeau@lirmm.fr, soit en déposant votre devoir durant le cours

1 Partie théorique

Exercice 1 Algorithmes pour la programmation linéaire.
Considérons la formulation suivante :

$$P_\beta = \begin{cases} \max z = 5x_1 + 2x_2 \\ 6x_1 + x_2 \geq 6 \\ 4x_1 + 4x_2 \geq 12 \\ x_1 + 2x_2 \geq 4 \\ x_i \geq 0, \quad \forall i \in \{1,2\} \end{cases}$$

1. Résoudre le problème P_β par la méthode du big M.
2. Résoudre le problème P_β par la méthode à deux phases.
3. **Difficile :**
   (a) Résoudre le problème P_β par la méthode dual-simplexe.
   (b) Soit le programme linéaire P_θ

$$P_\theta = \begin{cases} \max z = x_1 + 3x_2 \\ x_1 + x_2 \geq 3 \\ x_1 - 2x_2 \geq 5 \\ -2x_1 + x_2 \leq 5 \\ x_i \geq 0, \quad \forall i \in \{1,2\} \end{cases}$$

Résoudre le problème P_θ par la méthode dual-simplexe.

Solution.

1. Commençons par poser le problème sous forme standard :

$$P_\beta = \begin{cases} \max z = 5x_1 + 2x_2 + 0x_3 + 0x_4 + 0x_5 - M \cdot (y_1 + y_2 + y_3) \\ 6x_1 + x_2 - x_3 + y_1 = 6 \\ 4x_1 + 4x_2 - x_4 + y_2 = 12 \\ x_1 + 2x_2 - x_5 + y_3 = 4 \\ x_i \geq 0, y_j \geq 0, \quad \forall i \in \{1,\dots,5\}, \forall j \in \{1,2,3\}. \end{cases}$$

Ensuite on construit notre tableau du simplexe :

		c	5	2	0	0	0	$-M$	$-M$	$-M$
c^J	variables de base		x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3
$-M$	$x_1^1 = y_1$	6	6	1	-1	0	0	1	0	0
$-M$	$x_2^1 = y_2$	12	4	4	0	-1	0	0	1	0
$-M$	$x_3^1 = y_3$	4	1	2	0	0	-1	0	0	1
	$z(x)$	$-22M$	$-11M-5$	$-7M-2$	M	M	M	0	0	0

et on déroule l'algorithme :

- on rentre x_1,

- on sort y_1 car $1 < 3 < 4$,

$\downarrow$

			c	5	2	0	0	0	$-M$	$-M$	$-M$
c^J	variables de base			x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3
5	$x_1^2 = x_1$	1		1	$\frac{1}{6}$	$-\frac{1}{6}$	0	0	$\frac{1}{6}$	0	0
$-M$	$x_2^2 = y_2$	8		0	$\frac{10}{3}$	$\frac{4}{6}$	-1	0	$-\frac{4}{6}$	1	0
$-M$	$x_3^2 = y_3$	3		0	$\frac{11}{6}$	$\frac{1}{6}$	0	-1	$-\frac{1}{6}$	0	1
	$z(x)$	$-11M+5$		0	$-\frac{31}{6}M - \frac{7}{12}$	$-\frac{5}{6}M - \frac{5}{6}$	M	M	$\frac{11}{6}M + \frac{5}{6}$	0	0

- on rentre x_2,
- on sort y_3 car $\frac{11}{2} < 6 < \frac{80}{3}$,

$\downarrow$

			c	5	2	0	0	0	$-M$	$-M$	$-M$
c^J	variables de base			x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3
5	$x_1^3 = x_1$	$\frac{8}{11}$		1	0	$-\frac{2}{11}$	0	$\frac{1}{11}$	$\frac{2}{11}$	0	$-\frac{1}{11}$
$-M$	$x_2^3 = y_2$	$\frac{28}{11}$		0	0	$\frac{4}{11}$	-1	$\frac{20}{11}$	$-\frac{4}{11}$	1	$-\frac{20}{11}$
2	$x_3^3 = x_2$	$\frac{18}{11}$		0	1	$\frac{1}{11}$	0	$-\frac{6}{11}$	$-\frac{1}{11}$	0	$\frac{6}{11}$
	$z(x)$	$-\frac{28}{11}M + \frac{76}{11}$		0	0	$-\frac{4}{11}M - \frac{8}{11}$	$-M$	$-\frac{20}{11}M - \frac{7}{11}$	$\frac{15}{11}M + \frac{8}{11}$	0	$\frac{31}{11}M + \frac{7}{11}$

- on rentre x_5,
- on sort y_2 car $\frac{7}{5} < 8$ (l'autre rapport étant négatif on le compte pas),

$\downarrow$

			c	5	2	0	0	0	$-M$	$-M$	$-M$
c^J	variables de base			x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3
5	$x_1^4 = x_1$	$\frac{3}{5}$		1	0	$-\frac{1}{5}$	$\frac{1}{20}$	0	$\frac{1}{5}$	$-\frac{1}{20}$	0
0	$x_2^4 = x_5$	$\frac{7}{5}$		0	0	$\frac{1}{5}$	$-\frac{11}{20}$	1	$-\frac{1}{5}$	$\frac{11}{20}$	-1
2	$x_3^4 = x_2$	$\frac{12}{5}$		0	1	$\frac{1}{5}$	$-\frac{3}{10}$	0	$-\frac{1}{5}$	$\frac{3}{10}$	0
	$z(x)$	$\frac{39}{5}$		0	0	$-\frac{3}{5}$	$-\frac{7}{20}$	0	$M + \frac{3}{5}$	$M + \frac{7}{20}$	M

A ce stade, il est clair que les variables artificielles ne pourront plus entrer en base. Elles ont un coût dépendant de M et toutes les autres variables ont un coût indépendant de M. On peut donc les retirer et procéder sur le tableau réduit suivant (c'est surtout pour des raisons de clarté) :

			c	5	2	0	0	0
c^J	variables de base			x_1	x_2	x_3	x_4	x_5
5	$x_1^4 = x_1$	$\frac{3}{5}$		1	0	$-\frac{1}{5}$	$\frac{1}{20}$	0
0	$x_2^4 = x_5$	$\frac{7}{5}$		0	0	$\frac{1}{5}$	$-\frac{11}{20}$	1
2	$x_3^4 = x_2$	$\frac{12}{5}$		0	1	$\frac{1}{5}$	$-\frac{3}{10}$	0
	$z(x)$	$\frac{39}{5}$		0	0	$-\frac{3}{5}$	$-\frac{7}{20}$	0

- on rentre x_3,
- on sort x_5,

$\downarrow$

			c	5	2	0	0	0
c^J	variables de base			x_1	x_2	x_3	x_4	x_5
5	$x_1^5 = x_1$	2		1	0	0	$-\frac{1}{2}$	1
0	$x_2^5 = x_3$	7		0	0	1	$-\frac{11}{4}$	5
2	$x_3^5 = x_2$	1		0	1	0	$\frac{1}{4}$	-1
	$z(x)$	12		0	0	0	-2	3

Il ne reste probablement qu'une étape, faisons-la :

- on rentre x_4,
- on sort x_2, c'est la seule valeur positive,

$\downarrow$

			c	5	2	0	0	0
c^J	variables de base			x_1	x_2	x_3	x_4	x_5
5	$x_1^6 = x_1$	4		1	2	0	0	-1
0	$x_2^6 = x_3$	18		0	11	1	0	-6
0	$x_3^6 = x_4$	4		0	4	0	1	-4
	$z(x)$	20		0	8	0	0	-5

Dommage, on a encore un point pivot à faire rentrer (x_5 a valeur négative) donc on aimerait continuer de dérouler l'algorithme mais on a pas de critère pour trouver quelle variable sortir de base (en effet, toutes les valeurs dans la colonne du pivot sont négatives).

On conclut donc que l'espace des solutions est non borné. C'est-à-dire qu'il n'existe pas « un couple maximum » car on peut toujours en trouver un plus grand. Donc il n'y a pas un unique couple (x_1, x_2) qui maximise z.

2. Résolvons maintenant le problème P_θ par la méthode à deux phases.

Commençons par modifier le problème comme il faut pour la phase 1 :

$$P'_\theta = \begin{cases} \max z' = 0x_1 + 0x_2 + 0x_3 + 0x_4 + 0x_5 - (y_1 + y_2 + y_3) \\ 6x_1 + x_2 - x_3 + y_1 = 6 \\ 4x_1 + 4x_2 - x_4 + y_2 = 12 \\ x_1 + 2x_2 - x_5 + y_3 = 4 \\ x_i \geq 0, y_j \geq 0, \quad \forall i \in \{1, \ldots, 5\}, \forall j \in \{1, 2, 3\}. \end{cases}$$

Ensuite on construit notre tableau du simplexe :

		c	0	0	0	0	0	−1	−1	−1
c^J	variables de base		x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3
−1	$x_1^1 = y_1$	6	6	1	−1	0	0	1	0	0
−1	$x_2^1 = y_2$	12	4	4	0	−1	0	0	1	0
−1	$x_3^1 = y_3$	4	1	2	0	0	−1	0	0	1
	$z'(x)$	−22	−11	−7	1	1	1	0	0	0

et on déroule l'algorithme :

- on rentre x_1,
- on sort y_1 car $1 < 3 < 4$,

↓

		c	0	0	0	0	0	−1	−1	−1
c^J	variables de base		x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3
0	$x_1^2 = x_1$	1	1	$\frac{1}{6}$	$-\frac{1}{6}$	0	0	$\frac{1}{6}$	0	0
−1	$x_2^2 = y_2$	8	0	$\frac{10}{3}$	$\frac{4}{6}$	−1	0	$-\frac{4}{6}$	1	0
−1	$x_3^2 = y_3$	3	0	$\frac{11}{6}$	$\frac{1}{6}$	0	−1	$-\frac{1}{6}$	0	1
	$z'(x)$	−11	0	$-\frac{31}{6}$	$-\frac{5}{6}$	1	1	$\frac{11}{6}$	0	0

- on rentre x_2,
- on sort y_3,

↓

		c	0	0	0	0	0	−1	−1	−1
c^J	variables de base		x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3
0	$x_1^3 = x_1$	$\frac{8}{11}$	1	0	$-\frac{2}{11}$	0	$\frac{1}{11}$	$\frac{2}{11}$	0	$-\frac{1}{11}$
−1	$x_2^3 = y_2$	$\frac{28}{11}$	0	0	$\frac{4}{11}$	−1	$\frac{20}{11}$	$-\frac{4}{11}$	1	$-\frac{20}{11}$
0	$x_3^3 = x_2$	$\frac{18}{11}$	0	1	$\frac{1}{11}$	0	$-\frac{6}{11}$	$-\frac{1}{11}$	0	$\frac{6}{11}$
	$z'(x)$	$-\frac{28}{11}$	0	0	$-\frac{4}{11}$	−1	$-\frac{20}{11}$	$\frac{15}{11}$	0	$\frac{31}{11}$

- on rentre x_5,
- on sort y_2,

↓

			c	0	0	0	0	0	−1	−1	−1
c^J	variables de base			x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3
0	$x_1^4 = x_1$	$\frac{3}{5}$		1	0	$-\frac{1}{5}$	$\frac{1}{20}$	0	$\frac{1}{5}$	$-\frac{1}{20}$	0
0	$x_2^4 = x_5$	$\frac{7}{5}$		0	0	$\frac{1}{5}$	$-\frac{11}{20}$	1	$-\frac{1}{5}$	$\frac{11}{20}$	−1
0	$x_3^4 = x_2$	$\frac{12}{5}$		0	1	$\frac{1}{5}$	$-\frac{3}{10}$	0	$-\frac{1}{5}$	$\frac{3}{10}$	0
	$z'(x)$	0		0	0	0	0	0	1	1	1

A ce stade, toutes les variables artificielles sont sorties de base et la valeur optimale de z' est nulle. On peut donc passer à la phase 2 en retirant les variables artificielles du tableau avec la fonction objectif originale :

			c	5	2	0	0	0
c^J	variables de base			x_1	x_2	x_3	x_4	x_5
5	$x_1^4 = x_1$	$\frac{3}{5}$		1	0	$-\frac{1}{5}$	$\frac{1}{20}$	0
0	$x_2^4 = x_5$	$\frac{7}{5}$		0	0	$\frac{1}{5}$	$-\frac{11}{20}$	1
2	$x_3^4 = x_2$	$\frac{12}{5}$		0	1	$\frac{1}{5}$	$-\frac{3}{10}$	0
	$z(x)$	$\frac{39}{5}$		0	0	$-\frac{3}{5}$	$-\frac{7}{20}$	0

- on rentre x_3,
- on sort x_5,

↓

			c	5	2	0	0	0
c^J	variables de base			x_1	x_2	x_3	x_4	x_5
5	$x_1^5 = x_1$	2		1	0	0	$-\frac{1}{2}$	1
0	$x_2^5 = x_3$	7		0	0	1	$-\frac{11}{4}$	5
2	$x_3^5 = x_2$	1		0	1	0	$\frac{1}{4}$	−1
	$z(x)$	12		0	0	0	−2	3

- on rentre x_4,
- on sort x_2, c'est la seule valeur positive,

↓

			c	5	2	0	0	0
c^J	variables de base			x_1	x_2	x_3	x_4	x_5
5	$x_1^6 = x_1$	4		1	2	0	0	−1
0	$x_2^6 = x_3$	18		0	11	1	0	−6
0	$x_3^6 = x_4$	4		0	4	0	1	−4
	$z(x)$	20		0	8	0	0	−5

Il ne resterait plus qu'à faire rentrer x_5 pour obtenir la solution optimale mais on a aucune variable qui correspond à un critère d'entrée, elles sont toutes negatives. Donc x_5 ne peut pas rentrer en base et le problème est non borné.

3. On passe à la partie difficile.

(a) On commence par passer toutes les contraintes sous forme $\leq$:

$$\begin{cases} \max z = 5x_1 + 2x_2 \\ -6x_1 - x_2 \leq -6 \\ -4x_1 - 4x_2 \leq -12 \\ -x_1 - 2x_2 \leq -4 \\ x_i \geq 0, \quad \forall i \in \{1, 2\} \end{cases}$$

On n'a plus que des contraintes de type $\leq$ donc on a plus qu'à rajouter les variables d'écart, pas de variable artificielle nécessaire. Le problème devient :

$$\begin{cases} \max z = 5x_1 + 2x_2 + 0x_3 + 0x_4 + 0x_5 \\ -6x_1 - x_2 + x_3 = -6 \\ -4x_1 - 4x_2 + x_4 = -12 \\ -x_1 - 2x_2 + x_5 = -4 \\ x_i \geq 0, \quad \forall i \in \{1, \ldots, 5\} \end{cases}$$

Enfin, on construit le tableau initial :

			c	5	2	0	0	0
c^J	variables de base			x_1	x_2	x_3	x_4	x_5
0	$x_1^1 = x_3$		-6	-6	-1	1	0	0
0	$x_2^1 = x_4$		-12	-4	-4	0	1	0
0	$x_3^1 = x_5$		-4	-1	-2	0	0	1
	$z(x)$		0	-5	-2	0	0	0

On a ni une solution de base réalisable pour le dual ni une solution de base réalisable pour le primal car tous les termes sont négatifs. Donc on n'a pas de solution de base avec la méthode du dual simplexe et on ne peut pas chercher une solution au problème P_β.

Exercice 2 Dualité.
Considérez le programme linéaire le plus général envisageable donné ci-dessous :

$$\begin{cases} \min z = c_1 x_1 + c_2 x_2 \\ A_{11} x_1 + A_{12} x_2 \leq b_1 \\ A_{21} x_1 + A_{22} x_2 = b_2 \\ x_i \geq 0, \quad \forall i \in \{1, 2\} \end{cases}$$

où A est une matrice $(m_1 + m_2) \times (n_1 + n_2)$ et $c, x \in \mathbb{R}^{n_1+n_2}$ et $b \in \mathbb{R}^{m_1+m_2}$.
Caractériser le dual.

Solution.
Pour chaque contrainte de type $\leq$ on associe une variable duale $y_i \geq 0$ et pour chaque contrainte de type $=$ on associe une variable duale y_j pas forcément positive. Le dual est donc :

$$\begin{cases} \max w = b_1 y_1 + b_2 y_2 \\ A_{11} y_1 + A_{21} y_2 \geq c_1 \\ A_{12} y_1 + A_{22} y_2 = c_2 \\ y_1 \geq 0 \end{cases}$$

Au départ on avait m_1 contraintes de type $\leq$ et m_2 contraintes de type $=$, donc le dual a $m_1 + m_2$ variables (c'est le cas car b_1 (respectivement b_2) est un vecteur de taille m_1 (respectivement m_2)). De plus, on avait $n_1 + n_2$ variables dans le primal, donc le dual a $n_1 + n_2$ contraintes (c'est

le cas car A est une matrice $(m_1 + m_2) \times (n_1 + n_2)$, donc c est un vecteur de taille $n_1 + n_2$).

Exercice 3 Ensemble convexe.
Soit C_1 et C_2 deux convexes de $\mathbb{R}^{m+n}$. Montrer que l'ensemble

$$C = \{(x, y_1 + y_2) \mid x \in \mathbb{R}^m, y_1 \in \mathbb{R}^n, y_2 \in \mathbb{R}^n, (x, y_1) \in C_1, (x, y_2) \in C_2\}$$

est également convexe.

Solution.
Soient (x^1, y^1) et (x^2, y^2) deux points de C. Par définition de C, il existe $y_1^1, y_2^1, y_1^2, y_2^2$ tels que

$$(x^1, y_1^1) \in C_1, \quad (x^1, y_2^1) \in C_2, \quad (x^2, y_1^2) \in C_1, \quad (x^2, y_2^2) \in C_2,$$

et

$$y^1 = y_1^1 + y_2^1, \quad y^2 = y_1^2 + y_2^2.$$

Soit $\lambda \in [0, 1]$. Considérons le point

$$(x^\lambda, y^\lambda) = \lambda(x^1, y^1) + (1 - \lambda)(x^2, y^2).$$

On a

$$\begin{aligned} x^\lambda &= \lambda x^1 + (1 - \lambda)x^2, \\ y^\lambda &= \lambda y^1 + (1 - \lambda)y^2 \\ &= \lambda(y_1^1 + y_2^1) + (1 - \lambda)(y_1^2 + y_2^2) \\ &= (\lambda y_1^1 + (1 - \lambda)y_1^2) + (\lambda y_2^1 + (1 - \lambda)y_2^2). \end{aligned}$$

Notons $y_1^\lambda = \lambda y_1^1 + (1 - \lambda)y_1^2$ et $y_2^\lambda = \lambda y_2^1 + (1 - \lambda)y_2^2$. Par convexité de C_1 et C_2, on a

$$(x^\lambda, y_1^\lambda) \in C_1, \quad (x^\lambda, y_2^\lambda) \in C_2.$$

Ainsi, par définition de C, on a $(x^\lambda, y^\lambda) \in C$. Donc C est convexe.

Exercice 4 Modélisation et dualité.
Considérons un problème d'affectation avec m jobs et n travailleurs ($n \geq m$). Chaque job doit être affecté à exactement un travailleur. Soit p_{ij} le rendement obtenu si on affecte le job i au travailleur j, où $i \in \{1, \ldots, m\}$ et $j \in \{1, \ldots, n\}$. On cherche une affectation qui maximise le rendement total.

1. Donner le programme linéaire.
2. Donner la formulation du dual de ce problème.

Solution.

1. On introduit les variables x_{ij} qui valent 1 si le job i est affecté au travailleur j, et 0 sinon. Le programme linéaire s'écrit alors :

$$P_0 = \begin{cases} \max z = \displaystyle\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} & (1) \\ \displaystyle\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i \in \{1, \ldots, m\} & (2) \\ \displaystyle\sum_{i=1}^{m} x_{ij} \leq 1, \quad \forall j \in \{1, \ldots, n\} & (3) \\ x_{ij} \in \{0, 1\}, \quad \forall i, j & (4) \end{cases}$$

où les contraintes correspondent à :

(1) on maximise le rendement total
(2) chaque job est affecté (à un travailleur)
(3) chaque travailleur a au plus un job
(4) les jobs ne sont pas partiellement affectés

2. Pour obtenir la formulation du dual, on introduit les variables duales u_i (une par job) et v_j (une par travailleur). Le dual s'écrit alors :

$$D_0 = \begin{cases} \min w = \sum_{i=1}^{m} u_i + \sum_{j=1}^{n} v_j & (5) \\ u_i + v_j \geq p_{ij}, \quad \forall i, j & (6) \\ v_j \geq 0, \quad \forall j & (7) \end{cases}$$

où les contraintes correspondent à :

(5) on minimise la somme des coûts
(6) les coûts doivent couvrir les rendements
(7) les variables sont positives

c'est une autre manière intéressante de voir le problème.

Exercice 5 Programmation linéaire : Farkas.
Considérons le programme linéaire suivant, qui dépend de $\varepsilon \in \mathbb{R}$:

$$\begin{cases} \min z = 4x_1 - 2x_2 \\ x_2 \leq 3 \\ \varepsilon x_1 + (2 - \varepsilon)x_2 \leq 4 \\ x_i \geq 0, \quad \forall i \in \{1, 2\} \end{cases}$$

1. Montrer que le problème est réalisable $\forall \varepsilon \in \mathbb{R}$.
2. Pour quelles valeurs de ε la valeur optimale est-elle non bornée ?

Solution.

1. On remarque tout de suite que le point $x = (0, 0)$ satisfait les contraintes du programme linéaire pour toute valeur de ε. Donc le problème est réalisable pour tout $\varepsilon \in \mathbb{R}$.
2. Pour étudier la borne de la valeur optimale, on peut regarder le dual du programme linéaire :

$$\begin{cases} \max w = 3y_1 + 4y_2 \\ \varepsilon y_2 \geq 4 \\ y_1 + (2 - \varepsilon)y_2 \geq -2 \\ y_i \geq 0 \end{cases}$$

D'après le théorème de dualité de la programmation linéaire (4.4.2, n. 3), si le dual est irréalisable alors le primal est non borné. Étudions donc l'irréalisabilité du dual :

- Si $\varepsilon \leq 0$, la première contrainte du dual ne peut pas être satisfaite car $y_2 \geq 0$ et donc le dual est irréalisable. Donc le primal est non borné.
- Si $\varepsilon > 0$, la première contrainte du dual impose $y_2 \geq \frac{4}{\varepsilon} > 0$. En remplaçant dans la

deuxième contrainte on obtient :

$$y_1 + (2-\varepsilon)\frac{4}{\varepsilon} \geq -2 \iff y_1 \geq -2 - \frac{8}{\varepsilon} + 4 = 2 - \frac{8}{\varepsilon}.$$

Comme $\varepsilon > 0$, on a $2 - \frac{8}{\varepsilon} < 2$. Donc on peut toujours choisir y_1 suffisamment grand pour satisfaire cette contrainte. Donc le dual est réalisable et le primal est donc borné.

En conclusion, la valeur optimale est non bornée pour $\varepsilon \leq 0$.

Exercice 6 Résolution numérique.
Résoudre le programme linéaire suivant par la méthode Primal-Dual :

$$\text{Primal} = \begin{cases} \min z(x_1, x_2, x_3) = 2x_1 + x_2 + 2x_3 + 8x_4 \\ 2x_1 - x_2 + 3x_3 - 2x_4 = 3 \\ -x_1 + 3x_2 - 4x_3 = 1 \\ x_i \geq 0, \quad \forall i \in \{1,2,3,4\} \end{cases}$$

Solution.

2 Partie pratique

Exercice 7 Résolution d'un programme linéaire avec Julia.

1. Dans un premier temps nous allons regarder quelques commandes de base en Julia.
   (a) Pour lancer Julia :

```
julia
```

   (b) Introduction à la syntaxe. La syntaxe est assez simple :

```
a = 5
b = 67.67e42
c = a + 3 * b ^ 4
m = "Bonjour"
d = sin(atan(3))
toi = "Rosa"
println("Bonjour, $toi !")
e1 = [12; 34; 45]
e2 = [12 34 45]
d = true
f = [12.0; 34; "Youhou !"]
B = [1 2; 1 3; 4 5]
```

   (c) les fonctions sont bien définies :

```
function add(a, b)
    return a + b
end

f(x) = 42 * x^3

add(4, 2)
f(a)
```

Renseignements obligatoires sur la syntaxe de Julia peuvent être trouvés sur les pages suivantes :

- `https://julialang.org/`
- `https://zestedesavoir.com/articles/78/a-la-decouverte-de-julia/`

(d) Le package `JuMP` et le solveur `Cbe JuMP` sont des langages de modélisation pour l'optimisation mathématique intégré dans Julia. Pour l'utiliser il faut d'abord ajouter le package associé :

```
using Pkg
Pkg.add("JuMP")
Pkg.add("Cbc")
```

Julia est un langage de modélisation et ne possède pas un solveur de programmation mathématique intégré. `CPLEX` est un solveur de programmation mathématique de haute performance, puissant pour la programmation linéaire, la programmation en nombres entiers et la programmation quadratique. Mais le solveur `CPLEX` n'est pas gratuit et requiert une license (facile à obtenir pour les étudiants mais le système est bridé). Nous utiliserons donc plutôt `Cbc` qui est un solveur de programmation mathématique open-source moins puissant intégré dans Julia. Dans la suite, nous présentons les commandes pour utiliser `Cbc`. Pour utiliser un solveur à partir de Julia il faut ajouter le package associé.

(e) Création du modèle :

Utiliser les commandes présentées dans cette section pour définir dans Julia un modèle associé au programme linéaire P_1. Les variables x_1 et x_2 représentent respectivement la quantité de deux types différents de yaourt produits. La fonction objectif maximise le profit de la production :

```
Z = max 4x + 5y
2x+y <= 800
x+2y <= 700
y <= 300
x,y >= 0
```

(f) Packages :

```
using JuMP
using Cbc
```

(g) Définition du modèle :

```
m = Model(Cbc.Optimizer)
```

(h) Définition des variables, exemples de définition de variables :

```
@variable(m, x) # No bounds
@variable(m, x >= lb) # Lower bound only (note: 'lb <= x' is not valid)
@variable(m, lb <= x <= ub) # Both lower and upper bounds
@variable(m, x == fixedval) # Fixed variable
@variable(m, x[1:M, 1:N]) # Un vecteur de variables
@variable(m, x[1:5], Bin) # Un vecteur de variabes binaires
```

Ajouter à votre modèle les contraintes de P_1.

(i) Pour résoudre le modèle, il faut exécuter :

```
optimize!(m)
```

La valeur de la solution optimale du modèle, le temps d'exécution et la solution optimale, peuvent etre récupérés en utilisant les commandes suivantes :

```
Objective value(m)
    solve time(m)
    value.(x)
    value(x[i])
```

Voici un exemple, en supposant que x est le vecteur de variables, profit et weight sont des vecteurs de coefficients :

```
println("Objective is: ", objective_value(m))
println("Solve time = ", solve_time(m))
println("Solution is:")
for i = 1:5
    print("x[$i] = ", value(x[i]))
    println(", p[$i]/w[$i] = ", profit[i]/weight[i])
end
```

Nous pouvons à tout moment visualiser le modèle en utilisant la commande suivante :

```
print(m)
```

Nous pouvons aussi écrire la totalité du modèle dans un fichier nommé `ex1.jl` et exéctuer la commande suivante :

```
include("ex1.jl")
```

(j) Lecture de fichier :
La commande suivante place le contenu du fichier dans un tableau :

```
using DelimitedFiles
data = readdlm("data.txt")
```

On peut ensuite extraire une partie du tableau. Par exemple, la commande suivante place les lignes 5 à 10 de la seconde colonne de data dans le vecteur `cost` :

```
cost = data[5:10, 2]
```

(k) Pour afficher le nombre de noeuds dans un arbre :

```
XXX.get(model, XXX.NodeCount())
```

2. Sur le problème de **Bin Packing Problem**.
On considère le problème P_1 et sa variante P_2 : $P_1 : x_{ij} \in \{0,1\}$ avec $x_{ij} = 1$ si et seulement si l'objet i est placé dans la boîte j et $y_j \in \{0,1\}$ avec $y_j = 1$ si et seulement si la boîte j contient au moins un objet. On a n objets et m boîtes.

$$PNLE_{\text{bin packing}} = \begin{cases} \min z = \sum_{j=1}^{m} y_j \\ \sum_{j=1}^{m} x_{ij} = 1, \quad \forall i \in \{1,\dots,n\} \\ \sum_{i=1}^{n} a_i x_{ij} \leq W, \quad \forall j \in \{1,\dots,m\} \\ x_{ij} \leq y_j, \quad \forall i,j \\ x_{ij} \in \{0,1\}, \quad \forall i,j \\ y_j \in \{0,1\}, \quad \forall j \end{cases}$$

Noter que l'on peut combiner les deux contraintes $\sum_{i=1}^{n} a_i x_{ij} \leq W$ et $x_{ij} \leq y_j$ en une seule contrainte :

$$\sum_{i=1}^{n} a_i x_{ij} \leq W y_j, \quad \forall j \in \{1,\dots,m\}$$

c'est ce qu'on utilise dans P_2

(a) Considérer les instances Bin Packing Problem se trouvant à l'adresse `https://site.unibo.it/operations-research/en/research/bpplib-a-bin-packing-problem-library` (prendre Falkenaeur et Scholl).

A noter que le format est le suivant :

- n : nombre d'objets
- c : capacité des boîtes
- w_j : poids de l'objet j pour $j \in \{1, \ldots, n\}$

Comparer les valeurs des relaxations linéaires des deux modèles précédents pour les instances précédentes. Comparer le temps de calcul pour obtenir les solutions optimales.

(b) Ajouter des coupes pour accélérer les temps de calcul. Reprendre les tests avec les jeux de données.

Solution.

1. Commencons par la modélisation du problème P_1 :

```
# Install required packages
using Pkg
Pkg.add("JuMP")
Pkg.add("Cbc")

# Load required packages
using JuMP
using Cbc

# Problem modeling for P1
# model to model
# Z = max 4x + 5y
# 2x+y <= 800
# x+2y <= 700
# y <= 300
# x,y >= 0
model = Model(Cbc.Optimizer)
@variable(model, x >= 0)
@variable(model, y >= 0)
@objective(model, Max, 4x + 5y)
@constraint(model, 2x + y <= 800)
@constraint(model, x + 2y <= 700)
@constraint(model, y <= 300)

optimize!(model)

# Print results
println("Objective is: ", objective_value(model))
println("Solve time = ", solve_time(model))
println("Solution is:")
println("x = ", value(x))
println("y = ", value(y))
```

Le programme a l'air de fonctionner, les résultats sont cohérents :

```
Presolve 2 (-1) rows, 2 (0) columns and 4 (-1) elements
0  Obj -0 Dual inf 8.999998 (2)
2  Obj 2200
Optimal - objective value 2200
After Postsolve, objective 2200, infeasibilities - dual 0 (0), primal 0 (0)
Optimal objective 2200 - 2 iterations time 0.002, Presolve 0.00
```

```
Objective is: 2200.0
Solve time = 0.004999876022338867
Solution is:
x = 300.0
y = 200.0
```

2. On passe maintenant au problème de Bin Packing Problem :
   (a) On commence par créer une fonction qui lit les données d'un fichier et crée le modèle associé :

```
function read_bpplib_instance(path)
    data = readlines(path)
    # First line contains n
    n = parse(Int, data[1])
    # Second line contains capacity
    c = parse(Int, data[2])

    # Collect all remaining numbers, on the n following lines
    weights = []
    for line in data[3:end]
        append!(weights, parse.(Int, split(line)))
    end

    if length(weights) != n
        error("Error: expected $n weights but got $(length(weights))")
    end

    return n, c, weights
end
```

Ensuite on construit les modèles pour P_1 :

```
function solve_P1(n, c, weights; m=n)
    model = Model(Cbc.Optimizer)

    @variable(model, x[1:n, 1:m], Bin)
    @variable(model, y[1:m], Bin)

    @objective(model, Min, sum(y[j] for j in 1:m))

    # Each item exactly in one bin
    @constraint(model, [i=1:n], sum(x[i, j] for j=1:m) == 1)

    # bin capacities
    @constraint(model, [j=1:m], sum(weights[i] * x[i,j] for i=1:n) <= c)

    @constraint(model, [i=1:n, j=1:m], x[i,j] <= y[j])

    optimize!(model)

    return model
end
```

et pour P_2 :

```
function solve_P2(n, c, weights; m=n)
    model = Model(Cbc.Optimizer)

    @variable(model, x[1:n, 1:m], Bin)
    @variable(model, y[1:m], Bin)
```

```
    @objective(model, Min, sum(y[j] for j in 1:m))

    # Each item exactly in one bin
    @constraint(model, [i=1:n], sum(x[i, j] for j=1:m) == 1)

    # bin capacities
    @constraint(model, [j=1:m], sum(weights[i] * x[i,j] for i=1:n)
    <= c * y[j])

    optimize!(model)

    return model
end
```

Le code complet utilisé dans cette partie est présent en annexe A.

Les résultats même pour un petit sous ensemble de problèmes mettent plus de 2500 secondes à être calculés.

(b) On ajoute maintenant des coupes pour accélérer les temps de calcul. On ajoute les coupes de type *covering inequalities* :

$$\sum_{i \in C} x_{ij} \leq |C| - 1 y_j, \quad \forall j \in \{1, \ldots, m\}, \forall C \subseteq \{1, \ldots, n\} \text{ tel que } \sum_{i \in C} a_i > W$$

Le code pour cette partie est le suivant :

```
function add_cover_cuts!(model, x, y, weights, c; max_subset_size=4)
    n = length(weights)
    m = length(y)
    for k in 2:max_subset_size
        for C in combinations(1:n, k)
            if sum(weights[i] for i in C) > c
                for j in 1:m
                    @constraint(model, sum(x[i,j] for i in C) <=
                    (length(C)-1)*y[j])
                end
            end
        end
    end
end
```

Exercice 8 Résolution du pire cas : problème Klee-Minty.
Soit le problème linéaire suivant :

$$\begin{cases} \max z = \sum_{j=1}^{n} 10^{n-j} x_j \\ 2 \sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \leq 100^{i-1}, \quad \forall i \in \{1, \ldots, n\} \\ x_j \geq 0, \quad \forall j \in \{1, \ldots, n\} \end{cases}$$

1. Résoudre pour $n = 1, 2, 3$ avec la méthode des tableaux.
2. Utiliser Julia pour résoudre le problème avec $n \geq 4$. Pour cela vous procéderez à des tests. Soit T la borne de temps maximum autorisé (par exemple 60 secondes).
   (a) Faire varier $n \in \{5, 10, 15, 20, 30, 50, 100, 150, 200, 250, \ldots\}$ tout en respectant la borne T.
   (b) Afficher les temps de calcul en fonction de n.
   (c) Vérifier que les temps de calcul sont similaires à $\mathsf{O}(2^n)$

Solution.

1. • Pour $n = 1$:

		c	1
c^J	variables de base		x_1
1	$x_1^1 = s_1$	1	1
	$z(x)$	0	0

La solution optimale est $x_1 = 1$ avec $z = 1$.

• Pour $n = 2$, le problème devient :

$$\begin{cases} \max z = 10x_1 + x_2 \\ x_1 \leq 1 \\ 20x_1 + x_2 \leq 100 \\ x_j \geq 0, \quad \forall j \in \{1, 2\} \end{cases}$$

Sous forme standard on a

$$\begin{cases} \max z = 10x_1 + x_2 + 0x_3 + 0x_4 \\ x_1 + x_3 = 1 \\ 20x_1 + x_2 + x_4 = 100 \\ x_j \geq 0, \quad \forall j \in \{1, 2, 3, 4\} \end{cases}$$

Le tableau initial est :

		c	10	1	0	0
c^J	variables de base		x_1	x_2	x_3	x_4
0	$x_1^1 = x_3$	1	1	0	1	0
0	$x_2^1 = x_4$	100	20	1	0	1
	$z(x)$	0	−10	−1	0	0

On a x_1 qui entre et x_3 qui sort, on effectue le pivot :

		c	10	1	0	0
c^J	variables de base		x_1	x_2	x_3	x_4
10	$x_1^2 = x_1$	1	1	0	1	0
0	$x_2^2 = x_4$	80	0	1	−20	1
	$z(x)$	10	0	−1	10	0

On a x_2 qui entre et x_4 qui sort, on effectue le pivot :

		c	10	1	0	0
c^J	variables de base		x_1	x_2	x_3	x_4
10	$x_1^3 = x_1$	1	1	0	1	0
1	$x_2^3 = x_2$	80	0	1	−20	1
	$z(x)$	90	0	0	−10	1

On a x_3 qui entre et x_1 qui sort, on effectue le pivot :

		c	10	1	0	0
c^J	variables de base		x_1	x_2	x_3	x_4
0	$x_1^4 = x_3$	1	1	0	1	0
1	$x_2^4 = x_2$	100	20	1	0	1
	$z(x)$	100	10	0	0	1

La solution optimale est $x_1 = 0$, $x_2 = 100$ avec $z = 100$.

- Pour $n = 3$, le problème devient :

$$\begin{cases} \max z = 100x_1 + 10x_2 + x_3 \\ x_1 \leq 1 \\ 20x_1 + x_2 \leq 100 \\ 200x_1 + 20x_2 + x_3 \leq 10000 \\ x_j \geq 0, \quad \forall j \in \{1, 2, 3\} \end{cases}$$

Sous forme standard on a

$$\begin{cases} \max z = 100x_1 + 10x_2 + x_3 + 0x_4 + 0x_5 + 0x_6 \\ x_1 + x_4 = 1 \\ 20x_1 + x_2 + x_5 = 100 \\ 200x_1 + 20x_2 + x_3 + x_6 = 10000 \\ x_j \geq 0, \quad \forall j \in \{1, 2, 3, 4, 5, 6\} \end{cases}$$

Le tableau initial est :

		c	100	10	1	0	0	0
c^J	variables de base		x_1	x_2	x_3	x_4	x_5	x_6
0	$x_1^1 = x_4$	1	1	0	0	1	0	0
0	$x_2^1 = x_5$	100	20	1	0	0	1	0
0	$x_3^1 = x_6$	10000	200	20	1	0	0	1
	$z(x)$	0	−100	−10	−1	0	0	0

Il y a beaucoup d'autres tableaux (8 au total), le dernier est :

		c	100	10	1	0	0	0
c^J	variables de base		x_1	x_2	x_3	x_4	x_5	x_6
0	$x_1^8 = x_4$	1	1	0	0	1	0	0
0	$x_2^8 = x_5$	100	20	1	0	0	1	0
1	$x_3^8 = x_3$	10000	200	20	1	0	0	1
	$z(x)$	10000	100	10	0	0	0	1

La solution optimale est $x_1 = 0$, $x_2 = 0$, $x_3 = 10000$ avec $z = 10000$.

2. Pour $n \geq 4$, on utilise Julia pour résoudre le problème.

   (a) On crée une fonction qui génère le modèle Klee-Minty pour une dimension n donnée :

```
# using Pkg
# Pkg.add("JuMP")
# Pkg.add("Cbc")
# Pkg.add("Printf")
# Pkg.add("DataFrames")
# Pkg.add("CSV")
# Pkg.add("Dates")

using JuMP
using Cbc
using Printf
using DataFrames
using CSV
using Dates


function build_klee_minty_model(n)
    model = Model(Cbc.Optimizer)
    set_silent(model)  # no terminal printing
```

```
    @variable(model, x[1:n] >= 0)

    # Constraints:
    for i in 1:n
        if i == 1
            @constraint(model, x[1] <= 1)
        else
            @constraint(model, 2*sum(10^(i-j)*x[j] for j in 1:i-1) + x[i]
            <= 100^(i-1))
        end
    end

    # Objective:
    @objective(model, Max, sum(10^(n-j) * x[j] for j in 1:n))

    return model, x
end


function solve_klee_minty(n)
    model, x = build_klee_minty_model(n)
    start_time = now()
    optimize!(model)
    end_time = now()
    elapsed = end_time - start_time

    # Retrieve solution
    x_val = value.(x)
    obj_val = objective_value(model)
    return x_val, obj_val, elapsed
end


function run_tests(ns; max_time_sec=60)
    results = DataFrame(n=Int[], obj=Float64[], time_sec=Float64[])

    for n in ns
        println("Solving n = $n ...")
        try
            x_val, obj_val, elapsed = solve_klee_minty(n)
            elapsed_sec = Dates.value(elapsed)/1e9
            if elapsed_sec > max_time_sec
                println("  Exceeded max time of $max_time_sec seconds,
                skipping further large n")
                break
            end
            push!(results, (n, obj_val, elapsed_sec))
            println(@sprintf("  Obj = %.4f, Time = %.3f sec",
            obj_val, elapsed_sec))
        catch e
            println("  Error solving n=$n: $e")
        end
    end
    return results
end


# Test sizes
ns = [5, 10, 15, 20]#, 30, 50, 100, 150, 200, 250]
```

```
results = run_tests(ns)
CSV.write("klee_minty_results.csv", results)

println("\nAll results saved to klee_minty_results.csv")
println(results)
```

(b) On affiche les résultats dans le terminal et on les sauvegarde dans un fichier texte.

(c) Les temps de calculs sont en effet de l'ordre de $O(2^n)$ et dans mon cas le solveur n'arrive pas à résoudre le problème pour $n \geq 25$ en moins de 60 secondes.

Exercice 9 Formulation : trajets d'un convoyeur dans un entrepôt.
Dans un grand entrepôt, un convoyeur électrique est utilisé pour acheminer les arrivages (sous forme de palettes) de la zone d'entrée jusqu'à leur place propre de stockage dans l'entrepôt. Ayant placé une palette dans le stock, le convoyeur peut ensuite aller puiser dans le stock une palette d'un article demandé pour la transporter à la zone de sortie. Il retourne ensuite à la zone d'entrée et recommence avec une nouvelle palette à placer dans le stock.
Appelons a_i (avec $i = 1, \ldots, m$) le nombre de palettes à transporter de la zone d'entrée à la zone i de stockage et b_j (avec $j = 1, \ldots, n$) le nombre de palettes à transporter de la zone j de stockage à la zone de sortie. Soit enfin t_{ij} le temps nécessaire pour aller de la zone i à la zone j (où $a_i > 0$ et $b_j > 0$). Nous pouvons admettre que $\{i \mid a_i > 0\} \cap \{j \mid b_j > 0\} = \varnothing$ et que $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$ (sans restriction de généralité).
Déterminer dans quel ordre les palettes doivent être transportées dans l'entrepôt de manière à minimiser le temps total de transport.

Solution.
On peut modéliser ce problème suit dans Julia :

```
using JuMP, Cbc

# a = vector of entry-to-storage pallet counts
# b = vector of storage-to-exit pallet counts
# t = m x n matrix of travel times
function solve_conveyor_assignment(a, b, t)
    m = length(a)
    n = length(b)
    model = Model(Cbc.Optimizer)
    set_silent(model)  # avoid printing too much

    # Decision variables: x[i,j] = number of pallets assigned from entry i to exit j
    @variable(model, x[1:m, 1:n] >= 0, Int)

    # Each entry pallet count is satisfied
    @constraint(model, [i=1:m], sum(x[i,j] for j=1:n) == a[i])

    # Each exit pallet count is satisfied
    @constraint(model, [j=1:n], sum(x[i,j] for i=1:m) == b[j])

    # Objective: minimize total travel time
    @objective(model, Min, sum(t[i,j] * x[i,j] for i=1:m, j=1:n))

    optimize!(model)

    return model, value.(x), objective_value(model)
end

# exemple
a = [2, 3]          # 2 pallets for storage 1, 3 pallets for storage 2
b = [1, 4]          # 1 pallet to exit 1, 4 pallets to exit 2
```

```
t = [3 5;          # travel times from storage to exit
     2 1]

model, assignment, total_time = solve_conveyor_assignment(a, b, t)

println("Assignment matrix:")
println(assignment)
println("Total transport time: $total_time")
```

A Code source complet pour le bin packing

Le code source complet utilisé pour les exercices sur le bin packing est donné ci-dessous.

```
# Install required packages
using Pkg
# Pkg.add("JuMP")
# Pkg.add("Cbc")
# Pkg.add("Glob")
# Pkg.add("CSV")
# Pkg.add("DataFrames")
# Pkg.add("Printf")

# Load required packages
using JuMP
using Cbc
using Glob
using CSV, DataFrames, Printf


function build_P1(n, c, weights; m=n)
    model = Model(Cbc.Optimizer)
    # Stop printing in terminal
    # set_silent(model)

    @variable(model, x[1:n, 1:m], Bin)
    @variable(model, y[1:m], Bin)

    @objective(model, Min, sum(y[j] for j in 1:m))

    # Each item exactly in one bin
    @constraint(model, [i=1:n], sum(x[i, j] for j=1:m) == 1)

    # bin capacities
    @constraint(model, [j=1:m], sum(weights[i] * x[i,j] for i=1:n) <= c)

    @constraint(model, [i=1:n, j=1:m], x[i,j] <= y[j])

    return model
end

function build_P2(n, c, weights; m=n)
    model = Model(Cbc.Optimizer)
    # Stop printing in terminal
    # set_silent(model)

    @variable(model, x[1:n, 1:m], Bin)
    @variable(model, y[1:m], Bin)

    @objective(model, Min, sum(y[j] for j in 1:m))

    # Each item exactly in one bin
    @constraint(model, [i=1:n], sum(x[i, j] for j=1:m) == 1)

    # bin capacities
    @constraint(model, [j=1:m], sum(weights[i] * x[i,j] for i=1:n) <= c * y[j])

    return model
end

function read_bpplib_instance(path)
```

```
    data = readlines(path)
    # First line contains n
    n = parse(Int, data[1])
    # Second line contains capacity
    c = parse(Int, data[2])

    # Collect all remaining numbers, on the n following lines
    weights = []
    for line in data[3:end]
        append!(weights, parse.(Int, split(line)))
    end

    if length(weights) != n
        error("Error: expected $n weights but got $(length(weights))")
    end

    return n, c, weights
end

function lp_relax_value(model)
    relax_integrality(model)
    optimize!(model)
    return objective_value(model)
end

function solve_relaxations_for_instances(instance_paths)
    results = Dict()

    for path in instance_paths
        n, c, weights = read_bpplib_instance(path)

        model_P1 = build_P1(n, c, weights)
        model_P2 = build_P2(n, c, weights)

        optimize!(model_P1)
        optimize!(model_P2)

        results[path] = (objective_value(model_P1), objective_value(model_P2))
    end

    return results
end


function generate_falkenauer_paths(base_dir="../instances/Falkenauer")
    T_sizes = [60, 120, 249, 501]
    U_sizes = [120, 250, 500, 1000]

    paths = String[]

    # T instances
    for s in T_sizes
        for i in 0:19
            filename = @sprintf("Falkenauer_t%d_%02d.txt", s, i)
            push!(paths, joinpath(base_dir, filename))
        end
    end

    # U instances
    for s in U_sizes
        for i in 0:19
```

```
            filename = @sprintf("Falkenauer_u%d_%02d.txt", s, i)
            push!(paths, joinpath(base_dir, filename))
        end
    end

    return paths
end

function generate_scholl_paths(base_dir="../instances/Scholl")
    return glob("*.txt", base_dir)
end

function generate_small_falkenauer_set(base_dir="../instances/Falkenauer")
    sizes = [60]
    paths = String[]

    for s in sizes
        for i in 0:2
            filename_t = @sprintf("Falkenauer_t%d_%02d.txt", s, i)
            # filename_u = @sprintf("Falkenauer_u%d_%02d.txt", s, i)
            push!(paths, joinpath(base_dir, filename_t))
            # push!(paths, joinpath(base_dir, filename_u))
        end
    end

    return paths
end

function generate_small_scholl_set(base_dir="../instances/Scholl")
    paths = String[]
    filename_1 = @sprintf("N1C1W1_A")
    filename_2 = @sprintf("N1C1W1_B")
    push!(paths, joinpath(base_dir, filename_1))
    push!(paths, joinpath(base_dir, filename_2))

    return paths
end

function get_all_instance_paths()
    # falk_paths = generate_falkenauer_paths()
    # scholl_paths = generate_scholl_paths()
    falk_paths = generate_small_falkenauer_set()
    scholl_paths = generate_small_scholl_set()
    return vcat(falk_paths, scholl_paths)
end

function save_results(results, filename="lp_relax_results.csv")
    df = DataFrame(Path = String[], LP_P1 = Float64[], LP_P2 = Float64[])

    for (path, (lp1, lp2)) in results
        push!(df, (path, lp1, lp2))
    end

    CSV.write(filename, df)
end


# Main execution
all_paths = get_all_instance_paths()

results = solve_relaxations_for_instances(all_paths)
```

```
save_results(results)


# instance_paths = ["path/to/instance1.txt", "path/to/instance2.txt"]
# results = solve_relaxations_for_instances(instance_paths)
```