Réallocation des bureaux de l'université Paris Dauphine

Léa Cohen-Solal
Shana Baroukh
Dirigé par Clément ROYER

1 1e scénario : les 5 ailes (Partie 1)

Dans le premier scénario, notre attention se focalise exclusivement sur les cinq ailes de l'université, nommées A1, B1, C1, P1 et N1. Pour ce cas, notre implémentation va procéder aux attributions suivantes :

- Aile A1 : attribuée au numéro 1
- Aile B1 : attribuée au numéro 2
- Aile P1 : attribuée au numéro 3
- Aile C1 : attribuée au numéro 4
- Aile N1 : attribuée au numéro 5

On note n le nombre d'ailes, et p le nombre de phases
n = 5 et p = 5

1.1 Programme linéaire

1.2 Variables

Vecteurs x et P

Nous avons défini les vecteurs de variables suivant :
$\mathbf{x} \in \mathbb{R}^{n\times n\times(p+1)}$ et $\mathbf{P} \in \mathbb{R}^{n\times(n-1)\times(p+1)}$ tel que

$$\mathbf{x} = \begin{pmatrix} x_{110} \\ \vdots \\ x_{11p} \\ x_{120} \\ \vdots \\ x_{12p} \\ \vdots \\ x_{1np} \\ x_{210} \\ \vdots \\ x_{nnp} \end{pmatrix} \quad et \quad \mathbf{P} = \begin{pmatrix} P_{110} \\ \vdots \\ P_{11p} \\ P_{120} \\ \vdots \\ P_{12p} \\ \vdots \\ P_{1np} \\ P_{210} \\ \vdots \\ P_{nnp} \end{pmatrix}$$

Nous avons défini les fonctions '**index(i, j, k)**' et '**index_P(i, j, k)**' qui facilite l'accès à l'indice correspondant des vecteurs.

$$\forall(i,j,k) \in \{1,...,n\}^2 \times \{0,...,p\}$$

$$x_{ijk} = \begin{cases} 1 & \text{si les bureaux de l'aile i se deplacent vers l'aile j à la phase k} \\ 0 & \text{sinon} \end{cases}$$

$$\forall(i,j,k) \in \{1,...,n\} \times \{1,....,n-1\} \times \{0,...,p\}$$

$$P_{ijk} = \begin{cases} 1 & \text{si les bureaux initialement en aile j sont en aile i à la phase k} \\ 0 & \text{sinon} \end{cases}$$

Étant donné que dans cette section nous considérons une relaxation de la contrainte x_{ijk} (et donc celle de P_{ijk}), les variables ne seront pas binaires. Par conséquent nous auront les contraintes :

$$0 \leq x_{ijk} \leq 1 \quad \text{et} \quad 0 \leq P_{ijk} \leq 1$$

.

Il s'agit ici de minimiser les déplacements des bureaux a travers les ailes. Sachant que lorsqu'un bureau i reste à sa place pendant une phase k, $x_{iik} = 1$ mais il ne s'agit pas d'un déplacement. Il faudra donc faire attention à distinguer les 2 cas. On va donc introduire une matrice I telle que

$$I = \begin{pmatrix} 0 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 0 \end{pmatrix} \Rightarrow \quad x_{iik} * I_{iik} = 0$$

.

La fonction objective modélisant ce problème s'écrit :

$$\text{Minimiser} \quad \mathbf{x}^T.I$$

1.2.1 Contraintes

1. Initialisation des positions

   Comme donné dans l'énoncé les position initiales sont connues. Nous initialisons donc les positions des ailes à l'aide du position P.
   Par exemple pour l'aile A1 (correspondant à l'aile 1 dans la modélisation), on initialise sa position par $P_{110} = 1$. L'aile initialement en aile 1 est bien en aile 1 à la phase 0. De plus chaque bureau doit retourner dans son aile de départ à la fin du processus. De la même manière, si des bureaux initialement en aile i sont en aile i à la phase 0 alors ces mêmes bureaux initialement en aile i sont en aile i à la phase p.
   Voir contrainte 1, 1e problème

2. Pour chaque phase, il y a un 'mouvement' dans chaque aile

   Un mouvement peut être :

   - d'une aile i à une aile j
   - d'une aile i à une aile i (sur place)

   Cas particulier pour la phase 1 : Étant donné que personne n'est dans l'aile N1 en phase 0, il est donc impossible que des bureaux de l'aile 5 se déplacent à la phase 1.

   Voir contrainte 2, 1e problème

3. Contraintes linéaire pour les vecteurs positions

   Pour pouvoir retracer le chemin d'un bureau, le vecteur P est crucial, il faut donc bien l'actualiser à chaque phase. Si des bureaux initialement en aile i sont en aile j à la phase 3 ($P_{ji3} = 1$) et que $x_{j34} = 1$ alors à la phase 4 les bureaux initialement en aile i sont en aile 3 donc $P_{3i4} = 1$.

   Pour pouvoir actualiser le vecteur on utilise la contrainte

   $$P_{ijk} = \sum_{l=1}^{n} P_{ljk} * x_{lik}$$

   Cependant cette contrainte n'étant pas linéaire nous l'avons transformé en contrainte linéaire en introduisant une nouvelle variable $\mathbf{Y} \in \mathbb{R}^{n\times(n-1)\times(p+1)\times n}$ telle que

   $$Y_{ijkl} = P_{ljk} * x_{lik} \qquad \Rightarrow \begin{cases} Y_{ijkl} \leq P_{ljk}, \\ Y_{ijkl} \leq x_{lik}, \\ Y_{ijkl} \geq P_{ljk} + x_{lik} - 1 \end{cases}$$

   Voir contrainte 3, 1e problème

4. Les bureaux ne sont pas dans plusieurs ailes

   Chacun des bureaux (initialement dans une aile) ne peuvent se trouver pour une phase donnée dans une et une seule aile.
   Voir contrainte 4 , 1e problème

5. Une aile ne contient pas plusieurs bureaux
   De la même manière une aile ne peut contenir qu'un type de bureau pour une phase donnée (*Ex : presidency, Students association...*)
   Voir contrainte 5, 1e problème

6. Contraintes de relaxation Les variables P, x (donc Y) ne sont pas binaires ($\{0, 1\}$) mais sont telles que

$$0 \leq x_{ijk} \leq 1$$
$$0 \leq P_{ijk} \leq 1$$
$$0 \leq Y_{ijkl} \leq 1$$

   Voir contrainte 6, 1e problème

7. Les ailes en travaux

   Pour une phase donnée, si une aile est en travaux :

   - Les bureaux dans cette aile doivent se déplacer
   - Aucun bureau ne peut accéder à cette aile

   Par exemple pour la première phase, l'aile B est en travaux. Les contraintes sont donc :
   $\sum_{i=1}^{n} x_{i21} * I_{i20} = 0$
   $\sum_{j=1}^{n} x_{2j1} * I_{2j0} = 1$

1.2.2 Résultats

Nous avons obtenu comme valeur optimale **7.18** ce qui est plutôt cohérent au fait que nous avons trouvé à la main un plan de déplacement minimal de 8. La solution optimale de la relaxation linéaire est une borne inférieure pour la solution optimale du problème initial avec des variables binaires.

2 2e scénario : Les 13 bureaux

Dans cette configuration revisitée, les ailes sont représentées par plusieurs bureaux. Dans notre modèle, nous ne prenons pas en compte les déplacements collectifs des bureaux appartenant à une même aile. En effet, chaque bureau a la capacité de se déplacer de manière autonome, indépendamment des autres bureaux de la même aile. Cependant à la fin du processus, tous les bureaux d'une aile doivent être regroupés. Comme pour le premier scénario, nous avons numérotés les bureaux de sorte que les bureaux initialement vide soient les derniers dans la numérotation.
Ici n = 13 et p = 5

- bureau P3 : attribuée au numéro 1
- bureau A1 : attribuée au numéro 2
- bureau A2 : attribuée au numéro 3
- bureau B3 : attribuée au numéro 4
- bureau P2 : attribuée au numéro 5
- bureau B2 : attribuée au numéro 6

- bureau P1 : attribuée au numéro 7
- bureau C1 : attribuée au numéro 8
- bureau C2 : attribuée au numéro 9
- bureau B1 : attribuée au numéro 10
- bureau N1 : attribuée au numéro 11
- bureau N2 : attribuée au numéro 12
- bureau N3 : attribuée au numéro 13

2.1 Variables

Vecteurs x et P Les vecteurs sont construits de la même manière que pour le premier scénario. Il y a juste une petite nuance dans leur definition

$$\forall(i,j,k) \in \{1,...,n\}^2 \times \{0,...,p\}$$
$$x_{ijk} = \begin{cases} 1 & \text{si le bureau placé en i se deplace vers l'emplacement j à la phase k} \\ 0 & \text{sinon} \end{cases}$$

$$\forall(i,j,k) \in \{1,...,n\} \times \{1,....,n-1\} \times \{0,...,p\}$$
$$P_{ijk} = \begin{cases} 1 & \text{si le bureau initialement placé en j est à l'emplacement i à la phase k} \\ 0 & \text{sinon} \end{cases}$$

La fonction objective est la même que celle du 1e scénario : Minimiser $\mathbf{x}^T.I$

2.1.1 Contraintes

Comme pour le 1e scénario, nous avons les contraintes d'initialisation (1), des mouvements par bureau (2), des vecteurs positions (3), les bureaux ne sont pas à plusieurs emplacements (4), Un emplacement ne contient pas plusieurs bureaux (5), relaxation (6).

Les emplacements finaux

Les positions finales des bureaux sont prédéterminées et ne correspondent pas nécessairement à leurs emplacements initiaux. Ainsi, le bureau qui se trouvait initialement à l'emplacement P2 (5) doit se retrouver à l'emplacement C1 (8) en fin de processus. Cette exigence est modélisée par la contrainte suivante : $P_{85p} = 1$
Contrainte 2, 2e problème

Les ailes en travaux

Dans ce contexte, lorsqu'une aile est en travaux, tous les bureaux situés dans cette aile doivent se déplacer ailleurs. Par exemple dans la phase 2, l'aile P est en travaux. Les bureaux placés en P1, P2, P3 (1,5,7) doivent donc se déplacer ailleurs. La contrainte se modélise par :
$liste\_indices = [1,5,7]$
Pour chaque indice j dans la liste d'indice :
$\sum_{i=1}^{n} x_{ij2} * I_{ij0} = 0$
$\sum_{i=1}^{n} x_{ji2} * I_{ji0} = 1$

Contrainte 3, 2e problème

2.1.2 Résultats

Nous avons obtenu comme valeur optimale **16**

2.2 Moving plan

2.2.1 1e problème avec les 5 ailes

Nous avons choisis d'expliquer seulement le déplacements des 5 ailes pour plus de lisibilité mais le fonctionnement est le même pour le problème avec les 13 bureaux.

À chaque phase k, les bureaux qui étaient initialement dans l'aile j ont une chance p d'être placés dans l'aile i, représentée par P[i, j, k]. Pour exploiter ces probabilités et trouver un plan de déplacement qui se rapprochera de la valeur optimale calculée précédemment (7.18, une borne inférieure au nombre réel de déplacements nécessaires), nous cherchons maintenant à concevoir un plan qui nécessitera 8 déplacements.

1. bureaux candidats pour chaque phase Pour la phase k et des bureaux initialement en aile j donnés, nous avons identifié tous les emplacements i susceptibles de contenir les bureaux de j à cette phase. Nous avons sélectionné tous les emplacements i pour lesquels la probabilité $P[i, j, k]$ est supérieure ou égale à 0.1. Cela donne les listes suivantes :

   [[1], [2], [3], [4]]
   [[1, 5], [1, 4, 5], [3], [4, 5]]
   [[1, 2, 5], [2, 5], [1, 4], [4, 5]]
   [[1, 2, 5], [2, 5], [1, 3], [1, 5]]
   [[2, 4, 5], [2, 5], [3], [4, 5]]
   [[1], [2], [3], [4]]

   Pour la phase 1, les bureaux initialement en aile 2 sont potentiellement en 1, 4 ou 5.
   Nous remarquons bien que pour les phases 0 et 5 comme les allocations initiales et finales sont imposées alors les probabilités sont à 1.

2. Combinaisons possibles
   Pour chaque phase nous allons construire les combinaison possibles avec les bureaux candidats. Pour une phase k une combinaison [i1, i2, i3, i4] représente une allocation possible et probable pour des bureaux initialement en i1, i2, i3, i4. Par exemple en phase 1 une combinaison possible est [1,4,3,5] (les bureaux initlalement en aile 1 sont en 1, les bureaux initialement en aile 2 sont en 4 etc...). Par contre la combinaison [1,4,3,4] n'est pas possible car une aile ne contient pas plusieurs "types" de bureaux. On ecarte donc ces combinaisons impossibles.
   L'ensemble des combinaisons est très grand mais voici une partie d'entre eux :
   ([1, 2, 3, 4], [1, 4, 3, 5], [1, 2, 4, 5], [1, 2, 3, 5], [2, 5, 3, 4], [1, 2, 3, 4])
   ([1, 2, 3, 4], [1, 4, 3, 5], [1, 2, 4, 5], [1, 2, 3, 5], [4, 2, 3, 5], [1, 2, 3, 4])
   ([1, 2, 3, 4], [1, 4, 3, 5], [1, 2, 4, 5], [1, 2, 3, 5], [5, 2, 3, 4], [1, 2, 3, 4])
   ([1, 2, 3, 4], [1, 4, 3, 5], [1, 2, 4, 5], [2, 5, 3, 1], [2, 5, 3, 4], [1, 2, 3, 4])
   ([1, 2, 3, 4], [1, 4, 3, 5], [1, 2, 4, 5], [2, 5, 3, 1], [4, 2, 3, 5], [1, 2, 3, 4])
   ([1, 2, 3, 4], [1, 4, 3, 5], [1, 2, 4, 5], [2, 5, 3, 1], [5, 2, 3, 4], [1, 2, 3, 4])
   ([1, 2, 3, 4], [1, 4, 3, 5], [1, 2, 4, 5], [5, 2, 3, 1], [2, 5, 3, 4], [1, 2, 3, 4])

   Par exemple pour la première combinaison
   phase 0 : [1, 2, 3, 4]
   phase 1 : [1, 4, 3, 5] ⇒ déplacement de 2 vers 4 et de 4 vers 5
   phase 2 : [1, 2, 4, 5] ⇒ déplacement de 4 vers 2 et de 3 vers 4
   phase 3 : [1, 2, 3, 5] ⇒ déplacement de 4 vers 3
   phase 4 : [2, 5, 3, 4] ⇒ déplacement de 1 vers 2, de 2 vers 5 et de 5 vers 4
   phase 5 : [1, 2, 3, 4]) ⇒ déplacement de 2 vers 1 , de 5 vers 2
   Le nombre de déplacement pour cette combinaison est donc de 10. Elle ne sera pas choisie car son nombre de déplacements n'est pas optimal.

3. Combinaison minimisant le nombre de déplacements
   On va maintenant utiliser la fonction $calcul_deplacements(combinaison)$ qui comptera le nombre de déplacements pour chaque combinaison. Les combinaisons ayant un nombre de déplacements de 8 seront donc choisie.

2.3 Graphes

Nous avons modélisé les déplacements des bureaux avec un graphe comme représenté sur le schéma du sujet. La fonction **'new_configuration(configuration, position, k)'** permet de changer la position des sommets du graphes en fonction des déplacements. La fonction **'update_edges(position)'** permet quant à elle de reconstituer les arêtes du graphes pour que le graphe soit lisible de phase en phase. **NB :** Les sommets N1, N2 et N3 dans le graphe signifie qu'aucun bureau ne se trouve à cette endroit durant la phase.

2.4 Éviter le regroupement de 'Presidency' et 'Students association'

Les bureaux de la présidence sont initialement en C1 et C2 (8 et 9) et les associations étudiantes en P1 et P2 (5 et 7). Pour chaque phase on va donc empecher que :

- les bureaux initialement en 8 soient à côté des bureaux initialement en 5
- les bureaux initialement en 8 soient à côté des bureaux initialement en 7
- les bureaux initialement en 9 soient à côté des bureaux initialement en 5
- les bureaux initialement en 9 soient à côté des bureaux initialement en 7

Pour cela nous avons crée un dictionnaire $'near\_to'$ qui à chaque emplacement associe ses emplacements voisins. Par exemple les voisins de B3 sont A2 et B2 qui se traduit par $near\_to[4] = [3, 6]$
Si à une phase donnée les bureaux initialement en C1 se trouvent en B3 alors les bureaux initialement en P1 et P2 ne peuvent pas être en A2 ou B2.

3 Programme quadratique : Ajout d'une pénalité (Partie 2)

3.1 La fonction objective

Nous souhaitons reformuler notre problème sous forme quadratique en ajoutant une pénalité à la fonction objective. Cette pénalité mesure la différence entre l'allocation actuelle (z_k) et l'allocation finale (z_F) pour chaque phase. L'objectif est d'encourager l'allocation des bureaux dès les premières phases à se rapprocher de l'allocation finale.

La forme générale de la pénalité est la suivante :

$$\lambda \sum_{k=0}^{5} \|z_k - z_F\|^2$$

Où :

- λ est un poids positif qui représente le poids que l'on veut appliquer à la pénalité.
- z_k est le vecteur des allocations de bureaux à la phase k.
- z_F est le vecteur correspondant des allocations finales.

Dans notre code, la fonction objective est exprimée comme suit :

$$\text{Minimize} \quad x^T I + \lambda \cdot (\|P_{\text{penalty}}\|_2)^2$$

Où :

- `x.T @ I` représente la partie linéaire de la fonction objective provenant de la Partie 1.
- $\|P_{\text{penalty}}\|_2$ calcule la norme euclidienne du terme de pénalité pour mettre la fonction objective sous forme quadratique.
- λ, noté `l`, est le poids appliqué à l'ensemble du terme.

3.2 Les contraintes

3.2.1 Présentation des contraintes

Nous reprenons toutes les contraintes de notre programme linéaire, en ajoutant uniquement les contraintes liées à la pénalité. Les contraintes attribuent à la pénalité la différence entre l'affectation à la phase courante et celle à la phase finale :

$$P_{\text{penalty}}[i,j,k] = P[i,j,k] - P[i,j,p]$$

où p est la dernière phase.

Ces contraintes concernent le terme de pénalité (P_{penalty}) et est mise en œuvre de la manière suivante :

- $P_{\text{penalty}}[i,j,k]$ représente le terme de pénalité associé au bureau j, dans le bureau i à la phase k.
- $P[i,j,k]$ est la variable indiquant si le bureau initialement placé en j est affecté au bureau i à la phase k.
- $P[i,j,p]$ est la variable indiquant si le bureau initialement en j est affecté au bureau i à la dernière phase.

  Étant donné que nous connaissons les affectations finales, toutes les valeurs $P[i,j,p]$ sont connues dès le début.

3.3 Test pour λ= 1 et λ= 100

Pour $\lambda = 1$, le résultat obtenu est **44.20** tandis que pour $\lambda = 100$ le résultat obtenu est **2451**.
En manipulant le vecteur x des déplacements, on obtient 18,9 déplacements pour λ= 1 et 21,7 déplacements pour λ= 100. Le nombre de déplacements pour λ= 1 est donc plus petit que pour λ= 100.

3.4 Interprétation de la pénalité et de son poids

Dans notre modèle d'optimisation, le paramètre λ joue un rôle crucial en influençant le compromis entre la rapidité de convergence vers la solution optimale et la minimisation du nombre de déplacements de bureaux. Comprendre le comportement de λ nous permet d'ajuster notre approche selon le paramètre que nous voulons optimiser.

- Lorsque λ est grand, la fonction objective accorde une importance significative à la pénalité, favorisant ainsi une convergence rapide vers l'allocation finale des bureaux. Le modèle cherchera à converger rapidement vers la solution optimale, ce qui pourrait entraîner un nombre plus élevé de déplacements des bureaux.
- À l'inverse, lorsque λ est petit, la pénalité dans la fonction objective devient moins prépondérante par rapport à la partie linéaire du problème. Cela conduit à privilégier la minimisation du nombre total de déplacements de bureaux, même si cela peut entraîner une convergence plus lente vers l'allocation finale.

Le but est de trouver un équilibre dans notre choix de λ qui nécessite une évaluation des priorités du projet.

En ajustant judicieusement λ, notre modèle d'optimisation devient un outil flexible capable de s'adapter aux différentes préférences et exigences du processus de rénovation des bureaux.

Pour illustrer ce point, nous avons effectué des calculs pour déterminer le nombre minimal de déplacements pour chaque valeur de λ, en considérant l'ensemble des plans de déplacement possibles (voir Section sur le calcul du nombre minimal de déplacements pour $\lambda = 1$ et 100). Nous avons employé les mêmes méthodes que celles utilisées pour identifier un plan de déplacement dans le problème. Pour chaque combinaison envisageable, nous avons calculé le nombre correspondant de déplacements, puis sélectionné le minimum. Pour un λ de 1, le nombre minimal de déplacements s'élève à 18, tandis que pour un λ de 100, il est de 21, soit 3 déplacements supplémentaires par rapport à un λ de 1. Cette augmentation semble cohérente avec nos explications précédentes, indiquant que le λ est 'proportionnel' au nombre de déplacements.

3.5 Question 8 : Pénalité par rapport au vecteur initial

Ici la forme générale de la pénalité est la suivante :

$$\lambda \sum_{k=0}^{5} \|z_k - z_I\|^2$$

Où :

- λ est un poids positif qui représente le poids que l'on veut appliquer à la pénalité.
- z_k est le vecteur des allocations de bureaux à la phase k.
- z_I est le vecteur correspondant aux allocations initiales.

3.6 Test pour λ= 1 et λ= 100

Pour $\lambda = 1$, le résultat obtenu est **34.5** tandis que pour $\lambda = 100$ le résultat obtenu est **1205**.
En manipulant le vecteur x des déplacements, on obtient 20,6 déplacements pour λ= 1 et 25,14 déplacements pour λ= 100. Le nombre de déplacements pour λ= 1 est donc plus petit que pour λ= 100.

3.7 Différence de pénalité entre vecteur_final et vecteur_initial

Les 13 bureaux

Pour le 2e problème des 13 bureaux, étant donné que les bureaux ne doivent pas revenir à leur position initiale après les travaux, on a P(i,j,p) != P(i,j,0). Il n'est donc pas pertinent d'imposer une pénalité basée sur la position initiale contrairement à une pénalité fondée sur la position finale.
Il est clair que l'optimisation du nombre de déplacements n'est pas atteinte lorsque l'on applique une pénalité basée sur le vecteur initial, par opposition à une pénalité appliquée sur le vecteur final.
PVI représente la pénalité relative au vecteur initial.
PVF désigne la pénalité relative au vecteur final.
$\lambda = 1$: l'usage du PVI entraîne 2 déplacements supplémentaires par rapport au PVF.
$\lambda = 100$: l'application du PVI résulte en 3 déplacements supplémentaires par rapport au PVF.

Les 5 ailes

En revanche, pour le 1e problème concernant les 5 ailes, comme les bureaux doivent retrouver leur emplacement initial une fois les travaux terminés, on a P(i,j,p) = P(i,j,0). Imposer une pénalité en fonction de la position initiale est équivalent à une pénalité basée sur la position finale.

4 Programme SDP (Partie 3)

Dans cette partie nous utilisons les mêmes variables x et P en ajoutant un vecteur u et une matrice U tels que :
$x\_dimension = (n * n * (p + 1))$
$P\_dimension = (n * (n - 1) * (p + 1))$
$u\_dimension = x\_dimension + P\_dimension$
$U = matriceu\_dimension$ x $u\_dimension$

Le problème est relaxé par $U \succeq uu^\top$

4.1 La fonction objective

La forme générale de la fonction à minimiser est donnée par :

$$\mathbf{Tr}(diag(x) \cdot I)$$

, où I est obtenue grâce à la fonction `give_equality_SDP()`.

4.2 Les contraintes

Contraintes de relaxation

La matrice U est définie telle que

- $U_{ii} = 1 \ \forall i \in \{1, ..., u\_dimension\}$
- $U \succeq u.u^T \Rightarrow U - u.u^T \succeq 0$

On a donc posé

$$Y = \begin{bmatrix} U & u^T \\ u & z \end{bmatrix} \succeq 0$$

avec z = 1, de telle sorte à ce que la contrainte soit acceptée par cvx.

Lien entre x,P et u

Pour les $x\_dimension$ premiers éléments de u on a posé : $u[i] = 2.x[i] - 1$.
Pour les $P\_dimension$ derniers éléments de u on a posé $u[i] = 2.P[i] - 1$.

La fonction `index_u(choix, i, j, k)` est conçue pour sélectionner un élément soit de x (pour `choix = 1`) soit de P (pour `choix = 2`).

Nous avons préservé toutes les contraintes concernant x et P, à l'exception de celles qui nécessitaient un produit entre x et P, que nous avons dû convertir en un ensemble de contraintes linéaires.

Nous avons transformé la contrainte

$$P_{ijk} = \sum_{l=1}^{n} P_{ljk} \times x_{lik}$$

en

$$P_{ijk} = \sum_{l=1}^{n+1} U[\texttt{index\_u}(2, l, j, k-1), \texttt{index\_u}(1, l, i, k)]$$

L'objectif de reformuler ce programme sous une forme semi-définie positive (SDP) est de simplifier l'expression de ce type de contrainte. Ainsi, la matrice U permet de récupérer directement le produit souhaité entre x et P.

4.3 Question 10, comparaison du nombre de variables et de contraintes

Nous avons testé le modèle SDP uniquement avec le problème des 5 ailes.
n = 5 et p = 5

Nombre de variables et contraintes dans le modèle linéaire

x : Variable de n*n*(p+1) termes
P : Variable de n*(n-1)*(p+1) termes
Y : Variable de n*n*n*(p+1) termes

Nous avons donc un total de **1020 variables** et de **1702 contraintes.**

Nombre de variables et contraintes dans le modèle SDP

x : Variable de n*n*(p+1) termes
P : Variable de n*(n-1)*(p+1) termes
u : Variable de n*n*(p+1) + n*(n-1)*(p+1) termes
U : Variable de (n*n*(p+1) + n*(n-1)*(p+1))$\hat{}$2 termes
Y : même dimension que U

z : Variable de 1 terme

Nous avons donc un total de **146 341 variables** et de **703 contraintes**.

Bien que la transformation du problème ait entraîné une augmentation du nombre de variables par rapport au modèle linéaire, cela nous a permis de simplifier les contraintes.

Effectivement, dans le cadre du modèle linéaire, l'actualisation de P(i,j,k) implique la multiplication de deux variables, transformée ensuite en contraintes linéaires, ce qui a requis l'emploi d'un nombre conséquent de contraintes. À l'opposé, dans le modèle SDP, il a suffi d'initialiser la matrice Y semi-définie positive et de l'utiliser pour mettre à jour P.

On en conclut que **le modèle SDP utiliser plus de variables mais moins de contraintes que le modèle linéaire.**

4.4 Résultats

Nous avons obtenu un nombre de 4 déplacements. Des améliorations sur le modèle doivent être fait. Peut être exprimer toutes les contraintes en fonction de U et non en fonction de x et P.