Projet - Course d'avions

Compte rendu projet SOD321

Elaboré par :
Mehdi Ghrabli
Mohamed Issa

Encadré par :
Mme. Sourour Elloumi
M. Mathieu Verchère

3ème Année Techniques avancées

Année universitaire : 2021/2022

Table des matières

1 Les formulations du problème **4**

1.1 Formulation polynomiale 4

1.2 Formulation exponentielle 6

2 Cas tests **10**

2.0.1 Instance n=6 10

2.0.2 Instance n=20 12

2.0.3 Instance n=40 13

2.0.4 Instance n=70 14

Table des figures

1 Exemple de répartition 3

1.1 Trajectoire optimale sans élimination des sous-tours 5
1.2 Trajectoire optimale sans la contrainte des visites 6

2.1 Cas test avec n=6. 10
2.2 Résolution avec la formulation polynomiale 11
2.3 Résolution avec la formulation exponentielle. 11
2.4 Résolution avec la formulation exponentielle sans séparation. 11
2.5 Cas test avec n=20. 12
2.6 Résolution avec la formulation polynomiale 12
2.7 Cas test avec n=40. 13
2.8 Résolution avec la formulation polynomiale 13
2.9 Résolution avec la formulation exponentielle. 13
2.10 Cas test avec n=70. 14
2.11 Résolution avec la formulation polynomiale 14

Introduction

Etant donné un nombre n d'aérodromes appartenant chacune à une région parmi N_r, on se propose de calculer la distance minimale qu'on peut parcourir tout en visitant toutes les régions avec un nombre minimal d'aérodromes à visiter.

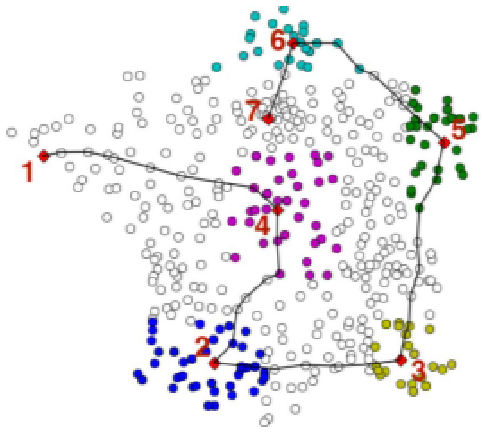


FIGURE 1 – Exemple de répartition

On s'inpire dans notre travail du modèle de voyageur de commerce sur un graphe orienté pour proposer deux fomulations : Une première avec un nombre polynomial de contraintes et une deuxiéme avec un nombre exponentiel.

Chapitre 1

Les formulations du problème

1.1 Formulation polynomiale

Pour la formulation avec un nombre polynomiale de contraintes, on utilise la modélisation suivante :

$$(P_{sous-tours}) : \begin{cases} \min \sum_{(i,j)\in\mathcal{V}} d_{ij}x_{ij} & \\ \sum_{i=1}^{n} y_i \geq A_{min} & \\ R \geq d_{ij}x_{ij} & \forall i,j \in [\![1,n]\!] \\ u_j \geq u_i + 1 - n(1-x_{ij}) & \forall i,j \in [\![1,n]\!] \\ \sum_{k\in\mathcal{R}_j} y_k \geq 1 & \forall j \in [\![1,Nr]\!] \\ \sum_{i=1}^{n} x_{ij} = y_j & \forall j \in \mathcal{V}\backslash\{d\} \\ \sum_{j=1}^{n} x_{ij} = y_i & \forall i \in \mathcal{V}\backslash\{f\} \\ \sum_{j=1}^{n} x_{fj} = 0 & \\ \sum_{j=1}^{n} x_{jd} = 0 & \\ x_{ii} = 0 & \forall i \in [\![1,n]\!] \\ y_d = 1 & \\ u_d = 0 & \\ x_{ij} \in \{0,1\}, y_i \in \{0,1\} & \forall i,j \in \mathcal{V} \end{cases}$$

Les variables du problème

- $\mathcal{V}$ represente l'ensemble d'aérodromes.
- le noeud de départ $d \in \mathcal{V}$
- le noeud d'arrivée $f \in \mathcal{V}$
- A_{min} le nombre minimal d'aérodromes à visiter.
- la matrice symétrique d calcule tout les distance entre deux aérodrome : d_{ij} est la distance entre l'aérodrome i et l'aérodrome j calculée à partir des données d'entrée.
- R la distance maximale qu'un avion peut parcourir sans se poser.
- N_r le nombre de régions à visiter (on ne compte pas la région indexée par 0)
- $\mathcal{R} = \mathcal{R}_1 \cup .. \cup \mathcal{R}_{N_r} \subset \mathcal{V}$ l'ensemble des régions.

Les variables du modèle

- Les x_{ij} sont des variables binaires qui prennent la valuer 1 si on se fait le trajet $i \to j$ durant le parcours optimal, 0 sinon.
- y un vecteur binaire où $y_i := 1$ ssi on visite l'aérodrome.
- Les u_i represenant le nombre de sommets visités depuis le sommet d.

Modélisation des contraintes

Nous rappelons que les contrintes à respecter sont :

- Visiter au moins A_{min} aérodromes.
- L'avion ne peut parcourir qu'une distance maximale de R avant de se poser.
- Visiter toutes les régions.

Ces contraintes sont modélisées par les inégalités $\sum_{i=1}^{n} y_i \geq A_{min}$, $R \geq d_{ij}x_{ij}$ et $\sum_{k \in \mathcal{R}_j} y_k \geq 1$ respectivement. En plus de ça, on va imposer des contraintes supplémentaires pour avoir une solution réalisable physiquement :

- *Elimination des sous tours* represntée par les contraintes : $u_j \geq u_i + 1 - n(1 - x_{ij})$. Ceci nous evite le cas d'avoir multiples trajectoire non-connexes du type :

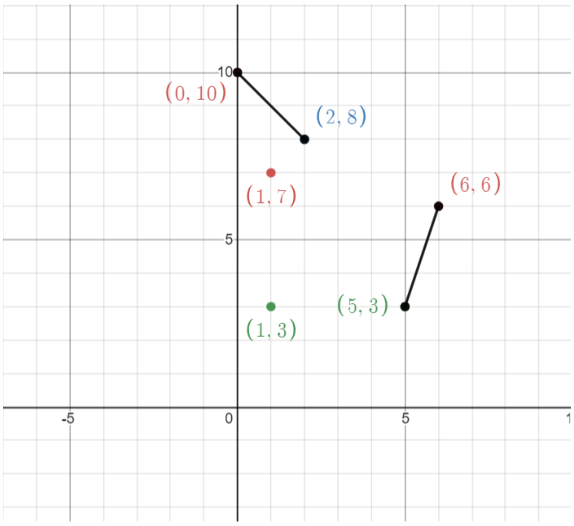


Figure 1.1 – Trajectoire optimale sans élimination des sous-tours

- Contraintes $\sum_{i=1}^{n} x_{ij} = y_j$ et $\sum_{i=1}^{n} x_{ij} = y_j$ pour marquer les visites des points, sinon, et si on ignore par exemple la première contrainte ($\sum_{i=1}^{n} x_{ij} = y_j$) on observe le graphe suivant

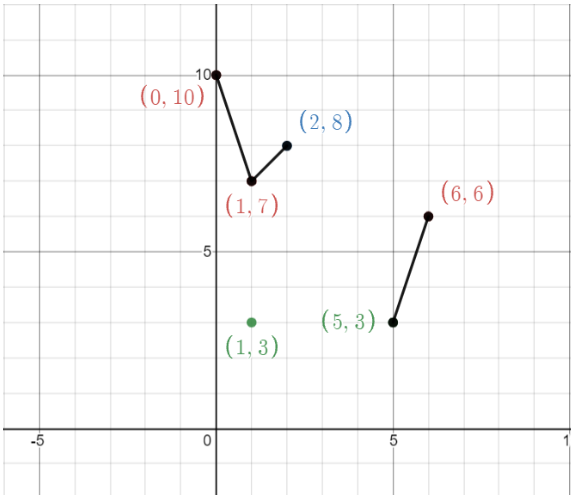


FIGURE 1.2 – Trajectoire optimale sans la contrainte des visites

On pourrait penser qu'on peut éviter ce comportement en éliminant les sous-tours, cependant, et dans ce cas, seulement l'arc $(5,3) \to (6,6)$ prend la valeur 1 (l'arc $(6,6) \to (5,3)$ prend la valeur 0) Donc on n'a pas de sous tours et le noeud $(6,6)$ n'est pas marqué comme visité ($y_6 = 0$). D'ailleurs, on peut vérifier ça aussi en observant que le chemin maintenant est plus long que dans la figure1.1 vu qu'on doit visiter encore un noeud pour verifier la contrainte $\sum_{i=1}^{n} y_i \geq A_{min}$. Dans ces deux, on ne visite que 4 noeuds.

— Contraintes de *début* et de *fin* du trajet : $\sum_{j=1}^{n} x_{fj} = 0$ et $\sum_{j=1}^{n} x_{jd} = 0$. A l'aide des contraintes mentionnées ci-dessus, on garantit avec ces équations là d'avoir aucun arc qui sort de f ni un arc qui entre en d ce qui définit le début et la fin.

— Les contraintes $x_{ii} = 0$ étaient mises pour éliminer les cas on prend des noeuds isolés, mais ceci est garanti grâce à la contrainte : $u_j \geq u_i + 1 - n(1 - x_{ij})$, en prenant $x_{ii} = 1$, on obtient $u_i \geq u_i + 1$.

— Les initialisations : $y_d = 1$ et $u_d = 0$

1.2 Formulation exponentielle

La formulation exponentielle donnée ici ne diffère de la formulation polynomiale qu'au niveau de l'expression de la contrainte d'absence des cycles. Le problème est donné par :

$$
(P_{sous-tours}) : \begin{cases}
\min \sum_{(i,j)\in\mathcal{A}} d_{ij}x_{ij} & \\
\sum_{i=1}^{n} y_i \geq A_{min} & \\
x_{ii} = 0 & \forall i \in [\![1,n]\!] \\
d_{ij}x_{ij} \leq R & \\
\sum_j x_{ij} = y_i & \forall i \in \mathcal{V}\backslash\{f\} \\
y_d = 1 & \\
\sum_{j=1}^{n} x_{fj} = 0 & \\
\sum_{j=1}^{n} x_{jd} = 0 & \\
y_j = \sum_{i=1}^{n} x_{ij} & \forall j \in \mathcal{V}\backslash\{d\} \\
\sum_{k\in\mathcal{R}_j} y_k \geq 1 & \forall j \in [\![1,Nr]\!] \\
\sum_{(i,j)\in E(S)} x_{ij} \leq |S|-1 & \forall S \subset \mathcal{V}, 2 \leq |S| \leq n-2 \\
x_{ij} \in \{0,1\}, y_i \in \{0,1\} & \forall i,j \in \mathcal{V}
\end{cases}
$$

Le nombre exponentiel des contraintes provient provient du nombre |S| des ensembles à explorer afin de garantir l'absence des sous-tours qui de l'ordre de 2^n. En réalité l'énumération de tous les ensembles est une méthode assez lente.

Ce qu'on fait c'est qu'on déroule un algorithme qui cherche une solution inadmissible en se restreignant à un sous ensemble $\mathcal{I}$ et en ajoutant de proche en proche les contraintes des sous-tours. Les étapes de l'algorithme sont les suivants :

1. Initialisation : $\mathcal{I} = \varnothing$,
2. Résoudre le problème $P_{\mathcal{I}} \to \tilde{x}$,
   (a) si la solution $\tilde{x}$ n'est pas réalisable : On résout le problème de séparation $SEP \to \tilde{S}$,
      i. Si $\tilde{S}$ est trouvé :
         — $\mathcal{I} \leftarrow \mathcal{I} + \tilde{S}$,
         — retour à 2.
      ii. sinon : arrêt
   (b) sinon : arrêt.

Le problème $P_{\mathcal{I}}$ est donné par :

$$(P_{\mathcal{I}}) : \begin{cases} \min \sum_{(i,j)\in\mathcal{A}} d_{ij}x_{ij} & \\ \sum_{i=1}^{n} y_i \geq A_{min} & \\ x_{ii} = 0 & \forall i \in [\![1,n]\!] \\ d_{ij}x_{ij} \leq R & \\ \sum_j x_{ij} = y_i & \forall i \in \mathcal{V}\backslash\{f\} \\ y_d = 1 & \\ \sum_{j=1}^{n} x_{fj} = 0 & \\ \sum_{j=1}^{n} x_{jd} = 0 & \\ y_j = \sum_{i=1}^{n} x_{ij} & \forall j \in \mathcal{V}\backslash\{d\} \\ \sum_{k\in\mathcal{R}_j} y_k \geq 1 & \forall j \in [\![1,Nr]\!] \\ \sum_{(i,j)\in E(S)} x_{ij} \leq |S|-1 & \forall S \subset \mathcal{I} \\ x_{ij} \in \{0,1\}, y_i \in \{0,1\} & \forall i,j \in \mathcal{V} \end{cases}$$

Le problème de séparation *SEP*, consiste à chercher un ensemble $\tilde{S}$, telle que l'inégalité associée à $\tilde{S}$ soit violée par $\tilde{x}$, c-à-d ($\sum_{(i,j)\in E(\tilde{S})} \tilde{x}_{ij} > |\tilde{S}|-1$). Pour cela on cherche à maximiser l'écart ($\sum_{(i,j)\in E(\tilde{S})} \tilde{x}_{ij} - |\tilde{S}|+1$).

Le problème de séparation *SEP* est un problème quadratique donné par :

$$(SEP) : \begin{cases} \max \sum_{(i,j)\in E(S)} a_i a_j \tilde{x}_{ij} - (\sum_{i=1}^{n} a_i - 1) & \\ \sum_{i=1}^{n} a_i \geq 2 & \\ a_i \in \{0,1\} & \forall i \in [\![1,n]\!] \end{cases}$$

L'algorithme que nous avons déroulé en Julia est donné par :

— Définir le problème maître : model=Model(Gurobi.Optimizer),
— Déclarer les variables du problème maître,
— Définir l'objectif du model,
— Ajouter toutes les contraintes sauf celle des sous-tours,
— Résoudre le problème model : JuMP.optimize!(model),
— fixer un nombre maximal d'itérations : max_iter=999,
— nb_iter=1,
— isOptim=false : =true lorsqu'on trouve une solution réalisable du (P_sous-tours),
— Tant que (!(isOptim) et (nb_iter<=max_iter))

1. nb_iter ← nb_iter+1,
2. Définir le problème SEP : sep=Model(Gurobi.Optimizer),
3. Déclarer les variables du problème SEP,
4. Définir l'objectif du sep,
5. Résoudre le problème sep : JuMP.optimize!(sep)
6. Si on a (JuMP.objective_value(sep)>0), c'est à dire qu'on a trouvé un ensemble $\tilde{S}$ tel que $\tilde{x}$ viole la contrainte des sous-tours,
   — on ajoute la contrainte des sous-tours restreinte à l'ensemble $\tilde{S}$ au modèle model,

— On résout de nouveau model : JuMP.optimize!(model).

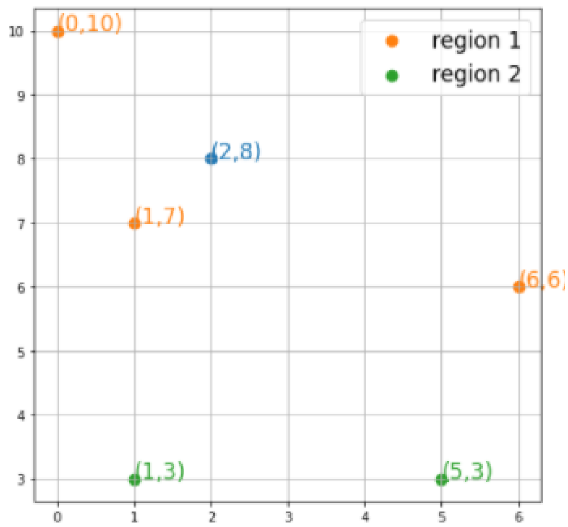
7. sinon : isOptim=true : On a trouvé une solution réalisable pour (P_sous-tours).

Chapitre 2

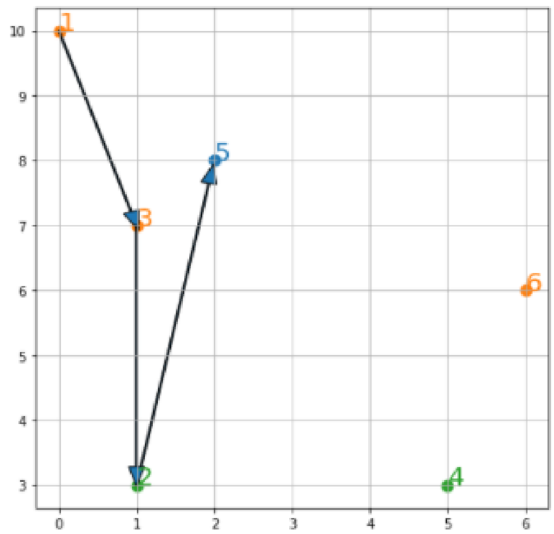
Cas tests

Dans cette partie, on essaie de résoudre le problème sous différentes configurations d'instances.

2.0.1 Instance n=6



(a) La répartition des aérodromes.



(b) La solution réalisable.

FIGURE 2.1 – Cas test avec n=6.

```
Explored 1 nodes (21 simplex iterations) in 0.07 seconds
Thread count was 12 (of 12 available processors)

Solution count 2: 12 13

Optimal solution found (tolerance 1.00e-04)
Best objective 1.200000000000e+01, best bound 1.200000000000e+01, gap 0.0000%

User-callback calls 84, time in user-callback 0.00 sec
Objective value: 12.0
```

Figure 2.2 – Résolution avec la formulation polynomiale

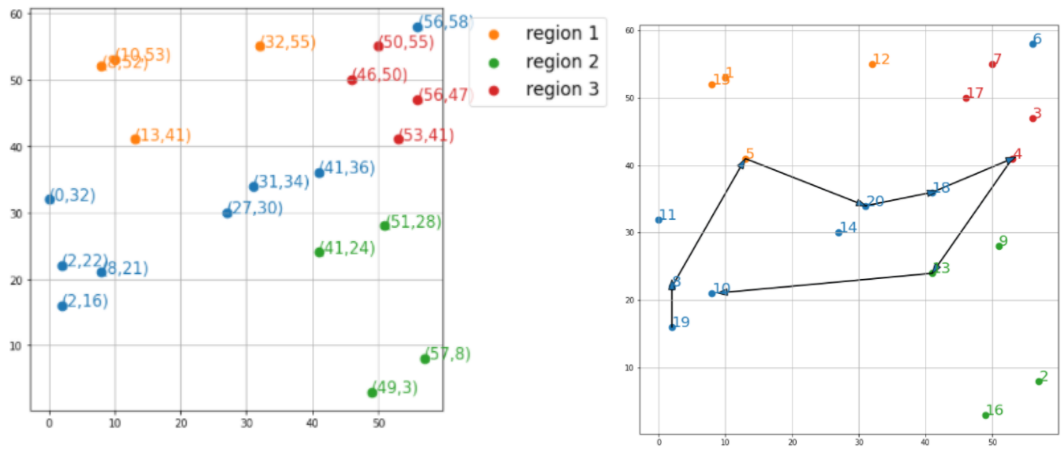
```
Explored 1 nodes (3 simplex iterations) in 0.06 seconds
Thread count was 12 (of 12 available processors)

Solution count 2: -0 -2
No other solutions better than -0

Optimal solution found (tolerance 1.00e-04)
Best objective -0.000000000000e+00, best bound -0.000000000000e+00, gap 0.0000%

User-callback calls 54, time in user-callback 0.00 sec
Objective value: 12.0
```

Figure 2.3 – Résolution avec la formulation exponentielle.

```
Explored 1 nodes (20 simplex iterations) in 0.00 seconds
Thread count was 12 (of 12 available processors)

Solution count 5: 12 13 15 ... 18

Optimal solution found (tolerance 1.00e-04)
Best objective 1.200000000000e+01, best bound 1.200000000000e+01, gap 0.0000%

User-callback calls 76, time in user-callback 0.00 sec
Objective value: 12.0
1 3
2 5
3 2
```

Figure 2.4 – Résolution avec la formulation exponentielle sans séparation.

2.0.2 Instance n=20



(a) La répartition des aérodromes.

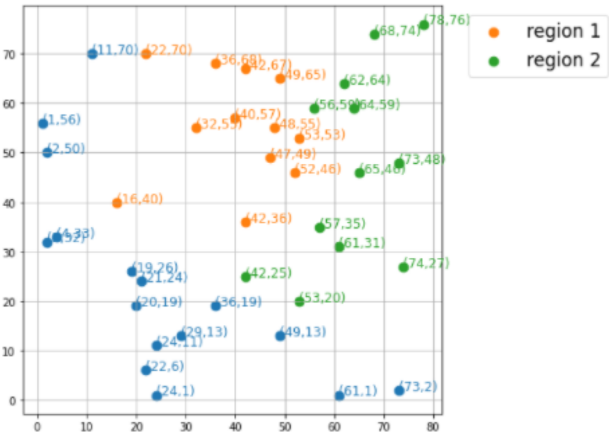
(b) La solution réalisable.

FIGURE 2.5 – Cas test avec n=20.

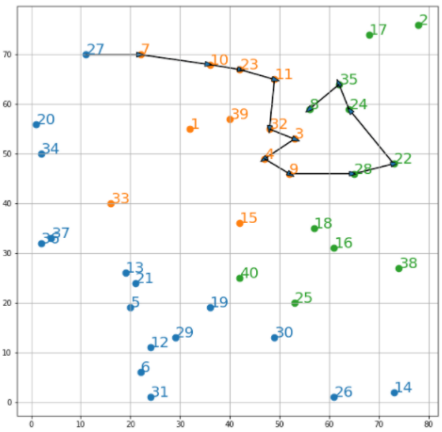
```
Explored 92864 nodes (669364 simplex iterations) in 11.02 seconds
Thread count was 12 (of 12 available processors)

Solution count 10: 122 123 124 ... 194

Optimal solution found (tolerance 1.00e-04)
Best objective 1.220000000000e+02, best bound 1.220000000000e+02, gap 0.0000%

User-callback calls 188236, time in user-callback 0.05 sec
Objective value: 122.0
```

FIGURE 2.6 – Résolution avec la formulation polynomiale

2.0.3 Instance n=40



(a) La répartition des aérodromes.



(b) La solution réalisable.

FIGURE 2.7 – Cas test avec n=40.

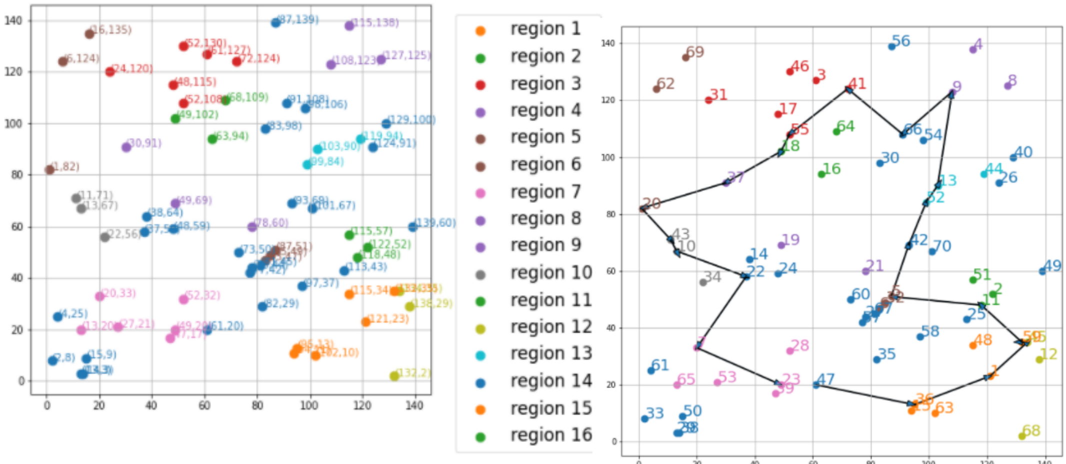
```
Explored 6314 nodes (58113 simplex iterations) in 1.42 seconds
Thread count was 12 (of 12 available processors)

Solution count 10: 112 113 116 ... 165

Optimal solution found (tolerance 1.00e-04)
Best objective 1.120000000000e+02, best bound 1.120000000000e+02, gap 0.0000%

User-callback calls 13038, time in user-callback 0.00 sec
Objective value: 112.0
```

FIGURE 2.8 – Résolution avec la formulation polynomiale

```
Explored 1 nodes (6 simplex iterations) in 0.00 seconds
Thread count was 12 (of 12 available processors)

Solution count 2: -0 -26
No other solutions better than -0

Optimal solution found (tolerance 1.00e-04)
Best objective -0.000000000000e+00, best bound -0.000000000000e+00, gap 0.0000%

User-callback calls 51, time in user-callback 0.00 sec
Objective value: 112.0
```

FIGURE 2.9 – Résolution avec la formulation exponentielle.

2.0.4 Instance n=70



(a) La répartition des aérodromes.

(b) La solution réalisable.

FIGURE 2.10 – Cas test avec n=70.

```
Explored 685128 nodes (24659163 simplex iterations) in 543.08 seconds
Thread count was 12 (of 12 available processors)

Solution count 10: 436 438 439 ... 453

Optimal solution found (tolerance 1.00e-04)
Best objective 4.360000000000e+02, best bound 4.360000000000e+02, gap 0.0000%

User-callback calls 1411490, time in user-callback 0.54 sec
Objective value: 436.0
```

FIGURE 2.11 – Résolution avec la formulation polynomiale

Conclusion

Les deux formulations polynomiale et exponentielle sont équivalents pour notre problème. Avec un nombre n assez petit on ne voit pas trop une différence du temps de calcul. En augmantant le nombre n, on s'aperçoit que la résolution avec la formulation polynomiale est plus rapide que la formulation exponentielle en nombre de contraintes.