SOD 321 - Optimisation Discrète
Projet - Course d'avions

Antoine GEY et Robin LABBÉ

November 9, 2021

1 Introduction

La Coupe Breitling 100/24 est une course aéronautique qui consiste à effectuer le maximum de posé-décollé dans différents aérodromes en 24h. Si le nombre de 100 n'est pas atteint, les pilotes doivent visiter au moins un aérodrome dans chaque région et minimiser la distance totale parcourue. Nous pouvons modéliser ce problème d'optimisation par un programme linéaire en nombres entiers similaire au problème du voyageur du commerce.

1.1 Notations

On définit les notations suivantes :

- $n \in \mathbb{N}^*$ est le nombre d'aérodromes
- $m \in \mathbb{N}^*$ est le nombre de régions à traverser
- $(x_i, y_i) \in \mathbb{R}$ les coordonnées de l'aérodrome i
- $Amin \in \mathbb{N}^*$ est le nombre d'aérodromes minimal à visiter
- $D \in \mathbb{R}^+$ est la distance que peut parcourir un avion sans se poser
- $r_i \in \mathbb{N}$ est la région dans laquelle se trouve l'aéroport i
- $d_{ij} = \left\lfloor \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right\rfloor \in \mathbb{N}$ la distance euclidienne (arrondie à l'entier inférieur) entre les aérodromes i et j
- i_0 désigne le numéro de l'aérodrome de départ
- i_f désigne le numéro de l'aérodrome d'arrivée
- $X = (x_{ij}) \in \mathcal{M}_n(\{0,1\})$ où $x_{ij} = 1$ si le pilote se rend de l'aérodrome i à l'aérodrome j durant son trajet, 0 sinon

2 Définition du problème

Le problème est similaire à celui du voyageur du commerce orienté. Nous choisissons donc de modéliser notre problème par un graphe orienté dans lequel deux aérodromes sont reliés s'il est possible pour l'avion de parcourir la distance qui les sépare.
On définit le graphe $G = (V, A)$ où $V = \{1, \dots, n\}$ représente l'ensemble des aérodromes, et $A = \left\{(i,j) \in V^2, i \neq j : d_{ij} \leq R\right\}$, les arêtes entre les aérodrômes.

2.1 Fonction à minimiser

La fonction à minimiser est donc la somme des distances parcourues lors du trajet :

$$Z = \sum_{(i,j) \in A} x_{ij} d_{ij}$$

2.2 Contraintes

Les contraintes sont les suivantes :

- On ne décolle/atterrit, qu'au plus une fois dans les aéroports sauf pour les aéroports de départ, respectivement d'arrivé où l'on décolle, respectivement atterit, exactement 1 fois
- On doit partir du départ i_0 et arriver à l'arrivé i_f
- La distance entre les aéroports visités ne doit pas être supérieure à la distance que peut parcourir l'avion.
- Le pilote doit visiter au moins une fois chaque région
- En dehors des aérodromes d'arrivée et de départ, l'avion atterrit et décolle
- Le nombre d'aéroports visités doit être supérieur ou égal à A_{min}
- $x_{ij} \in \{0, 1\}$

2.3 Formulation avec contraintes des sous-tours

La fonction à minimiser est la distance totale parcourue par l'avion :

$$\min \sum_{(i,j)\in A} d_{ij} x_{ij}$$

On n'atterit au plus une fois dans chaque aéroport :

$$\sum_{j:(i,j)\in A} x_{ij} \leq 1, \forall\, i \in V$$

On ne décolle qu'au plus une fois depuis aéroport :

$$\sum_{j:(j,i)\in A} x_{ji} \leq 1, \forall\, i \in V$$

Contraintes sur le départ :

$$\sum_{j:(i_0,j)\in A} x_{i_0 j} = 1$$

Contraintes sur l'arrivée :

$$\sum_{j:(j,i_f)\in A} x_{j i_f} = 1$$

Contrainte sur la distance maximale que peut parcourir l'avion :

$$\forall (i,j),\ d_{ij} x_{ij} \leq D$$

On décolle et atterrit des aéroports en dehors du départ et de l'arrivé :

$$\sum_{j:(i,j)\in A} x_{ij} - x_{ji} = 0,\ \forall\, (i,j) \neq (i_0, i_f)$$

Toutes les régions sont visitées au moins une fois :

$$\sum_{i\in r_k} x_{ij} + \sum_{j\in r_k} x_{ij} \geq 1, \forall\, r_k$$

Nombre minimum d'aéroports à visiter, $\mathbb{1}$ étant la fonction indicatrice :

$$\sum_{i,j:(i,j)\in E} x_{ij} \geq A_{\min} - 1 + \mathbb{1}_{\{i_0 = i_f\}}$$

x_{ij} binaires :

$$x_{ij} \in \{0, 1\}$$

On a ensuite deux possibilités pour empêcher les cycles :

- Formulation sous-tours, nombre exponentiel de contraintes :

$$\sum_{(i,j)\in A, i\in S, j\in S} x_{ij} \leq |S| - 1,\ \forall S \subset V,\ 2 \leq |S| \leq |V| - 2$$

- Formulation polynomiale :

$$\forall i \neq i_0, j \neq i_f : (i,j) \in V, u_j \geq u_i + 1 - n\left(1 - x_{ij}\right),\ u_i \in \mathbb{Z}$$

  Avec u_j le nombre d'aéroports visités pour arrivée en j depuis i_0.

2.4 Méthodes de séparation

Etant donné un point $\tilde{x}, \tilde{y}$, touver un ensemble S tel que

$$\sum_{i\in\mathcal{S}, j\in\mathcal{S}, (i,j)\in Arcs} \tilde{x}_{ij} > |\mathcal{S}| - 1$$

En notant a_i la variable binaire = 1 ssi $i \in S$, on cherche donc a solution optimale du problème suivant:

$$SEP \begin{cases} \max \Delta = \sum_{i=1}^{N}\sum_{j=1}^{N} \tilde{x}_{ij} * a_i * a_j - \sum_{i=1}^{N} a_i + 1 \\ \text{s.t.} \\ \quad \sum_{i=1}^{N} a_i \geq 1 \\ \quad a_i \in \{0,1\} \end{cases}$$

Les contraintes (1), peuvent être améliorées par les inégalités de sous-tours généralisées suivantes: (x étant une variable de selection d'arc, et y de selection de sommets)

$$\sum_{i\in\mathcal{S}, j\in\mathcal{S}} x_{ij} \leq \sum_{i\in\mathcal{S}} y_i - y_{i_0} \quad \mathcal{S} \subset \{1,\ldots,N\}, i_0 \in \{1,\ldots,N\} : i_0 \in \mathcal{S}$$

Leur signification est la suivante : étant donnés un ensemble S et un sommet i_0 de S. Si le parcours passe par i_0 ($y_{i_0} = 1$) alors le nombre d'arcs à l'intérieur de S ne dépasse pas le nombre de sommets de S parcourus -1. Sinon ($y_{i_0} = 0$) alors l'inégalité est vérifiée aussi mais elle est redondante. Séparation des inégalités (2) Etant donné un point $\tilde{x}, \tilde{y}$, touver un ensemble S et un élément i_0 de S tel que l'inégalité est violée. En notant a_i la variable binaire = 1 ssi $i \in S$ et une nouvelle variable binaire $h_i = 1$ ssi i est l'unique sommet qui joue le rôle de i_0, on cherche donc (a, h) solution optimale du problème suivant :

$$SEP_{gen} \begin{cases} \max \Delta_g = \sum_{i=1}^{N}\sum_{j=1}^{N} \tilde{x}_{ij} * a_i * a_j - \sum_{i=1}^{N} \tilde{y}_i a_i + \sum_{i=1}^{N} \tilde{y}_i h_i \\ \text{s.t.} \\ h_i \leq a_i \quad i = 1,\ldots,N \\ \sum_{i=1}^{N} h_i = 1 \\ a_i, h_i \in \{0,1\} \end{cases}$$

2.4.1 Principe de résolution

Le principe de résolution avec la séparation de base ou séparation généralisée est le suivant :

- 1. On calcul l'optimal pour le problème principal
- 2. On calcule le maximum du problème de séparation
- 3. Tant que le maximum du problème de séparation est positif, on ajoute une contrainte au problème maître et l'on retourne en 1.

Il est intéressant de remarquer que l'on peut résoudre la séparation uniquement à partir de la résolution du problème principal en nombre entier, mais aussi, on peut résoudre le problème de séparation à partir de la relaxation continue du problème principal. Effectivement, dans les deux cas on ajoute des inégalités valides pour la résolution.

Il est intéressant d'utiliser la relaxation continue du problème maître pour résoudre le problème de séparation et l'ajout de contraintes. En effet, numériquement, il est beaucoup plus rapide de résoudre la relaxation continue que le problème maître en variable binaires. L'ajout de contraintes à partir de la relaxation continue finit par nous fournir l'optimal de la relaxation continue. A partir de là on continue la résolution avec le problème de séparation en repassant en nombres entiers pour le problème principal. En conclusion, cette procédure utilisant la relaxation continue est efficace numériquement car on ajoute rapidement beaucoup de contraintes, ce qui aide par la suite à résoudre le problème principal en nombre entiers.

3 Résolution sur Julia

On modélise le problèmes sous les deux formes possibles (contraintes polynomiales et exponentielles) sur Julia. On utilise un script python pour visualiser les trajets optimaux après export de la matrice $X = (x_{ij})$ depuis Julia. On a le résultat suivant pour l'instance avec 40 aéroports :

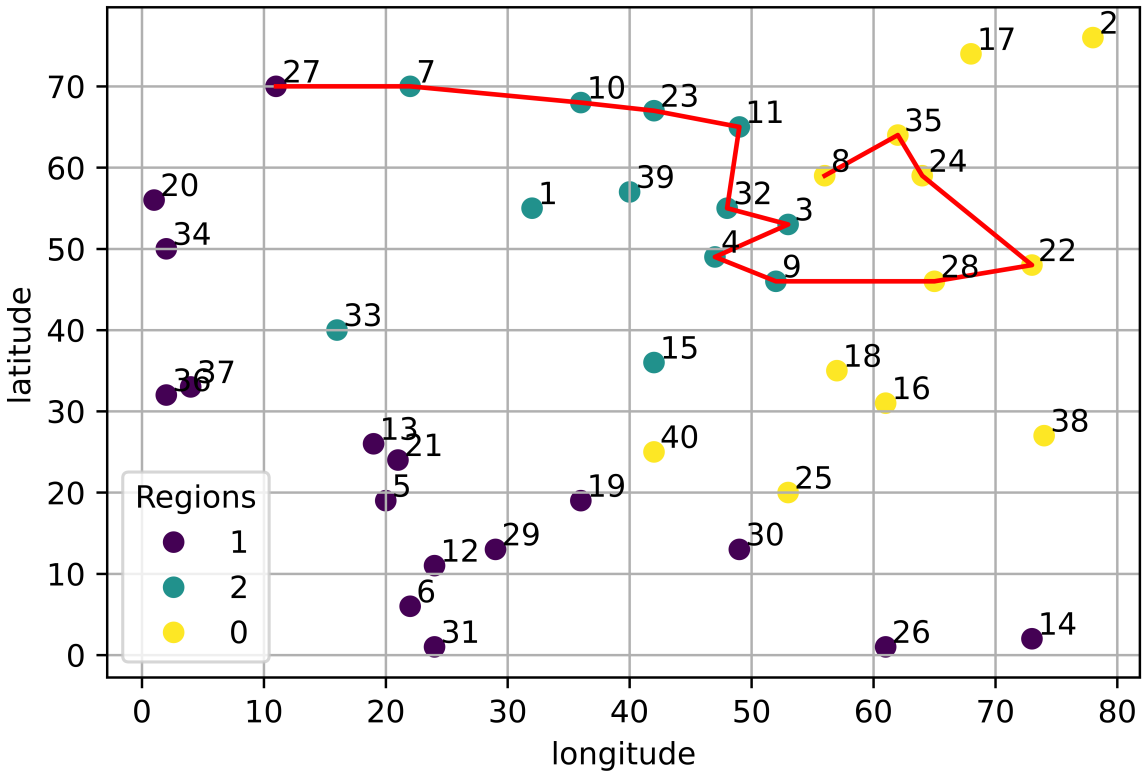

Figure 1: Trajet instance 40

Le trajet passe bien par toutes les régions et aucune partie du trajet ne semble être aberrante du point de vue de l'optimalité (c'est rassurant). Les autres tracés de chemins optimaux sont disponibles en annexes.

Au niveau des temps de calcul on résume nos résultats dans le tableau suivant, on a mis des points d'interrogation là où les calcul étaient très long et pour lesquels on n'a pas abouti :

	instance 6	instance 20	instance 40	instance 70
polynomial	0.006 s	5.73	2.43 s	1859 s
exponentiel SEP	0.0336 s	?	23.35 s	?
exponentiel SEP_{gen}	0.0336 s	16.38 s	2.82 s	?
SEP_{gen} + polynomial	0.03	0.97 s	2.17 s	75.3 s
v_{opt}	12	122	112	436

Discussion sur les performances des modèles

La résolution polynomiale nous permet d'obtenir des résultats en des temps acceptables, même si on doit attendre 30min pour la résolution avec 70 instances. Le nombre de contraintes s'acccroissant aussi de façon polynomiale, le temps de résolution pour 70 et pour des instances de taille plus grande exploserait. Il ne serait pas raisonnable de continuer à utiliser la méthode telle quelle.

Lorsque nous avons commencé à implémenter la méthode exponentielle sans utiliser la relaxation continue du problème maître, la résolution était encore plus longue et fastidieuse, nous avons donc du implémenter la résolution avec la relaxation continue pour obtenir des temps de calcul acceptables.

La méthode exponentielle avec la SEP de base est assez inefficace, même en utilisant la relaxation continue. Pour l'instance 20, assez compliquée et l'instance 70, la méthode avec SEP de base n'aboutit pas en temps raisonnable. Nous avons stoppé la résolution après 30min d'attente.

La méthode exponentielle avec la SEP_{gen} est bien plus efficace que la méthode avec la SEP de base. jusqu'à une taille d'instance 40, la résolution est très rapide avec des temps comparables à la résolution polynomiale, sauf pour instance 20, on note quand même que 16 secondes d'attente reste une belle performance car la méthode de séparation de base n'aboutit pas. Pour l'instance 70 la méthode avec SEP_{gen} n'aboutit pas en temps raisonnable. Nous avons stoppé la résolution après 30min d'attente.

N'obtenant pas de solution en temps satisfaisant pour l'instance 70, nous avons utilisé une dernière méthode "SEP_{gen}+polynomial". Dans un premier temps nous avons choisi que la méthode SEP_{gen} + polynomial consiste à ajouter des contraintes pendant un certains temps (180 s) puis à résoudre le problème de manière polynomiale. Le problème de séparation généralisé est résolu avec la solution de la relaxation continue du problème maître. Si on ne fixe pas un seuil de durée, l'ajout de contraintes prend énormément de temps dans le cas de l'instance 70. La résolution avec cette méthode est très rapide pour les différentes instances, nous avons mis 229s pour résoudre l'instance 70. Il est intéressant de constater que pendant les 180s nous avons passé beaucoup de temps sans améliorer l'optimal de la relaxation du problème maître.
Cette méthode est licite car les inégalités ajoutées via SEP sont valides pour la formulation polynomiale. En théorie on pourrait d'abord ajouter toutes les contraintes de SEP puis résoudre le problème de manière polynomiale (c'est d'ailleurs ce qui est fait pour les instances 6, 20 et 40 même avec le seuil), en pratique cela prend trop de temps pour des problèmes de grand taille (70).
Cependant, mettre un seuil n'aurait pas de sens pour le cas d'une très grande instance où pendant plus de 180s nous pouvons rajouter très rapidement des contraintes qui améliorent la relaxation continue. On pourrait se retrouver dans un cas, où l'on "n'a pas ajouté assez de contraintes" avec la méthode de séparation pour accélérer la résolution du problème principal par la méthode polynomiale.

Ainsi nous avons choisi un 2e critère d'arret. La méthode "SEP_{gen}+polynomial" est donc définie comme ceci :
Dans un premier temps la méthode consiste à ajouter des contraintes jusqu'à ce que l'optimal de la relaxation continue ne soit pas amélioré pendant 10 itérations (ou que l'optimum du problème de séparation soit négatif), puis à résoudre le problème principal en variable binaire avec la méthode polynomiale.
L'avantage de cette méthode est qu'elle est très rapide quelque soit la taille de l'instance. Combiner les deux méthodes divise par 25 le temps de résolution comparé à la méthode polynomiale. La condition d'arret est aussi plus naturelle, car si l'on passe trop de temps à ajouter des contraintes au problème principal, qui n'améliorent pas la relaxation continue, nous perdons du temps pour la résolution. Effectivement, avec cette méthode, on ajoute très rapidement énormément de contraintes qui vont servir à réduire le polyèdre de résolution pour la méthode polynomiale. Autre avantage, on ne résout qu'une fois le problème principal en variable binaire.

4 Conclusion

Nous avons à travers ce projet découvert comment utiliser le langage de programmation Julia conjointement au solveur Gurobi. La combinaison de julia et gurobi rend claire et efficace l'implémentation de problèmes d'optimisations.
Ce problème similaire à celui du voyageur du commerce est NP-difficile. Sa résolution est complexe car dès que la taille des instances augmentent un peu, le temps de résolution devient gigantesque. Nous avons donc du chercher à optimiser la résolution (notamment en méthode exponentielle) en améliorant la qualité du problème de séparation, puis en utilisant la relaxation continue pour accélérer l'ajout de contraintes au problème principal, enfin nous avons combiné la résolution avec la méthode exponentielle d'abord pour ajouter des controntraintes et réduire le polyedre de résolution et utiliser la méthode polynomiale pour terminer la résolution. Cette derniere méthode nous fournit des résultats très corrects même pour des grandes instances, chose qui n'était pas permise en utilisant uniquement la méthode polynomiale ou la méthode exponentielle.

5 Annexes

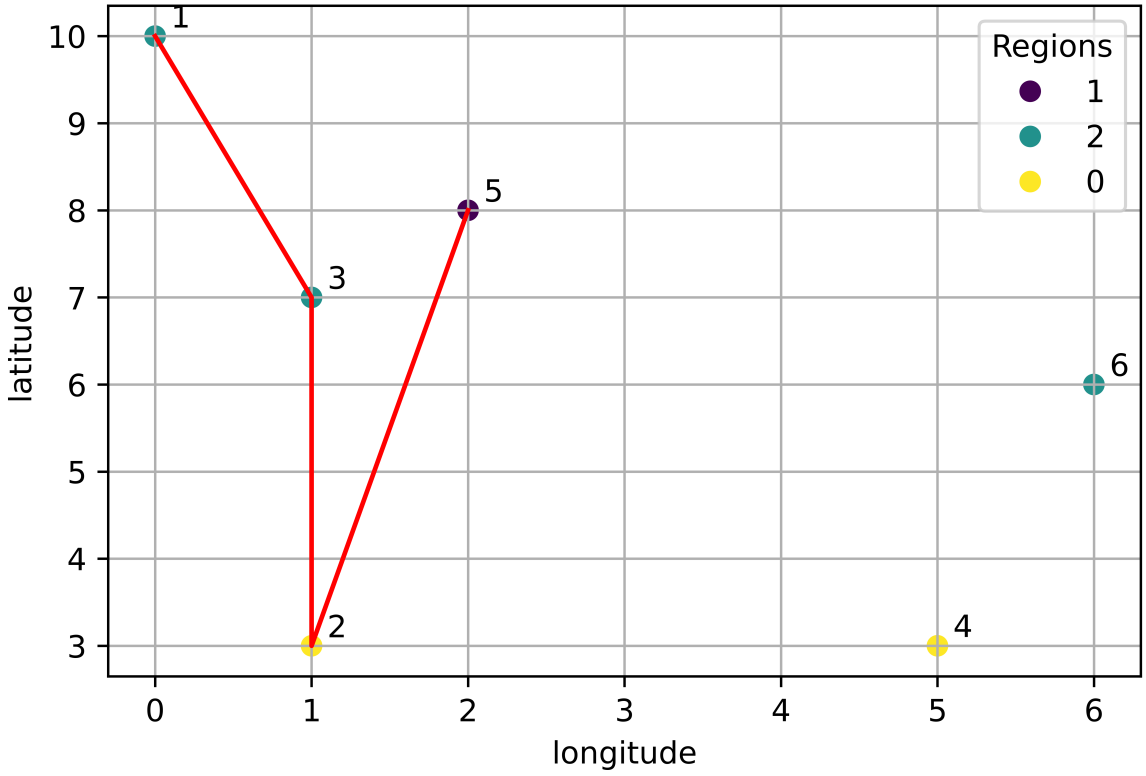

Figure 2: Trajet instance 6

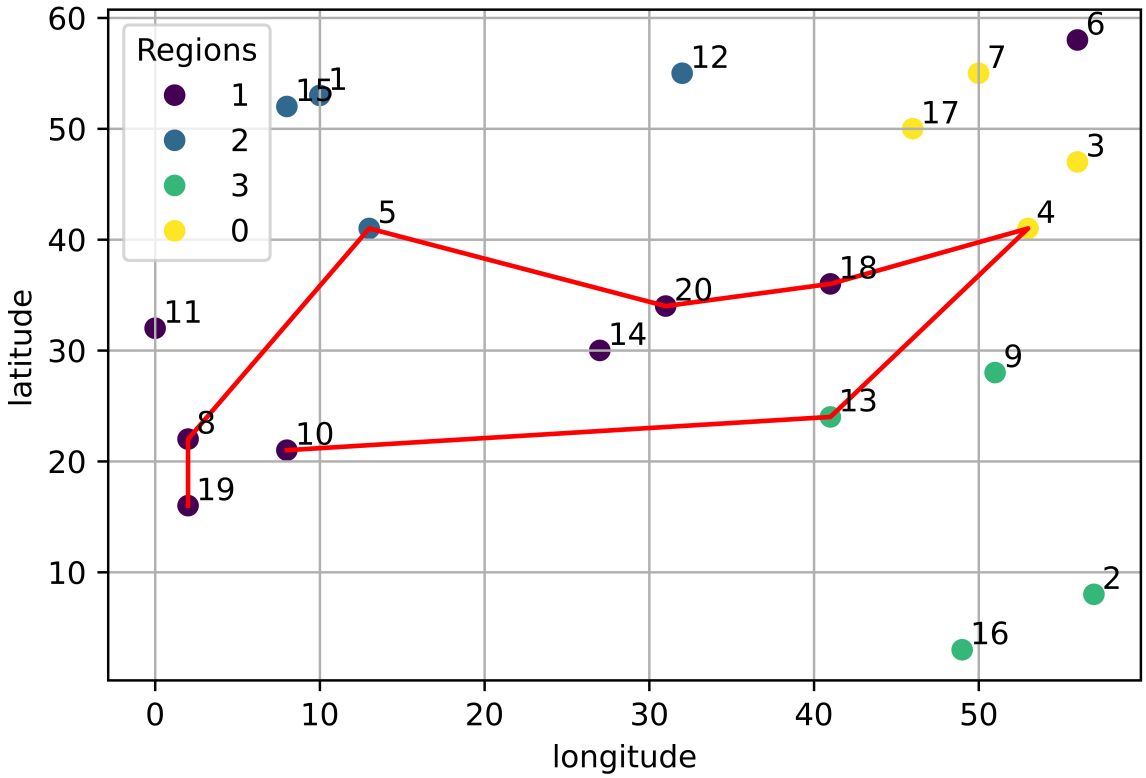

Figure 3: Trajet instance 20

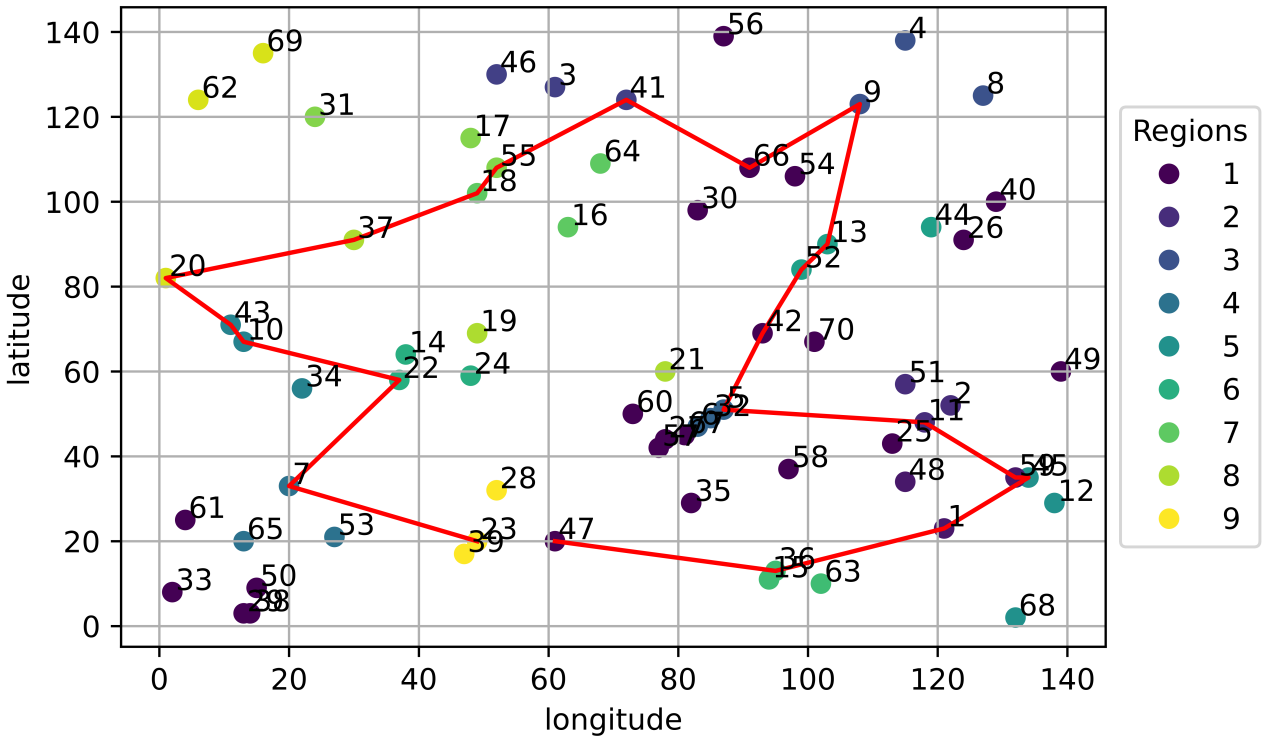

Figure 4: Trajet instance 70