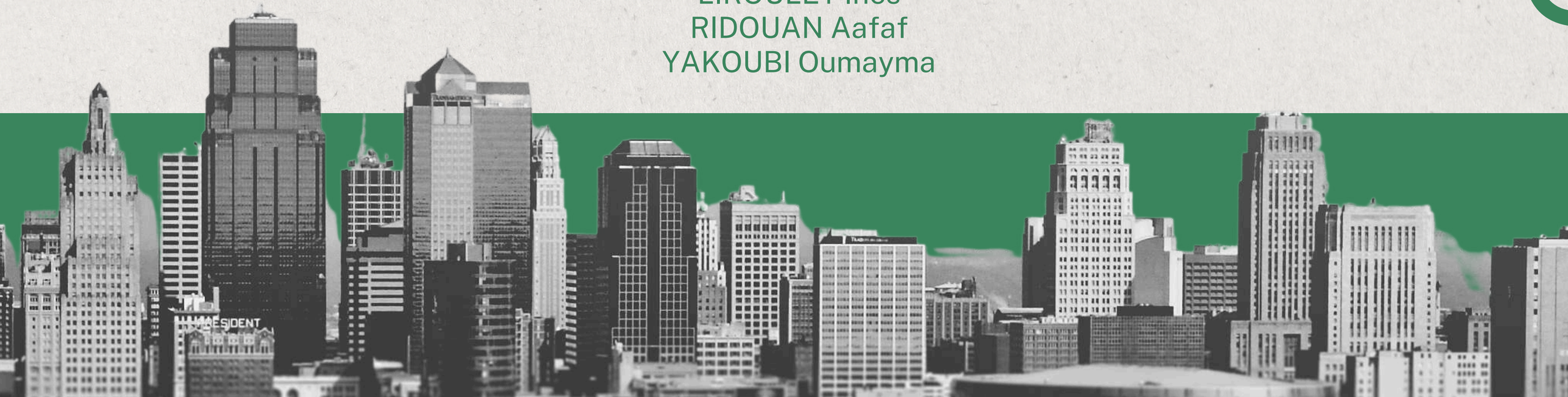
Master 2 Intelligence Artificielle

Problème du voyageur de commerce *(avec fenêtres de temps)*

Combinatoire, Complexité et Graphes

Prof. Samba Ndojh NDIAYE

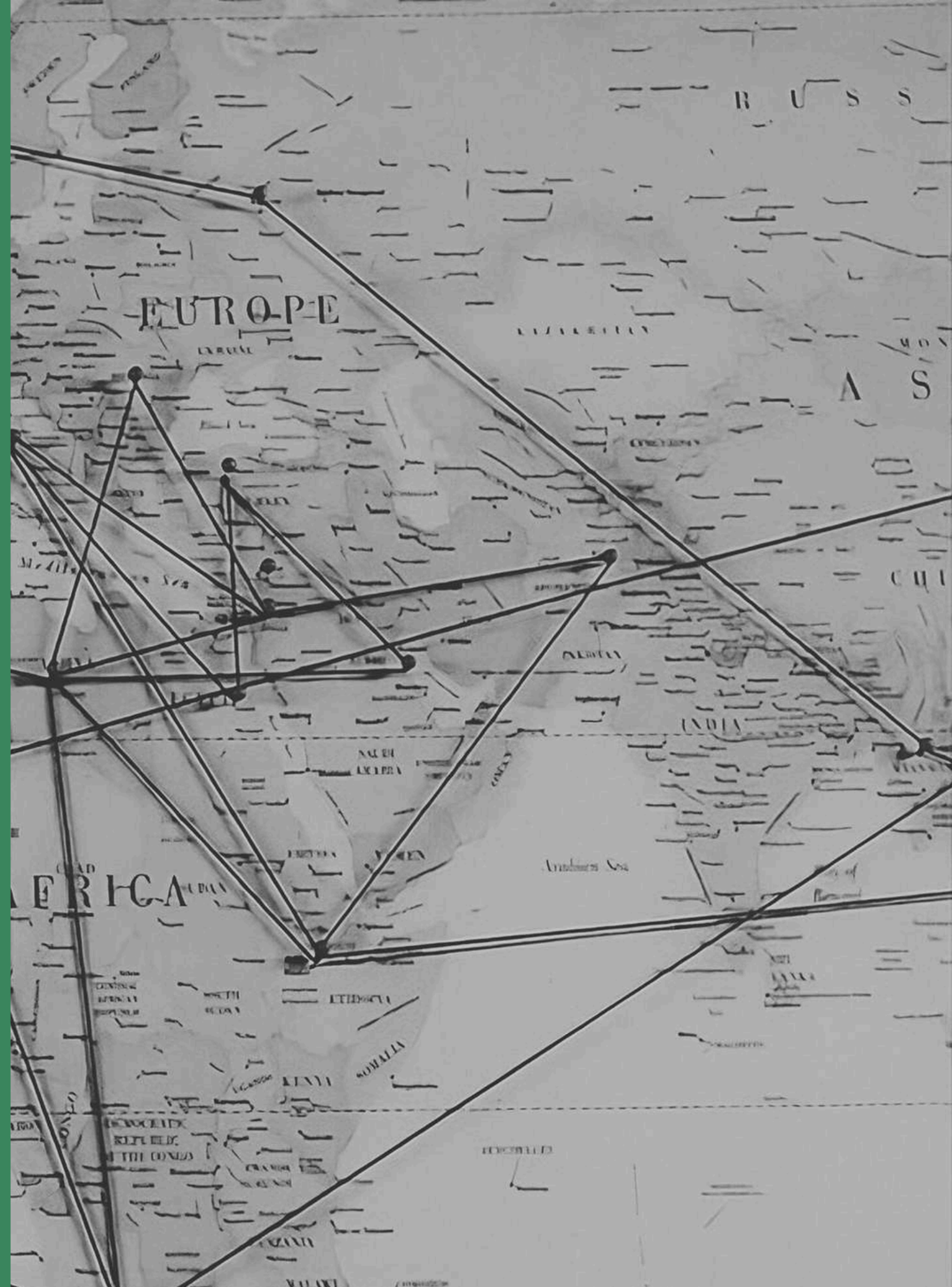
LIROULET Inès
RIDOUAN Aafaf
YAKOUBI Oumayma



Plan

- Description du problème
- Modélisation du problème
- Benchmark utilisé
- OR-tools : solution complète
- Solutions incomplètes
- Comparaison

TSP : Description du problème



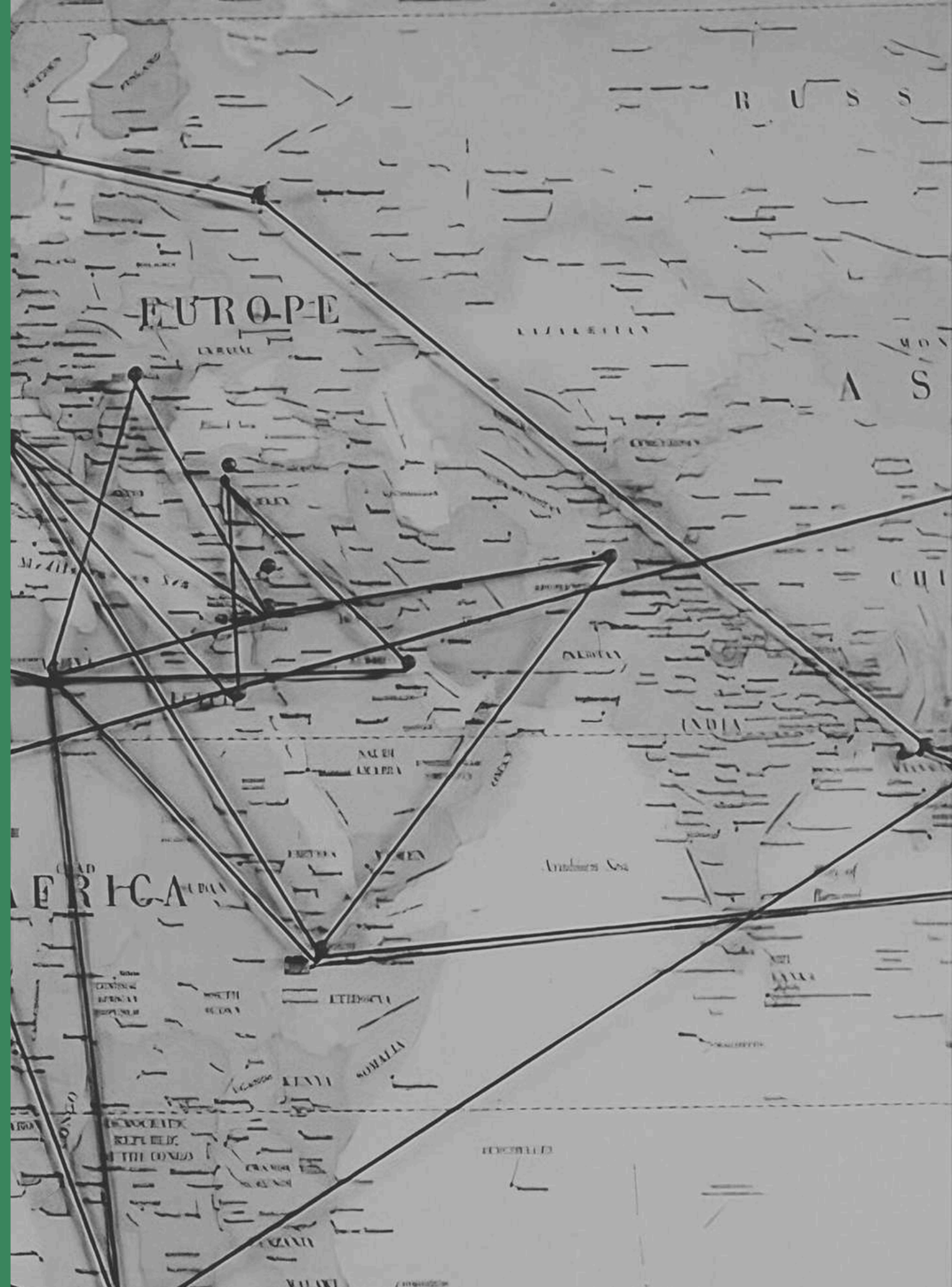
Problème du voyageur de commerce

*“Un voyageur de commerce doit visiter une et **une seule fois** un nombre fini de villes et revenir à son point d’origine. Trouvez l’ordre de visite des villes qui **minimise** la **distance totale parcourue** par le voyageur”*

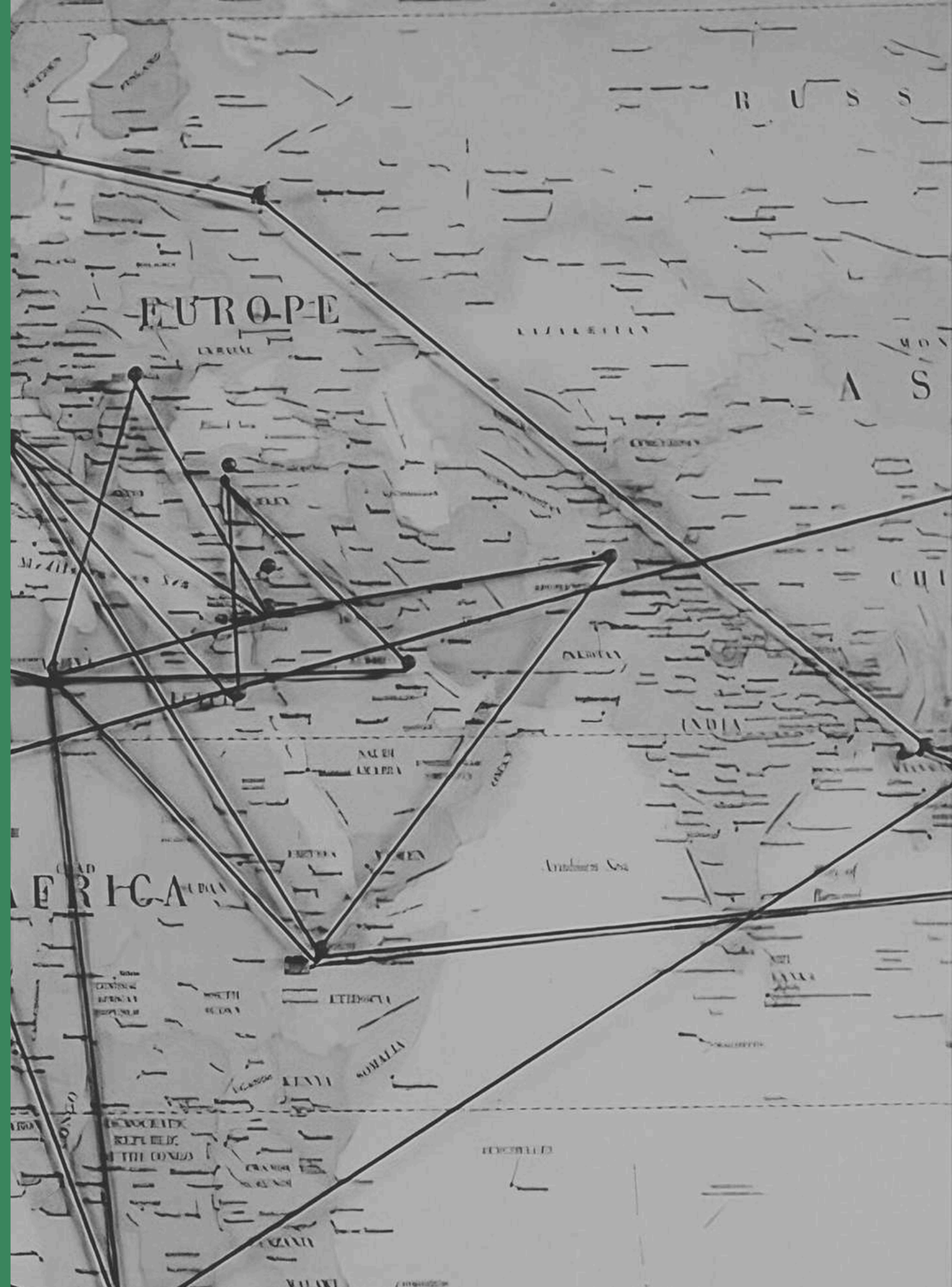
- Un problème de la classe **NP-complet**
- L’optimisation de la solution est **NP-Difficile**

Pour **n** villes dans le problème, le nombre de parcours possibles est égal à : **(n-1)!/2**

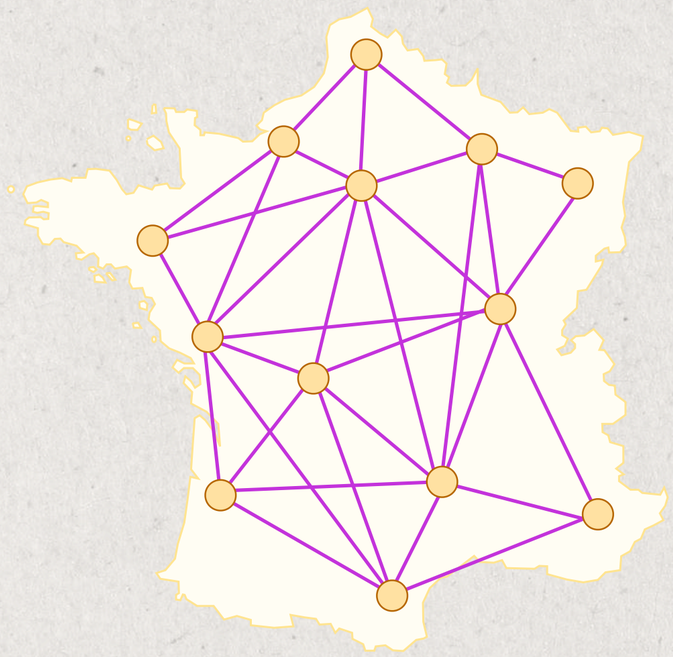
- **5** villes : **12** parcours possibles
- **10** villes : **181 440** parcours possibles
- **20** villes : **60 × 10^15** parcours



Modélisation du problème



Modélisation du problème

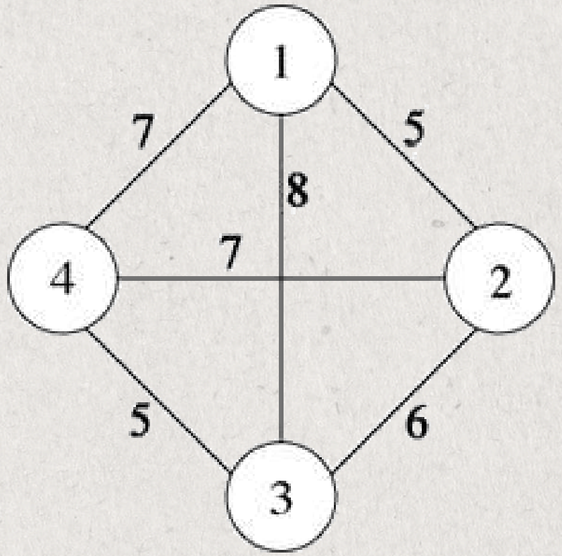


- Graphe où les sommets représentent les villes et les arêtes les passages entre les villes
- Un **poids** est associé aux arêtes pour représenter la **distance** entre les villes
- Les arrêtes sont **symétriques** : même distance en allant de A et B qu'en allant de B vers A
- Le graphe est **non-orienté** : on considère que l'on peut passer d'une ville à l'autre dans le sens que l'on veut (→ les arêtes n'ont pas de direction)

But : trouver un **cycle** dans le graphe passant par tous les sommets une fois exactement (**cycle hamiltonien**) avec un **coût minimal**

Source : https://en.wikipedia.org/wiki/Travelling_salesman_problem

Modélisation du problème



0	5	8	7
5	0	6	7
8	6	0	5
7	7	5	0

Source : https://interstices.info/le-probleme-du-voyageur-de-commerce/

Modélisation du problème

Contraintes du problème

Contrainte 1 :
Ne pas passer par une ville plus qu'une seule fois.

Contrainte 2 :
Revenir au point de départ.

Contrainte 3 :
Fenêtres de temps : chaque ville doit être visitée dans un intervalle de temps défini.

Modélisation du problème

Variables

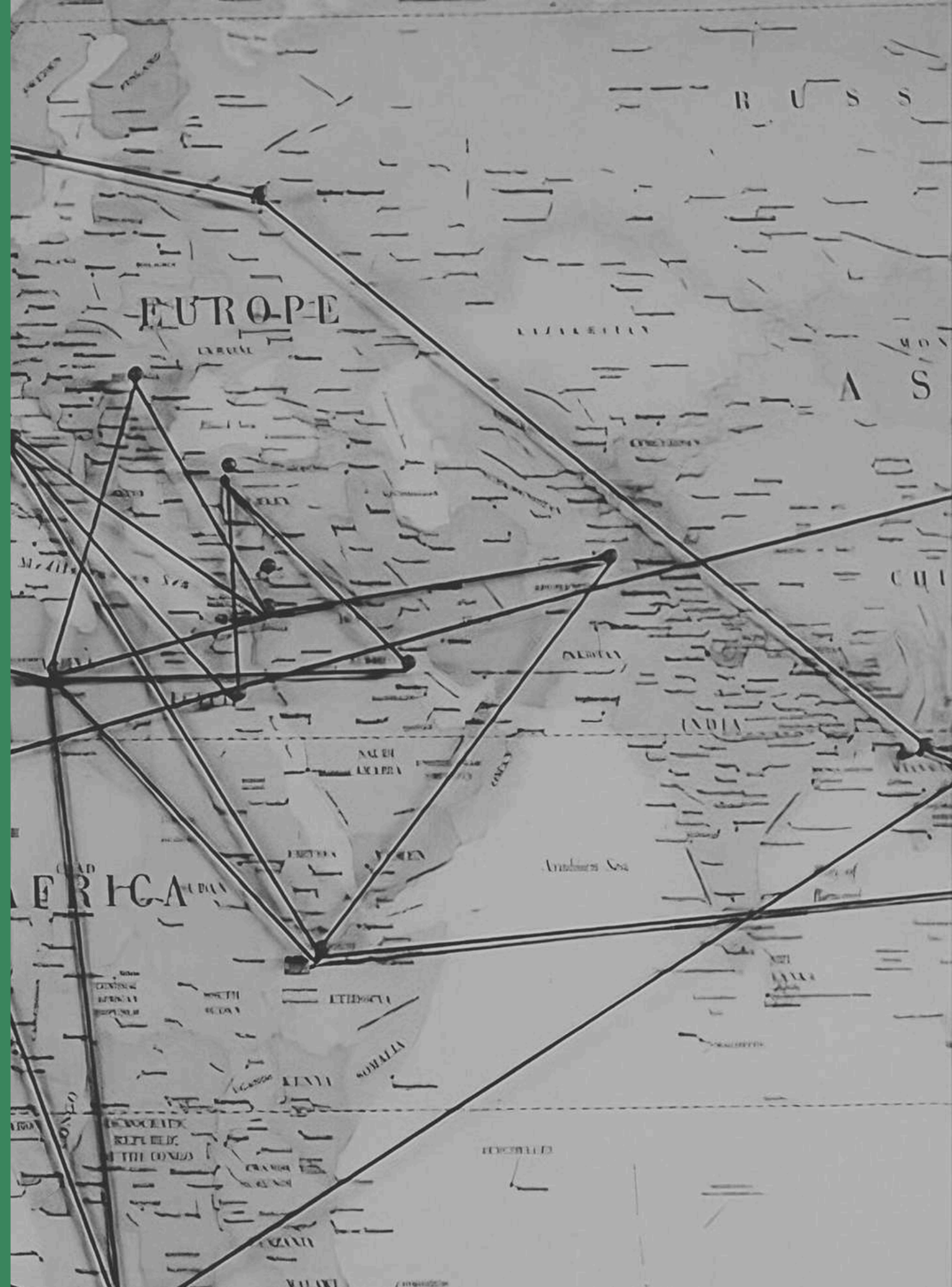
Nombre de villes à visiter (N) : Le nombre total de villes que le voyageur doit visiter.

Matrice des distances : Une matrice symétrique de dimension NxN, où chaque élement représente la distance entre deux villes.

Matrice des temps de trajet: En supposant une vitesse de déplacement constante de 80km/h, cette matrice NxN indique le temps nécessaire pour parcourir la distance entre chaque paire de villes.

Matrice des fenêtres temporelles : Une matrice de dimensions Nx2, où chaque ligne représente une ville et les deux colonnes indiquent respectivement l'heure d'ouverture et l'heure de fermeture pendant lesquelles la ville peut être visitée.

Benchmarks



Benchmarks

Valeurs possibles des variables

Nombre de villes à visiter (N)

Small

4 villes
2 x 10 villes
2x 20 villes
2x 25 villes

Medium

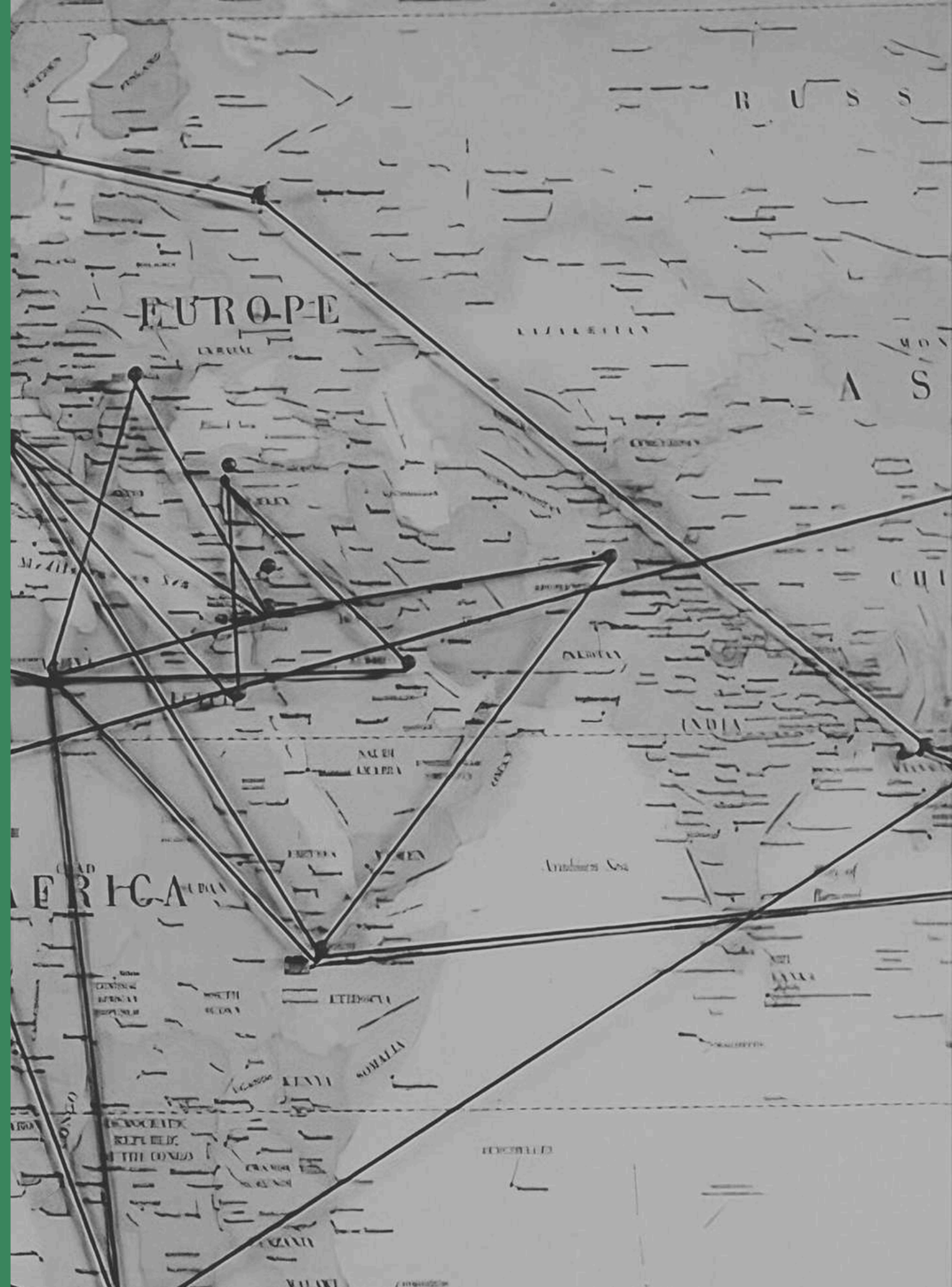
30 villes
41 villes
61 Villes

Large

81 villes
101 villes

→ 13 matrices différentes en tout

Méthode complète



Méthode complète

1. Modèle de données contenant :

- La matrice de distances (NxN).
- Les fenêtres temporelles.
- Nombre de véhicules (problème de routage du véhicule, 1 pour le TSP).
- Dépôt (ville de départ et arrivée).

2. Définition de la Fonction Coût

- Minimiser la distance totale.
- La distance entre deux villes est convertie en temps de trajet (distance / vitesse).

3. Dimensions et Contraintes

- Ajout d'une dimension temporelle pour gérer les contraintes de temps avec un waiting time (0, 30, 60, 120).
- Application des fenêtres temporelles pour chaque ville.

4. Résolution et Extraction de la Solution

(trajet optimal) avec un coût total minimal en respectant les fenêtres temporelles.

Google OR-Tools

Méthode complète

Résultats

Catégorie	Nombre de villes	Temps d'exécution (ms)	Waiting time(min)	Distance totale(km)
small	4	9.104490280151367	0	74.0
small	10	9.497642517089844	120	174.0
small	20	6272.384166717529	120	344.0
medium	40	28.17344665527344	0	484.0
medium	60	51.24545097351074	30	230.0
large	80	133.0280303955078	60	241.0
large	100	136.08479499816897	0	677.0

Méthode complète
Résultats

0 Time(0,0) -> 3 Time(25,25) -> 1 Time(55,55) -> 2 Time(62,62) -> 0 Time(74,74)

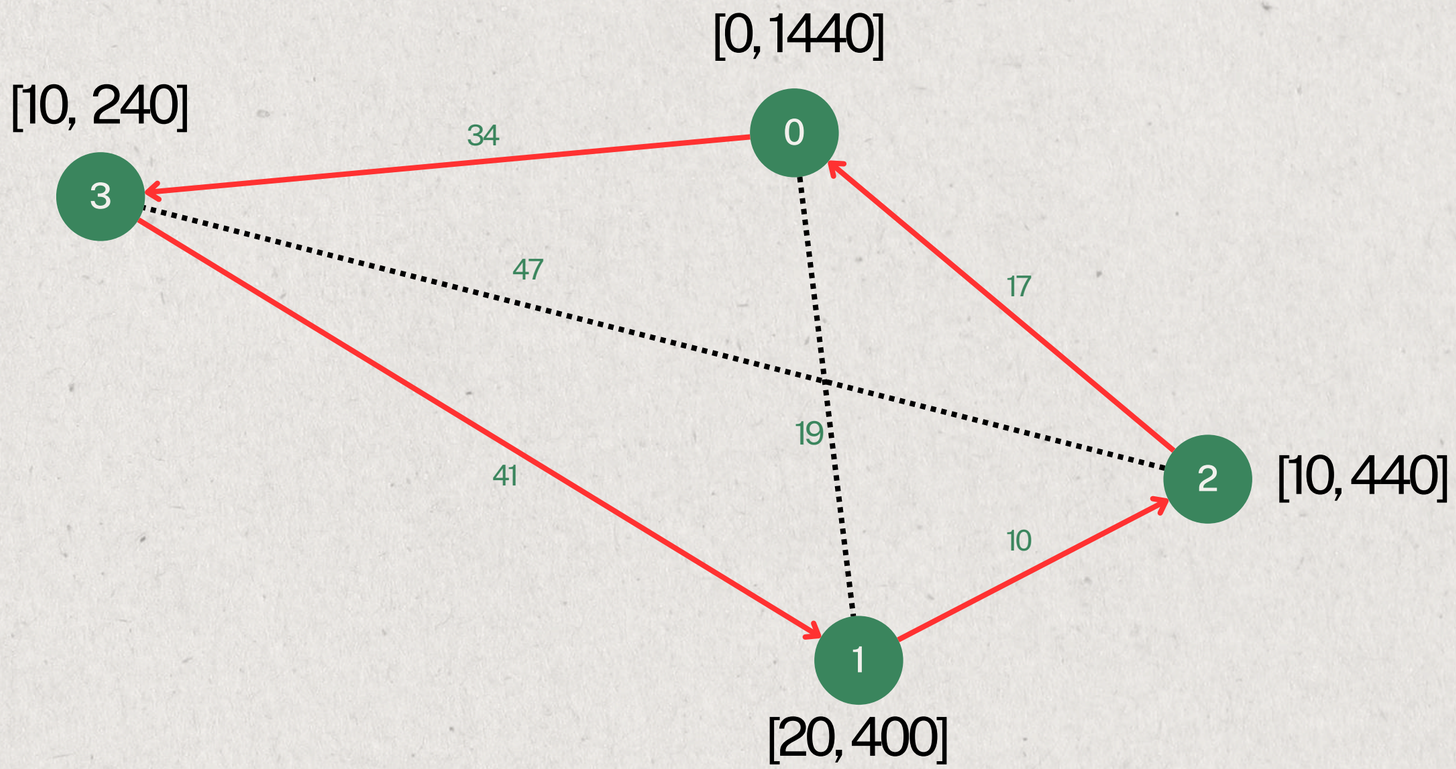
Matrice de distances

0 19 17 34
19 0 10 41
17 10 0 47
34 41 47 0

Matrice des fenêtres de temps

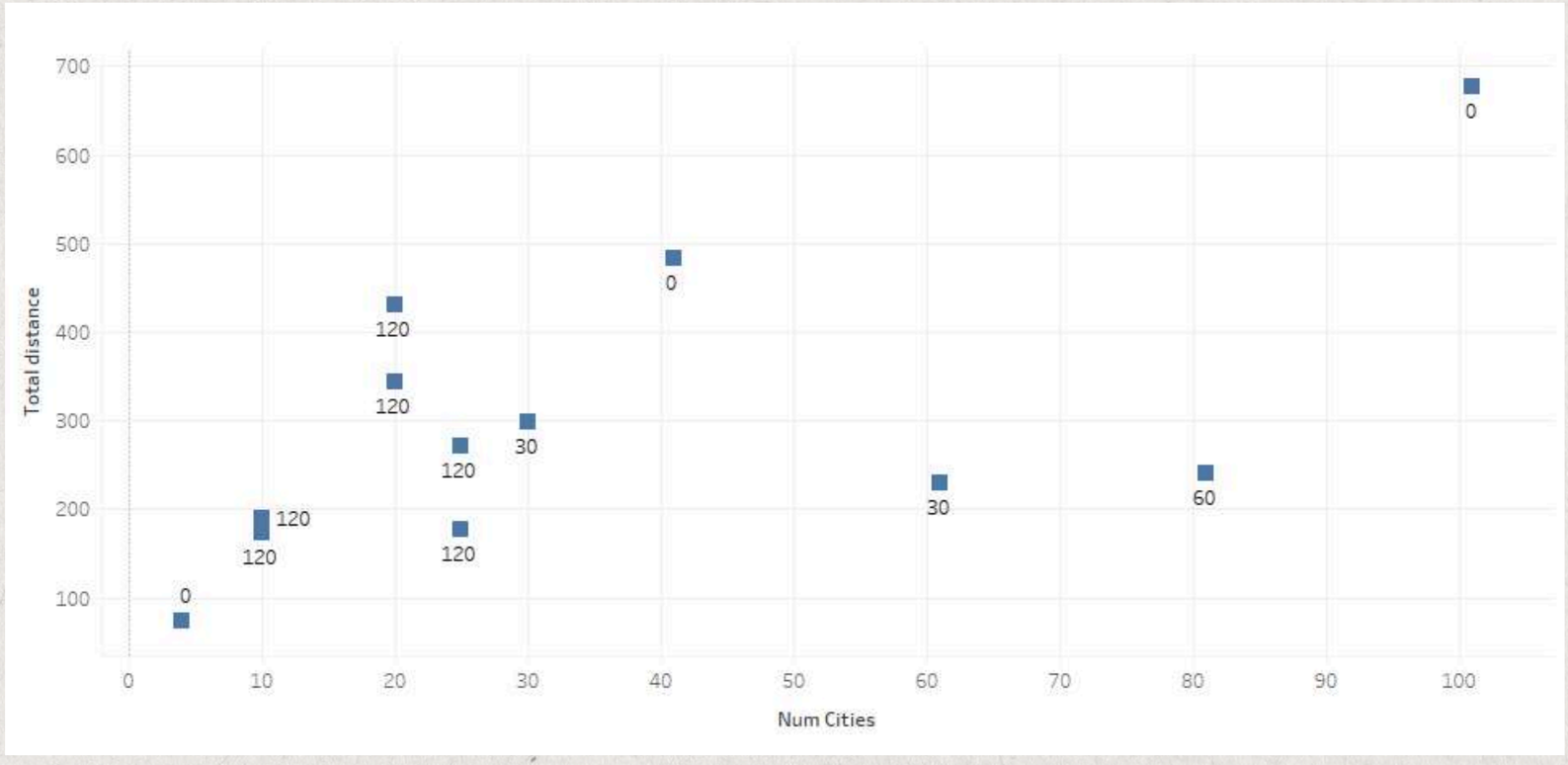
0 1440
20 400
10 440
10 240

Temps d'attente = 0



Méthode complète
Résultats

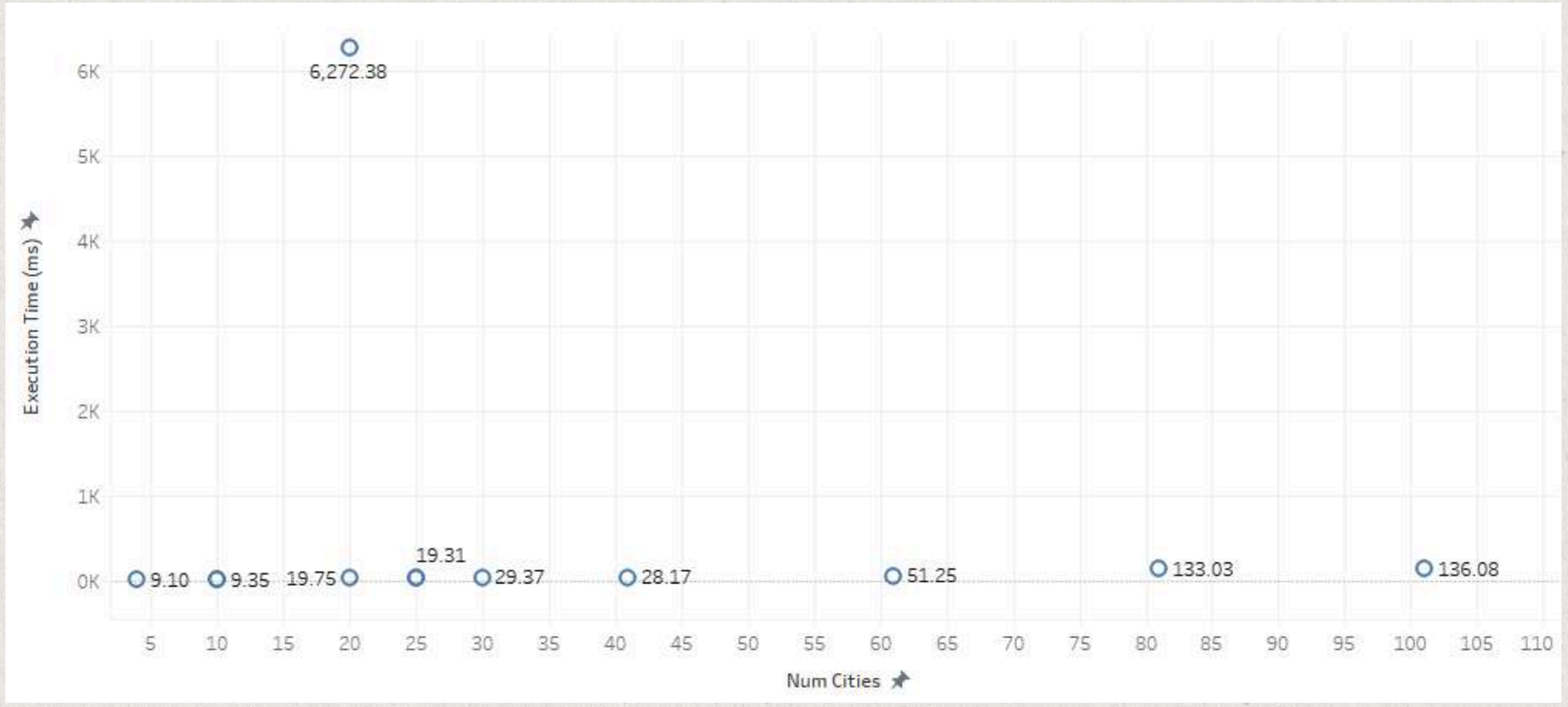
Nous avons effectué des tests avec différentes valeurs du paramètre 'temps d'attente maximum' et retenu celle qui optimise notre fonction objective (en minimisant la distance parcourue).



Relation entre le Nombre de Villes et la Distance Totale avec Temps d'Attente

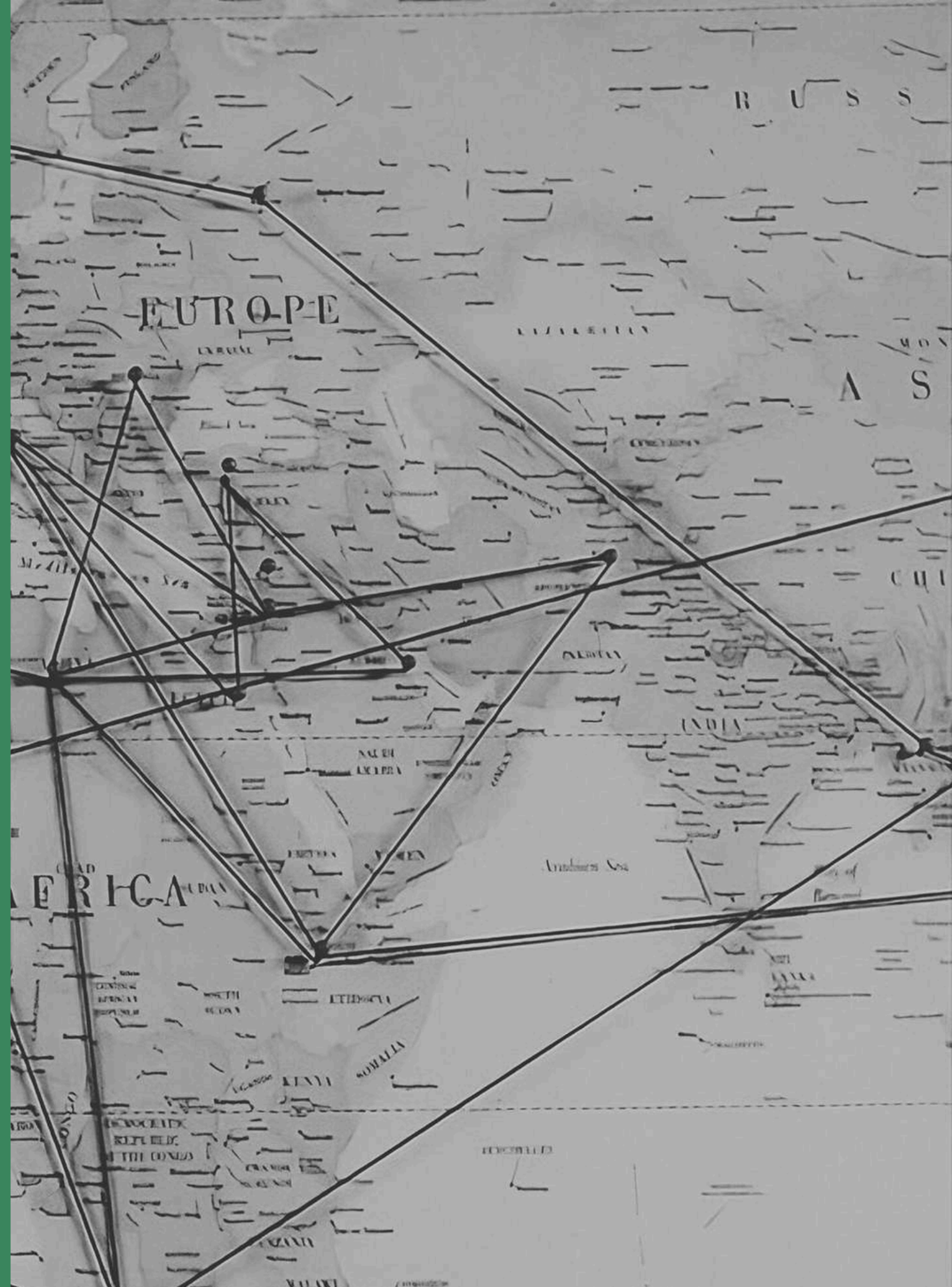
Méthode complète
Résultats

Le temps d'exécution augmente considérablement avec l'augmentation du nombre de villes. L'extrême valeur observée pour un nombre de villes égal à 20 s'explique par des fenêtres temporelles très strictes, ce qui contraint le programme à chercher davantage de possibilités pour trouver la solution optimale.



Evolution du temps d'execution (en ms) par rapport au nombre de villes

Méthodes incomplètes



Méthodes incomplètes

Approche 1

1. Modèle de données contenant :

- La matrice de distances (NxN).
- Les fenêtres temporelles.
- La matrice du temps entre les villes.

Remarque:
Pour une même valeur d'epsilon, nous avons exécuté le programme 10 fois et retenu la distance minimale obtenue.

2. L’heuristique utilisée :

- Identifier les villes disponibles à l'instant t.
- Choisir parmi les villes diponibles la ville la plus proche.

3.Intensification et Diversification:

Nous avons défini le paramètre epsilon en utilisant trois valeurs différentes :

- epsilon = 0.3 => Intensification -- | Diversification ++
- epsilon = 0.6 => Intensification ++| Diversification --
- epsilon = 1 => Que de l’intensification (glouton pur)

Méthodes incomplètes
Approche 2

1. Modèle de données contenant :

- La matrice de distances (NxN).
- Les fenêtres temporelles.
- La matrice du temps entre les villes.

2. L’heuristique utilisée :

- Identifier les villes disponibles à l'instant t.
- Parmi ces villes, sélectionner celle ayant la fenêtre temporelle la plus courte.
- En cas d'égalité, privilégier la ville la plus proche

3. 100% Intensification

Méthodes incomplètes
Approche 3

1. Modèle de données contenant :

- La matrice de distances (NxN).
- Les fenêtres temporelles.
- La matrice du temps entre les villes.

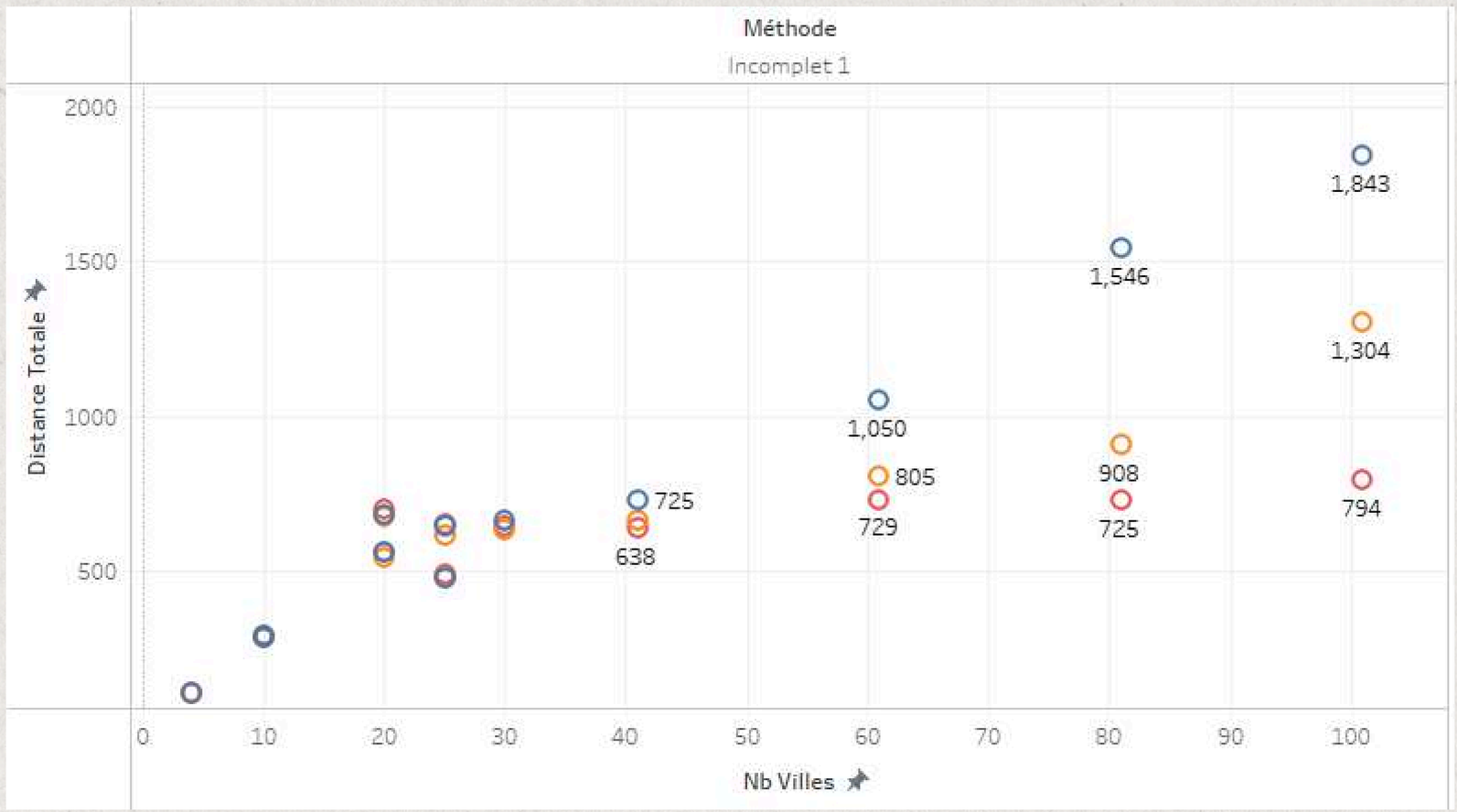
2. L’heuristique utilisée :

- Identifier les villes disponibles à l'instant t.
- Sélectionner aléatoirement une ville parmi celles qui n'ont pas encore été visitées.
- Générer plusieurs trajectoires pendant une durée de temps définie (paramètre timeout), puis on sélectionne la meilleure (plus courte distance)

3. 100% Diversification

Résultats

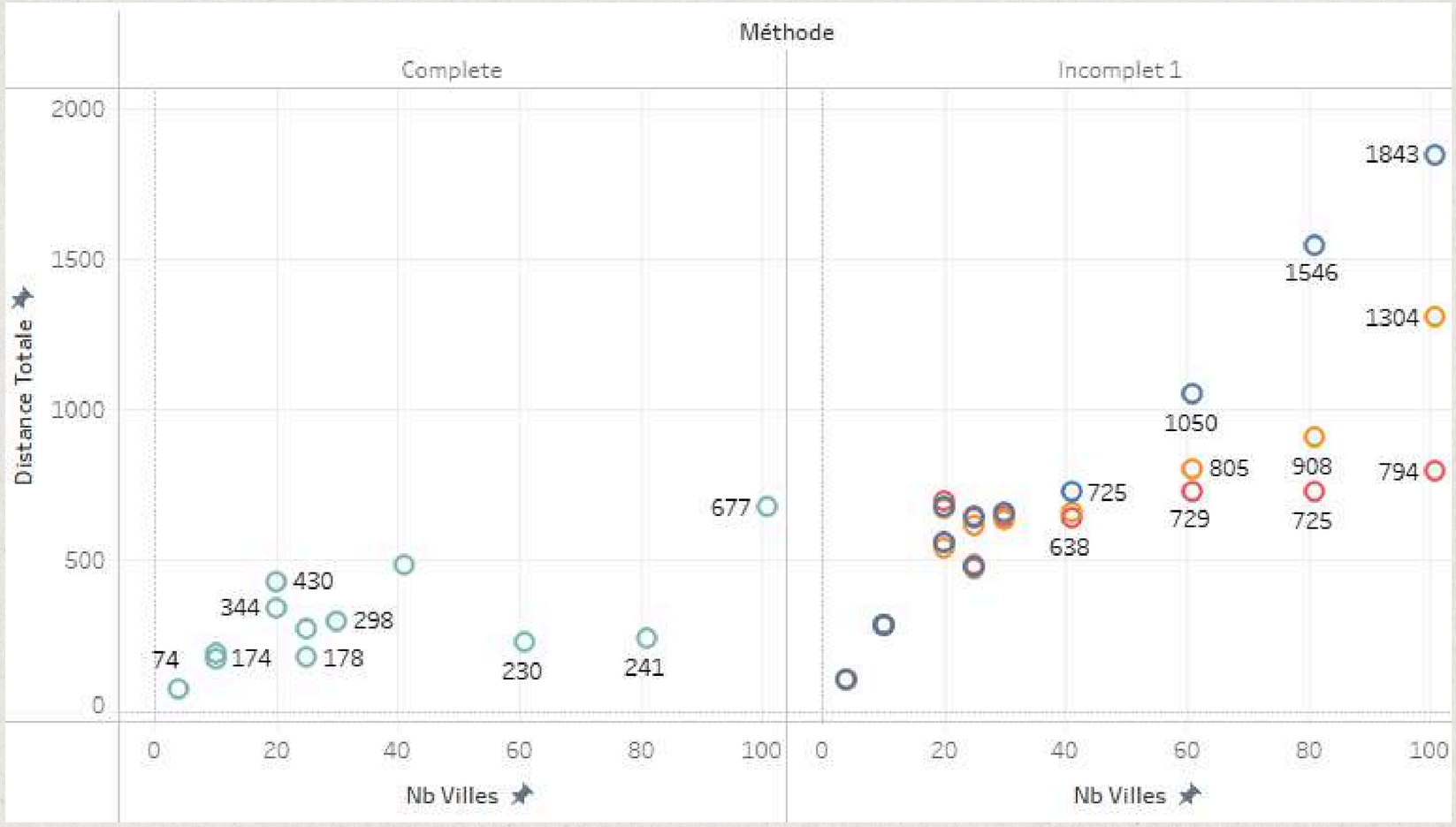
Méthode incomplète - Approche 1



Evolution de la distance avec le nombre de villes pour trois valeurs d'epsilon

Résultats

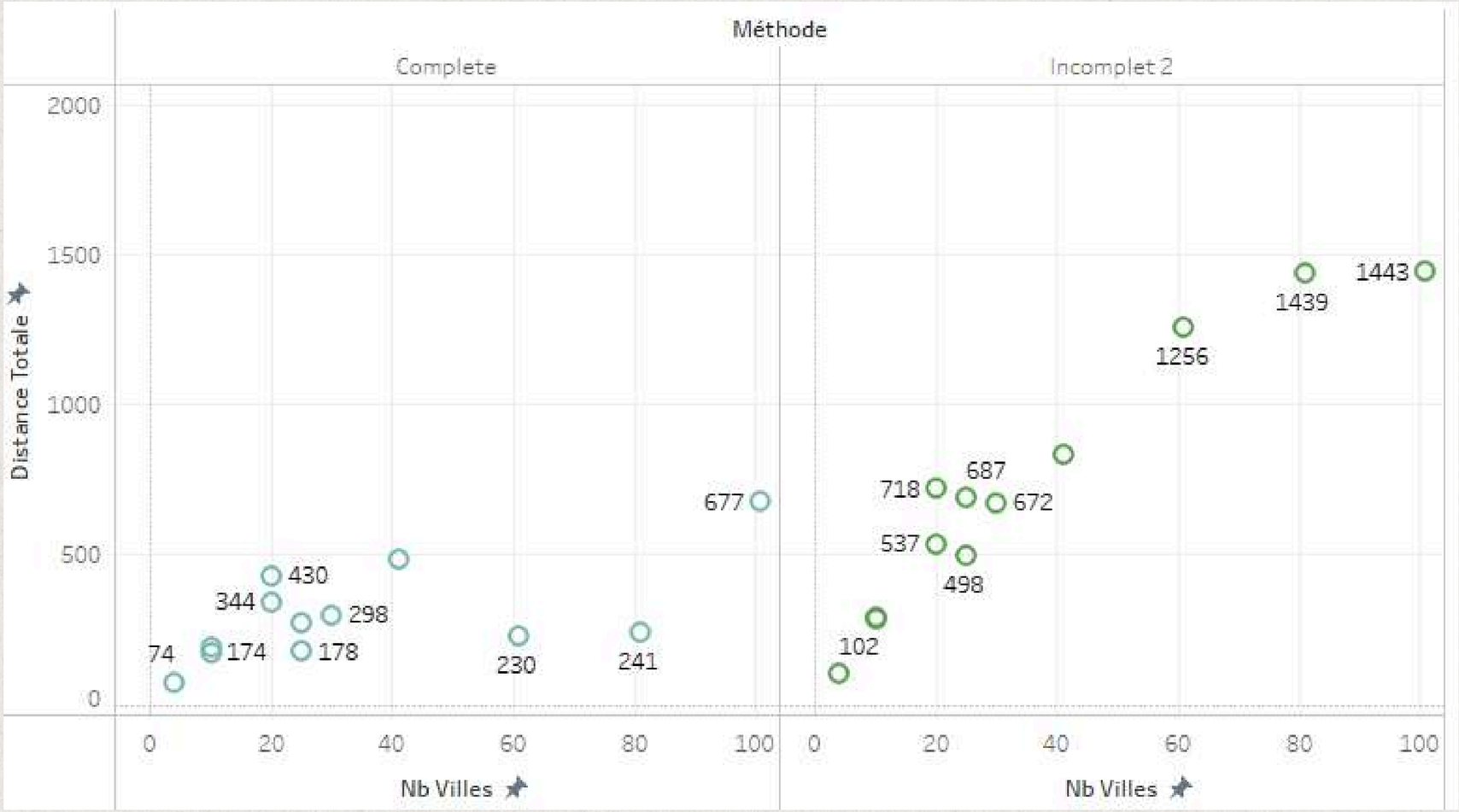
Méthode incomplète - Approche 1 vs Méthode complète



Comparaison entre la méthode complète et incomplète 1 en terme de la distance totale

Résultats

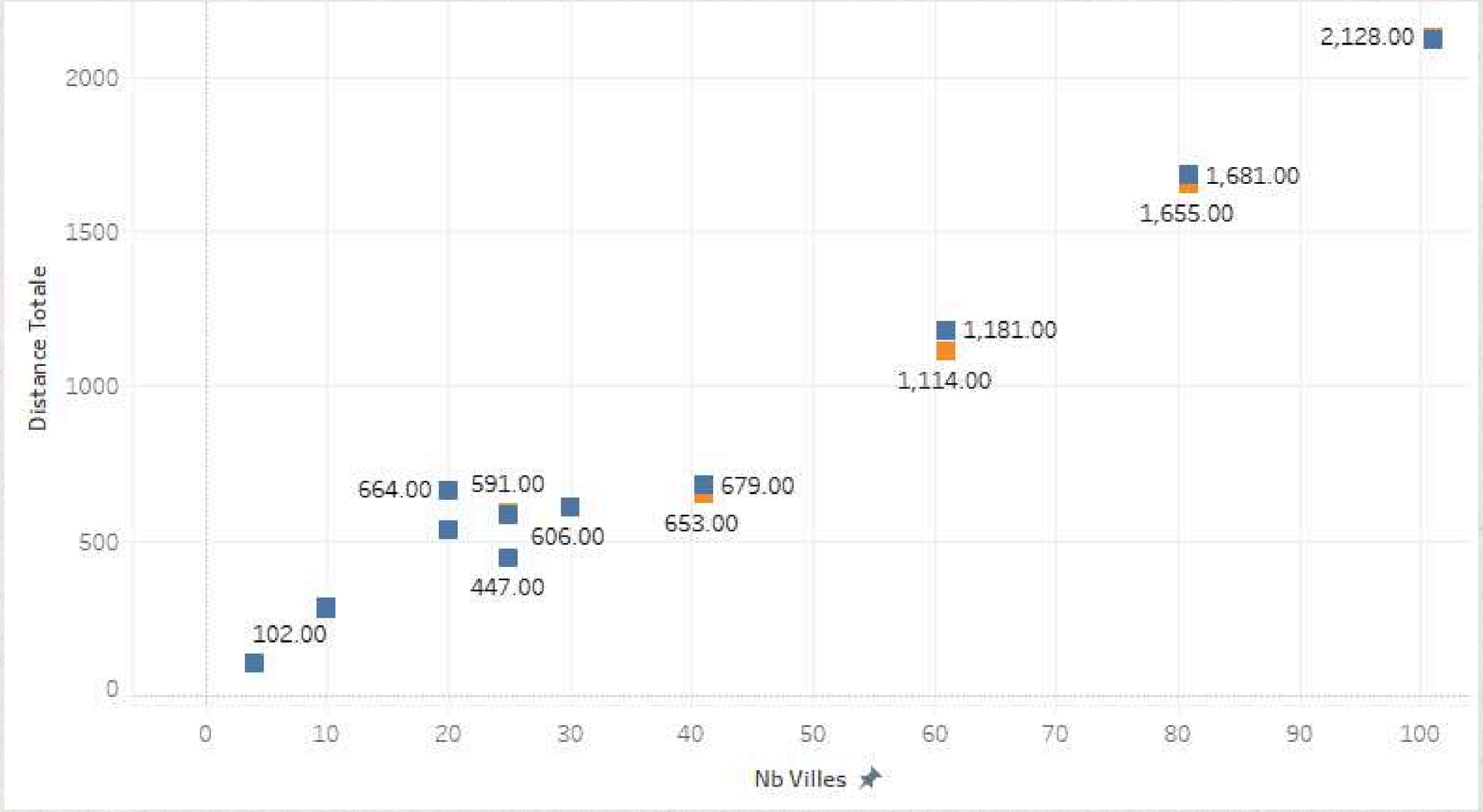
Méthode incomplète - Approche 2 vs Méthode complète



Comparaison entre la méthode complète et incomplète 2 en terme de la distance totale

Résultats

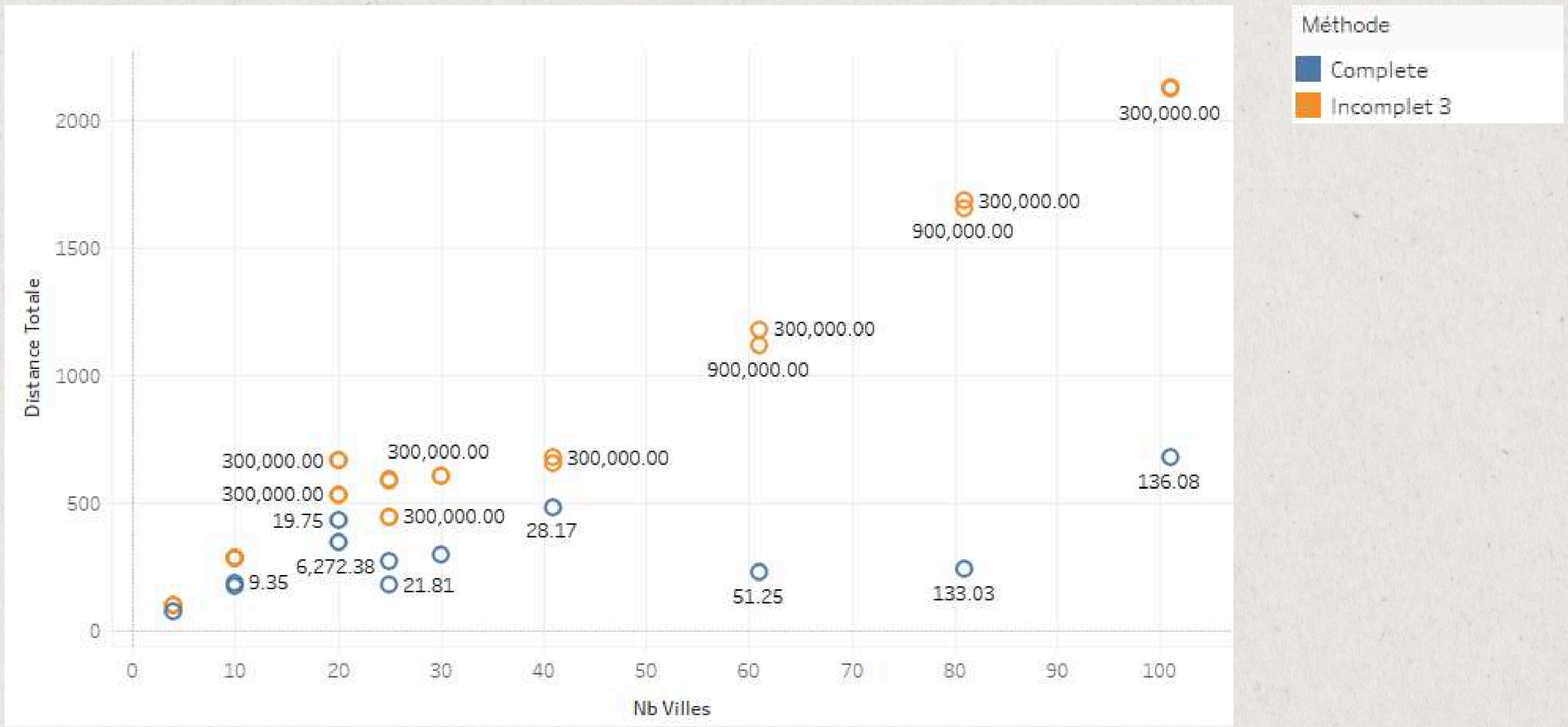
Méthode incomplète - Approche 3



Evolution de la distance totale avec le nombre de villes pour 5 et 15 min en temps limite d'exécution (timeout)

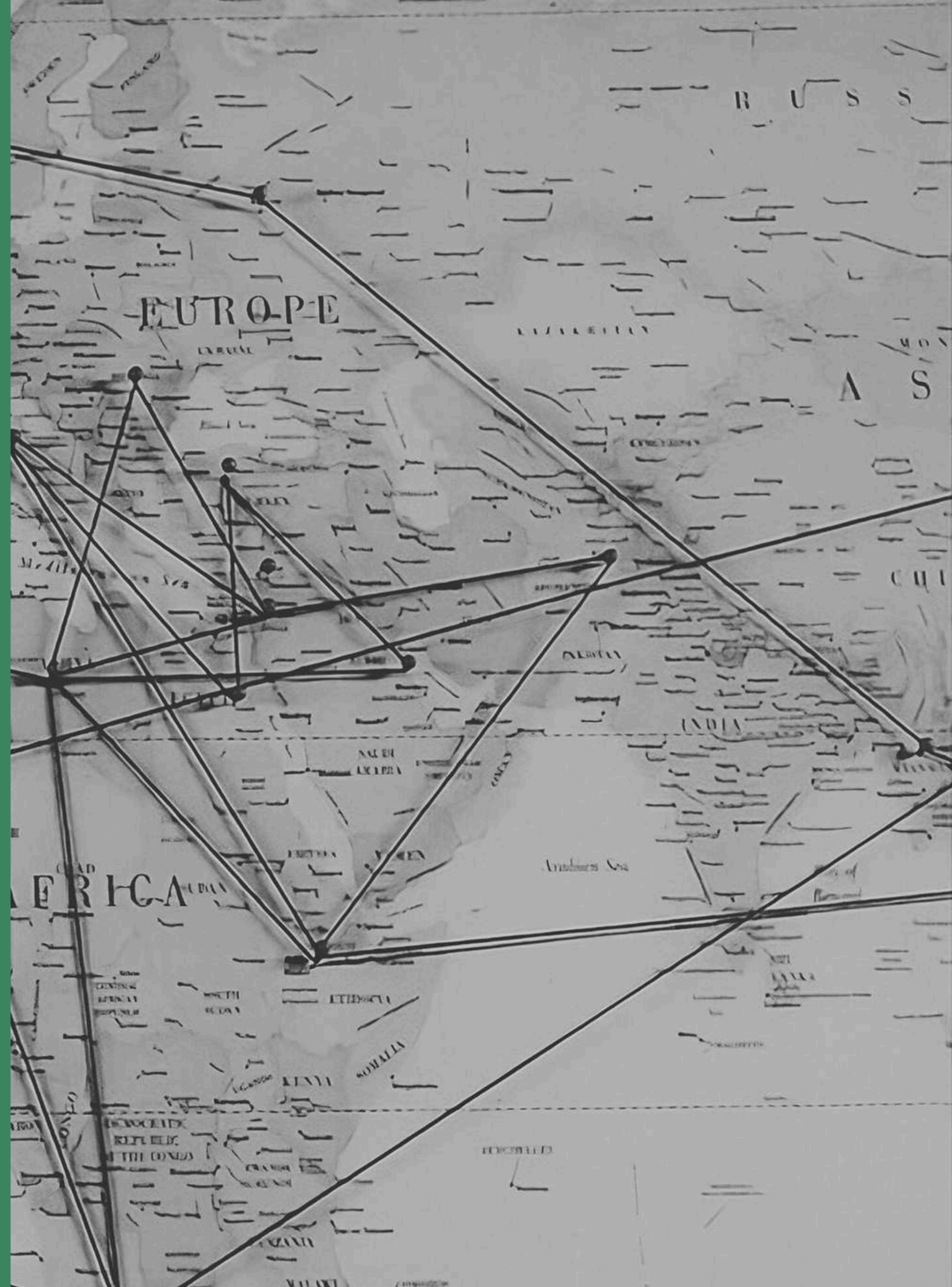
Résultats

Méthode incomplète - Approche 3 vs Approche complète



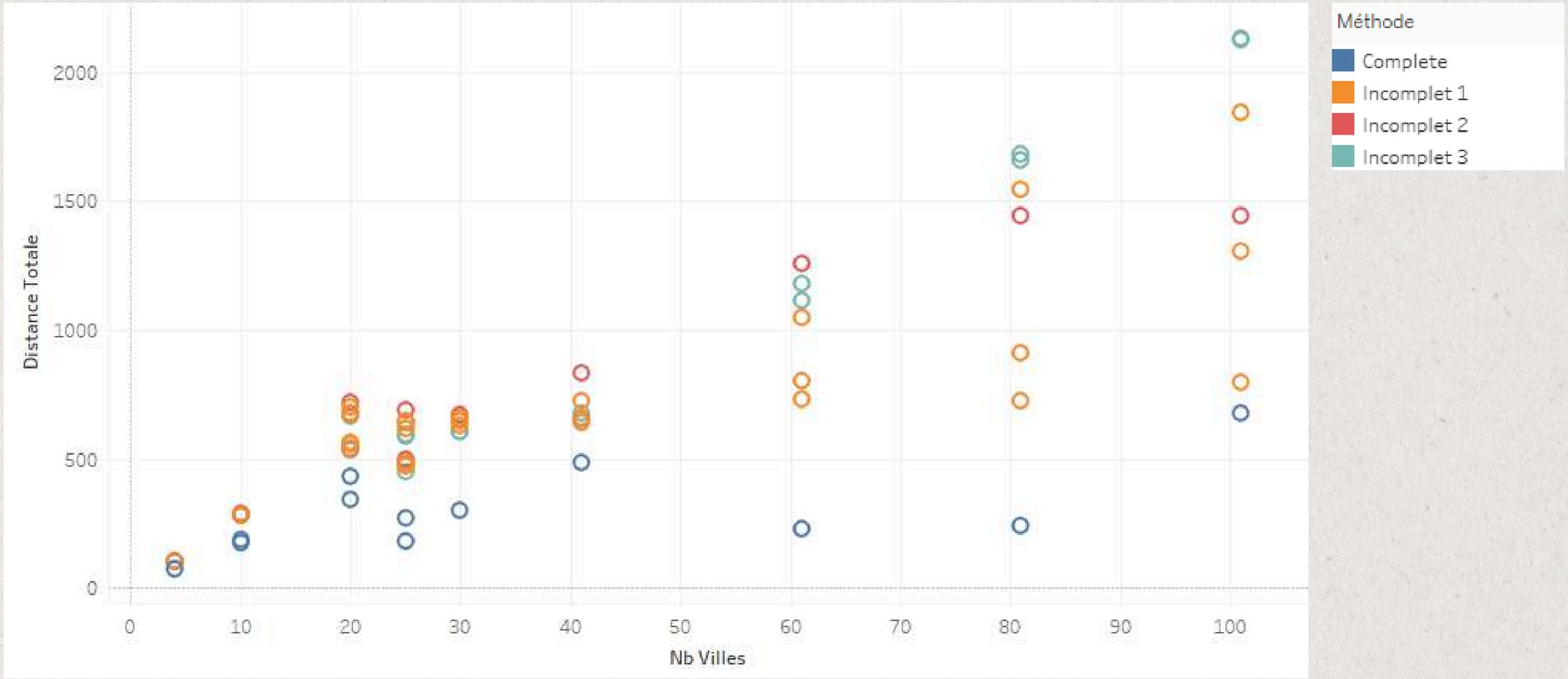
Comparaison entre la méthode complète et incomplète 3 en terme de la distance totale et du temps d'exécution

Comparaison



Comparaison des Résultats

Complète VS incomplète



Comparaison entre les méthodes incomplètes et la méthode complète en terme de la distance totale

Merci pour votre attention