Velib Project

Lilia IZRI
Yacine MOKHTARI

06 Juin 2022

1 Introduction

Dans ce problème d'optimisation, nous devons dans un premier temps, récolter des données en temps réels de différentes stations vélibs de l'Ile-de-France et dresser un planning quotidien que devrait suivre chaque inspecteur des deux ateliers *Alfortville* et *Villeneuve-La-Garenne* afin de traiter en priorité les stations ayant le taux d'anomalies le plus élevés tout en maximisant le nombre de traitements/réparations ou stations traitées. Biensûr, cela va de soit que tout cela sera contraint à la durée de travail dont dispose chaque employé. Puis, dans un second temps trouver le chemin optimal que devra parcourir chaque employé.

Notre objectif est donc d'utiliser le language OPL afin de résoudre ce type de problème.

2 Modèle I: Partitionnement

2.1 Prétraitement

Tout d'abord, comme demandé dans l'énoncé, nous complétons les fichiers qui permettent de récuperer les status des stations pour ainsi générer des instances à notre modèle.

Après cela, nous parcourons les stations à notre dispotion et nous gardons uniquement les stations qui appartiennent bien au workshop que l'on passe modèle, qui un taux d'anomalies ¿ 0 et qui ont bien leur attribut `is_returning` à 1. Nous enregistrons les stations dans un set de tuple ordonné ou chaque tuple correspond à l'id d'une station, son taux d'anomalies et le nombre d'anomalies. Le set ordonné nous permet d'avoir les stations classées de telle sorte à ce que nous avons les stations les plus prioritaires en premier (Taux et nb d'anomalies les plus élevés).

Tout en parcourant ces stations, nous gardons également les distances de celles ci dans des variables gloables afin de les réutiliser par la suite.

Après cela, Le but principal du fichier `partitioning.mod` étant de répartir les k stations les plus urgentes à traiter aux inspecteurs d'un même atelier, la grande partie du travail à réaliser consistait donc à estimer k. Pour ce faire, nous nous sommes dit que:

- **Calculer le nombre k qui représente le nombre de stations qu'on pourrait éventuellement visiter en une journée de travail:**
  Pour cela, nous étions parti de cette formule-ci pour approximer la valeur de k :

$$(t_i \times AM)k + \frac{(k-1) \times DM}{VM} + AR \times NI \leq periodLength \times NI$$

  Effectivement, cette formule nous donne une borne supérieur (en quelques sortes une ssur-estimation) du nombre k.

  - **AM** : nombre d'anomalie en moyenne*.
  - **DM** : la distance moyenne* entre les stations.
  - **VM** : Vitesse moyenne de déplacement (`averageDrivingSpeed`)

– **AR** : Qui désigne le temps d'un aller (de l'atelier vers la première station) et le retour (de la dernière station vers l'atelier). Les dites stations étant encore inconnues, nous approchant cela par 2 × le temps moyen entre l'atelier et les stations*.

– **NI** : Nombre d'inspecteurs ratachés à une station (`numberOfInspectors`)

*: Nous concidérons l'ensemble des stations ratachées à cet atelier, d'où notre appelation de cette approximation *surestimation*.

Nous pouvons ainsi simplifier notre approximation comme suit :

$$(t_i \times AM)k + \frac{DM \times k}{VM} - \frac{DM}{VM} \leq periodLength \times NI - AR \times NI$$

$$k(t_i \times AM + \frac{DM}{VM}) \leq (periodLength - AR) \times NI + \frac{DM}{VM}$$

On peut ainsi donc approcher la valeur de k en prenant la partie entière inférieure :

$$k = \lfloor \frac{(periodLength - AR) \times NI + \frac{DM}{VM}}{(t_i \times AM + \frac{DM}{VM})} \rfloor$$

Finalement, puisque l'on possède un set ordonné des stations selon le taux d'anomalies et le nombre, il suffit que l'on retienne les K premières stations.

2.2 Variables du modèle:

Une fois notre k estimé, nous pouvons définir les variables de notre modèle qui permettront la résolution du problème de partitionnement. Pour ce faire, nous avons défini plusieurs variables:

- **Une variable principale** : Une matrice de taille `numberOfInspectors`×`k` appellée `repartitions`. Celle-ci representra les partitionnements des k stations sur les inspecteurs de l'atelier. Cette matrice vaudra 1 à la case (i, j) si la station j a été affectée à l'inspecteur i et 0 sinon.

- **Deux variables pour les distances des partitions** : `distMoyAtelier` et `distMoyEntreStations` toutes deux de tailles `numberOfInspectors`, representront respectivement la distance moyenne entre les stations de chaque partition et l'atelier du modèle, la distance moyenne entre les stations de chaque partition.

- **Deux variables d'accomodation pour les temps des partitions** : `tempsMoyVersAtelier` et `tempsMoyEntreStations` toutes deux encore de tailles `Nombre d'inspecteurs`, representront respectivement le temps moyen entre les stations de chaque partition et l'atelier du modèle, le temps moyen entre les stations de chaque partition. Celles ci, sont facilement déductibles à partir des variables précédentes de distances $t = \frac{d}{v}$.

- **Une variable pour les diametres** : `diametresPartitions` de taille `Nombre d'inspecteurs`, qui representera le diametre maximale de chaque partition.

- **Une variable pour compter le nombre stations que l'on traite** : `nombre_stations_a_traiter` de taille `k`, qui represente le nombre de stations que l'on traite à chaque fois.

- **Une dernière variable qui sera notre objectif à minimiser** : `objectifAMinimiser` , qui est simplement la somme des diamètres de chaque partition moins le nombre de stations que l'on souhaite traiter. En effet, nous cherchons à minimiser les diamètres de chaque partition et à maximiser le nombre de stations traitées. Le diamètres étant une valeur relativement grande, nous appliquerons un facteur de 0.001.

- Nous avons d'autres variables/constantes que l'on ne détaillera pas. Celles-ci sont issues principalement de la partie prétraitement et nous servent afin d'extraire les informations utiles à la résolution de modèle. A titre d'exemple nous avons, une matrice de distance entre toutes les stations, ainsi qu'un vecteur/tableau de distance entre chaque station et atelier.

2.3 Contraintes

Les contraintes du modèles se font ainsi de façon naturelle et très directe :

- **C1:** Une station parmis les k séléctionnées ne doit être visitée que par au plus un seul agent. Cette contrainte de différence est traduite en disant que la somme de chaque colonne ne doit pas excéder 1.
- **C2 & C3:** Nous devons nous assurer que, respectivement, la distance moyennes entre les stations d'une même partitions (`distMoyEntreStations`) et la distance moyenne entre les stations et l'atelier () pour un même inspecteur est bel et bien égale à la moyenne distances des stations affectées à ce même inspecteur dans la variable `repartitions` et entre ces dernières et l'atelier.
- **C4:** S'assurer que la valeur du diamètre pour un inspecteur est égale à la distance maximale entre les stations qui lui ont été affecté dans la matrice `repartitions`.
- **C5:** La contrainte la plus attendues, est que l'inspecteur ne doit pas travailler plus de `periodLength` (cela revient à dire que, l'aller-retour, ses deplacements entre toutes ses stations et le temps de réparer le anomalies est inférieur ou égale à `periodLength`

2.4 Post-traitement

D'une part, nous afficherons à chaque execution du modèle et pour chaque inspecteur, un tableau représentant ses stations. Exemple:
- Inspecteur 1: [1, 0, 0, 1] (Equivaut à la première et la dernière station ont été affectées au premier inspecteur).
Nous afficherons egalement les distances moyennes entre stations et atelier de sa partitions, les temps moyens, ainsi que son temps d'inspection.

D'autre part, nous écrivons les résultats du modèle dans un fichier `.dat` avec le format demandé dans l'énoncé du projet dans le dossier `velib/data/dat/results/partitioning/`.

3 Résultats - Model I:

Nous avons testé quelques instances avec differents paramètres et nous avons constaté que pour des paramètres assez élevés (Où le nombre de stations traitables serait donc plus important) le modèle peut tourner longtemps avant de converger. Mais pour des paramètres assez raisonnable (par ex l'instance: `velib/data/dat/partitioning_instances/ instance_220503-1434_al_10_2_180_25.dat`, on obtient des résultats assez convaincaints: Chaque inspecteur (parmi les 2) traite environ 2-3 stations pour une durée d'inspection d'environ 1h chacun. En rajoutant à cela leurs temps de trajet, on obtient un temps total de travail un peu plus petit que `periodLength`.

En testant notre modèle sur l'instance mentionnée précedemment, nous obtenons ces résultats:

```
 -------
 -- Inspecteur n° 1 - Sa Répartition:
 [0 0 0 0 0 0 1 0 0 0 0 1 0 0 0]
- Distance moyenne entre les stations de sa partition : 1833m.
- Distance moyenne entre l'atelier et ses stations : 8132m.
- Temps d'inspection: 130min.
- Temps moyen entre pour parcourir toutes les stations de sa partition :5min.
- Temps moyen entre une station et l'atelier : 20min.
 -------
 -------
 -- Inspecteur n° 2 - Sa Répartition:
 [0 0 0 0 1 0 0 1 0 0 1 0 0 0 0]
- Distance moyenne entre les stations de sa partition : 4225m.
- Distance moyenne entre l'atelier et ses stations : 9714m.
- Temps d'inspection: 100min.
- Temps moyen entre pour parcourir toutes les stations de sa partition :21min.
- Temps moyen entre une station et l'atelier : 24min.
 -------
```

Figure 1: Résultats du premier modèle

Sachant qu'il faut multiplier le temps vers l'atelier × 2 et que `periodLength` vaut 180, les résultats semblent cohérents.

Voici un exemple de fichier résultant:

```
partitionnement={
<5, 58391194, 48.856, 2.3579, 4012, Tiron - Rivoli >
<8, 101024234, 48.828, 2.3843, 13050, Quai Panhard et Levassor>
}
```

Figure 2: Aperçu du fichier résultant.

4 Modèle II: Tour optimal

4.1 Prétraitement

Nous n'avons malheureusement pas eu le temps de traiter la partie prétraitement afin d'extraire les temps réels ainsi que de creer l'ensemble des stations d'une partition.
Cependant, afin de pouvoir créer un modèle capable de résoudre le problème du tour, nous avons supposé avoir une matrice de temps `matriceTemps` en notre possession qui décrit les temps entre les stations et l'atelier. Ainsi qu'une instance `'velib/data/dat/tour_instances/instance.dat'` que nous avons créée à la main.

4.2 Variables du modèle:

- **Une variable principale pour définir l'ordre de parcours**. Pour cela, nous définissons une matrice `matriceOrdre` de taille `Ordre` × `NbStations+1`. Où ordre, est un range allant de 0 à NbStations à traiter + 1 (On appelera l'ensemble des stations à traiter avec le workshop `lieux`).
- **Une variable critère d'oprimisation** `tempsTrajet` qui represente le temps de trajet (que l'on cherche donc à minimiser).
- Comme annoncé précédemment, on suppose que l'on dispose d'une instance qui correspond à la structure `tour_stations` et de la matrice de temps correspondante à cette instance que l'on déclare au début du fichier. Ce qui implique que le modèle ne marche qu'avec des instances qui correspondent à la taille de cette matrice.

4.3 Contraintes

Les contraintes du modèles se font ainsi de façon naturelle et très directe :

- **C1:** On doit avoir uniquement Un "1" par ligne et par colonne, sauf, pour le cas du workshop ou il possède deux 1 dans sa colonne.
- **C2:** Nous devons nous assurer que, l'on commence le trajet à partir du workshop et que l'on finit bien dans celui ci. Il faudrait donc fixer la valeur de 1 dans la case de la première ligne (ordre = 0) au niveau du workshop et pareillement pour la dernière ligne (ordre = n) avec n = Nbstations + 1.
- **C3:** Il faut que le temps de trajet corresponde bien à la somme des temps entre deux lieux consécutifs.

4.4 Post-traitement

Au moment du post-traitement, On affiche l'ordre dans le quel l'on doit parcourir les stations décrites par l'instance.

5 Resultats - Modèle II

Le modèle marche sur des petites instances que l'on a pu testées. Mais nous ne saurons dire s'il serait efficace sur de plus grandes instances.

Si l'on suppose que l'on possède 4 stations et cette matrice pour les temps entre les stations et l'atelier:

```
int matriceTemps [lieux][lieux] = [[1000,    10,     1,     10,   10],
                                   [  10,  1000,    10,     10,   10],
                                   [  10,     1,  1000,     10,   10],
                                   [   1,    10,    10,   1000,   10],
                                   [  10,    10,    10,      1, 1000]];
```

Figure 3: Matrice de temps supposée.

Où la case (i, j) représente le temps entre le lieu i et le lieu j avec $(i, j) \in$ {stat1, ..., stat4, Workshop}. On obtient ce résultat:

```
OBJECTIVE: 14
L'ordre pour parcourir:
Workshop --> stat4 --> stat1 --> stat3 --> stat2 --> Workshop.
<<< post process
```

Figure 4: Résultats du deuxième modèle.

Qui est bien le chemin qui fait le moins de temps.

6 Conclusion

Ce projet nous aura permis de travailler sur de vraies données et ainsi faire face aux problèmes que l'on pourrait rencontrer avec ce type de données afin de réaliser un modèle permettant d'optimiser certaines tâches.