Notice d’utilisation du prototype

I) Environnement requis :

- Environnement de développement : Windows 10
- Version de python utilisée : 3.10.7
- Librairies utilisées :
  - pandas 2.0.0
  - jupyterLab 3.6.1
  - matplotlib 3.7.1
  - folium 0.15.0
  - numpy 1.23.3
  - openrouteservice 2.3.3

Attention, pour openrouteservice, nécessite une API_KEY récupérable sur leur site (nécessite la création d’un compte).

II) Génération d’instance :

Cette partie à pour but d’expliquer comment construire une instance, ou plus particulièrement, comment utiliser le script « *menu.py* » pour pouvoir construire une instance à partir des tableurs détaillés dans la documentation d’instance.

Pour rappel, les données d’une instance sont les suivantes :

- Les tableurs résumant les différents acteurs (plateformes, cantines, producteurs, transformateurs).
- Les coûts d’ouverture de plateforme,
- Les coûts de transport entre deux acteurs,
- Les filières de produits
- La capacité des véhicules

Toutes ces informations sont stockés dans un dossier du nom de l’instance dans « *in/instance* ».

1) Créer un dossier dans *« in/instance/ »*:

Pour créer une nouvelle instance, commencez par créer un dossier du nom de l’instance dans « *in/instance* ». Vous pouvez y insérez les différents tableurs par la suite, veuillez les nommer de la manière suivante :

- « *e.csv* » pour le tableur des établissements*,
- *« f.csv »* pour le tableur des fournisseurs*,
- *« t.csv »* pour le tableur des transformateurs*,
- *« n.csv »*pour le tableur des plateformes*,

- « *d.csv* » pour le tableur des commandes*.

Les colonnes nécessaires pour chaque tableur sont détaillées dans la documentation d'instance.

Un dernier fichier nommé « *data.txt* » est utilisé. Ce dernier enregistre les différents coûts et d'autres informations comme le nombre d'acteurs de chaque type.

2) « *menu.py* »

Le script *« menu.py »* au point d'entrée du prototype est un utilitaire de vérification et de construction d'instances. A partir de lui, vous pouvez déterminer si les informations contenues dans le dossier de l'instance sont suffisantes pour pouvoir démarrer la résolution.

Note très importante : ce script ne va faire que regarder la présence des colonnes nécessaires pour les différentes opérations, **le contenu des colonnes n'est jamais vérifié**. Par conséquent, c'est lors de la rentrée de ces colonnes qu'il vous faut assurer qu'elle soit intégralement rempli.

Le script permet notamment de construire le fichier « *data.txt* », à partir des tableurs déjà présents. Une construction au fur et à mesure est possible, mais privilégié la création du fichier en un seul tour.

C'est aussi à partir de ce script que vous pouvez aussi générer les différents informations, et les commandes.

Pour résumer, le fonctionnement du script est le suivant :

Vérification des fichiers dans le dossier → Si nécessaire et possible, génération des coordonnées pour chaque tableur→ Si nécessaire, génération du tableur des commandes avec génération des demandes des cantines et affectation des filières aux producteurs

Si « data.txt » dans dossier, vérification des informations présentes dans le fichier → récupération des infos présentes + détection infos manquantes → si infos manquantes, proposer les générations et/ou entrées manuelles si possible → sauvegarde des nouvelles informations dans le fichier « data.txt »

Une fois ces deux étapes résolues, le script retourne à la détection d'instance, et vous permet de recommencer. A la fin de l'exécution, ou au tout début, si l'instance est prête, ce message s'affiche.

```
Instance prête pour résolution, modifier main.py pour pouvoir l'utiliser
```

Note : Si non modifié, le script n'est pas capable encore de lire directement une colonne «Coût d'ouverture» comme indiqué dans la documentation précédente. Il vous sera proposé de la rentrer manuellement dans le script.

Il ne sera pas détaillé l'usage du programme puisqu'il est volontairement simpliste et explicite sur les informations manquantes, et proposera spontanément les opérations possibles.

3) Usage des générateurs

Une petite note sur les différents générateurs, la plupart sont des formules, il n'y a alors pas trop de limites par rapport à elle à l'exception de deux d'entre eux :

« x », « y » :

Ce générateur utilise le service **Nominatim** d'*OpenRouteService Fondation*. Par faute de capacité pour leur serveur, il n'est pas recommandé d'user de leurs services pour un grand/très grand requetage. Comme indiqué précédemment, les serveurs peuvent vous bloquer temporairement l'usage, le script de génération vous avertira si le requetage est interrompu par le serveur.

Dans tous les cas, vous aurez les adresses qui n'ont pas pu être trouvées. Un nouveau tableur vient prendre la place de l'ancien, qui a pour en préfixe « *old_* ».

A cause du fait que la vérification du contenu des colonnes ne se fait pas, même si le script vous prévient avoir terminé et que l'instance est prête, passer tout de même le temps de vérifier rapidement les localisations, et s'il a tout trouvé, et le cas inverse, récupérer vous même les informations manquantes. Le script trouve environ 80 % des adresses correctement formées, ce travail ne doit pas mettre trop de temps.

Coût de transport

Comme pour les localisations, nous utilisons *OpenRouteService* pour pouvoir récupérer des matrices de durées et de distance.

Contrairement au précédent générateur, celui-ci a besoin d'une **API_KEY** pour pouvoir effectuer les requêtes. Pour cela, diriger vous sur le site d'OpenRouteService, le site vous dirigera pour créer un compte pour pouvoir réclamer votre clé.

Une quantité de requêtes est instaurée sur votre clé. Nous utilisons le service de récupération de matrices qui permet de grandement économiser du budget de calcul. En clair, tant qu'il n'y a pas plusieurs milliers d'acteurs différents, la contrainte n'est pas présente.

Une fois votre **API_KEY** récupéré, allez dans *« src/constant.py »* et enregistrer là à **APIKEY_OPENROUTE**. Techniquement, *« src/constant.py »* ne devrait pas être dans le dépôt git. Assurez vous de verrouillez le dépôt pour qu'il ne soit accessible qu'au public.

III) Résolution

Une fois que toutes les générations sont faites, et que l'utilitaire vous confirme qu'une instance est utilisable, ouvrez « *main.py* »

```python
import sys

sys.path.insert(0, '')
from src.constant import PATH_OUT

from src.resolution.resolve import control

from src.entity.instance.instance import Instance

#INSEREZ LE NOM DE L'INSTANCE ICI
name = "nope"
budget_calc = 40000
nb_perm = 30

instance = Instance(name)

instance.load_instance()

control(PATH_OUT,instance,budget_calc,nb_perm)
```

Entrez dans « *name* » le nom de l'instance à résoudre, dans « *budget_calc* » le budget de calcul accordé pour chaque perturbation, et « *nb_perm* » le nombre de perturbation. Une fois fait, exécutez le fichier.

Une fois la résolution terminée, un dossier du même nom que l'instance apparaît dans « *out/* ». Ce dernier contiendra plusieurs éléments, des cartes.html construites avec Folium, où le nom est expliqué dans la documentation technique, un dossier solution où vous pouvez récupérer des solutions sauvegardées et en texte brut pour d'autres usages.

IV) « *main.ipynb* »

Ce notebook Jupyter est un document utilisé pour tester différentes fonctions/vérifier certaines solutions si nécessaire. Il est en dehors de la résolution du problème, vous pouvez l'utilisez pour ce que vous désirez, voir le supprimer.