Devoir 2 - Alimentation de fours à pizza

Remise le 17/03/24 — 2024 (avant minuit) sur Moodle pour tous les groupes.

Consignes

- Le devoir doit être fait par groupe de 2 au maximum. Il est fortement recommandé d'être 2.
- Lors de votre soumission sur Moodle, donnez votre rapport en format **PDF** ainsi que vos fichiers de code à la racine d'un seul dossier compressé nommé (`matricule1_matricule2_Devoir2.zip`).
- Indiquez vos noms et matricules en commentaires au dessus des fichiers .py soumis.
- Toutes les consignes générales du cours (interdiction de plagiat, etc.) s'appliquent pour ce devoir.
- Il est permis (et encouragé) de discuter de vos pistes de solution avec les autres groupes. Par contre, il est formellement interdit de reprendre le code d'un autre groupe ou de copier un code déjà existant (StackOverflow ou autre). Tout cas de plagiat sera sanctionné de la note minimale pour le devoir.

Enoncé du devoir

Célestin, votre fidèle ami, magnat de l'immobilier et artiste à ses heures perdues, a été très satisfait de votre travail lorsque vous l'avez aidé à placer des tableaux pour son exposition. Il a décidé de vous présenter à Marco, pasteur le dimanche et gérant d'une chaîne de pizzerias artisanales en semaine. Contrairement à l'opinion publique qui veut que la sauce tomate soit le nerf de la guerre dans cette industrie, le charbon nécessaire à alimenter les fours est souvent en réalité la ressource la plus critique.

⚠**Le problème du devoir étant plus complexe que le précédent, assurez-vous de bien comprendre l'énoncé, ainsi que les différentes contraintes due problème, avant de commencer votre implémentation.**

Tenir dans le temps

Depuis le lancement de délicieuses pizzas aux fruits de mer, la chaîne restaurants de Marco est victime de son succès et les fours doivent être alimentés en permanence pour pouvoir subvenir à la demande de ses clients. Il a besoin de vous pour gérer ses stocks de charbon à moindre coût pendant un nombre T de jours.

Dépenses et contraintes liées à la livraison

Marco a à sa disposition une flotte de M camions à benne identiques de capacité Q pour alimenter ses n restaurants chaque matin. Ces camions coûtent à Marco environ 1$ par kilomètre de distance parcourue (distance euclidienne). Chaque camion doit rentrer à la mine en fin de journée pour être prêt pour ses livraisons du lendemain, tous les véhicules ne doivent pas obligatoirement effectuer des livraisons tous les jours et chaque pizzeria ne doit être livrée qu'une seule fois.

Dépenses et contraintes liées à la mine de charbon

Les camions vont chercher leur marchandise à l'exploitation minière détenue par Luigi, le cousin de Marco, qui *produit* r_0 kilogrammes de charbon par jour et veut à chaque fois idéalement se débarrasser de tout son charbon. Luigi charge à son cousin c_0 dollars par kilogramme de charbon que les camions laissent

derrière eux. Il n'est évidemment pas possible de prendre plus de charbon que Luigi n'en propose. On pose la convention que la mine est référé par l'indice 0. Le coût lié à la mine au temps t ($\text{cout}_{t,0}$) dépend ainsi de la quantité de charbon en stock à la mine au temps t ($\text{quantite}_{t,0}$), de ce qui est produit par jour (r_0, cette quantité est fixe), de ce qui est livré au temps t (totalLivré_t), et de ce que Luigi charge à Marco par kilo non livré c_0. On a ainsi la relation suivante :

$$\text{cout}_{t,0} = (\text{quantite}_{t,0} + r_0 - \text{totalLivré}_t) \times c_0 \tag{1}$$

Contraintes de capacité

Une fois les camions chargés, ils sont déployés pour déposer des quantités de charbon pour alimenter les four des pizzerias. Chaque pizzeria est référée par un indice $i \in 1, \ldots, n$. Chacune d'entre elles est équipée d'un four avec une capacité en charbon maximale U_i pour lequel la quantité ne doit pas tomber sous une valeur minimale (L_i). De taille différente, chaque four d'une pizzeria i *consomme* r_i kilogrammes de charbon par jour. On considère uniquement les valeurs post-livraison et post-consommation pour les seuils de quantités, avec $\text{livraison}_{i,t}$ indiquant la quantité livrée à la pizzeria i le jout t. Chaque pizzeria i possède également son stock de charbon qui varie de jour en jour en fonction de ce qui est consomme. On a ainsi, pour chaque jour t et pour chaque pizzeria i, les relations suivantes :

$$\text{quantité}_{i,t} = \text{quantité}_{i,t-1} + \text{livraison}_{i,t} - r_i \tag{2}$$

$$\text{quantité}_{i,t} \geq L_i \tag{3}$$

$$\text{quantité}_{i,t-1} + \text{livraison}_{i,t} \leq U_i \tag{4}$$

Dépenses liées au stockage du charbon

Vous conviendrez que le charbon a une odeur forte qui peut couper l'appétit, une grosse quantité de charbon stockée sur place fera fuir certains clients. Les pertes financières pour chaque pizzeria i liées au charbon sont estimées à c_i dollars par kilogramme de charbon non consommé par le four au cours de la journée. Dans les faits, pour une journée t et une pizzeria i on calcule le coût comme suit :

$$\text{cout}_{i,t} = (\text{quantité}_{i,t-1} + \text{livraison}_{i,t} - r_i) \times c_i \tag{5}$$

Votre mission

Marco vous met au défi de programmer pour T jours les routes des camions ainsi que les quantitées livrées à chaque client tout en vous assurant que :

1. les seuils de capacité des pizzerias ne sont pas excédés.

$$\left(\text{quantité}_{i,t} + \text{livraison}_{i,t} - r_i \geq L_i\right) \wedge \left(\text{quantité}_{i,t} + \text{livraison}_{i,t} \leq U_i\right) \forall_{t \in \{1,\ldots,T\}} \tag{6}$$

2. les capacités de chaque camion k ne sont jamais excédées.

$$\text{quantitéTransportée}_{k,t} \leq Q \tag{7}$$

3. les camions passent la nuit au dépôt. Cela s'exprime par le fait que la première et dernière position de chaque camion correspondent à la mine de Luigi (le dépôt).
4. les pizzerias ne sont livrées qu'une seule fois par jour.

Notez bien qu'il n'est pas obligatoire que chaque pizzeria soit visitée chaque jour. Votre objectif est de faire en sorte que Marco doive débourser le moins d'argent possible. Le coût total d'un plan de déploiement correspond ainsi à la somme des trois entités suivantes :

- Des coûts engendrés par la quantité de charbon dans les pizzerias pour chaque jour ($\text{cout}_{i,t}$). Vous pouvez voir ce coût comme un coût de stockage pour chaque pizzeria.
- De l'argent que demandera Luigi à Marco après que tous les camions aient été chargés. Vous pouvez voir ce coût comme un coût de stockage au depôt.
- Des coûts liés aux déplacements des camions entre chaque position (pizzeria ou dépôt). On pose $D_{t,k}$ le nombre déplacements effectués par un camion k entre deux positions au jour t, et travelCost_d le coût d'un déplacement d.

Le coût total est alors exprimé comme suit.

$$\text{coût} = \sum_{t=1}^{T} \sum_{i=0}^{n} \text{cout}_{i,t} + \sum_{t=1}^{T} \sum_{k=1}^{M} \sum_{d=1}^{D_{t,k}} \text{travelCost}(d) \tag{8}$$

Format des instances

Marco opère dans différentes villes, il vous demande de l'aider pour chacune des configurations et vous fournit ces dernières sous le format suivant :

```
(n+1) T  Q  M
0  <x_mine> <y_mine>    <quantite_initiale>              <r_0> <c_0>
1  <x_1>    <y_1>       <quantite_initiale>  <U_1> <L_1> <r_1> <c_1>
...
n  <x_n>    <y_n>       <quantite_initiale>  <U_n> <L_n> <r_n> <c_n>
```

La première ligne du document fourni par Marco indique les variables globales du problème soit le nombre d'endroits où peuvent passer les camions, le nombre de jours pour lesquels il a besoin de vous, la capacité maximale des véhicules et la quantité de camions disponible.

La ligne suivante indique l'identifiant de la mine (0) ainsi que ses coordonnées cartésiennes, la quantité initiale de charbon dont dispose Luigi, la production journalière de la mine et le coût chargé par Luigi pour chaque kilogramme de charbon qu'il doit garder. Les lignes suivantes décrivent les pizzerias de manière similaire à la mine avec comme information supplémentaires les bornes entre lesquelles la quantité de charbon doit être maintenue.

Format des solutions

Il vous est demandé de fournir pour chaque configuration un fichier de solution qui pour chaque pas de temps, spécifie les différentes routes, clients visités et quantités délivrées. Deux retours à la ligne séparent chaque bloc correspondant à un pas de temps. Une fonction utilitaire *Instance.generate_file(solution)* vous est fournie pour générer les fichiers.

⚠Il n'est pas nécessaire de fournir la mine au début et à la fin de votre route, le vérificateur de solution va toujours considérer un cycle!

```
<T> <M>
<t_1_n_clients_1> (<id_client_1>, <q_1>) ... (<id_client_n>, <q_n>)
...
<t_1_n_clients_M> (<id_client_1>, <q_1>) ... (<id_client_n>, <q_n>)
```

```
...

<t_T_n_clients_1> (<id_client_1>, <q_1>) ... (<id_client_n>, <q_n>)
...
<t_T_clients_M> (<id_client_1>, <q_1>) ... (<id_client_n>, <q_n>)
```

Exemple illustratif

A titre d'exemple, vous avez la situation suivant avec la mine (en bleu) sur l'image et trois pizzerias (en rouge) ainsi que leurs état d'inventaire.

```
4   3   500    1
0   154.0  417.0  510                  193    0.30
1   172.0  334.0  130    195    0       65    0.23
2   267.0  87.0   70     105    0       35    0.32
3   355.0  444.0  48     72     0       24    0.23
```

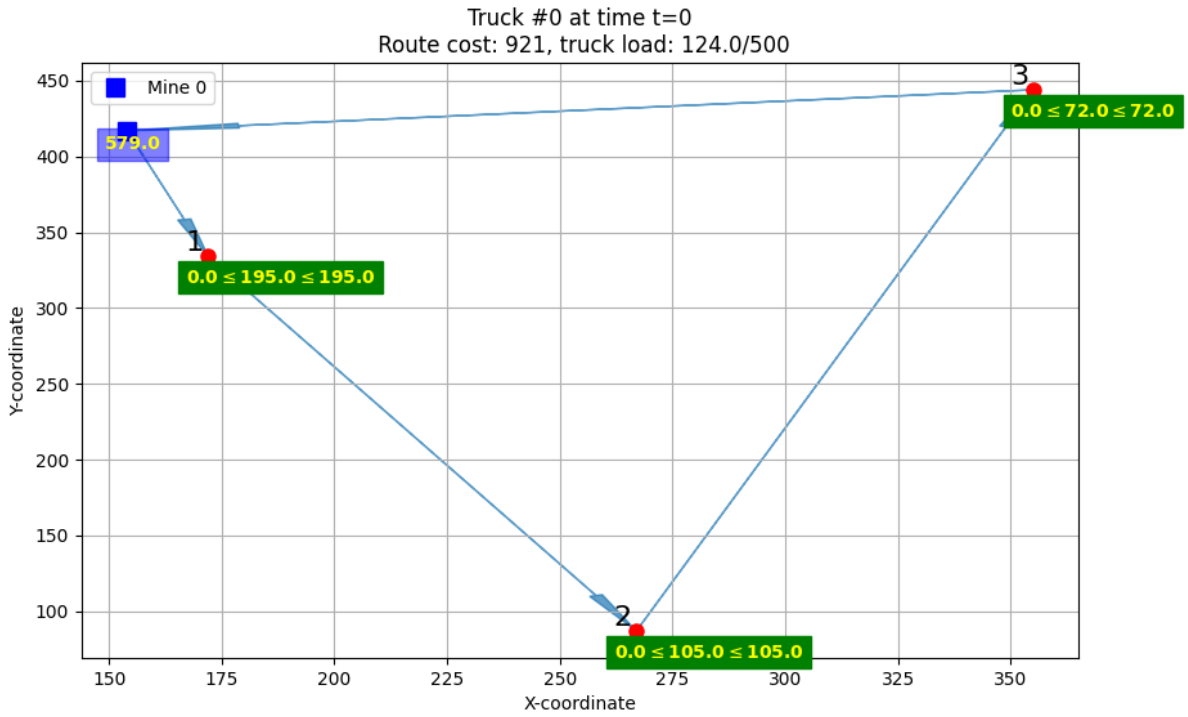

FIGURE 1 – Visualisation d'un pas de la solution pour l'exemple

Vous remarquerez que la solution suivante proposée pour cette situation est assez simpliste, elle propose que tous les camions s'arrangent pour livrer à chaque client un M-ième de leur consommation journalière (ici 1/2, arrondi à l'entier inférieur). Cette solution est valide et coûte 3536.22 dollars à Marco. (190.02 pour les coûts générés par la mine, 5520 pour ceux générés par les camions et 168.72 pour les pertes liées aux quantités de charbon dans les pizzerias).

```
0
0 3 (1, 65.0) (2, 35.0) (3, 24.0)


1
0 3 (1, 65.0) (2, 35.0) (3, 24.0)


2
0 3 (1, 65.0) (2, 35.0) (3, 24.0)
```

La validité d'une solution peut être vérifiée à l'aide des images générées par *Instance.visualize(sol)* qui vous aidera à générer la Figure 2. Des utilitaires vous sont proposés dans le code pour obtenir le détail des coûts et de la validité.

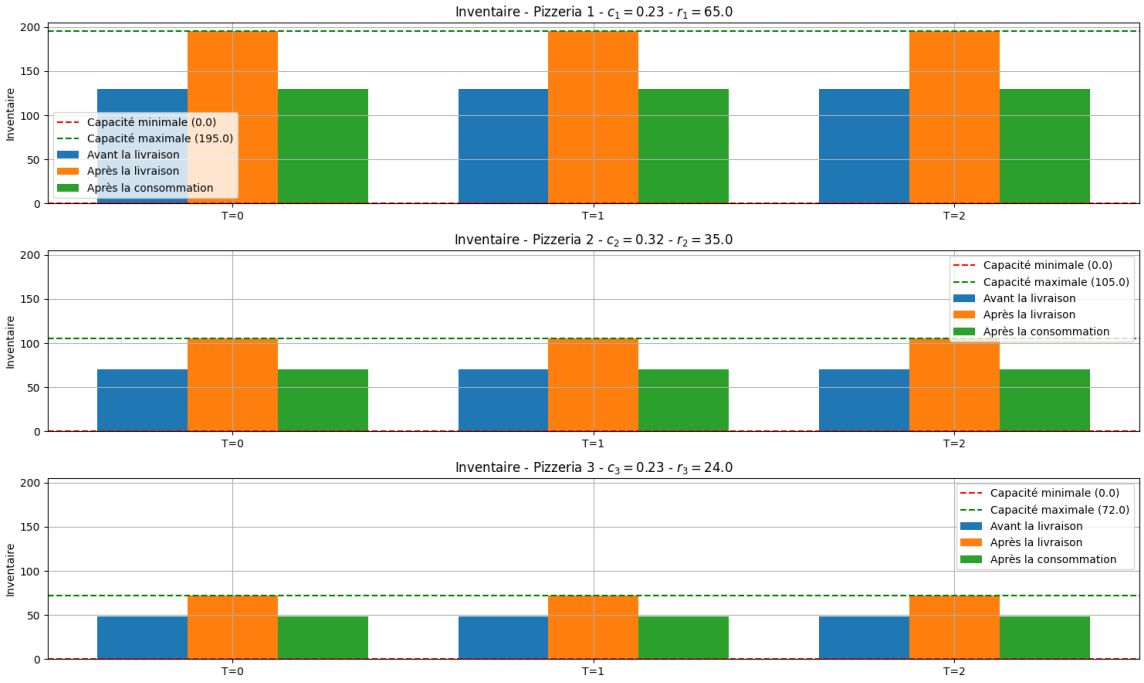

FIGURE 2 – Visualisation de la quantité de charbon des pizzerias pour l'exemple.

Implémentation

Vous avez à votre disposition un projet python. Plusieurs fichiers vous sont fournis :

- `utils.py` qui implémente la classe `Instance` pour lire les instances, sauvegarder les solutions, les valider et les visualiser.
- `main.py` vous permettant d'exécuter votre code sur une instance donnée. Ce programme enregistre également votre meilleure solution dans un fichier au format texte et sous la forme d'une image.
- `solver_naive.py` : implémentation d'un solveur naïf qui réserve un mur par oeuvre d'art.
- `solver_advanced.py` : implémentation de votre méthode de résolution qui sera évaluée pour ce devoir.

— `autograder.py` : un tout petit programme vous permettant de facilement calculer votre note compte tenu de vos performances sur les instances fournies.

Notez bien qu'un vérificateur de solutions est disponible. Vous êtes également libres de rajouter d'autres fichiers au besoin. Au total, 5 instances sont disponibles dans les fichiers `instances/<instance>.txt`. Il vous est demandé de produire les fichiers de solution correspondants `instances/<instance>.txt` pour chacune de ces instances.

Pour vérifier que tout fonctionne bien, vous pouvez exécuter le solveur naïf comme suit.

```
python3 main.py --agent=naive --infile=instances/super_easy.txt
```

Il est possible que pour visualiser les figures vous ayez besoin d'installer `imageio`.

```
pip install imageio
```

Un fichier `solutions/super_easy.txt`, une animation `solution_timesteps.gif` et les illustrations d'évolution d'inventaire seront générés.

Production à réaliser

Vous devez compléter le fichier `solver_advanced.py` avec votre méthode de résolution. Au minimum, votre solveur doit contenir un algorithme de recherche locale. C'est à dire que votre algorithme va partir d'une solution complète et l'améliorer petit à petit via des mouvements locaux. Réfléchissez bien à la définition de votre espace de recherche, de votre voisinage, de la fonction de sélection et d'évaluation. Vous devez également intégrer une métaheuristique de votre choix pour vous s'échapper des minima locaux (autre qu'un simple redémmarage aléatoire). Vous êtes ensuite libres d'apporter n'importe quelle modification pour améliorer les performances de votre solveur. Une fois construit, votre solveur pourra ensuite être appelé comme suit.

```
python3 main.py --agent=advanced --infile=./instances/super_easy.txt
```

Pour éviter de générer les images de visualisation à chaque fois, vous pouvez utiliser l'option `--no-viz`.

Un rapport **succinct** (3 pages de contenu, sans compter la page de garde, figures, et références) doit également être fourni. Dans ce dernier, vous devez présenter votre algorithme de résolution, vos choix de conceptions, vos analyses de complexité. Reportez-y également les résultats obtenus pour les différentes instances. On vous demande finalement de fournir un dossier `solutions` avec vos solutions au format attendu par Marco (décrit plus haut) respectant la structure `solutions/<instance_name>.txt`.

Critères d'évaluation

L'évaluation portera sur la qualité du rapport et du code fournis, ainsi que sur les performances de votre solveur sur les différentes instances. Concrètement, la répartition des points (sur 20) est la suivante :

— **10 points sur 20** sont attribués à l'évaluation des solutions fournies par votre solveur. Cinq d'entre elles vous sont fournies, la dernière restera cachée pour vérifier les capacités de généralisation de

votre solveur. Pour chaque instance vous devrez fournir des **solutions valides** pour pouvoir obtenir des points. Les scores que vous obtiendrez devront se situer entre les valeurs (**LB**) et (**UB**) du tableau. Obtenir la valeur **LB** vous permettra d'obtenir 100% de la note pour cette instance, obtenir une valeur supérieure ou égale à **UB** ne vous fera gagner aucun point (0%). La perte des points par rapport à **LB** est linéaire entre les deux bornes. A titre d'exemple, pour l'instance D, j'ai obtenu une instance valide de coût 4425.4, ma note pour cette instance est de $2(1 - \frac{4425.4-LB}{UB-LB}) = 1.48/2$

La dernière colonne du tableau indique le temps maximum qui sera accordé à votre programme pour fournir une réponse.

Instance	Pondération	T	M	Q	n_pizzerias	LB	UB	Temps max.
A	1/10	3	4	87	5	1640.93	3281.86	5min
B	2/10	3	4	87	5	2475.88	4951.76	5min
C	2/10	3	3	96	5	2061.27	4122.54	5min
D	2/10	3	5	192	10	2911.11	8733.33	10min
E	2/10	3	5	247	15	3487.12	10461.36	10min
X	1/10	3	?	?	?	?	?	10min

TABLE 1 – Résultats attendus pour la notation

— **10 points sur 20** sont attribués à l'évaluation de votre rapport, les éléments indispensables pour obtenir une bonne note sont évidemment :
  — Une formalisation de **votre espace de recherche** (est-il connecté ?).
  — **Une quantification chiffrée** de l'impact des mécaniques implémentées. (Vous échappez-vous bien des extrema locaux ? Si oui, prouvez-le, si non, montrez *formellement* pourquoi.) (Vous avez une stratégie qui permet de réduire la complexité d'une routine ? Prouvez que ce que vous faites est utile avec des *mesures de temps*, de *mémoire* ou en *donnant une notation asymptotique*.)
  — Privilégiez graphes, tableaux et formules pour transmettre l'information de manière concise.
  — Évitez à tout prix les explications de nature qualitatives, rapporter qu'un algorithme "va plus vite", est "plus performant" ou "donne de meilleures solutions" demande peu de travail, arriver à le prouver permet d'obtenir une bonne note.

Une fois à la racine de votre projet, vous pouvez calculer vos scores automatiquement avec :

```
python3 autograder.py
```

Les solutions des instances doivent se trouver dans `./solutions/` et respecter la nomenclature demandée.

⚠Il est attendu que vos algorithmes retournent une solution et des valeurs correctes. Par exemple, il est interdit de modifier artificiellement les dimensions des murs, tableaux ou autre. Un algorithme retournant une solution non cohérente est susceptible de ne recevoir aucun point.

Conseils

Voici quelques conseils pour le mener le devoir à bien :

1. Tirez le meilleur parti des séances de laboratoire encadrées afin de demander des conseils.
2. Inspirez vous des techniques vues au cours, et ajoutez-y vos propres idées.
3. Procédez par étape. Réfléchissez d'abord sur papier à une approche, implémentez une recherche locale simple, puis améliorez la avec vos différentes idées.

4. Tenez compte du fait que l'exécution et le paramétrage de vos algorithmes peut demander un temps considérable. Dès lors, ne vous prenez pas à la dernière minute pour réaliser vos expériences.

Remise

Vous remettrez sur Moodle une archive zip nommée `matricule1_matricule2_Devoir1` contenant :

— Votre code commenté au complet.
— Un dossier `solutions` avec vos solutions au format attendu par Célestin (décrit plus haut) respectant la structure `solutions/<instance_name>.txt`.
— Votre rapport de présentation de votre méthode et de vos résultats, d'au maximum 3 pages.