Devoir 3 - Planification de la production alimentaire

Remise le 21/04/25 (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_Devoir3.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.

Énoncé du devoir

Après avoir impressionné TONDA par le rendement que peut créer votre expertise, de nouveaux industriels de la grande distribution tentent de vous contacter. Vous faites ainsi la connaissance de Marco, un magna de la restauration italienne et plus particulièrement des pizzérias. Ayant fait succès dans la restauration de haute qualité grâce à une recette unique et un réseau de distribution bien organisé, il fait maintenant appel à vous pour diversifier sa clientèle. Après plusieurs investissements faramineux, il lui est possible de produire en masse ses recettes de pizzas dans des usines pour les distribuer en grandes surfaces sous la forme de produits surgelés. Les usines ne possèdent malheureusement qu'une seule ligne de production, mais elles peuvent néanmoins être reparamétrées chaque jour pour créer de nouveaux types de produits. Votre but sera de planifier une séquence S de configurations de cette ligne pour pouvoir répondre aux demandes d'approvisionnement des magasins tout en minimisant les coûts engendrés par le stockage ou les transitions entre les configurations.

Réponse à la demande

Votre planification pour une usine se déroule sur un nombre de jours J. Vous avez également à votre disposition, un ensemble de C configurations possibles pour la ligne de production. Chacune de ces configurations est associée à un produit qui vous permettra de satisfaire la demande de vos clients. À titre d'exemple, une configuration permettra de produire des pizzas aux champignons, une autre des pizzas Margherita, etc. Vous avez la connaissance de toutes les demandes journalières qui doivent être satisfaites sur un horizon temporel défini. Pour chaque jour $j \in J$, la demande est un vecteur $d_j = \langle d_{j,1}, \ldots, d_{j,C} \rangle$ avec $d_{j,c} = 1$ lorsqu'un lot du produit correspondant à la configuration $c \in C$ est requis ce jour-là et $d_{j,c} = 0$ sinon. Un lot de produit pourrait par exemple être une quantité de 1000 pizzas Margherita. Une séquence S de production est définie comme une séquence ordonnée de configurations $c \in C \cup \{-1\}$ et de taille J. La valeur -1 indique que rien n'est produit lors du jour $j \in J$ (i.e., aucune configuration n'est mise). Intuitivement, il s'agit du nombre de lots que vous avez produits et qui sont accumulés pour une utilisation future. Finalement, on définit $n_c^S \in \mathbb{N}$ comme étant le nombre d'unités du produit c générées sur toute la période de planification selon la séquence S. À partir de là, deux contraintes doivent être respectées pour que votre séquencement (i.e., une solution au problème) soit valide :

1. Le nombre de produits conçus en usine doit être strictement égal au nombre de demandes sur l'horizon de planification.

$$\sum_{j \in J} d_{j,c} = n_c^S \quad \forall c \in C. \tag{1}$$

2. La production ne doit pas être en retard sur la demande.

$$d_{j,c} \leq a_j^S \quad \forall j \in J \wedge \forall c \in C. \tag{2}$$

Coûts d'opération et de stockage

Malheureusement, les opérations vous permettant de moduler votre ligne de production ont chacune un coût respectif. En effet, la modulation peut nécessiter un changement des matières premières utilisées, la nécessité de modifier les machines utilisées ou encore un nettoyage complet de la ligne. En ce sens, les coûts de modulation entre toutes les configurations ne sont donc pas les mêmes. On notera donc $T_{c,c'}$, le coût de transition d'une configuration $c \in C$ vers une transition $c' \in C$. Notez que $T_{c,c'}$ et $T_{c',c}$ ne sont pas forcément égaux et que si $c = c'$ alors $T_{c,c'} = 0$. On définit $p_j^S \in C$ comme la configuration précédant celle du jour $j \in J$ sur la séquence S. Pour le premier jour, on considère que la configuration précédente est identique. En plus de la modulation, stocker les produits avant leur jour de demande possède un coût de stockage H (en pratique, le coût peut-être lié à l'utilisation de congélateurs supplémentaires et à l'énergie requise pour leur fonctionnement). Si une unité n'est pas consommée le jour de sa production, il faut alors la stocker, ce qui engendre ce coût. Le coût est le même pour toutes les unités de chaque produit. On peut donc quantifier le coût de stockage d'un jour $j \in J$ sur une séquence S par l'expression suivante.

$$H = \sum_{c \in C} \max\left(0, a_{j,c}^S - d_{j,c}\right). \tag{3}$$

Votre mission

Votre mission sera donc de trouver une séquence S satisfaisant toutes les contraintes et minimisant le coût de production total, avec c_j^S étant la configuration utilisée le jour j dans la séquence S.

$$\min_S \left(\sum_{j \in J} T_{p_j^S, c_j^S} + H \cdot \left[\sum_{c \in C} \max\left(0, a_{j,c}^S - d_{j,c}\right) \right] \right) \tag{4}$$

Format des instances

Chaque instance est nommée selon le format suivant : `instance_<instance_letter>_<J>_<C>_<H>.txt`. Les fichiers contenant les instances du problème sont organisés en $3 + 2 * C$ lignes comme ci-dessous.

```
J
C
d_0^1 d_0^2 d_0^3 d_0^4 ... d_0^{J}
d_1^1 d_1^2 d_1^3 d_1^4 ... d_1^{J}
...
o_{C-1}^1 o_{C-1}^2 o_{C-1}^3 ... o_{C-1}^{J}
H
T^{1,0} T^{1,1} T^{1,2} T^{1,3} ... T^{1,C}
T^{2,0} T^{2,1} T^{2,2} T^{2,3} ... T^{2,C}
...
T^{C,0} T^{C,1} T^{C,2} T^{C,3} ... T^{C,C}
```

La première ligne contient le nombre J de jours pour faire le planning. La seconde ligne contient le nombre C de produits, et donc de configurations, possibles. Les C lignes suivantes contiennent J entiers et décrivent les demandes journalières de l'horizon de planification. La ligne suivante contient le coût de stockage H. Pour finir, les C dernières lignes contiennent C entiers, décrivant le coût de transition pour un type à un autre.

Ainsi, dans l'instance supplémentaire `trivial.txt` affichée ci-dessous, l'horizon est de 15 jours et 5 configurations sont possibles. Lors de la huitième journée, il est demandé de livrer les produits liés à la première et la quatrième configuration. Moduler la ligne de production de la deuxième à la cinquième configuration provoque un coût de 167.

Une représentation graphique de l'instance est proposée dans la figure 1. La première frise chronologique met en évidence les demandes de livraisons pour chaque jour de l'horizon de planification avec une couleur attribuée à chaque type de produits. La matrice indique les coûts de transition d'une configuration à l'autre et gradue les couleurs des cellules selon la valeur du coût. L'index de la ligne correspond à la configuration initiale et celui de la colonne correspond à la configuration ciblée par la transition.

```
15
5
0 0 0 0 0 0 0 1 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0 0 0 0 1 0 0 1
0 0 0 0 0 0 1 0 0 0 0 1 0 1 0
0 0 0 0 0 0 0 1 0 0 1 0 0 0 1
0 0 0 0 0 0 0 0 1 0 0 1 0 0 1
10
0 105 154 130 100
146 0 135 139 167
101 183 0 193 113
188 112 111 0 103
179 117 161 124 0
```

Format des solutions

Le format d'une solution attendue comporte une seule ligne contenant les configurations journalières de la séquence S. Si un jour n'a pas de production, la valeur -1 est indiquée. Une fonction `save_solution()` vous est fournie pour générer ce fichier.

```
t_1 t_2 ... t_{J}
```

Exemple illustratif

À titre d'exemple, voici la solution retournée par le solveur naïf sur l'instance triviale.

```
1 2 0 3 4 3 1 2 4 0 2 1 3 4 -1
```

Une illustration de cette solution est donnée à la Figure 2. Le solveur ajoute toutes les configurations nécessaires les unes à la suite des autres, le plus tôt possible, sans prendre en compte les coûts de transition ou de stockage. Le coût total de cette solution est de 2231. Par exemple, il serait préférable de moduler la ligne de

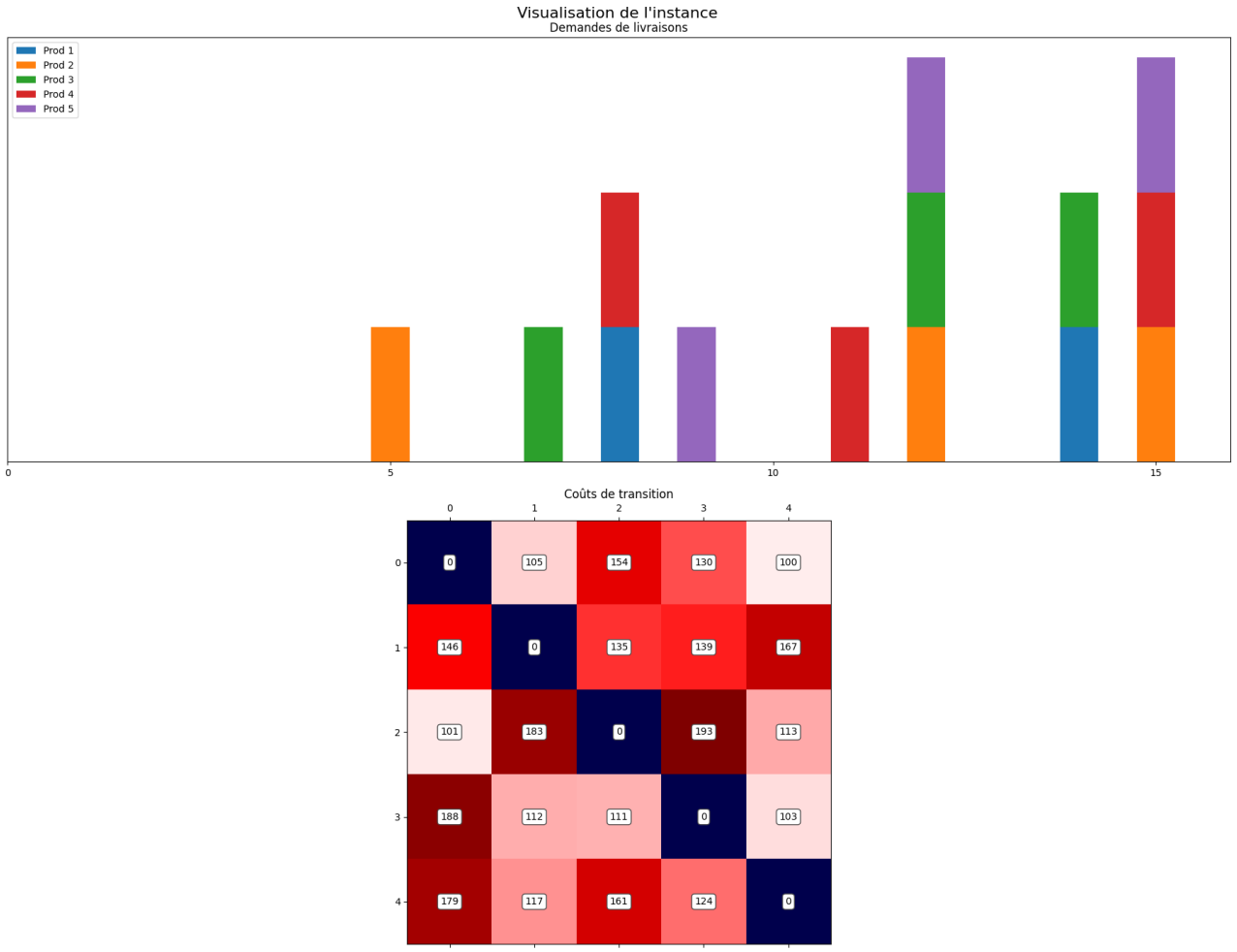

FIGURE 1 – Représentation graphique de l'instance `trivial`.

la première configuration vers la cinquième, sachant qu'il s'agit de la transition la moins coûteuse de l'instance et que les demandes des deux produits sont souvent proches. À titre de comparaison, vous pourrez voir la visualisation d'une meilleure solution sur la Figure 3 présentant un coût de 1519 grâce à un meilleur agencement des configurations. Des fonctions utilitaires vous sont données pour générer ces figures.

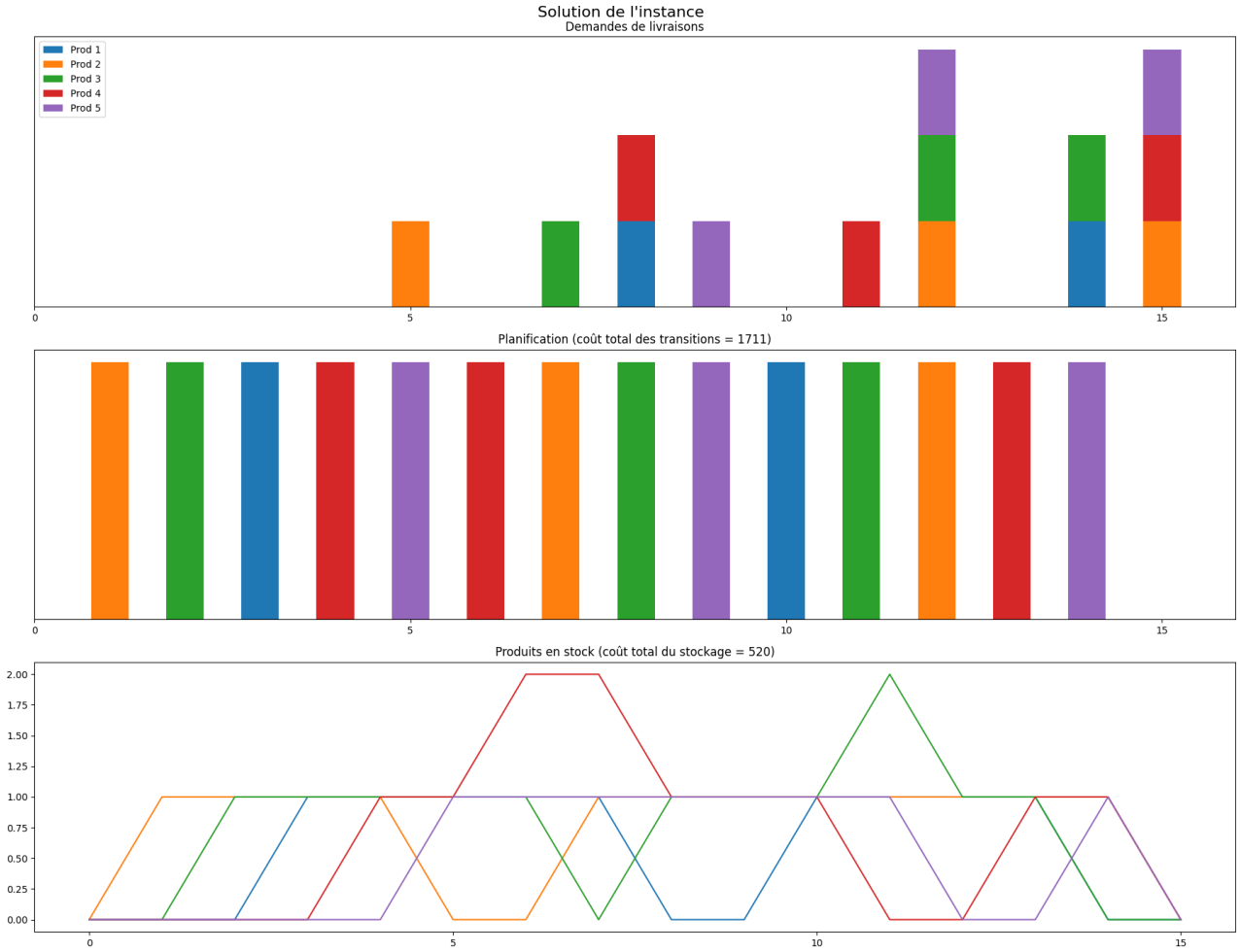

FIGURE 2 – Visualisation de la solution illustrative pour l'instance `trivial`.

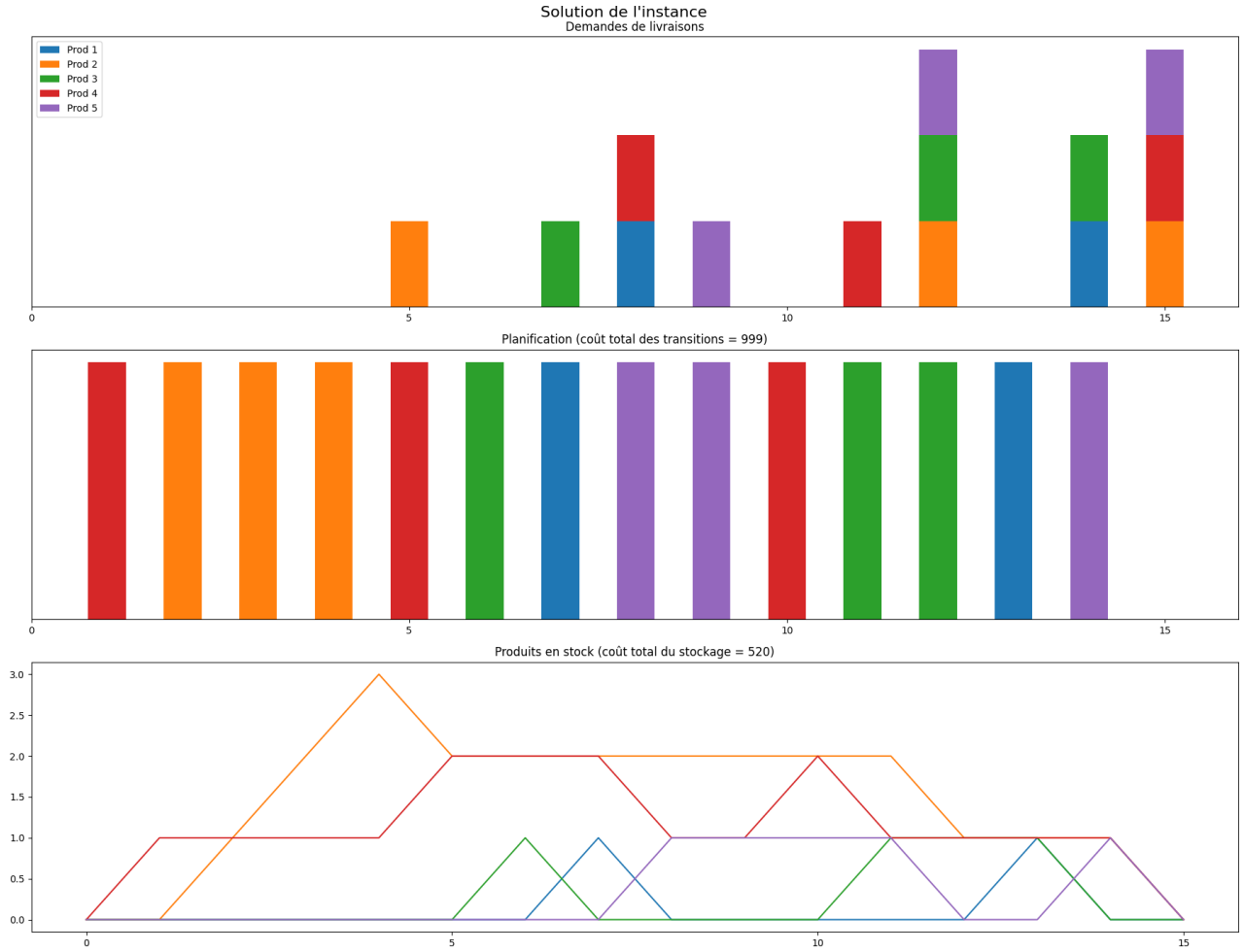

FIGURE 3 – Visualisation d'une solution plus avancée pour l'instance `trivial`.

Implémentation

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

- `requirements.txt` qui contient les librairies nécessaires au projet.
- `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 ajoutant toutes les routes après le centre tant que les contraintes le permettent.
- `solver_advanced.py` : implémentation de votre méthode de résolution qui sera évaluée pour ce devoir.
- `autograder.py` : un 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, 6 instances sont disponibles dans les fichiers `instances/<instance>.txt` en plus de l'instance triviale `instances/trivial.txt`. Il vous est demandé de produire les fichiers de solution `solutions/<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/trivial.txt
```

Un fichier `solutions/trivial.txt` décrit plus haut et une image `visualization_trivial.png` du même format que la Figure 2 seront générés.

Un environnement virtuel sous **Python 3.11** est recommandé afin d'installer les librairies du projet grâce à la commande suivante.

```
pip3 install -r requirements.txt
```

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 minimums locaux **différente de celle utilisée dans le devoir précédent**. Vous êtes ensuite libres d'apporter n'importe quelle modification pour améliorer les performances de votre solveur. Il vous est permis de reprendre votre métaheuristique précédente, mais vous devrez alors la combiner avec une autre métaheuristique. Par exemple, si vous avez réalisé une recherche Tabou, il est autorisé de faire une recherche Tabou avec un GRASP ou avec un LNS ou encore les trois ensembles. L'utilisation de librairies python externe est autorisée, mais doivent être utilisée avec parcimonie, si une librairie remplace entièrement les attendus de base, votre code ne sera pas considéré comme remplissant les exigences du devoir. Une fois construit, votre solveur pourra ensuite être appelé comme suit.

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

L'option `-no-viz` peut être ajoutée afin de ne pas avoir à générer la visualisation à chaque fois.

Un rapport **succinct** (2-3 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, et toute autre information que vous jugez pertinente. À titre d'exemple, **il est attendu que vous analysiez la complexité des fonctions principales de votre algorithme**, comme par exemple la sélection d'un voisin en fonction de la taille du voisinage. Reportez-y également les résultats obtenus pour les différentes instances. Finalement, vous devez fournir un dossier `solutions` avec vos solutions au format telles qu'écrites plus haut et en 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. Parmi ces instances, six d'entre elles vous sont fournies et une dernière restera cachée pour vérifier les capacités de généralisation de votre solveur. 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 21564, ma note pour cette instance est de $(1 - \frac{21564-LB}{UB-LB}) \approx 0.62/1$.

Le tableau ci-dessous indique toutes les caractéristiques des instances ainsi que certaines informations de l'instance cachée. La dernière colonne indique le temps de résolution maximal accordé à chaque instance.

Instance	Pondération	J	C	H	LB	UB	Temps max.
A	1/10	30	10	10	1471	2270	5 min
B	1/10	100	10	10	10340	17309	5 min
C	1/10	150	15	10	25076	38477	5 min
D	1/10	150	15	10	18098	27348	5 min
E	2/10	200	15	10	16127	22233	10 min
F	2/10	200	15	10	18289	24448	10 min
X	2/10	200	15	10	?	?	10 min

TABLE 1 – Informations sur les instances et résultats attendus pour la notation

— **10 points sur 20** sont attribués à l'évaluation de votre rapport. Sans être exhaustif, les éléments indispensables pour obtenir une bonne note sont :

1. Une formalisation et une analyse de **votre espace de recherche et de vos voisinages**. (taille, connectivité, etc.)
2. **Une quantification chiffrée et/ou argumentée** de l'impact des mécaniques implémentées (p.ex., vous échappez-vous bien des extrema locaux? Si oui, prouvez-le, sinon, 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*).

3. **Privilégiez** graphes, tableaux et formules pour transmettre l'information de manière concise.
4. **Évitez à tout prix** les explications de nature qualitatives, rapporter qu'un algorithme "va plus vite", qu'il est "plus performant" ou "donne de meilleures solutions" est flou et incomplet. À l'inverse, arriver à le prouver ou à le quantifier expérimentalement est nécessaire dans un rendu scientifique.

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. N'oubliez pas de générer à nouveau vos solutions pour évaluer une nouvelle méthode ou un changement.**

⚠**Il est attendu que vos algorithmes retournent une solution et des valeurs correctes. Par exemple, il est interdit de modifier artificiellement le nombre de nœuds, arêtes 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_Devoir2.zip` contenant :
- Votre code commenté au complet.
- Un dossier `solutions` avec vos solutions au format attendu par Marco (décrit plus haut) respectant la structure `solutions/<instance_name>.txt`.
- Votre rapport de présentation de votre méthode et de vos résultats respectant les modalités énoncées plus haut.