Devoir 1 - Création d'un horaire académique

Remise le 20/02/2022 **(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_Devoir1.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

Construire un horaire académique n'est pas une tâche aisée. Ce devoir vise à vous sensibiliser à cette difficulté, et à proposer une méthode pour en construire automatiquement. Pour établir un horaire, on doit assigner un ensemble de cours à un créneau horaire spécifique. Cependant, en raison des contraintes des professeurs et des étudiants, certains cours ne peuvent en aucun cas se chevaucher. Dans le cadre de ce projet, les incompatibilités sont indiquées par paire de cours. Par exemple, le cours INF6102 ne peut pas avoir lieu en même temps que INF8215 parce qu'il est donné par le même professeur.

L'objectif de ce devoir est de réaliser un horaire académique sans conflit, et qui utilise le moins de créneaux horaires possibles. A cette fin, vous implémenterez une méthode de recherche locale en y ajoutant un mécanisme permettant de sortir des optima locaux.

Différentes instances vous sont fournies. Elles sont nommées selon le schéma `horaire_X_N_E.txt` avec X le nom de l'instance, N le nombre de cours et E le nombre d'incompatibilités. Chaque fichier d'instance contient $E+2$ lignes et a le format suivant.

```
N
E
i_1 j_1
i_2 j_2
...
i_E j_E
```

Ainsi, la première ligne (N) indique le nombre de cours à caser, et la deuxième (E) indique le nombre de conflits existant. Chaque ligne suivante ($i\,j$) indique un conflit entre le cours i et le cours j. De plus, notez que les conflits sont mutuels : un conflit ($i \rightarrow j$) implique le conflit ($j \rightarrow i$).

Le format attendu d'une solution est un fichier de $N+3$ lignes renseignant : le nombre de cours (N), le nombre d'incompatibilités (E), le nombre de créneaux horaires utilisés dans la solution (K), et l'indication du créneau horaire utilisé pour chaque cours : (`cours,créneau`). Le format est présenté ci-dessous. Notez que la valeur K correspond au coût que l'on souhaite minimiser : on souhaite utiliser le moins de créneaux possibles. Les valeurs N et E sont les mêmes que celles du fichier d'instance, et ont principalement pour objectif de simplifier la correction.

```
N
E
K
n_1 c_1
n_2 c_2
...
n_N c_N
```

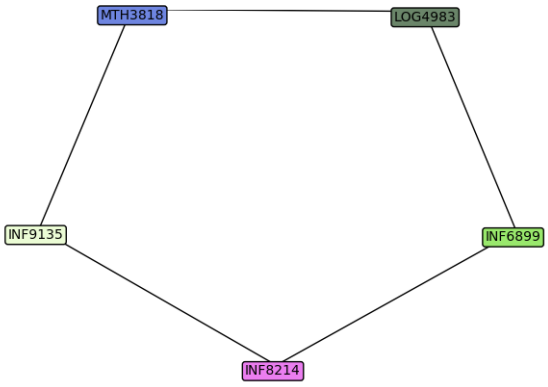
A titre d'exemple, l'instance suivante (`horaire_X_5_5.txt`) contient 5 cours et 5 conflits.

```
5
5
INF6899 LOG4983
INF6899 INF8214
MTH3818 LOG4983
MTH3818 INF9135
INF8214 INF9135
```

Une solution possible au problème est la configuration suivante, qui implique 5 créneaux horaires (résolution naïve).

```
5
5
5
INF6899 1
LOG4983 2
INF8214 3
MTH3818 4
INF9135 5
```

Pour faciliter la lecture d'une solution, un outil de visualisation vous est fourni : chaque couleur correspond à un créneau utilisé, chaque nœud à un cours, et chaque conflit par une arête entre deux cours.



Implémentation

Vous avez à votre disposition un projet python. Dans ce projet, une `solution` est un dictionnaire de taille N où les clés sont les cours auxquels sont associé les périodes de temps qui leur sont assignées. Plusieurs fichiers vous sont fournis :

- `schedule.py` qui implémente la classe `Schedule` pour lire les instances, construire et stocker vos solutions
- `main.py` vous permettant d'exécuter votre code sur une instance donnée. Ce programme stocke 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 assigne un créneau différent à chaque cours.
- `solver_advanced.py` : implémentation de votre méthode de résolution pour ce devoir.
- `courses.txt` : qui contient la liste des cours que vous serez susceptibles de traiter.

Vous êtes également libres de rajouter d'autres fichiers au besoin. De plus, 5 instances sont mises à votre disposition :

- `horaire_A_11_20.txt` : pour tester votre implémentation (ne rapporte aucun point)
- `horaire_B_23_71.txt` : utilisé pour évaluer votre solveur (optimum connu égal à 4)
- `horaire_C_169_3328.txt` : utilisé pour évaluer votre solveur (optimum connu égal à 13)
- `horaire_D_184_3916.txt` : utilisé pour évaluer votre solveur (optimum inconnu)
- `horaire_E_645_13979.txt` : utilisé pour évaluer votre solveur (optimum inconnu)

Votre code sera également évalué sur une instance cachée (`horaire_X`), de taille légèrement inférieure à la dernière. Pour vérifier que tout fonctionne bien, vous pouvez exécuter le solveur naïf comme suit.

```
python3 main.py --agent=naive --infile=instances/horaire_A_11_20.txt
```

Un fichier `solution.txt` et une image `visualization.png` 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 un mécanisme de votre choix pour vous s'échapper des minima locaux. 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/horaire_A_11_20.txt
```

Un rapport succinct (2 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, et reportez les résultats obtenus pour les différentes instances.

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 de votre solveur. L'instance B rapporte 1 point si l'optimum est trouvé, et 0 sinon. L'instance C rapporte 2 points si l'optimum est trouvé et diminue progressivement jusqu'à 0 en fonction de la qualité de la solution. Les instances D, E rapportent 2 points si l'optimum est obtenu et diminue progressivement jusqu'à 0 en fonction de la qualité de la solution. Si aucun groupe ne trouve l'optimum pour ces instances, la solution du meilleur groupe est prise comme référence des 2 points. L'instance cachée (X) rapporte entre 0 et 2 points en fonction d'un seuil raisonnable défini par le chargé de laboratoire. Le temps d'exécution est de 20 minutes par instance. Finalement, 1 point est consacré à l'appréciation générale de votre implémentation (bonne construction, commentaires, etc).
— 10 points sur 20 sont attribués pour le rapport. Pour ce dernier, les critères sont la qualité générale, la qualité des explications, et le détail des choix de conception.
— 2 points bonus seront attribués au groupe ayant le meilleur résultat pour l'instance cachée (X).

⚠Il est attendu que vos algorithmes retournent une solution et un coût correct. Par exemple, il est interdit de modifier artificiellement les contraintes entre les noeuds ou autre. Un algorithme retournant une solution non cohérente est susceptible de 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. Tenez compte du fait que l'exécution de vos algorithmes peut demander un temps considérable. Dès lors, ne vous prenez pas à la dernière minute pour réaliser vos expériences.
4. Bien que court dans l'absolu, prenez garde au temps d'exécution. Exécuter votre algorithme sur les 4 instances données prend 1h20. Organisez au mieux votre temps de développement/évaluation.

Remise

Vous remettrez sur Moodle une archive zip nommée `matricule1_matricule2_Devoir1` contenant :
— Votre code commenté au complet
— Vos solutions aux différentes instances nommées `solutionX` où X est le nom de l'instance
— Votre rapport de présentation de votre méthode et de vos résultats, d'au maximum 2 pages.