Devoir 1 - Étude de marché

Remise le 02/03/25 (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_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.

Énoncé du devoir

Votre ancienne amie du secondaire Jia reprend contact avec vous afin de vous faire part d'une proposition que vous ne pouvez refuser. Elle aurait mis la main sur plusieurs jeux de données représentant chacun une population dans une région et les liens sociaux entre chacun des individus qui la composent (font-ils du sport ensemble, ont-ils des loisirs en commun, le même travail, etc.). Selon elle, en l'utilisant dans le cadre d'une analyse de marché, vous pourriez obtenir des regroupements selon leurs similitudes et vendre cela extrêmement cher à toutes sortes de jeunes entrepreneurs.

Chacune des données prend la forme d'un graphe non orienté et sans boucles $G = (V, E)$, avec V l'ensemble des nœuds de taille N et E l'ensemble des arêtes de taille M. On considère chaque arête de relation comme ayant le même poids. Vous mettrez donc à disposition vos connaissances en résolution de problèmes combinatoire pour déterminer, sur chacune des régions, les groupements de populations ayant le plus de cohésions entre eux. Plus la qualité de vos regroupements sera fiable, plus vous pourrez négocier à la hausse et vous en mettre plein les fouilles.

Votre mission

Vous allez devoir trouver, pour chaque graphe de relations, un ensemble de regroupements C dont les éléments $c \in C$ sont des listes d'individus $e \in E$. Votre ensemble de regroupement doit toujours respecter les propriétés suivantes :

1. Chaque individu doit faire partie d'un seul et unique regroupement. Formellement, on a donc pour chaque paire $(c, c') \in C$ avec $c \neq c'$ la contrainte $c \cap c' = \emptyset$.
2. Hormis les regroupements de type singleton qui ne comprennent qu'un seul individu, un individu doit forcément avoir un lien avec une personne de l'ensemble auquel il est attribué. Formellement, on a donc $\forall c \in C, \forall e_i \in c, \exists e_j \in c \setminus e_i$ tel que $(e_i, e_j) \in V$.

⚠**En suivant strictement ces deux propriétés, une solution où un individu ne serait pas représenté dans un regroupement sera jugée invalide.**

Comme expliqué, votre objectif sera de **maximiser le coefficient de cohésion des solutions de chacune de vos populations**. Cette valeur est comprise dans l'intervalle $[-1, 1]$. Une valeur haute indique que la

densité de connexion est forte entre les individus d'un même regroupement, tandis qu'une valeur basse et négative indique que cette densité est plus forte entre les individus de regroupement différents. La qualité de cohésion Q sera donc donnée par cette formule :

$$Q = \frac{1}{2M} \sum_{i=1}^{N} \sum_{j=1}^{N} \delta_{c_i c_j} \left(A_{ij} - \frac{k_i k_j}{2M} \right) \tag{1}$$

On note k_i le degré d'un nœud i (i.e., le nombre d'arêtes liées au nœud), A_{ij} la matrice d'adjacence du graphe, c_i le regroupement auquel appartient le nœud i et $\delta_{c_i c_j}$ l'indicateur de *Kronecker*. Ce dernier est défini comme suit :

$$\delta_{ij} = \begin{cases} 1 & \text{si } c_i = c_j \\ 0 & \text{sinon} \end{cases} \tag{2}$$

L'intuition derrière cette formule est de comptabiliser la fraction d'arêtes liées à chaque paire de nœuds d'un même groupement par rapport au nombre total d'arêtes dans le graphe. Sachant que, selon la forme de la somme, ce score sera pénalisé si deux nœuds i et j font partie d'un même groupe, mais n'ont aucun lien d'adjacence ($A_{ij} = 0$). Votre score n'augmentera donc pas nécessairement grâce à la taille de vos groupes, mais plutôt par la densité de lien entre eux. Pour vous donner une idée d'évolution de la cohésion selon la qualité d'un groupement, vous pouvez observer les exemples des figures 1 et 2 dans les sections suivantes.

Format des instances

Chaque population à analyser est inscrite dans un fichier d'instance de $N+1$ lignes correspondant au format suivant.

```
N M
<a_1_1> ... <a_1_i>
<a_2_1> ... <a_2_j>
...
<a_n_1> ... <a_n_k>
```

La première ligne du document fourni par Jia indique les variables globales du problème, soit le nombre d'individus et le nombre de liens au total. En suivant, les N lignes suivantes indiqueront les nœuds ayant un lien avec celui de la ligne courante. Ainsi, dans l'instance supplémentaire `trivial_1.txt` affichée ci-dessous, le nœud 3 est en lien avec les nœuds 1, 2, 4, 5 et 6. Remarquez que la dernière ligne est vide, cela veut dire que le nœud 8 n'a aucun lien avec un autre nœud. Remarquez également que les liens sont symétriques. Par exemple, les nœuds 1, 2, 4, 5 et 6 sont également en relation avec le nœud 3.

```
8 10
2 3
1 3 4
1 2 4 5 6
2 3 5 7
3 4 6
3 5
4

```

Format des solutions

Il vous est demandé de fournir pour chaque configuration un fichier de solution. Le fichier doit contenir $L+1$ lignes. La première ligne indique le nombre de regroupements de votre solution ($|C|$), et les lignes suivantes indiquent les nœuds présents dans chaque regroupement. Une fonction utilitaire `generate_file()` vous est fournie pour générer ce fichier.

```
<|C|>
<group_1_1>    ...    <group_1_i>
<group_2_1>    ...    <group_2_j>
...
<group_l_1>    ...    <group_l_k>
```

Exemple illustratif

À titre d'exemple, voici la solution qu'un solveur naïf pourrait retourner sur l'instance `trivial_1.txt`. Ici, le solveur attribue tous les individus à un ensemble singleton afin de respecter les propriétés requises.

```
8
1
2
3
4
5
6
7
8
```

Une illustration de cette solution est donnée à la Figure 1. Sur cette figure, vous trouverez également le coefficient de cohésion, qui s'améliorera avec la qualité de vos regroupements. À gauche, la figure montre le graphe de l'instance avec une coloration dépendante de votre solution. À droite, les nœuds gardent la même coloration, mais sont en plus regroupés en fonction de leur groupe assigné par la solution. Dans cette représentation, chaque groupe est considéré comme un nœud d'un supergraphe circulaire permettant de mettre en relief le nombre de groupes se trouvant dans la solution et le nombre de liens entre chaque paire de groupe. À titre de comparaison, vous pourrez voir la visualisation d'une meilleure solution sur la Figure 2. Remarquez également l'évolution de la cohésion entre ces deux solutions. À présent, les nœuds d'une même couleur sont rapprochés sur la figure de droite. Des fonctions utilitaires vous sont données pour générer ces figures.

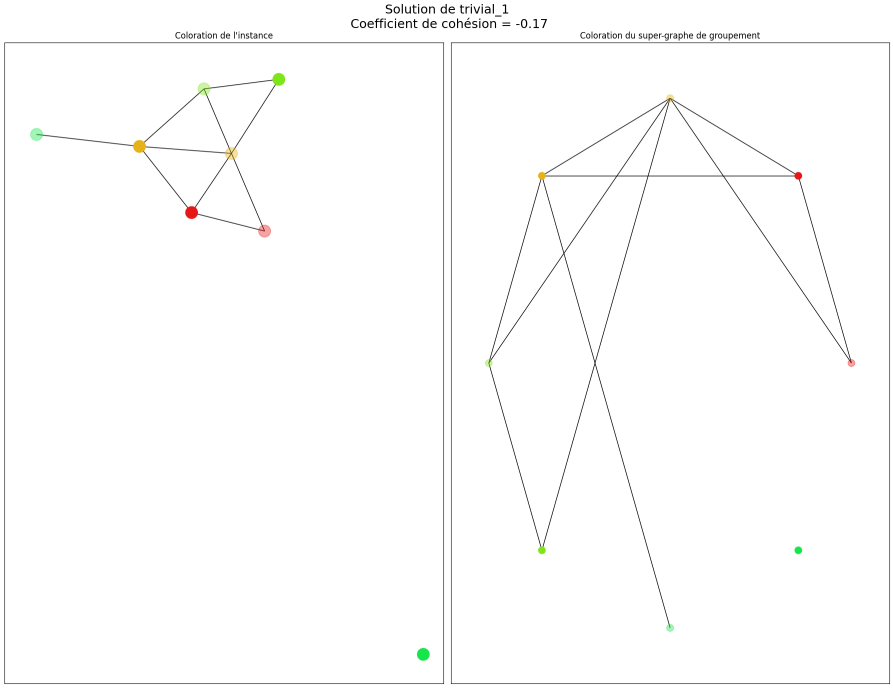

FIGURE 1 – Visualisation de la solution illustrative pour l'instance `trivial_1`.

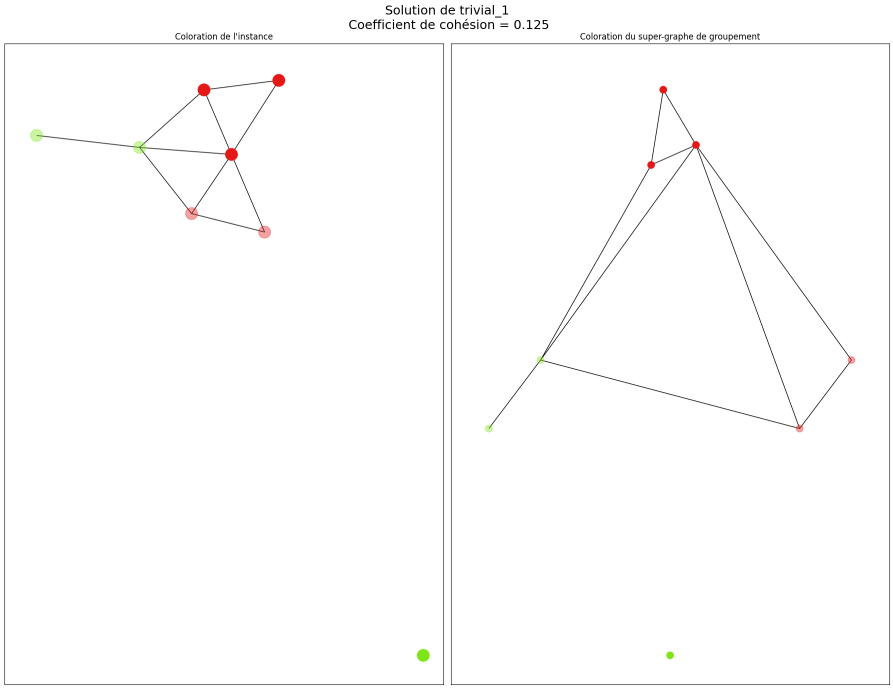

FIGURE 2 – Visualisation d'une solution plus avancée pour l'instance `trivial_1`.

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 qui réserve un regroupement par individus.
- `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, 15 instances sont disponibles dans les fichiers `instances/<instance>.txt` en plus des instances triviales `instances/trivial_1.txt` et `instances/trivial_2.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_1.txt
```

Un fichier `solutions/trivial_1.txt` décrit plus haut et une image `visualization_trivial_1.png` du même format que la Figure 1 seront générés. À partir des instances de plus de 4500 nœuds, la visualisation est désactivée, car elle demande beaucoup de temps. Si vous le souhaitez, vous pouvez le changer grâce à un paramètre du fichier `main.py` comme suit.

```
python3 main.py --agent=naive --infile=./instances/trivial_1.txt --viz-node-limit
=8000
```

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 devrez également intégrer un mécanisme de votre choix pour que votre solveur s'échappe des minimas locaux. Vous êtes ensuite libre d'apporter n'importe quelle modification pour améliorer vos performances. 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_1.txt
```

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. 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, 15 d'entre elles vous sont fournies, et 5 resteront cachées pour vérifier les capacités de généralisation de votre solveur. Vous commencez avec 10 points et **vous perdez 0.5 points** pour chaque instance en dessous du seuil de qualité défini dans le temps indiqué ou que votre solution est invalide. Le tableau ci-dessous liste les valeurs à atteindre pour obtenir la note maximale si vous ne décevez pas votre amie sur les instances secrètes. Votre algorithme doit fournir une solution sous la limite de temps visible dans ces mêmes tableaux (5 minutes seront accordées pour chaque instance cachée). Les instances cachées comportent entre 77 et 22963 nœuds et ont entre 254 et 48436 liens.

Instance (max 3min)	Nombre de nœuds	Nombre d'arêtes	**Seuil de qualité demandé**
instanceA	34	78	**0.39**
instanceB	105	441	**0.51**
instanceC	198	2742	**0.44**
instanceD	297	2148	**0.375**
instanceE	453	2025	**0.43**
instanceF	1133	5451	**0.54**
instanceG	1024	3056	**0.78**
instanceH	2038	6033	**0.84**
instanceI	4096	12264	**0.85**
instanceJ	4941	6594	**0.92**
instanceK	8361	15751	**0.84**
instanceL	8192	28229	**0.95**
instanceM	8192	24547	**0.87**
instanceN	10680	24316	**0.86**
instanceO	16726	47594	**0.82**

— **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** (p.ex., est-il connecté?).
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, 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*).

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", est "plus performant" ou "donne de meilleures solutions" est flou et incomplet. A l'inverse, arriver à le prouver où 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_Devoir1.zip` 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 respectant les modalités énoncées plus haut.