Devoir 2 - Optimisation du placement de micro-puces

Remise le 20/03/2022 **(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 la soumission sur Moodle, donnez votre rapport en **PDF** ainsi que votre code **commentés** 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.

Énoncé du devoir

Dans un ordinateur, une carte-mère est un circuit imprimé sur lequel viennent se fixer toute une série de composants électroniques (connecteurs de cartes, micro-processeurs, mémoire centrale, carte graphique, etc.). Ces composants échangent entre eux une certaine quantité d'information, que l'on nommera *flux*. Idéalement, on souhaite que ce transfert d'information se réalise rapidement. Cette rapidité de transfert du flux entre deux composants est dépendante de la *distance* entre l'emplacement sur la carte mère où sont placés les deux composants. Ainsi, plus les composants sont éloignés, plus le transfert sera long.

L'objectif de ce devoir est de proposer une manière de disposer l'entièreté des composants électroniques requis sur une carte mère afin de minimiser la vitesse de transmission totale du flux d'information. Posons les paramètres suivants :

- I : l'ensemble des composants devant être placés sur la carte mère
- J : l'ensemble des zones de la carte mère où peuvent être placés les composants. Notons que ce nombre est identique au nombre de composants.
- $f(i_1, i_2)$: le flux d'information entre les composants $i_1 \in I$ et $i_2 \in I$.
- $d(j_1, j_2)$: la distance entre les sites $j_1 \in J$ et $j_2 \in J$.

Pour chaque zone d'emplacement $j \in J$, on introduit une variable de décision $x_j \in I$ indiquant le composant $i \in I$ disposé à l'emplacement j. Le problème peut être formalisé mathématiquement comme suit.

$$\begin{aligned} \text{minimize} \quad & \sum_{j_1 \in J} \sum_{j_2 \in J} f(x_{j_1}, x_{j_2}) \times d(j_1, j_2) \\ \text{subject to} \quad & \texttt{allDifferent}(x) \\ & x_j \in I \qquad\qquad \forall j \in J \end{aligned}$$

Prêtez bien attention à la fonction objectif. On considère l'ensemble des zones d'emplacement, et on minimise la somme des distances pondérées par le flux d'information entre les composants situés à ces positions. La contrainte indique que chaque emplacement doit contenir un composant différent.

Différentes instances vous sont fournies. Elles sont nommées selon le schéma `mother_X_N.txt` avec X le nom de l'instance, et N le nombre de composants devant être placés. Chaque fichier d'instance contient $2N+3$ lignes et a le format suivant.

```
N

i(1,1)    i(1,2)    ...    i(1,N)
i(2,1)    i(2,2)    ...    i(2,N)
...
i(N,1)    i(N,2)    ...    i(N,N)

j(1,1)    j(1,2)    ...    j(1,N)
j(2,1)    j(2,2)    ...    j(2,N)
...
j(N,1)    j(N,2)    ...    j(N,N)
```

Ainsi, la première ligne (N) indique le nombre de composants à caser, les N lignes suivantes forment une matrice indiquant les flux entre chaque paire de composants, et les N dernières lignes forment une matrice indiquant les distances entre chaque paire d'emplacements. De plus, notez que ces matrices **ne sont pas symétriques**. Il s'agit d'une propriété dont vous devez tenir compte dans l'élaboration de votre algorithme de résolution.

Le format attendu d'une solution est un fichier de 3 lignes renseignant : le nombre de composants (N), le coût total de la solution (C) selon la fonction objectif, et un vecteur indiquant le composant mis à chaque emplacement ($x_1, x_2, \ldots, x_N$). La valeur N est la même que celles du fichier d'instance, et a principalement pour objectif de simplifier la correction.

```
N
C
x_1   x_2    ...    x_N
```

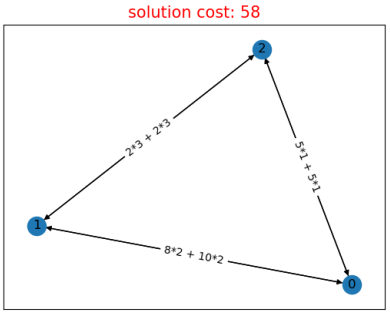
A titre d'exemple, l'instance suivante (`mother_X_3.txt`) contient 3 composants et 3 localisations.

```
3

0 1 2
1 0 3
2 3 0

0 10 5
8 0  2
5 2  0
```

Une solution possible au problème est la configuration suivante, qui indique de placer le composant i_0 à l'emplacement j_0, le composant i_2 à l'emplacement j_1, et le composant i_1 à l'emplacement j_2. Cette solution a un coût de $(1 \times 5 + 1 \times 5) + (2 \times 10 + 2 \times 8) + (3 \times 2 + 3 \times 2) = 58$. Notez que la numérotation commence avec l'indice 0.

```
3
58
0 2 1
```

Pour faciliter la lecture d'une solution, un outil de visualisation vous est fourni : chaque noeud correspond à un emplacement, le label du noeud correspond au composant qui a été placé, et chaque arête par sa contribution (`flux` × `distance`) sur le coût de transmission.



Implémentation

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

- `mother_card.py` qui implémente la classe `MotherCard` 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` qui implémente une résolution aléatoire du problème.
- `solver_advanced.py` qui implémente votre méthode de résolution du problème.

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

- `mother_A_5.txt` d'optimum connu égal à 58. Cette instance ne rapporte aucun point et est donnée pour réaliser vos tests.
- `mother_B_20.txt` d'optimum connu égal à 3683.
- `mother_C_30.txt` d'optimum connu égal à 13178.
- `mother_D_70.txt`
- `mother_E_80.txt`

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

```
python3 main.py --agent=naive --infile=instances/mother_A_5.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 amélioré par au moins une des métaheuristiques suivantes : *recherche tabou*, *algorithmes génétiques*, ou *GRASP*. Vous êtes libres de choisir la métaheuristique que vous souhaitez implémenter. Comme pour le premier devoir, réfléchissez bien à la définition de votre espace de recherche, de votre voisinage, de la fonction de sélection et d'évaluation. Vous êtes ensuite libres d'apporter n'importe quelle modification pour améliorer les performances de votre solveur, par exemple en combinant votre approche avec une autre métaheuristique. Une fois construit, votre solveur pourra ensuite être appelé comme suit.

```
python3 main.py --agent=advanced --infile=instances/mother_A_5.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 conception, et reporter 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. 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 et d'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.