Devoir 3 - Danse finale

Remise le 15/04/24 — 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_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.

Enoncé du devoir

Grâce à vous, Célestin et Marco sont devenus richissimes, leur objectif est à présent de devenir célèbres! Dans cette optique ils ont tous deux investi dans une compagnie de danseurs brésiliens en perte de vitesse. La troupe était au bord de la faillite à cause de la dépression nerveuse de ses danseurs.

Au cours d'un spectacle, chaque danseur doit enchainer les chorégraphies et dépendamment de la composition de ces dernières, certains d'entre eux doivent rapidement enfiler de nouvelles tenues. Cet évènement stressant est à éviter le plus possible pour assurer le bon déroulement de la performance.

Célestin et Marco font une dernière fois appel à vous pour les aider à améliorer les conditions de travail de leurs danseurs. Ils vous fournissent les plans des spectacles avec une liste reprenant les couleurs des tenues pour chaque chorégraphie. Votre objectif est de déterminer dans quel ordre ordonner les chorégraphies pour minimiser le nombre de fois que les danseurs doivent enfiler de nouvelles tenues.

De manière absolue, Célestin et Marco vous demandent de leur remettre une feuille indiquant quelles tenues sont portées pendant chaque chorégraphie. Certaines chorégraphies ne requièrent pas tous les danseurs, un danseur peut garder sa tenue et se cacher en coulisse pendant la performance de ses camarades. De cette manière il n'a pas à l'enfiler à nouveau si il doit remonter sur scène plus tard avec la même tenue.

Format des instances

Formellement, on peut résumer chaque spectacle par une matrice binaire $C \in \{0,1\}^{N \times M}$ avec N le nombre de tenues et M le nombre de chorégraphies. L'élément $C_{i,j}$ vaut 1 si la tenue i doit être portée dans la chorégraphie j. On pose également D comme étant le nombre de danseurs. Les fichiers d'instances spécifient M, N, D et ensuite la matrice C.

```
M
N
D
c(1,1) ... c(1,M)
...
c(N,1) ... c(N,M)
```

Format des solutions

De manière similaire, une solution est une matrice binaire $K \in \{0,1\}^{N \times M}$ où chaque élément $k_{i,j}$ vaut 1 si la tenue i est portée, sur scène ou en coulisses, lors de la j-ème chorégraphie de votre solution. Notez bien que la j-ème chorégraphie de votre solution ne correspond pas forcément à la j-ème chorégraphie du fichier d'entrée. La validité d'une solution tient à deux critères :

- Le nombre de danseurs est respecté en tout temps : $\forall j \in M : (\sum_{i=1}^{N} k_{i,j}) \leq D$
- A chaque chorégraphie $c_{*,x}$ dans C on peut faire correspondre une colonne $k_{*,y}$ de K différente telle que $(c_{*,x} \wedge k_{*,y}) = c_{*,x}$ où $\wedge$ est l'opérateur et logique appliqué élément par élément.

En d'autres mots, il doit être possible de créer M paires *distinctes* $(c_{*,x}, k_{*,y})$ de sorte à ce que les éléments positifs (= 1) de $c_{*,x}$ soient aussi positifs dans $k_{*,y}$ pour toutes les paires.

```
k(1,1), ...,k(1,M)
...
k(N,1), ...,k(N,M)
```

Exemple illustratif

A titre d'exemple, vous avez la situation suivante avec 3 danseurs, 5 chorégraphies et 4 costumes. Chaque colonne correspond à une chorégraphie, et chaque 1 indique qu'un costume spécifique est nécessaire pour la chorégraphie. Notez également, que chaque colonne a au plus 3 costumes (borne supérieure sur le nombre de danseurs).

```
5
4
3
1 0 1 1 0
0 1 0 0 1
1 1 0 0 1
1 0 0 1 1
```

Une solution possible est proposée ci dessous, et également illustrée dans la Figure 1.

```
1,0,1,0,1
0,1,1,1,0
1,1,0,1,0
1,0,0,1,1
```

Il y a plusieurs choses à noter dans cette solution :

- Il est possible de retrouver toutes les chorégraphies demandées par Célestin et Marco. Les trois premières sont dans l'ordre dans lesquelles elles ont été données, les deux dernières sont inversées dans la solution.
- On remarque que pour ménager l'un des danseurs, on autorise l'un d'entre eux à porter le costume 2 pendant la chorégraphie #3 même si ce costume ne figure pas dans la composition (la 3e colonne de l'input $(1,0,0,0)$ devient $1,1,0,0$). Ce n'est pas grave, le danseur pourra se cacher en coulisses
- On remarque que l'on impose un stress inutile pour un danseur. On lui demande en effet d'enlever sa tenue pour les chorégraphies #2 et #3 durant lesquelles il ne danse pas alors qu'il doit la remettre pour les chorégraphies #4 et #5 (dans la solution, on pourrait transformer la dernière ligne en $1,1,1,1,1$

pour remédier à ce problème).

— La grille '*stage composition*' vous permet de voir quelles tenues sont portées pendant chaque chorégraphie pendant les danseurs. Il faut surtout faire attention à ne pas avoir plus de costumes utilisés que de nombre de danseur par chorégraphie (que ce soit sur scène ou en coulisse).

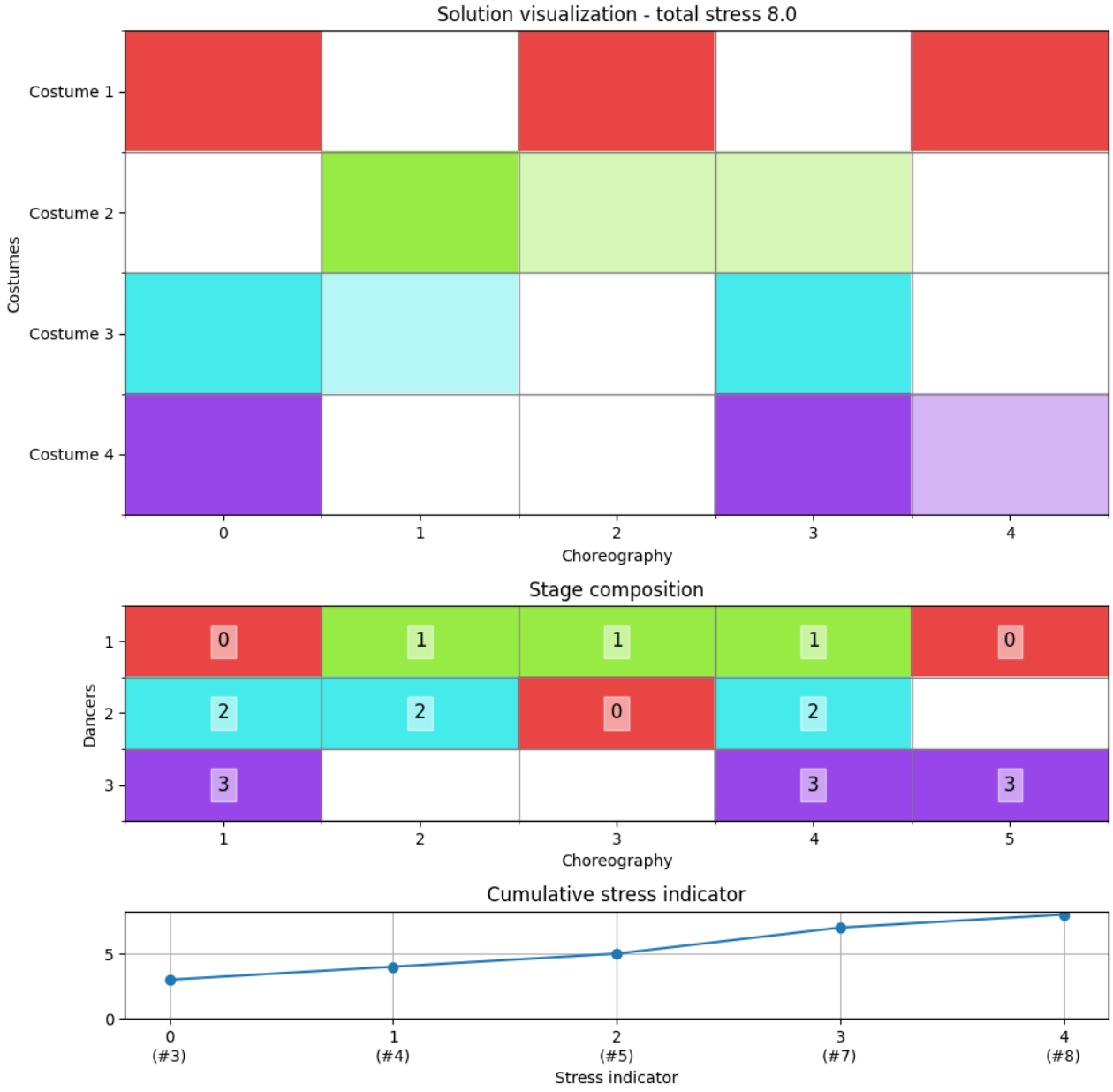

FIGURE 1 – Exemple de solution pour l'instance *toy.txt*

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 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.

Afin de vérifier la validité de vos solutions, des algorithmes liés au package `scipy` sont utilisés.

```
pip install scipy
```

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

```
python3 main.py --agent=naive --infile=instances/toy.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 minima 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. Si vous le souhaitez, vous pouvez reprendre votre métaheuristique précédente, mais vous devez 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. Une fois construit, votre solveur pourra ensuite être appelé comme suit.

```
python3 main.py --agent=advanced --infile=./instances/toy.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 A, j'ai obtenu une instance valide de coût 16, ma note pour cette instance est de $3 \times (1 - \frac{16-14}{19-16}) = 1.36/3$. Si aucun groupe n'atteint **LB**, alors le score du meilleur groupe est utilisé comme référence pour le calcul. La dernière colonne du tableau indique le temps maximum qui sera accordé à votre programme pour fournir une réponse.

Instance	Pondération	N	M	D	LB	UB	Temps max.
A	3/10	10	10	4	14	19	2min
B	2/10	15	20	6	25	40	10min
C	2/10	30	40	15	137	213	10min
D	2/10	40	60	20	250	401	15min
X	2/10	15	20	6	?	?	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 le nombre de chorégraphies, etc. 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.