Devoir 2 - Résolution de problèmes combinatoires

Remise le 29 mars (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 fichier python `solver_advanced.py`, et vos 4 modèles Minizinc (`.mzn`) à la racine d'un seul dossier compressé (matricule1_matricule2_Devoir2.zip). Ne changez pas les noms des fichiers. Attention, sur Mac cela crée parfois un second zip dans le dossier compressé, qui contient lui vos fichiers. Cela empêchera la correction de votre devoir. Veillez bien à ce que vos soumissions soient à la raçine du dossier compressé.
- Indiquez votre nom et matricule en en-tête des fichiers 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 solutions 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.

Conseils

Ce devoir est volontairement *challenging* pour obtenir la totalité des points. Voici quelques conseils pour le mener à bien :

1. Prenez vous y à l'avance. Il y a également la courbe d'apprentissage de Minizinc à considérer.
2. Travaillez efficacement en groupe, et répartissez vous bien les tâches.
3. Tirez le meilleur parti des séances de laboratoire encadrées afin de demander des conseils.
4. Pour la partie recherche locale, ne consacrez pas tout votre temps à l'amélioration de votre solution. Gardez cela pour la fin une fois que le reste est clôturé.
5. Pour la partie de modélisation, identifiez quelles contraintes globales pourraient vous aider.

Bonne chance!

Partie 1 : Construction d'un horaire académique (10 pts)

Construire un horaire académique n'est pas une tâche aisée. Cette question vise à vous sensibiliser à cette difficulté, et à proposer une méthode efficace 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 devoir, les incompatibilités sont indiquées par paire de cours. Par exemple, le cours INF6102 ne peut pas avoir lieu en même temps que INF8175 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 (i.e., une situation particulière à résoudre) 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
```

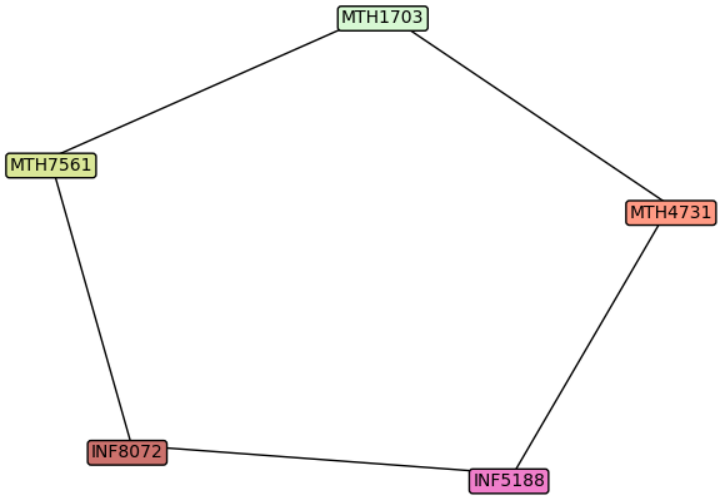
A titre d'exemple, l'instance suivante (`horaire_X_5_5.txt`) contient 5 cours et 5 conflits.

```
5
5
MTH4731 INF5188
MTH7561 MTH1703
MTH4731 MTH1703
MTH7561 INF8072
INF5188 INF8072
```

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

```
5
5
5
MTH4731 1
INF5188 2
MTH7561 3
MTH1703 4
INF8072 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 associés 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.

Vous êtes également libres de rajouter d'autres fichiers au besoin. De plus, 4 instances sont mises à votre disposition :

- `horaire_A_11_20.txt`, optimum connu égal à 4, **Score à battre égal à 5**.
- `horaire_B_23_71.txt`, optimum connu égal à 5, **Score à battre égal à 6**.
- `horaire_C_121_3960.txt`, optimum connu égal à 11, **Score à battre égal à 25**.
- `horaire_D_558_13979.txt`, optimum connu égal à 31, **Score à battre égal à 40**.

```
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
```

Il est conseillé d'utiliser la version **3.11** de Python pour ce devoir. Des **librairies supplémentaires sont également requises**, nous vous conseillons de créer un nouvel environnement virtuel pour ce devoir et de les installer comme ceci :

```
python -m venv venv
source venv/bin/activate # Pour Windows: ./venv/Script/activate
pip install matplotlib networkx
```

Critères d'évaluation

Le barème de cotation est le suivant.

- **La note minimale de 0 sur 10** si votre code ne compile pas.
- **Entre 1 et 4** si votre agent ne bat pas un agent aléatoire.
- **Entre 5 et 8** si votre agent bat l'agent aléatoire mais ne bat pas les scores seuils de **5, 6, 25, 40**, respectivement.
- **Entre 8 et 10** si votre agent bat les scores seuils. La différence entre un 8 et un 10 se fera sur la clarté et la lisibilité de votre implémentation (présence de commentaires, code structuré, etc.)

Vous avez le droit à un temps d'exécution de 5 minutes par instance.

⚠**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.**

Partie 2 : Modélisation en programmation par contraintes (10 pts)

Si ce n'est pas déjà fait, vous aurez besoin de télécharger MiniZinc et son environnement de développement. Il est recommandé de travailler avec la version **2.8.7** de MiniZinc. Un tutoriel se trouve également sur ce lien, il est conseillé de le lire attentivement avant d'attaquer les exercices.

Pour ce deuxième devoir, il vous est demandé de modéliser différents problèmes, de difficulté croissante, sur MiniZinc. Pour chaque problème, un fichier de base pour le modèle (`.mzn`), les fichiers vérificateurs de solutions (`.mzc`), ainsi que les différentes instances ou configurations à résoudre (`.dzn`), vous seront fournis. Vous êtes fortement encouragés à utiliser l'API de MiniZinc et d'identifier les contraintes globales les plus pertinentes pour résoudre les problèmes.

Lorsque vous lancez votre modèle dans MiniZinc, avoir le fichier `.mzc` correspondant ouvert dans les onglets vous permettra d'obtenir une vérification des solutions en sortie afin de vous indiquer si vos contraintes et la fonction objectif sont bien respectées. Le message **CORRECT** devrait s'afficher lorsque la solution renvoyée par votre modèle respecte les consignes. Attention, les vérificateurs ont besoin d'avoir une fonction objectif définie pour fonctionner sur les problèmes d'optimisation, il est donc probable que les premiers lancements de vos modèles se fassent sans les utiliser (redémarrer Minizinc s'ils ont déjà été ouverts dans les onglets). Les vérificateurs ne vous garantissent pas de trouver une solution optimale, leur rôle est seulement de vous assurer que vos contraintes sont respectées. Pour vous assurer l'optimalité, le solveur doit s'arrêter de lui-même, sans limite de temps, sur une solution pour certifier son optimalité.

Les fonctions de sortie des modèles Minizinc (`.mzn`) ont été implémentées pour vous, vous n'avez donc pas à les modifier, mais vous êtes autorisés à le faire.

⚠**Veillez à ne surtout pas modifier les variables déjà données au risque de ne plus pouvoir utiliser les vérificateurs, si vous avez besoin de variables supplémentaires il vous suffit de les déclarer.**

Ajoutez des commentaires lorsque vous jugez que c'est nécessaire. Cela peut aider le correcteur à comprendre une partie de votre modèle si ce dernier est incorrect, mais aussi vous aider à indiquer quelles contraintes sont déjà implémentées dans votre modèle. Vos modèles devraient être capable de résoudre chaque configuration en **moins de 2 minutes** sur une configuration similaire aux ordinateurs de la salle de travaux pratiques.

Problème 0 : code à 4 chiffres (introductory level - 1 pt)

Votre examen de programmation par contraintes arrive très bientôt! Après avoir bien pris le temps de vous familiariser avec les concepts de ce paradigme et la documentation du langage de modélisation, vous décidez de vous confronter aux exercices d'entraînement fournis. L'un d'entre eux attire particulièrement votre attention, on vous demande de trouver un **nombre** composé de 4 **chiffres** qui satisfait les critères suivants :

1. C'est un nombre pair.
2. Le chiffre 0 n'est pas présent dans le nombre.
3. Les 4 chiffres sont différents.
4. Le chiffre à la position des milliers est supérieur à celui à la position des centaines.
5. Le chiffre à la position des dizaines est inférieur à celui à la position des unités.
6. Le chiffre à la position des centaines est supérieur à celui à la position des unités.
7. La somme des 4 chiffres est supérieure à 15.
8. Le produit des 3 derniers chiffres (chiffre à la position des centaines×chiffre à la position des dizaines × chiffre à la position des unités) doit être minimisé.

Implémentez un modèle générique pour résoudre ce problème. Un bon modèle doit contenir **au moins une contrainte globale**. Trouver la **solution optimale** en moins de 2 minutes avec le solveur `Gecode` vous assure d'avoir le point de l'exercice.

Problème 1 : Une troupe capricieuse (easy level - 3 pts)

Ouf, l'examen est passé et vous avez obtenu votre diplôme avec brio !

Vous travaillez à présent pour Paul-Édouard, un metteur en scène de théâtre excentrique aux multiples facettes. Il aurait besoin de vos talents en optimisation combinatoire pour résoudre un problème d'attribution de rôles et de costumes dans la troupe qu'il dirige. En effet sa société bat de l'aile, il ne possède plus que quelques costumes en stock et seuls certains acteurs au caractère bien trempé acceptent de jouer pour lui à la condition qu'il réponde à chacun de leurs caprices. Vous devrez lui montrer vos talents en concevant un modèle de programmation par contraintes capable de résoudre les conflits sur chaque pièce qu'il dirige.

Il vous faudra donc **attribuer un et un seul rôle à chaque acteur** sachant que **chaque acteur décide du costume qu'il prendra pour chacun des rôles**. Par exemple, l'acteur A peut demander le costume Rouge pour le premier rôle alors que l'acteur B souhaiterait le costume Bleu s'il choisit ce même rôle. Vous devez donc **éviter tout conflit de rôle ou de costumes** entre vos acteurs. L'ambiance de la troupe étant quelque peu tendue, les acteurs voisins n'arrivent pas à se pifer lorsqu'ils jouent. Il vous faut vous assurer que **deux acteurs voisins, selon l'ordre lexicographique, ne jouent pas un rôle adjacent**. Attention, le premier et le dernier acteur ou rôle de leur liste respective ne sont pas considérés comme adjacents. Par exemple, l'acteur A qui prend le rôle 2 créera un conflit avec l'acteur B si celui-ci prend le rôle 1 ou 3, et inversement. Cependant, A ne créera pas de conflits avec un acteur C qui choisit un de ces rôles ou un acteur D qui serait le dernier de la liste. La figure d'exemple ci-dessous montre deux solutions invalides pour deux types de conflits différents et une solution valide sans conflits.

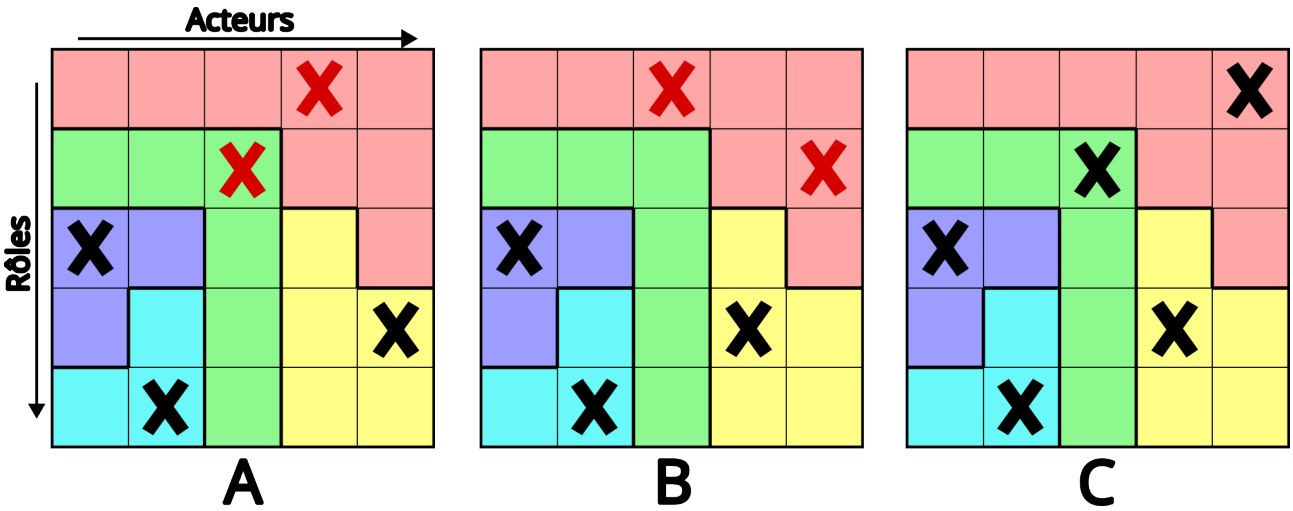


FIGURE 1 – Illustration de trois attributions de rôles pour une pièce avec 5 acteurs, rôles et costumes. La solution A présente un conflit entre les acteurs 3 et 4 car ils prennent deux rôles adjacents (2 et 1). La solution B présente un autre type de conflit, les acteurs 3 et 5 choisissent un rôle différent, et ne sont pas contraint de prendre un rôle non adjacent, mais le même costume rouge est exigé. Dans la solution valide C, les cinq acteurs sont respectivement attribués aux rôles 3, 5, 2, 4 et 1 ainsi qu'aux costumes Violet, Bleu, Vert, Jaune et Rouge.

Implémentez un modèle générique pour résoudre ce problème en vous appuyant sur le fichiers `.mzn` fournis. Un bon modèle doit contenir **au moins une contrainte globale**. Trois configurations vous sont données dans des fichiers `.dzn`. Trouver une **solution valide** en moins de 2 minutes avec le solveur `Gecode` pour une configuration vous rapporte 1 point.

Problème 2 : Une usine (trop) à la pointe (normal level - 3 pts)

Après avoir mené avec grand succès la troupe de Paul-Édouard et permis à son entreprise de reprendre la route du succès, ce dernier en profite pour vous introduire un autre de ces plans pour étendre sa gloire et son portefeuille. Il a décidé d'investir dans une usine entière de fabrication de costumes, équipée à la pointe de la cosmétique grâce à la technologie de l'industrie 4.0. Son but est d'utiliser sa troupe pour promouvoir un tout nouveau genre de costumes encore inédit. Cependant, il n'avait pas anticipé la consommation d'énergie de son usine. Afin de rendre son entreprise plus respectueuse de la planète (et de son porte-feuille) mais d'arriver à fabriquer ses produits dans un délai minimal, il vous demande d'ordonnancer la production de chacun de ses projets.

Chaque projet est découpé en **tâches pouvant être effectuées dans n'importe quel ordre et en parallèle**. Chacune des tâches possède une **durée** en heures et un **besoin en énergie**. Chaque projet possède une **limite d'énergie spécifique** qu'il vous faudra respecter en tout temps. Il est également très coûteux de démarrer une tâche, vous devrez donc vous assurer que **pas plus de deux tâches ne démarrent en même temps**. Votre objectif sera d'ordonnancer les tâches sur l'horizon temporel fourni et de **minimiser le temps avant la fin du projet**, c'est-à-dire le moment où la dernière tâche en cours se termine (aussi appelé le *makespan*).

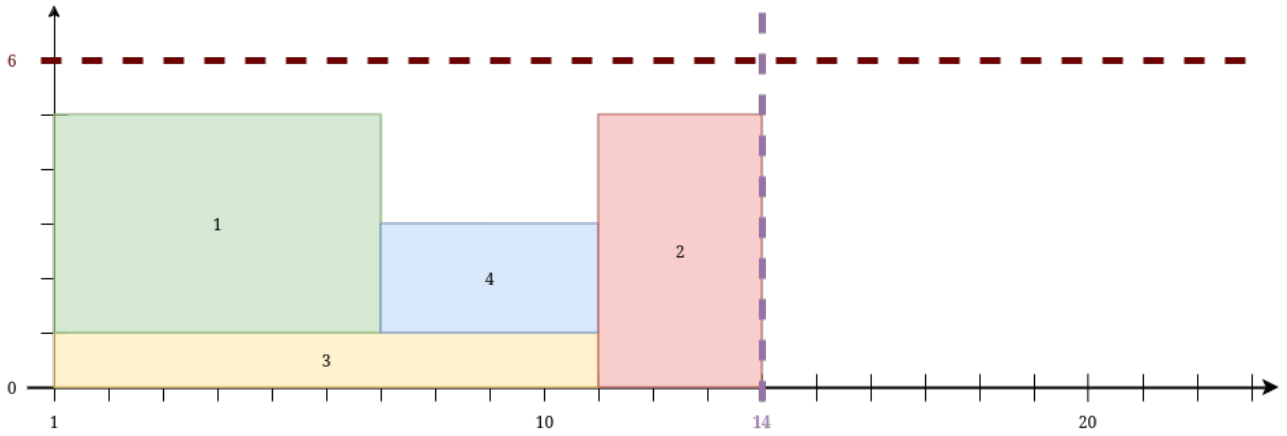


FIGURE 2 – Illustration d'une solution optimale avec 4 tâches. L'horizon temporel maximal est de 23 heures (abscisse) et la limite d'énergie est de 6 (ordonnée). Les tâches ont une durée respective de 6, 3, 10, et 4. Elles ont une consommation respective de 4, 5, 1 et 2. L'optimisation permet de finir le projet à la quatorzième heure, plutôt qu'à la vingt-troisième, en parallélisant correctement les tâches sans dépasser la limitation de ressource ou de tâches lancées en même temps.

Indication : le but de ce problème est de vous faire utiliser un maximum de contraintes globales, le modèle permettant de réussir l'exercice peu donc être très court si vous utilisez les bonnes contraintes. Pour vous aider, regardez du côté des contraintes de *scheduling*.;-)

Implémentez un modèle générique pour résoudre ce problème. Votre modèle devra contenir **au moins deux contraintes globales**. Trois configurations vous sont données dans des fichiers `.dzn`. Trouver une **solution optimale** en moins de 2 minutes avec le solveur `Gecode` pour une configuration rapporte 1 point.

Problème 3 : La grande tournée internationale (hard level - 3 pts)

Le succès vous sourit à nouveau et Paul-Édouard est aux anges de voir vos compétences lui rapporter gros. À présent, il est temps de faire passer sa troupe à un nouveau niveau et de promouvoir à un niveau plus large ses costumes d'exception. Il a prévu de grandes tournées dans toutes les villes de pays différents. Mais comme à son habitude, un gros problème de logistique s'impose, car les dates de la tournée sont extrêmement proches. Il vous demande donc d'organiser le chemin à effectuer pour chaque tournée afin d'éviter le fiasco.

Chaque tournée présente plusieurs villes à visiter, une distance entre chacune d'entre elles et une fenêtre de temps pendant laquelle il est autorisé d'arriver dans une ville. La distance est simplifiée afin de décrire **le temps dont la troupe à besoin pour passer d'une ville à l'autre** (comprenant la durée du spectacle), vous permettant de n'utiliser qu'une seule unité dans tout le problème.

Votre trajet doit **passer par toutes les villes une seule fois**. La première ville de la configuration est considérée comme votre **point de départ**. Le chemin construit commence donc par **la ville en suivant du point de départ** et **se termine par un retour à la première ville**. Afin d'entrer dans une ville, votre **temps de trajet cumulé au moment de votre arrivée doit se trouver dans la fenêtre de temps correspondante**. Si arriver en retard sur la fenêtre de temps est interdit, il se peut que votre troupe se présente trop tôt dans une ville. Pour cela, vous devez prendre en compte **un temps d'attente possible lors de votre cumul** pour accepter ces situations dans votre solution. L'objectif sera de **minimiser le temps total de la tournée**.

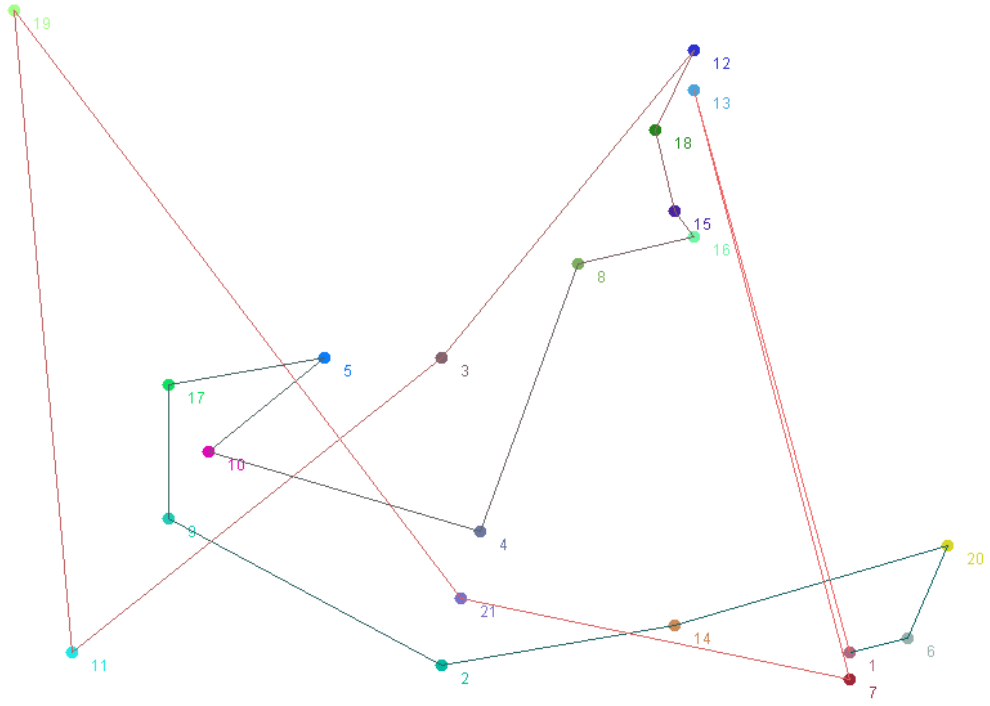


FIGURE 3 – Illustration d'une tournée optimale avec 20 villes et le point de départ (1). Le chemin de la tournée est [6, 20, 14, 2, 9, 17, 5, 10, 4, 8, 16, 15, 18, 12, 3, 11, 19, 21, 7, 13, 1].

Indication : ce problème demande beaucoup de temps de recherche au solveur, principalement pour les deux dernières configurations. Afin de pouvoir les réaliser dans le temps impartis, il vous sera primordial de trouver une façon d'accélérer la résolution sans risquer de perdre l'optimalité. Pour cela, nous vous

conseillons d'implémenter des contraintes redondantes et/ou d'essayer des heuristiques de recherche compatibles avec le solveur. Vous pourrez vous appuyer sur la documentation de Minizinc ainsi que sur le laboratoire 4 du cours pour vous donner une idée de l'utilisation de ces méthodes. Prenez également le temps de comprendre chaque variable de décision, la résolution sera nettement accélérée si chacune joue un rôle bien défini et est contrainte de la bonne façon.

Implémentez un modèle pour résoudre ce problème. Votre modèle devra contenir **au moins une contrainte globale**. Trois configurations vous sont données dans des fichiers `.dzn`. Trouver une **solution optimale** en moins de 2 minutes avec le solveur `Chuffed` pour une configuration rapporte 1 point.

Votre route vers la gloire et de la richesse est maintenant terminée!