Projet d'optimisation

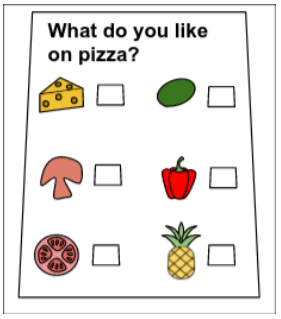
One Pizza is all you need

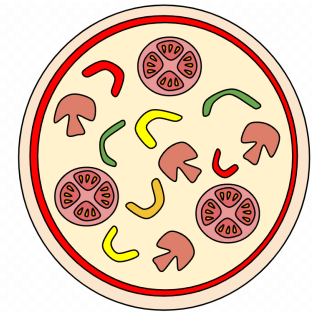
Le sujet est une adaptation du problème proposé lors de la session d'entrainement au *Google Hash Code* 2022 [1]. Le *Google Hash Code* est un concours d'optimisation proposé tous les ans, dans lequel les candidats, en équipe de 1 à 4 personnes, ont 4h pour obtenir la meilleure solution possible sur différentes instances d'un problème difficile.

Votre rendu doit être déposé sur la page arche dédiée à l'UE Optimisation au plus tard le **dimanche 24 avril 2022**.

1 Énoncé du problème

Vous souhaitez ouvrir une petite pizzeria. En fait, votre pizzeria est si petite que vous avez décidé de ne proposer qu'**une seule recette de pizza**. Vous devez décider quels ingrédients inclure dans votre recette.





Chacun de vos clients potentiels a des ingrédients qu'il aime, et peut-être des ingrédients qu'il n'aime pas. Un client viendra dans votre pizzeria **si les deux conditions suivantes sont remplies** :

1. Tous les ingrédients qu'il aime sont sur la pizza ;
2. Aucun des ingrédients qu'il n'aime pas n'est sur la pizza.

Chaque client est d'accord pour que des ingrédients supplémentaires qui ne soient ni dans ses ingrédients aimés, ni dans ses ingrédients détestés, soient présents sur la pizza.

Votre tâche est de choisir les ingrédients à mettre sur votre seul type de pizza, afin de maximiser le nombre de clients qui achèteront votre pizza.

1. `https://codingcompetitions.withgoogle.com/hashcode/round/00000000008f5ca9/00000000008f6f33`

1.1 Format des données

Les données du problème (nombre de clients potentiels, préférences des ingrédients) sont décrites dans un fichier texte au format suivant :
La première ligne contient un entier $1 \leq C \leq 10^5$ correspondant au nombre total de clients.
Les $2 \times C$ lignes suivantes décrivent les préférences des clients dans le format suivant :

- Une première ligne contient un entier $1 \leq L \leq 5$, suivi de L noms d'ingrédients qu'un client aime, délimités par des espaces ;
- Une deuxième ligne contient un entier $1 \leq D \leq 5$, suivi de D noms d'ingrédients qu'un client n'aime pas, délimités par des espaces.

Chaque nom d'ingrédient comprend entre 1 et 15 caractères ASCII. Chaque caractère est une lettre minuscule (a-z) ou un chiffre (0-9).

Exemple de fichier de données (instance A, voir tableau ci-dessous)

```
----------------------
3
2 cheese peppers
0
1 basil
1 pineapple
2 mushrooms tomatoes
1 basil
----------------------
```

Plusieurs instances du problème, correspondant à différents niveaux de difficultés sont disponibles (sur Arche) :

Problème	Ingrédients	Clients	Score indicatif
A - Un exemple	6	3	2
B - Basique	6	5	5
C - Grossier	10	10	5
D - Difficile	600	9368	1750
E - Élaboré	10000	4986	2000

À noter que les scores indicatifs sont des valeurs de scores possibles d'atteindre, mais ne sont pas nécessairement les valeurs optimales. Nous vous encourageons donc à les battre.

1.2 Format des solutions

Une solution doit être constituée d'une ligne composée d'un seul chiffre $0 \leq N$ suivi d'une liste de N noms d'ingrédients séparés par des espaces. Ils correspondront aux ingrédients que vous avez sélectionné pour votre recette de pizza. Cette liste d'ingrédients ne doit contenir que des ingrédients mentionnés par au moins un client, sans doublons.

Sur la page Arche de l'UE Optimisation, vous trouverez les différents fichiers d'entrée, un exemple de fichier de solution pour l'instance A, ainsi qu'un script Python `evaluation.py` qui, étant donné une instance du problème et une solution proposée, calcule et affiche le score de la solution. Il fonctionne grâce à la ligne de commande suivante :

```
python3 evaluation.py <instance> <solution>
```

où `instance` et `solution` sont les chemins vers les fichiers .txt contenant l'énoncé du problème et votre solution pour cet énoncé. Si la solution n'est pas valide ou ne correspond pas à l'instance du problème, le script affichera un message d'erreur.

Exemple `python3 evaluation.py A_an_example.txt exemple_solution_A.txt` produit la sortie suivante :

```
Nombre de clients : 3
Nombre total d'ingredients : 6

La solution proposée contient 4 ingrédients
Score de cette solution : 2
```

2 Travail à réaliser

Pour ce projet, le travail consiste à modéliser le problème des pizzas afin de pouvoir lui appliquer les différentes méta-heuristiques vues en cours.

Le rendu consiste en :

— Un rapport explicatif (détaillé en section 2.1)

Puis différents codes, dans le langage de votre choix [2], prenant en argument un fichier d'instance du problème (A à E) et générant une solution à cette instance, pour **au moins** :

— L'énumération de toutes les solutions possibles (partie 2.2)

Et, au choix, **deux parmi trois** métaheuristiques de la partie 2.3 :

— L'algorithme génétique ;

— Le recuit simulé ;

— La recherche tabou.

Facultativement, vous pouvez également proposer des réponses (en code comme dans votre rapport) aux questions supplémentaires de la partie 2.4.

Le tout devra être déposé sous la forme d'une archive sur la page arche dédiée à l'UE Optimisation au plus tard le **dimanche 24 avril 2022**.

2. Tant que ce code reste facile à exécuter sur un Ubuntu 18 et que les instructions pour le lancer sont fournies

Par exemple, si vous choisissez de faire le projet en python et que vous choisissez l'algorithme génétique et le recuit simulé, l'archive finale devra contenir au moins :

- `rapport.pdf` [3]
- `enumeration.py`
- `algo_genetique.py`
- `recuit_simule.py`

Les solutions que génèrent vos programmes seront réévaluées lors de la correction grâce au script `evaluation.py`. N'hésitez pas à également inclure dans l'archive des .txt contenant vos meilleures solutions (par exemple, `D_genetique.txt`).

2.1 Rapport

Pour expliquer vos solutions algorithmiques et vos programmes, rédigez un court rapport de quelques pages, qui devra au moins contenir des éléments de réponse aux questions posées dans les parties 2.2 et 2.3. N'hésitez pas à décrire en détails vos solutions, vos choix de paramètres, vos difficultés et n'importe quelle information ou astuce supplémentaire qui vous semblerait pertinente à la résolution du problème.

2.2 Petites instances du problème : recherche explicite

- Comment représentez-vous une solution au problème ?
- Supposons que l'on ait N ingrédients disponibles au total. Quelle est la taille de l'espace des solutions (recettes de pizza) possibles ?
- Sachant que $N < 10$ pour les problèmes A, B et C, proposez une méthode exacte.
- Donnez une estimation du temps que cette méthode mettrait pour s'exécuter sur l'instance E [4].

2.3 Grosses instances du problème : métaheuristiques

Pour les instances plus grosses du problèmes (D et E), nous proposons donc d'utiliser des heuristiques pour obtenir des solutions approchées.

2.3.1 Algorithme Génétique

- Quelle est la taille de votre population ?
- Comment définissez-vous l'opérateur de croisement ?
- Comment définissez-vous l'opérateur de mutation ? Quelle probabilité de mutation donne les meilleurs résultats ?
- Que choisissez-vous comme critère d'arrêt ?
- Précisez toute information au sujet de votre algorithme qui vous paraîtrait pertinente dans le rapport

3. ou `rapport.txt, rapport.md, rapport.odt, rapport.docx, ...`
4. On peut estimer qu'un ordinateur standard effectue 10^9 opérations par seconde

2.3.2 Recuit Simulé

— Quels mouvements entre solutions autorisez-vous ?
— Quelle est la valeur de départ de la température ?
— Comment décroît la température au cours du temps ?
— Quel est le critère d'arrêt ?
— Précisez toute information au sujet de votre algorithme qui vous paraîtrait pertinente dans le rapport

2.3.3 Recherche Tabou

— Quels mouvements entre solutions autorisez-vous ?
— Quel temps choisir pour les éléments dans la liste tabou ?
— Quel est le critère d'arrêt ?
— Précisez toute information au sujet de votre algorithme qui vous paraîtrait pertinente dans le rapport

2.4 Pistes pour aller plus loin

Proposez une modélisation du problème en programmation linéaire en nombre entier.

Proposez une modélisation sous la forme d'un graphe de contraintes entre clients.

Améliorez votre méthode exacte (partie 2.2) avec une approche Branch&Bound.
Quelle estimation du score pourrait-on utiliser pour borner certaines branches ?

Proposez un algorithme glouton pour résoudre le problème. Arrivez-vous à faire mieux que la métaheuristique codée en partie 2.3 ? Et en lançant une métaheuristique en partant de cette solution ?