Rapport du projet RP : Google Hash Code 2019, Photo Slideshow

Sylvain Sénéchal, Jean Lapostolle

Mai 2019

1 Introduction

1.1 Organisation du code

Certaines parties du projet sont réalisées en Python, d'autres en Javascript (nodeJs). Nous avons utilisé 2 langages de programmation, car Javascript, pour des raisons personnelles permettait de plus rapidement tester certaines idées, mais aussi parce que Python s'avérait être sensiblement plus lent sur certains algorithmes. Python reste cependant utile notamment pour utiliser Gurobi que nous avions déjà utilisé en MOGPL.
Pour lancer le code javascript, il faut installer nodeJs :

```
sudo apt-get install nodejs
```

Puis lancer la commande pour lancer le script

```
node scriptJS.js
```

1.2 Première approche du problème, analyse

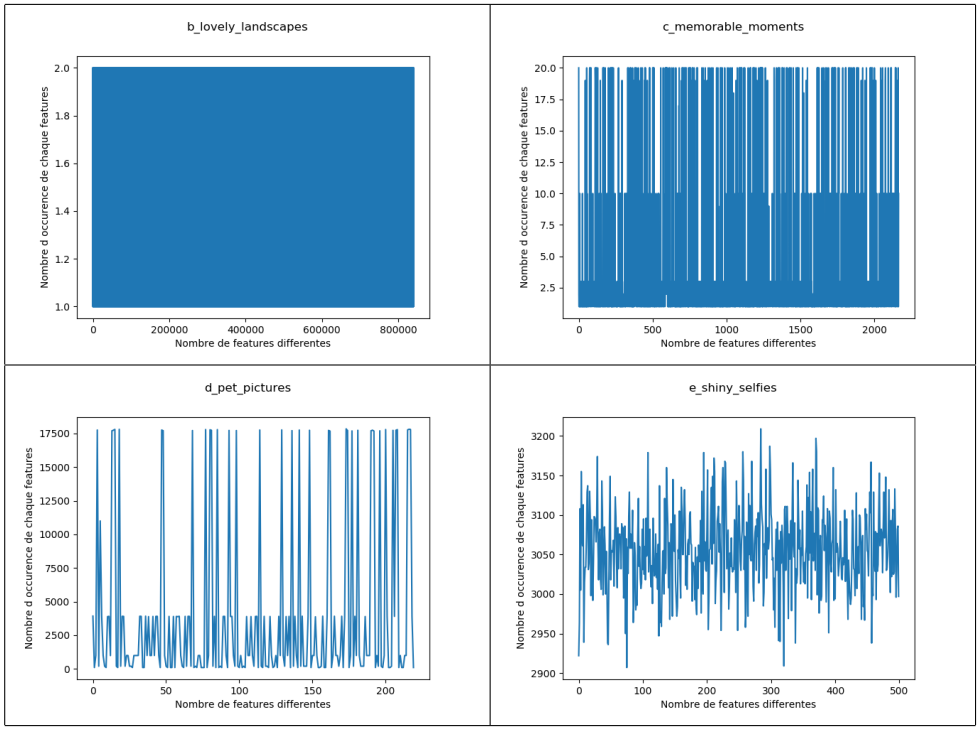
Pour donner un sens à nos résultats tout au long du projet, nous avons cherché sur le leaderboard de google les résultats du top pour avoir un ordre de grandeur : 1 222 000 points pour la première place (en faisant la somme du score de chaque dataset). Pour chaque dataset, les meilleurs résultats tournaient en général autours de :

- B lovely landscapes : 235 000 points
- C memorable moments : 1800 points
- D pet pictures : 440 000 points
- E shiny selfies : 550 000 points

Nous avons codé la fonction pour créer une présentation linéaire classique qui respecte l'ordre d'apparition des photos, pour obtenir ainsi une borne inférieure des résultats :

- B lovely landscapes : 12 points
- C memorable moments : 152 points
- D pet pictures : 190 961 points
- E shiny selfies : 112 468 points

On a ainsi à ce stade des bornes qui donnent une bonne idée du score qu'il faudrait atteindre au maximum et au minimum. On remarque également que le score du dataset B est particulièrement faible.
On réalise alors une analyse des features sur chaque dataset :



On remarque que les dataset ont chacun leur particularités, c'est-à-dire que la réparition des features est très différentes d'un dataset à l'autre, on pourrait sûrement adapter un algorithme particulier pour chaque dataset. Dans le B, on a près d'un million de features, qui ne sont présentes que 2 fois maximum, dans le C on a 2000 features présentes 2, 10 ou 20 fois chacunes de même que dans le D on a 200 features présentes 3000 ou 17500 fois la plupart du temps. Enfin le dataset E est plus régulier avec 500 features présentes 3000 +/- 100 fois

chacunes. Les résultats de la présentation linéaire de B s'expliquent donc très bien après avoir regardé les features présentes dans son dataset.

2 Complexité du problème

3 Résolution par diverses techniques

Les résultats sont présentés dans un tableau à la fin du document

3.1 Méthode gloutonne

Ici on commence par tenter une méthode gloutonne qui donne souvent des résultats assez acceptables sur des problèmes de type TSP. La méthode gloutonne utilisée est cependant en $O(n^2)$, ce qui pose un problème puisque les datasets ont environ 100 000 images, soit sur une double boucle 10 milliards d'itérations à faire en ordre de grandeur (sans compter que le calcul de la qualité de la transition à chaque itération est non négligeable en temps de calcul). Pour réduire ce temps, on peut utiliser la même méthode gloutonne avec une profondeur de recherche fixée, c'est-à-dire qu'une fois qu'une image est sélectionnée, on ne maximise la transition en ne choisissant que parmi les 50 prochaines images pout une profondeur de recherche de 50. On tend ainsi vers un algorithme en O(n) pour une profondeur qui tend vers 1.

3.2 Descente de gradient

La descente de gradient pour ce problème ne permet que difficilement d'améliorer une solution, en effet la recherche d'une bonne permutation de vignette permettant d'améliorer la solution courante et est en $O(n^2)$, et une amélioration n'apporte en général qu'un point de plus dans le score. En partant d'une solution déjà relativement bonne (le résultat de l'algorithme glouton), les permutations réalisées sont tout de suite plus utiles mais il semble que l'algorithme glouton renvoie souvent une solution déjà bloquée sur un optimum local. Seul le dataset C est suffisamment petit pour qu'on puisse travailler sur 100 pourcent de ses images et obtenir une légère amélioration.

3.3 Algorithme génétique

L'algorithme génétique est restreint au dataset B d'images strictement horizontales pour fortement simplifier les croisements génétiques. Ce type d'algorithme reste cependant assez peu efficace et assez lourd en calculs sur des instances constituées de nombreuses images de par la nature de son fonctionnement. Le dataset B étant par sa nature moins intéressant à étudier que les autres, on a testé l'algorithme génétique sur l'ensemble du dataset C de 1000 images, et en

partant d'une solution aléatoire de score 130, avec une population de 100 individus on arrive à un score de 230, ce qui reste assez faible comparé au résultat de l'algorthme glouton (1711).

3.4 Méthode custom

Etant donné les particularités du dataset B, on peut tenter une résolution avec un autre algorithme où l'on va travailler sur les mots-clé plutôt que sur les images : On commence par construire une table de hachage qui associe à chacune des 1 millions de features du dataset B une liste correspondant aux index des images possédant cette feature. Puis on construit la présentation ainsi : Pour chaque transition on sélectionne les images qui ont au moins une feature en commun avec la slide actuelle, puis on maximise la transition parmi ces images. Enfin on met à jour la table de hashage.
Cette méthode est intéressante sur ce dataset parce que poour chaque mot-clé, il n'y a que 2 images maximum qui le possède, et on est ici directement capable d'identifier ces 2 images pour les regrouper. On obtient sur le dataset B un score de 95628, bien meilleur qu'avec les autres solutions.

4 Formulations PLNE

Le solveur utilisé est Gurobi. Les performances sont telles qu'on peut résoudre le problème sur une instance d'environ 1 millier d'image en un temps raisonable.
Soit G les images.
Soit n images, c_{ij} le coût de la transition entre les images i et j.
Soit

$$\mathrm{x}_{ij} = \begin{cases} 1 & \text{si les images i et j se suivent dans le slide} \\ 0 & \text{sinon} \end{cases}$$

Le PLNE est le suivant :

$$max \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\text{Avec les contraintes : } \begin{cases} \sum_{j=1}^{n} x_{ij} = 1 & \forall j \in G \\ \sum_{i=1}^{n} x_{ij} = 1 & \forall i \in G \\ \sum_{j=1}^{n} \sum_{i=1}^{n} x_{ij} \leq n-1 & \\ \mathrm{x}_{ij} \in [\![0;1]\!] & \end{cases} \quad (1)$$

$$\begin{aligned}
&\text{max :} && \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
&\text{contraintes :} && \sum_{j:e_i S_j} x_j \geq 1, && i = 1, \ldots, n \\
& && x_j\{0,1\}, && j = 1, \ldots, m
\end{aligned}$$

La dernière contrainte assure la suppression des sous tours, en effet, il faut N-1 arc pour relier N points, si on compte plus de N-1 arcs dans la solution, il y a forcément des sous-tours. Gurobi optimise la suppression de sous-tours en commençant par résoudre le PL sans prendre en compte la contrainte de sous-tours. Puis une fois une solution trouvée, soit il n'y a pas de sous tours et la solution est directement optimale, soit il y a un sous-tours et on rajoute une contrainte qui est qu'il ne doit pas y avoir de sous-tours de longueur égale au plus petit sous-tour trouvé. Cette technique permet d'accélérer la résolution du PL. Pour le dataset B, on résoud le problème avec 900 images en une dizaine de secondes.

5 Analyse des résultats et conclusion

Dataset	bInf	bSup	glout.500 depth	Descente grd	génétique	PLNE	Custom
B	12	235000	7374	-	-	-	95628
C	152	1800	1711	1713	-	-	-
D	190961	440000	356932	-	-	-	-
E	112468	550000	395932	-	-	-	-

Table 1: Resultats sur la totalité des datasets

Dataset	bInf	bSup	glout.500 depth	Descente grd	génétique	PLNE	Custom
B	0	-	54	54	15	57	9
C	0	-	4	4	-	-	-
D	1855	-	3596	3610	-	-	-
E	1100	-	3831	3831	-	-	-

Table 2: Resultats sur 1 pourcent des datasets

La résolution par PLNE s'avère meilleure que les autres comme attendu, mais la particularité de la répartion des mots-clés du dataset B ne permet malheuresement pas d'obtenir une différence de résulats important sur une petit nombre d'images.
Pour chaque dataset, nous possèdons au moins une technique permettant d'obtenir des résultats satisfaisants, soit par un algorithme glouton, soit par une méthode dédiée spécialement au dataset qui s'adapte mieux à ses contraintes. C'est assez prévisible puisque les photos n'ont pas forcément été générée aléatoirement, mais Google a probablement fait en sorte de nous mettre en difficulté. Il pourrait d'ailleurs être intéressant de mélanger l'ordre des images lorsqu'on récupère les dataset puisque l'ordre a peu être aussi été choisi d'une certaine façon handicapante par Google.
Pour accélérer la vitesse d'exécution de l'algorithme glouton on pourrait simplement prendre les images par nombre de mots-clés décroissants, ou bien maximiser la transition avec une fonction de coût plus rapide à calculer.
Il serait aussi judicieux pour aller plus loin d'utiliser un langage de programmation compilé et de réfléchir plus judicieusement les structures de données utilisées.