Mini-projet de C++ : Google Hashcode 2016
Rapport de l'équipe 13

Vincent Cotineau Anthony Griffon Benjamin Landry
Hugo Pigeon Pierre Pétillon

16 décembre 2016

Ce projet pose un problème complexe, qui demande une certaine réflexion pour arriver à un résultat donnant un score élevé. Il faut également faire attention à la façon d'implémenter la méthode de résolution, pour que le traitement se fasse le plus rapidement possible. Cela conduit à faire des choix de modélisation, qui ont une grande influence sur les algorithmes développés et donc sur le temps de calcul.

Pour résoudre le problème posé, nous avons choisi de développer un algorithme naïf, simple à implémenter, afin d'avoir une première solution correcte. Pour améliorer les résultats obtenus et le temps de traitement, nous avons réfléchi en parallèle à un algorithme plus complexe. Dans ce rapport, nous allons présenter ces différents algorithmes.

1 Algorithme naïf

Le but de cette solution est d'obtenir un premier résultat à partir d'un principe relativement simple. Nous avons effectué quelques optimisations pour minimiser le temps de calcul, mais ce n'était pas l'objectif prioritaire de cette solution.

1.1 Principe

Le principe est donc le suivant : chaque satellite prend la photo la plus proche qu'il peut atteindre. Ainsi, on traite les satellites un par un, chacun effectuant ses déplacements sur toute la simulation en dirigeant en permanence sa caméra vers la prochaine photo qu'il peut prendre. Pour trouver la prochaine photo à prendre, on simule le déplacement du satellite sur chaque tour. On calcule à chaque fois la fenêtre de coordonnées que la caméra peut atteindre et on vérifie si une photo se trouve dans cette fenêtre. Si c'est le cas, que la photo n'a pas déjà été prise et qu'il est possible de la prendre au tour courant, alors on arrête la recherche. Sinon, on continue à simuler le déplacement du satellite, jusqu'à trouver une photo ou jusqu'à arriver à la fin du temps imparti. Une fois que la prochaine photo a été

trouvée, on exécute réellement le déplacement du satellite, en dirigeant la caméra vers les coordonnées de la photo trouvée.

1.2 Optimisation

Afin d'accélérer l'exécution de cet algorithme, nous avons mis au point une optimisation qui diminue grandement le temps de traitement, sans changer le résultat. Ainsi, nous avons choisi d'utiliser le principe de la recherche dichotomique pour trouver rapidement la prochaine photo à prendre. En effet, il s'agit de la partie la plus longue du programme : avec une recherche séquentielle, pour chaque satellite on parcourt toutes les images à chaque tour.

Le principe est donc d'avoir une liste de toutes les images, que l'on trie selon leur latitude puis selon leur longitude. A partir de là, on peut faire une recherche dichotomique sur la latitude, ce qui permet d'arriver rapidement à une image dont la latitude est dans la fenêtre de la caméra. On sait alors que les images qui se situent dans la fenêtre, si elles existent, sont classées autour de celle que l'on vient de trouver. On fait alors une recherche séquentielle sur toutes ces images proches jusqu'à en trouver une qui est dans la fenêtre, ou jusqu'à ce que l'on sorte de celle-ci.

Il n'est cependant pas possible d'effectuer une recherche dichotomique pour trouver directement l'image de manière systématique avec ce principe. En effet, dans certains cas la recherche dichotomique peut supprimer de son espace de recherche des images dont la latitude est dans la fenêtre. Par exemple, si on arrive sur une image dont la latitude est correcte mais la longitude est trop grande, la recherche dichotomique considère que toutes les images dont la latitude est supérieure à celle-ci ne sont pas dans la fenêtre. Or c'est faux dans la plupart des cas : on peut avoir une image dont la latitude est supérieure mais toujours dans la fenêtre et dont la longitude est aussi dans la fenêtre.

2 Algorithme amélioré avec utilisation d'un ratio

Le but de cet algorithme est d'utiliser les différentes informations que nous avons sur les collections d'images à savoir le nombre de points à gagner, le nombre de tours et le nombre d'images à photographier. Ainsi, nous nous servons de ces informations pour créer une priorité sur les photos à prendre.

2.1 Principe

Le principe de cet algorithme est de se baser sur la solution naïve présentée précedemment en améliorant l'ordre de la sélection des photos à prendre pour les satellites. Ainsi, pour chaque satellite on identifie pour chaque tour, tout d'abord une photo ayant un ratio (NbPoints/NbPhotosAPrendre) supérieure et on la prend en photo si elle existe. Si tel n'est

pas le cas, on choisit la photo la plus proche et on continue au prochain tour à trouver la photo la plus valable en moyenne.

2.2 Détails d'implémentation

Tout comme l'algorithme naïf, nous avons une liste de toutes les images du jeu de données. Cependant, nous choisissons d'abord de les trier via le ratio calculé pour chaque image avec un rapport entre le nombre de points de la collection et le nombre de photos à prendre. Si nous sommes dans un cas d'égalité de ratio, nous trions les images par leur latitude et leur longitude.

Une fois la liste d'images triées, nous recherchons la meilleure image possible à prendre selon le ratio avec un intervalle de tours limité. Ainsi, on vérifie si la meilleure image n'a pas une distance de tours relativement élévée grâce à un seuil. Si tel est le cas, on vérifie si il n'existe pas d'autres photos plus proche. S'il n'y a pas de photos comprise dans le seuil de tours, on incrémente le tour afin de trouver une solution.

Cependant, cet algorithme n'est malheureusement pas idéal car il se base sur l'hypothèse que la photo la plus valable permet de trouver le chemin le plus valable. Pour améliorer cet algorithme, il faudrait pouvoir comparer les chemins au fur et à mesure de l'avancée de l'algorithme et modifier le chemin à chaque fois que l'on en trouve un qui est plus intéressant.

2.3 Résultats

	Jeu de données	**Temps**	**Score**	**Pourcentage**
Algorithme 1				
	forever_alone	X secondes	44000	35.5
	constellation	X secondes	44000	35.5
	overlap	X secondes	44000	35.5
	weekend	X secondes	44000	35.5
Algorithme Ratio				
	forever_alone	X secondes	48000	34.44
	constellation	X secondes	64313	2.15
	overlap	X secondes	14114	1.21
	weekend	X secondes	50612	0.75
Meilleurs résultats				
	forever_alone	X secondes	44000	35.5
	constellation	X secondes	44000	35.5
	overlap	X secondes	44000	35.5
	weekdend	X secondes	44000	35.5