Rapport Du Projet Taxi

Ze CommiT

Membres du groupe :
Loïc BERTOLOTTO,
Vincent DELMOTTE,
Nathan MEULLE,
Clément POUYETO.

16 – 22 Décembre 2019

Plan du rapport

Rapport technique

Structure du projet

Pour mieux organiser le projet, nous avons créé trois packages regroupant les différentes fonctionnalités.

Le premier package "io" comprend les classes Entrée, analyseEntree et Sortie. Ces classes n'interfèrent pas avec l'algorithme, elles traitent les données en entrée et formatent les données de sortie.

Le second package "grille" comporte les éléments que l'algorithme va modifier. Les classes "position", "course" et "voiture" sont liées à la classe "carte" car elle comporte une liste de véhicules, une liste de courses avec des positions différentes.

Enfin le package "main" constitué de la classe "algorithme", "score" et "main" permet l'implémentation de plusieurs algorithmes facilement interchangeables.

Description des algorithmes

Pour répondre à la problématique posée par ce projet, nous avons commencé par appliquer un algorithme relativement simple avec une faible complexité : c'est un algorithme de type glouton.

Il s'agit de gloutonVehicule qui est relativement simple : on trie d'abord toutes les courses par rapport à l'étape de départ des courses. Puis on parcourt les véhicules un par un. Pour chaque véhicule, on parcourt la liste des courses triées et dès qu'une course est réalisable par le véhicule, celle-ci lui est attribuée, et le véhicule continue de parcourir la liste des courses jusqu'à ce qu'il ne puisse plus en réaliser, puis on fait de même pour les autres véhicules.

Nous avons ensuite amélioré cette version en gloutonBonus : le véhicule commence par parcourir les courses en ne prenant que celles qui lui rapportent le bonus de ponctualité, puis reparcourt la liste des courses pour lui attribuer seulement les courses réalisables. Cela nous a permis de gagner des points sur certains fichiers où les bonus de ponctualité avaient leur importance comme sur le fichier 4 ou le fichier 5.

Les résultats étants corrects mais insuffisants et au vu de la marge au niveau du temps d'exécution de nos algorithmes, nous avons décidé de revoir un tout nouvel algorithme avec plus de complexité : gloutonDistance.

Ce dernier commence par supprimer de la liste des courses les courses considérées comme trop longues. Puis, comme gloutonVehicule, il parcourt les véhicules un à un. Pour chaque véhicule il ne garde que les courses réalisables, puis trie cette liste de courses par rapport au maximum entre la distance séparant le véhicule du début de la course et l'attente

que le véhicule aura avant de commencer la course. Nous parcourons ensuite au plus 12 premières courses de la liste et nous regardons si une course peut obtenir le bonus de ponctualité. Si ce n'est pas le cas on attribue la première course réalisable, on la supprime de la liste ainsi que toutes les courses qui ne sont plus réalisables puis on retrie la liste de course qu'on reparcourt.

Enfin, le dernier algorithme implémenté, gloutonCourse, reprend l'algorithme gloutonBonus mais dans l'autre sens, c'est-à-dire qu'on parcourt les courses et pour chaque course on parcourt tous les véhicules. L'algorithme commence par effectuer un premier tri des courses en fonction de l'étape de départ de chaque course ainsi que de la distance les séparant du point (0,0) puis attribue à tous les véhicules une course de la liste. On retrie la liste de course en fonction de leur tour de départ et de leur longueur afin de réaliser en priorité les courses qui commencent tôt et qui sont les plus courtes, puis on parcourt cette liste de courses. A chaque course, on trie les véhicules sur le même critère que triDistanceCourse. On commence par parcourir la liste en regardant si le bonus de ponctualité peut être validé (pour les 340 premiers véhicules de la liste, nombre limite obtenu par dichotomie), puis on reparcourt en attribuant la course au premier véhicule de la liste pouvant réaliser la course.

Description des difficultés rencontrées

Nous avons rencontré des difficultés dans l'amélioration de notre algorithme d'attribution des courses. Nous avons en effet essayé plusieurs algorithmes avant d'en trouver un permettant d'obtenir de meilleurs scores.

Dans quelle mesure notre code est modulable/extensible ?

Notre code permet de mettre en place très facilement un nouvel algorithme : il suffit de l'implémenter et d'indiquer sous quelles conditions l'appliquer.

Peut-on changer le format d'entrée / de sortie ?

Les données d'entrée peuvent être modifiées tant que celles-ci respectent le format demandé. La sortie est elle aussi conforme au format demandé dans le cahier des charges.

Est-il facilement possible de remplacer un algorithme par un autre ?

Notre méthode « traitement() » permet de sélectionner facilement un algorithme plutôt qu'un autre. Cela permet également d'alterner entre plusieurs algorithmes selon certaines conditions dépendant des données d'entrée.

Est-il facile d'ajouter une nouvelle métrique d'optimisation ?

Notre code est composé de plusieurs parties permettant d'isoler l'une d'entre elles sans difficulté afin de l'optimiser.

Quelle est la meilleure chose de notre code ?

Notre algorithme de répartition des courses est très rapide et permet tout de même d'obtenir des scores élevés. De plus les méthodes mises en place afin de faire les vérifications quant à la possibilité de réalisation de course et l'attribution d'une course à une voiture permet de recréer de nouveaux algorithmes facilement sans risques d'erreurs de sortie.

Quelle est la pire chose de notre code ?

Certaines méthodes sont un peu longues et nécessiteraient d'être subdivisées.

Quels sont les avantages/inconvénients d'utiliser un langage orienté objet dans ce cas ?

Coder en java orienté objet permet une distinction facile des différentes classes. Cela permet de structurer de façon claire le projet : il est ainsi plus facile d'identifier et comprendre le rôle de chaque partie du code.

Néanmoins passer par des objets nous impose une structure plus contraignante et diminue les performances du fait de la création des nombreux objets (véhicules, courses, carte) et la manipulation de ces derniers.

Quelles seraient les différences si notre projet avait été écrit en C ?

Le code aurait été bien plus court et probablement en un seul bloc ce qui aurait été moins lisible et moins compréhensible.

Nous aurions potentiellement rencontré des difficultés dans le début de la mise en place du code.

Cependant le temps d'exécution aurait sans doute pu être diminué, mais au vu des temps obtenus ce gain serait relativement négligeable.

Quelles sont les limites de performances rencontrées ?

Le premier algorithme que nous avons mis en place (gloutonVehicule) a été construit, dès le début, en vue d'obtenir une faible complexité et des scores élevés : nous parcourons chaque véhicule puis chaque course pour l'attribuer directement si celle-ci est réalisable.

Ce faible temps d'exécution nous a permis d'avoir une marge assez importante nous permettant d'augmenter la puissance de nos algorithmes au dépend de la performance.

Nous n'avons donc pas été confronté par la suite à des soucis d'optimisation mais avons eu plus de mal dans le développement d'un nouvel algorithme capable de surpasser les scores précédemment obtenus.

Analyse des résultats obtenus

Quels sont les avantages et les inconvénients de nos approches ?

Nous avons d'abord établi un algorithme (gloutonVehicule) qui, d'un point de vue commercial est intéressant car il propose une solution fonctionnelle rapide et performante : celle-ci peut donc fonctionner sur des machines de faibles capacités.

Nous avons ensuite créé des variantes de notre premier algorithme permettant de meilleures performances. De plus, en fonction de l'entrée, l'algorithme le plus adapté est appelé. Ces nouveaux algorithmes sont toutefois légèrement plus longs que le premier implémenté mais restent très rapides avec de meilleurs résultats (par exemple environ 116 millions de points sur le fichier 4 au lieu de 102 millions avec un temps d'exécution inférieur à 3 secondes).

Dans quelle mesure avons-nous obtenu de bons ou de mauvais résultats ?

Nous avons obtenu de très bons scores pour chaque input notamment ceux avec des bonus élevés. De plus, 4 de nos scores sont obtenus avec un seul algorithme ce qui montre que cet algorithme peut s'adapter à plusieurs configurations de ville, ce qui en fait un algorithme assez puissant.

Comment évolue notre temps d'exécution ? Est-il prévisible ?

Le temps d'exécution de notre code est difficile à prévoir : il dépend de la simplicité ou non à arranger les courses afin d'obtenir un score optimal.

Description du benchmark

Notre benchmark effectue des tests sur des blocs importants du code ainsi que sur l'intégralité du programme ce qui permet de cerner les parties améliorables.

Organisation

Gestion du projet à plein temps

Au sein de l'équipe Ze-CommiT, nous avons été assez bien organisés : aussi bien avec les Milestones journalières sur GitHub qu'au niveau de la répartition du travail avec l'attribution à chacun d'entre nous de classes à réaliser.

Nous avons également pu effectuer des tests avant chaque livraison afin d'être confiant dans les scores fournis.

Auto-évaluation selon différents critères

- Organisation des tâches : 4,75/5 : Bonne organisation avec des Milestones et des issues sur GitHub chaque jour.
- Tests : 4/5 : Nous avons réalisé de nombreux tests avant chaque mise en place d'un nouvel algorithme.
- Orienté objet : 4,5/5 : Nous avons bien organisé nos classes. Utiliser un héritage dans la construction de nos algorithmes aurait peut-être été intéressant…
- Rendus journaliers 5/5 : une petite erreur sur 1 des 25 rendus
- Documentation 4/5 : Nous avons ajouté des commentaires pour décrire chaque classe, chaque méthode et chaque condition.

Auto-évaluation de la répartition

- 100/100/100/100 : Chaque membre de l'équipe a contribué de façon égale à la réussite de ce projet.

Conclusion

Qu'avons-nous appris grâce à ce projet ?

A travers ce projet nous avons appris à réaliser des benchmarks afin d'évaluer l'optimisation de notre code.

Nous avons également progressé dans la mise en place rapide d'un code minimal mais fonctionnel.

Quelles connaissances provenant d'autres cours avons-nous utilisées ?

Nous avons principalement utilisé nos connaissances en mathématiques afin de concevoir un algorithme ayant une complexité moindre, ainsi que certains cours de Programmation Orienté Objet pour l'utilisation des fonctions lambda.

Quelles leçons retenir de ce projet ?

Grâce à ce projet nous avons pu apprendre l'importance d'allier performance et rapidité dans notre code.