Rapport Projet RO

BladeFlyer II : Conquest of water

Table des matières

I – Introduction....1
II – Partie 1 : Présentation générale....1
1.1 Rappel sujet....1
1.2 Organisation générale de notre projet....2
III – Partie 2 : Résolution....2
2.1 Résumé de nos étapes de résolution....2
2.2 Regroupements, distances....3
III – Partie 3 : Analyse....5
3.1 Compléxité....5
3.2 Performances....5
3.3 Améliorations et difficultés....5
IV – Conclusion....6

I – Introduction

Ce rapport présente notre travail concernant le projet de RO : BladeFlyer II – Conquest of water. Le sujet de ce projet portait sur la mise en place d'une solution pour un problème de type voyageur de commerce. Dans ce rapport, nous présenterons d'abord rapidement le sujet ainsi que l'idée générale de la résolution. Ensuite nous parlerons en détail des algorithmes et des structures de données que nous utilisons. Enfin, nous discuterons de la complexité temporelle et spatiale de notre solution puis nous mettrons en valeur les améliorations possibles avant de conclure.

II – Partie 1 : Présentation générale

1.1 Rappel sujet

Le sujet du projet nous plaçait dans un monde post-apocalyptique dans lequel des communautés humaines tentent de survivre en pompant de l'eau dans certains points grâce à des drones. Ces points de pompages sont dispersé sur une carte, le but du projet était de proposer un chemin permettant au drone de prendre de l'eau en parcourant la distance la plus courte possible.

Dans un premier temps, il n'y a qu'un seul drone avec une capacité de portage limité. Les distances entre les points d'eau sont regroupés dans un distancier. Le voyageur de commerce ici se faisait en fait entre différents regroupements de point d'eau pour lesquels le drone pouvait prendre toute l'eau.

Donc en résumé, nous devions, à partir du distancier, des informations sur les points d'eau et le drone, générer des regroupements de points d'eau, calculer les distances pour parcourir ces points d'eau depuis la base et enfin proposer un chemin.

1.2 Organisation générale de notre projet

Notre résolution s'organise comme ceci : nous avons un fichier de données dans lequel se trouvent des informations (distancier, quantité d'eau à un point et capacité du drone). Nous avons un fichier **gestion_donnees.hpp** dans lequel il y a toutes les structures de données et les algorithmes que nous utilisons pour générer les données utiles tels que les regroupements, etc.

Ensuite, un fichier **bladeflyerii.cpp** contient le code qui lance la récupération des données et leur traitement, puis instancie un modèle glpk et le résout. Les différentes instances (données) sont dans un sous-dossier Data.

Nous avons décidé d'écrire une grande partie du code en C++. Nous sommes un peu plus à l'aise avec les structures de données et les fonctionnalités présentes en C++. Ecrire le code en C aurait été tout aussi bien (voire même plus précis au niveau de la gestion de la mémoire) mais nous avons décidé de rester sur quelque chose que nous connaissions un peu mieux.

Enfin il y a un Makefile contenant les quelques commandes nécessaires à la compilation du projet.

III – Partie 2 : Résolution

2.1 Résumé de nos étapes de résolution

De manière succincte, notre résolution s'organise comme ceci :

- lecture des données depuis un fichier .dat
- création de l'ensemble des regroupements possibles
- calcul des distances internes à chaque regroupement avec un algorithme de force brute
- génération des données utiles à la résolution en glpk (nombre de variables, nombre de contraintes, indices des variables dans la matrice creuse des contraintes,...)
- résolution du problème avec les méthodes de la bibliothèque glpk (et calcul de performances)

Nous avons dans notre fichier gestion_donnees.hpp différentes structures de données et différents algorithmes permettant à la fin de récupérer les données qui pourront être envoyées à glpk.

donnees_f est une structure permettant de récupérer directement les données du fichier .dat tel quel.

Ensuite, nous générons un ensemble de points de pompage en associant simplement un indice à chaque quantité d'eau à relever (donc un point de pompage).

A partir de notre ensemble de points de pompages nous créons l'ensemble des regroupements qui ne dépassent pas la capacité du drone (avec une méthode décrite plus loin dans le rapport).

Après cela, nous calculons les distances minimales internes aux différents regroupements avec un algorithme de force brute (décrit au point 2.2) .

Puis nous calculons à partir de l'ensemble des regroupements, l'ensemble des indices qui seront utilisés dans la matrice creuse des contraintes.

Enfin, nous mettons toutes ces données dans une seule structure que nous utilisons dans les différentes méthodes de glpk.

Concernant la résolution du problème avec glpk. Nous avons repris le modèle linéaire qui était fourni dans le sujet. Il y a une variable de décision binaire pour chaque regroupement. La fonction objetcif est la minimisation de la somme des regroupements multipliés par leur distance interne respective. Les contraintes sont pour assurer que nous passons une et une seule fois par chaque point de pompage. S' il y a n point de pompages alors il y a n contraintes. Ces contraintes sont des égalités à 1 des sommes des variables de décisions qui concernent le point de pompage de la contrainte. Pour pouvoir trouver les indices des variables concernées, nous avons écrit un algorithme qui parcourt les regroupements et trie dans un tableau à deux dimensions les indices de ces regroupements en fonction du point de pompage visé.

Nous avons à la fin ceci (exemple arbitraire) :

point de pompage	regroupement conernés
1	1 4 9 12 13
2	1 2 3 5 8 10
3	3 4 6 7 11
...	...

De cette facon nous pouvons facilement utiliser dans la matrice creuse les indices des regoupements pour déterminer quelles variables de décision doivent se retrouver dans quelle contrainte.

2.2 Regroupements, distances

Dans cette partie, nous allons présenter un peu plus en détails certains de nos algorithmes.

a) Les regroupements

Pour générer tous les regroupements nous utilisons l'algorithme suivant :

Nous partons de l'ensemble des points de pompage n. Nous avons 4 entiers : i, j , imax et jmax.

imax vaut 2^n-1, c'est l'ensemble des parties d'un ensemble à n éléments. jmax vaut n-1, c'est le nombre de points possible (mais on commence les indices à 0 donc -1).

Ensuite tant que i est inférieur ou égal à imax, tant que j est inférieur ou égal à jmax, si après le décalage de j de i bits vers la droite, un "ou" logique avec 1 est vrai alors c'est qu'il y avait un 1 dans la position j.

Nous pouvons alors mettre dans un regroupement la valeur de l'indice d'une pompe en j.

Ensuite nous recommencons jusqu'à faire tous les j pour un i.

Avant de passer au i suivant, nous regardons si la somme des quantité d'eau du regroupement que nous venons de créer dépasse la capacité du drone (avec une somme sur les capacité des points de pompages du regroupement). Si cette quantité est inférieur ou égale alors nous ajountons à l'ensemble des regroupements le regroupements crée juste avant.

Puis nous relancons avec i + 1.

Finalement, pour avoir les regroupements, nous utilisons le fait que les pompes soient indicées de 1 à n. L'écriture binaire de toutes ses valeurs donne les parties de cet ensemble de valeurs.

Exemple avec {1 2 3}, on va jusqu'à 2^3 -1 = 7

1 = 001 →{1}

2 = 010 → {2}

3 = 011 → {1,2}

4 = 100 → {3}

5 = 101 → {1,3}

6 = 110 → {2,3}

7 = 111 → {1,2,3}

En éliminant les regroupements qui coûtent trop cher, nous avons tous les regroupements possible pour l'instance donnée.

b) Calcul des distances de chaque regroupements

Ici il s'agissait, pour un regroupement de points de pompage, de déterminer le plus court chemin visitant une fois chaque point et revant à la base. C'est en fait un problème de voyageur de commerce encore une fois.

Nous avons d'abord essayé de résoudre le problème en créant une instance glpk pour chaque regroupements. Mais il très difficile de résoudre ce problème avec une modélisation linéaire. Nous devons tôt ou tard nous confronter au problème du test des sous-tours. Les méthodes pour résoudre ce problème s'orientent alors vers de la programmation dynamique, des algorithmes de coupe, etc.

Nous avons malheuresement passé beaucoup de temps sur cette idée mais nous n'avons pas réussi à produire quelque chose. Nous avons décidé alors de nous orienté sur un algorithme qui pourrait produire le même résultat qu'une modélisatio navec glpk. Il existe de nombreuse version d'algorithmes de force brute permettant de faire les calculs pour le problème de voyageur de commerce.

Nous avons essayé d'écrire une version mais elle ne marche pas du tout. Les résultats sont faux en plus d'être parfois différents sur plusieurs itérations de la même instance. Nous n'avons pas réussit à écrire une version fiable de l'algorithme de force brute pour le voyageur de commerce.

Il était possible de reprendre l'ordre des points de pompage dans un regroupements pour l'ordre de visite dans le voyageur de commerce. Mais nous avons passé trop de temps à essayer l'algortihme de force brute et nous n'avons aps cette version non plus.

Une de nos idée pour le calcul des distances était d'utiliser une structure arboresencte dans laquelle les "branches" correspondait à un chemin possible. Puis nous éliminions les branches les plus "chères" jusqu'à garder celle qui avait le poids le plus petit, ce correspondait au chemin le plus court.

Une autre facon était de reprendre les contraintes d'une modélisation en programmation linéaires. Nous testions les chemins possibles en en gardant dans des tableaux d'indicateurs les sommets et les arcs déjà visité afin d'éliminer les arcs ou les noeuds déjà visités par un chemin optimal. A la fin nous aurions eu le chemi nle plus court avec l'assurance que chaque noeuds aurait été visité une seule fois.

Le code que nous avons concernant le calcul de distance resemble à l'idée des chemisn et des tableaux d'indicateurs. Mais il n'est pas vraiment fonctionnel et nous avons malheursement pas pu le terminer.

III – Partie 3 : Analyse

3.1 Compléxité

Le voyageur de commerce est un problème NP-difficile, ce qui signifie qu'il peut coûter tres cher en temps et en mémoire selon l'instance. Dans notre projet nous avons beaucoup de structures de données différentes, souvent avec des tableaux pour conserver tous les indices des regroupements ou des points de pompages. Ceci coûte assez cher en mémoire. Par exemple, si il y a 50 lieux, alors il y aura potentiellemnt 2^{50} regroupements possible que nous devrons tous stockés (avec notre implémentation actuelle).

En terme de temps, nous avons beaucoup d'algorithmes qui ont des boucles pour imbriqués. Desfois nous parcourons l'ensemble des regroupements. Comme dit plus haut, cela peut vite coûter très cher.

Pour le calcul des distances. On sait que la contrainte des sous-tours génère un nombre exponentiel de contraintes. En passant par des algorithmes de force brute l'implémentaion peut changer beaucou pde choses. Comme nous n'avons pas terminé cette partie nous ne pouvons pas nosu exprimer pleinement sur le sujet mais s'il aurait fallu tester tous les sous-tours possible dans des boucles imbriquées, comme nous avions commencé à le faire, cela aurait pris un temps énorme sur certaines instances.

3.2 Performances

Comme nous n'avons pas réussit à écrire l'algorithme pour le calcul des distances, nous n'avons malheuresement pas pu faire des mesures de performances. Nous avions mis, dans notre fichier principal résolvant le problème, des macros pour calculer le temps écoulé entre l'instanciation des structures de données avec la résolution et l'affichage des résultats. Nous aurions ainsi pu avoir une idée précise du temps qu'il aurait fallu pour résoudre le problème selon les différentes instances proposées (de taille 10, 20, 30, ...).

Cependant, étant la compléxité de nos algorithmes et la compléxité qu'aurait pu avoir notre algortihme de force brut, il est fort probable qu'à partir d'insatnces de taille moyenne (15, 20, ..), le temps de calcul soit plutot long.

3.3 Améliorations et difficultés

Une des première améliorations que nous pourrions ajouter au projet est evidemment de propsoer un algortihme de force brute optimisé pour calculer efficacement les distances de chaque regroupements.

Ensuite nous pourrions réduire le nombre de données stockées car certaines ne servent que d'intermédiraires entre différents algorithmes. Cela aurait pour effet de ne pas trop prendre trop de mémoire sur des instances plus grandes.

Enfin, certains de nos alogrithmes pourraient être encore améliorer afin d'être plus rapide. Par exemple notre algorithme pour déterminer les sous-tours perd un peu de temps lorque nous enlevons les regroupements qui coûtent trop cher. Nous repassons en revus un ensemble de point de pompage alors que nous pourrions faire ce parcours dans les boucles calculant ces dit regroupements.

Nous avons eu du mal parfois à détermienr quel structures de données pourraient être les plus pratiques pour résoudre simplement le problème. Nous nous sommes parfois perdu entre nos différents algorithmes et structures de données. Une autre améliorations possible serait d'éclaircir toutes nos structures de données et nos algorithmes.

IV – Conclusion

Finalement, avec un problème tel que le voyageur de commerce nous avons nous rendre compte de la difficultés que certains porblème d'optimisation peuvent engendrer. Nous sommes désolé de ne pas avoir terminé le projet correctement. Avoir un algorithme pour calculer les distances serait une des premières choses que nous ajouterions dans le projet. Cependant, cela nous aussi permis d'avoir un petit apercu du nombre de calcul et de la compléxité de certains algorithmes derrière de grosse applications (calcul d'itineraires,etc...).