Compte-rendu de projet de Recherche Opérationnelle

Nathan Salaun, Alexandre Boudine

2016-2017

Table des matières

1 Introduction **2**

2 Le problème à résoudre **2**
2.1 Explication du problème 2
2.2 Variables du problème 2
2.3 Fonctionnement de notre programme 2

3 Première étape : les regroupements **3**
3.1 Qu'est-ce qu'un regroupement ? 3
3.2 Notre algorithme 3

4 Deuxième étape : les permutations **4**

5 Troisième étape : résolution du problème **5**

6 Conclusion **5**

1 Introduction

Dans le cadre du cours de Recherche Opérationnelle de notre Licence Informatique, il nous a été demandé de réaliser un projet regroupant toutes les compétences du module. Ce projet consiste en la modélisation puis en la résolution d'un problème conséquent, expliqué dans la section suivante.

Le but premier étant d'utiliser la librairie C de `glpk`, le rendu final est sous la forme d'un fichier source à compiler puis exécuter avec les jeux de données fournis avec le sujet. Nous avons cependant décidé de partir sur du code C++ car il inclut des fonctionnalités utiles pour nos algorithmes. Comme le C++ est rétro-compatible avec le C, nous n'avons eu aucun souci à utiliser la librairie `glpk` ni même à adapter le code squelette donné avec le sujet.

2 Le problème à résoudre

2.1 Explication du problème

Dans ce sujet nous devons optimiser le déplacement d'un drône dans un univers post-apocalyptique. Celui ci est capable de récupérer de l'eau et de le ramener à la base, cependant, il possède une capacité limitée.

Nous avons à disposition un ensemble de puits situés à des distances variables, le déplacement du drône entre les différents puits est donc plus ou moins coûteux. De plus, tous les puits ne possèdent pas la même quantité d'eau disponible.

Le but est évidement de récupérer toute l'eau disponible dans tout les puits tout en minimisant le coût de déplacement du drône.

2.2 Variables du problème

Le problème part de 4 variables :
— `n` : nombre total de points de collecte, sans compter la base (point 0)
— `Ca` : capacité maximale du drone
— `v[i]` : volume d'eau présent dans le point `i` (variables `d1,d2,d3,...,dn` du sujet)
— `c[i][j]` : distance entre les points `i` et `j`

2.3 Fonctionnement de notre programme

Notre programme de modélisation et de résolution du problème est divisé en trois parties, en suivant les indications du sujet.

Premièrement, notre algorithme génère les regroupements (chemins) de points de collectes dont la capacité ne dépasse pas celle du drône.

Ensuite, nous effectuons des permutations pour voir quel est le coût (distance) minimum de chaque regroupement. En effet, certains chemins sont plus

rapides que d'autres si on inverse les points de collecte dans un seul regroupement. Il nous faut donc trouver quel est le chemin le moins cher dans chaque regroupement.

Enfin, un programme GLPK s'occupe de faire un partitionnement pour ordonner tous les regroupements et parcourir la carte pour prendre chaque point de collecte une fois. Il donne donc les groupes de tournées les plus optimaux.

3 Première étape : les regroupements

3.1 Qu'est-ce qu'un regroupement ?

Comme expliqué dans la section précédente, la première étape dans la résolution de notre problème est la génération des regroupements. Un regroupement est une liste non ordonnée de points de collecte. Chaque regroupement doit représenter un "chemin" qui part de la base (ici nommé point 0) et qui y retourne (la base n'est pas représentée dans le regroupement).

Comme on considère que le drone prend toute l'eau de chaque point de collecte, la somme des capacités de chaque point d'un regroupement doit être inférieure ou égale à la capacité maximale du drone.

3.2 Notre algorithme

Nous avons écrit un algorithme qui génère tous les regroupements possibles (donc qui respectent les contraintes énumérées ci-dessus) en fonction des variables du problème. Plus `n` est grand, plus il y aura de regroupements, et plus `Ca` est grand, plus les regroupements seront longs.

Par exemple, avec les variables suivantes :
— `n` = 5
— `Ca` = 10
— `v` = {2, 4, 2, 4, 2}

On obtient les regroupements suivants :
— {1}
— {2}
— {3}
— {4}
— {5}
— {1, 2}
— {1, 3}
...
— {2, 4}
— {2, 5}
...
— {3, 5}
— {4, 5}

...
— {1, 2, 3}
— {1, 2, 4}
...
— {2, 4, 5}
— {3, 4, 5}
— {1, 2, 3, 5}
— {1, 3, 4, 5}

Le fonctionnement de l'algorithme est simple : il commence par remplir la liste des regroupements avec les `n` premiers regroupements, de taille 1 (ici {{1}, {2}, {3}, {4}, {5}}).

Puis, il itère sur chaque regroupement de la liste en partant du premier. Si le dernier élément du regroupement (appelé `j`) est inférieur à `n`, alors il ajoute à la liste les regroupements suivants : pour chaque `k` de `j+1` jusqu'à `n`, il copie le regroupement actuel en ajoutant `k` à la fin. Il vérifie également que le coût du nouveau regroupement ne dépasse pas la capacité du drone, auquel cas il n'est pas ajouté. Une table des capacités est utilisée pour associer chaque regroupement à sa capacité, et ainsi ne pas tout recalculer à chaque fois.

La boucle qui itère sur tous les regroupements est responsable de l'arrêt de l'algorithme. Comme des nouveaux regroupements sont ajoutés dans la boucle, on ne peut pas prédire combien de tours elle fera, chaque regroupement ajouté ajoutant également un tour de boucle. L'algorithme s'arrête quand tous les regroupements ont atteint la taille ou la capacité maximale : à ce moment, plus aucun regroupement n'est ajouté et la boucle arrive à la fin de la liste.

4 Deuxième étape : les permutations

Chaque regroupement doit être ensuite associé à son coût (la somme de toutes les distances lorsqu'on part de la base, parcourt le chemin du regroupement puis qu'on revient à la base).

Nous avons posé précédemment qu'un regroupement était non ordonné, ce qui signifie que, par exemple, les regroupements {1, 2, 3} et {2, 3, 1} étaient identiques. Seulement, leur coût n'est pas le même : il faut donc effectuer toutes les permutations possibles sur chaque regroupement afin de voir quel ordre est le moins cher à parcourir.

Nous utilisons la fonction `next_permutation` de la bibliothèque standard `algorithm` de C++ afin de réaliser nos permutations ; on commence par calculer le coût de notre regroupement de base, puis pour chacune de ses permutations, si son coût est plus petit que celui gardé précédemment alors on garde celui-ci à la place. Cela permet de calculer le coût minimum du regroupement.

5 Troisième étape : résolution du problème

Pour résoudre le problème à partir des données que nous avons pu assemblées jusque là, nous procédons à la mise en place d'un programme GLPK.

L'objectif de ce programme est de déterminer quelles tournées (groupes de puits) doivent être effectuées par le drône pour minimiser le coût total, tout en passant par tous les puits.

Pour ce faire, nous établissons un tableau dans lequel chaque ligne représente un point de collecte (puit) et chaque colonne représente une tournée possible (toutes les tournées qui ont étés déterminées par la partie 1). Le coût minimal déterminé à la partie 2 est associé à chaque variable (tournée).

Enfin, on met en place une matrice creuse dans laquelle on note quels puits sont contenus dans chaques tournées.

L'objectif est de minimiser la somme des coûts des tournées selectionnées tout en faisant en sorte que chaque puit soit contenu une et une seule fois dans l'ensemble des tournées sélectionnées.

6 Conclusion

Ce projet nous a permis de nous rentre compte de l'ampleur du travail lorsqu'il s'agit de s'attaquer à un problème d'envergure. Cela nous a également offert l'occasion de mettre en oeuvre toutes les compétences acquises durant ce module (modélisation, réalisation d'un programme linéaire, adaptation en GLPK puis en C).

Nous aurions cependant pu améliorer notre programme de résolution. Par exemple, nous aurions pu choisir de remplacer l'algorithme de permutations par un problème de type voyageur de commerce, à résoudre via une autre instance d'un problème GLPK (un problème par regroupement).