MAHIER Loïc
JEHANNO Clément

groupe 601B

Rapport de projet [*]: Recherche opérationnelle

[*]rapport réalisé sous LaTeX

Sommaire

1. Introduction

Trumpland étant ravagée il est de notre devoir de survivre face à l'adversité du milieu. Suite à l'écatombe deux pseudos informaticiens tirés de leurs études avant même de commencer leur masters doivent mettre en place une stratégie optimale pour récupérér de l'eau afin de vivre. Le problème est le suivant : un drone possédant une capacité d'eau maximale doit aller visiter plusieurs lieux de pompage et ramener toute l'eau en effectuant un chemin minimum.
Dans le cadre de notre projet nous avons procédés de deux manières : premièrement, afin de récupérer toutes les regroupements possibles par notre drone (nous entendons par possible qui ne dépasse pas la capacité d'eau que peut porter le drone) nous avons fait un algorithme qui calcule tous les regroupements possibles. Ensuite, en connaissant ces regroupements nous avons cherché la permutation qui nous permettait de parcourir ce regroupement de la manière la plus courte possible. Notre deuxième approche fut la manière dont nous avons résolu le problème, une fois toutes nos données récupérées il faut les traiter, pour se faire nous sommes passés par la bibliothèque glpk en c. Notre code est un mélange de c et de c++ car certaines libraires du c++ (next_permutation, vector, etc.) nous sont plus familières mais le besoin du c s'est fait ressentir pour l'utilisation de la libraire glpk.

2. Partie algorithmique

2.1 Algorithme d'énumération des regroupements possible

Ce premier algorithme nous permet de générer tous nos regroupements possibles. Il nous permet donc de générer tous les sous-ensembles de nos lieux qui ne dépassent pas la capacité du drone. Par exemple, le parcours des lieux 1,2,4,5 sera exclu de nos regroupements car ici le volume d'eau est de 12 ce qui dépasse la capacité du drone.

ensembleDesPartiesPossibles

input : Donnees *p
output : vector<vector<int> >
Data : vector<vector<int» subset, vector<int> poid, vector<int> empty

```
1  poid.push_back(0)
2  subset.push_back(empty)
3  for i allant de 0 à p->lieux.size() do
4  |  vector < vector < int >> subsetTemp = subset
5  |  vector < int > poids = poid
6  |  for j allant de 0 à subsetTemp.size() do
7  |  |  poids[j]+ = p− > capacite[i]
8  |  |  if poids[j] <= p -> capaciteDrone then
9  |  |  |  subsetTemp[j].push_back(i + 1)
10 |  |  |  subset.push_back(subsetTemp[j])
11 |  |  |  poid.push_back(poids[j])
12 |  |  end
13 |  end
14 end
15 subset.erase(subset.begin())
16 returnsubset
```

On commence par parcourir tous nos lieux. Pour chaque lieu on ajoute la capacité, si la capacité ne dépasse pas celle du drone alors on ajoute donc notre à vecteur déjà existant le lieu qu'on vient de visiter. Notre vecteur déjà existant contient les lieux qu'il a déjà parcouru et donc qu'il a déjà stocké afin de générer les sous ensembles adéquat. Ensuite, on push_back le vecteur tmp sur lequel on travaille dans notre subset. On ajoute aussi le poids dans le vecteur de poids.

2.2 Algorithme d'énumération des tournées

Une fois que on connait les regroupements possibles il faut savoir quelles sont tournées, autrement dit, l'ordre dans lequel on parcourt chaque regroupement de sorte à faire le moins de chemin possible.

permutationsLesPlusPetites

input : Donnees *p, vector<vector<int> > regroupement, vector<int> &longueurTournee
output : vector<vector<int> >
Data : vector<vector<int> > subset, unsigned int c = 0, int s = 0, vector<int> tmp, int min = INT_MAX

```
1  while c < regroupement.size() do
2      s = 0
3      if regroupement[c][0] != (-10) then
4          s+ = p− > distancier[0][regroupement[c][0]]
5          for i allant de 0 à regroupement[c].size() do
6              s+ = p− >
                distancier[regroupement[c][i]][regroupement[c][i + 1]]
7          end
8          s+ = p− > distancier[regroupement[c].back()][0]
9          if min > s then
10             min = s
11             tmp = regroupement[c]
12         end
13     else
14         subset.push_back(tmp);
15         longueurTournee.push_back(min);
16         tmp.clear();
17         min = INT_MAX;
18     end
19     + + c
20 end
21 returnsubset
```

3. Structures de données

Nous avons repris la structure de données fournie dans le squelette mais l'avons passés en c++ car nous sommes plus à l'aise. Nous avons donc une structure de données qui contient un vecteur de lieux dans lequel seront donc tous nos lieux. Un entier capaciteDrone qui va varier suivant les jeux de données sur lequel nous travaillons. Le vecteur capacité nous permet de savoir combien d'eau stock chaque lieu, le vecteur aura donc en position i la quantité d'eau du lieu i. Notre vecteur de vecteur distancier contient la matrice qui nous est donné en paramètre, c'est à dire la distance pour aller du point i au point j.

4. Analyse des résultats

WHALA ON EST DANS LE KK

5. Améliorations

6. Conclusion