

# Projet : Un problème de tournée de drones dans un contexte de déconfinement

PREKA Bruno, ZELLE Lars Yannick (681C)

29 mars 2021

## 1 Introduction

Dans le contexte sanitaire actuel, un scénario fictif suggère l'isolement *total* des personnes atteintes de la Covid-19 dans leur domicile. Des drones sont alors réquisitionnés pour livrer des courses à domicile.

Dans chaque zone géographique, un unique dépôt (qu'on notera 1 par la suite) stocke des ressources alimentaires et médicamenteuses. Plusieurs contraintes se dessinent dans ce problème appartenant à la classe des problèmes de tournées de véhicules. La contrainte garantissant que chaque client est visité une seule fois, ainsi que celle de la charge supportée par un drone, y interviennent. Le but du problème est de minimiser la distance parcourue par l'ensemble des drones.

On peut résoudre ce problème à travers deux méthodes possibles :

1. Dite *exacte*, elle assimile le problème en un problème d'optimisation combinatoire, mais qui peut demander un temps d'exécution très important aux yeux des décideurs ;
2. *Approchée*, elle est une variante de la méthode de Clark et Wright.

### 1.1 Structures de données générales

#### 1.1.1 Données du problème

A partir de la fonction `lecture_donnees`, on traduit les instances numériques fournies par la structure de données `donnees` dont les champs sont dédiés au nombre de clients `nbClients` (y compris le dépôt), la `capacité` du drone, la `demande` des clients et le distancier `distance`. On retrouvera régulièrement les types suivants :

- `nbClients` et `capacite` sont des entiers (on utilisera le type `Int64`)
- `demande` est un vecteur d'entiers `Vector{Int64}` : la demande du client  $i$  est renseigné dans ce vecteur, ce qui facilitera son accès.
- `distance` est une matrice à coefficients entiers `Matrix{Int64}`. Ce type est équivalent à `Array{Int64,2}`, ce qui sera utile pour manipuler des sous-matrices (restrictions de matrice).

#### 1.1.2 Ensemble des regroupements possibles : $S$ ou $\mathcal{S}$

On rappelle que l'ensemble des regroupements possibles est défini par :

$$\mathcal{S} := \left\{ S \subseteq \llbracket 2, n \rrbracket : \sum_{j \in S} d_j \leq \text{capacite} \right\}$$

Il est clair que la manipulation d'ensembles (type `Set`) est simple, mais elle demande des temps d'exécution plus importants. Il est donc plus que nécessaire de recourir à un "vecteur de vecteurs" au lieu d'un "ensemble d'ensembles", d'où le type `Vector{Vector{Int64}}` pour l'ensemble des regroupements.

### 1.1.3 Ensemble d'ensembles d'indices de tournées visitant le $i$ -ème client : $\text{AllS}_i$ ou $\{S_i \subseteq \llbracket 1, |\mathcal{S}| \rrbracket\}$

La contrainte garantissant que chaque client soit visité une seule fois est donnée par :

$$\sum_{j \in S_i} x_j = 1, i \in \llbracket 2, n \rrbracket,$$

où  $S_i \subseteq \llbracket 1, |\mathcal{S}| \rrbracket$  est le sous-ensemble d'indices des tournées contenant le client  $i$ . Pour les mêmes raisons que ce qui précède, on opte pour un vecteur de vecteurs d'entiers, au lieu d'un ensemble d'ensembles d'entiers, d'où le type `Vector{Vector{Int64}}`.

Pour un client  $i$ , l'accès à l'ensemble des indices des tournées le desservant se fait par  $\text{AllS}_i[i] = S_i$ .

### 1.1.4 Association tournée-distance minimale : 1

La fonction objectif (à minimiser) s'écrit :

$$z = \sum_{j=1}^{|\mathcal{S}|} l_j x_j,$$

où  $l_j$  désigne la distance minimale du regroupement  $j \in \llbracket 1, |\mathcal{S}| \rrbracket$ , calculée par la fonction de résolution du problème de voyageur (dit TSP). Ainsi, une telle expression requiert de lier cette distance à son regroupement via un accès immédiat. C'est pourquoi, le vecteur 1, constitué de couples (*tournée*,  $distance_{\min}$ ), intervient. Il est de type `Vector{Tuple{Vector{Int64}, Int64}}`.

Enfin, en termes d'accès, on a :  $\forall j \in \llbracket 1, |\mathcal{S}| \rrbracket, l_j = 1[j][2]$ .

## 2 Résolution exacte

### 2.1 Fonction d'énumération des regroupements possibles des clients

On appelle cette fonction `getSubsets_recursive(P, S, capacite, demande, index, d, distances)`, où :

- `P` et `toadd` sont les vecteurs des indices à ajouter pour compléter un regroupement, `toadd` étant une copie de `P` ;
- `S` est l'ensemble des regroupements ;
- `capacite` désigne la capacité du drone ;
- `demande` est le vecteur de la demande des clients ;
- `index` est un curseur ;
- `d` est l'accumulateur courant de la demande pour respecter la condition de  $\mathcal{S}$  ;
- `distances` est le distancier.

Derrière cette fonction, l'algorithme consiste à itérer (via `index`) sur toutes les demandes des clients et vérifier si la demande du client  $i \in \llbracket 2, n \rrbracket$ , accumulée à `d`, est conforme à la capacité. Ainsi, pour le client  $i$  :

1. Si les demandes sommées sont en-dessous du seuil, on sauvegarde l'ensemble `P` des clients à ajouter dans `toadd` et on ajoute le client suivant  $i + 1$  (à ajouter) à `toadd`. Sinon, rien n'est fait et on passe à l'étape 4.
2. On construit l'ensemble  $\mathcal{S} = \mathcal{S}$  en concaténant/réunissant `S` et `[toadd]`.
3. On répète le processus sur `S` et le client suivant  $i + 1$  et on compare le nouvel ensemble `Snew` de regroupements : si `Snew` accueille de nouveaux regroupements, alors on les ajoute à `S`.

4. Une fois ce processus récursif terminé sur ce sous-groupe de clients  $\{i, i + 1, \dots, n\}$ , on recommence pour le sous-groupe  $\{i + 1, \dots, n\}$ .

Par conséquent, il en résulte l'ensemble des regroupements attendu, à savoir  $\mathcal{S} = \mathbf{S}$ .

La fonction `getSubsets(capacite,demande,distances)` vient encapsuler la fonction récursive par l'appel `getSubsets_recursive([],[],capacite,demande,1,0,distances)`, avec `index = 1` et la distance initialisée à 0.

## 2.2 Fonction déterminant la tournée et sa distance minimale dans un regroupement $\mathbf{Si}$

On appelle cette fonction `determineShortestCycle(Si,d)`, où :

- $\mathbf{Si}$  est un regroupement de clients à visiter (de type `Vector{Int64}`);
- $\mathbf{d}$  est le distancier (`Matrix{Int64}`).

Puisque  $\mathbf{Si} \subseteq \llbracket 2, n \rrbracket$ , il n'inclut pas le dépôt 1. Ainsi, la réunion avec  $\{1\}$  est primordiale pour passer par la fonction `solveTSPEXact(d)` de résolution du TSP.

De plus, il faut restreindre  $\mathbf{d}$  au regroupement donné, d'où une sous-matrice `newd = d[Si, Si]`. Une telle opération est permise par l'équivalence entre les types `Matrix{.}` et `Array{., 2}`, comme évoqué précédemment.

`solveTSPEXact` retourne alors un couple (*tournée*,  $dist_{\min}$ ). Néanmoins, elle ne raisonne pas sur l'ensemble d'indices  $\mathbf{Si}$ , mais bien sur  $\llbracket 1, |\mathbf{Si}| \rrbracket$ .

Un réajustement des indices doit conserver la cohérence des indices dans les tournées. Pour ce faire, on sait que le premier indice traité par `solveTSPEXact` correspond au premier de  $\mathbf{Si}$ , idem pour le deuxième, ainsi de suite... Puis, dans l'ordre, on a les correspondances suivantes :

- le premier client  $k_1$  de *tournée*  $\longrightarrow$  client d'indice  $k_1$  dans  $\mathbf{Si}$  ;
- ...
- le  $i$ -ème client  $k_i$  de *tournée*  $\longrightarrow \mathbf{Si}[k_i]$
- ...

## 3 Résolution approchée (brève)

## 4 Analyse des résultats issus des instances numériques ( $A$ et $B$ )