Blade Flyer : A la conquête de l'eau

Dylan MONNEAU
Florent GAILLARD

Mars 2017

Sommaire

1 Introduction

Ce projet a pour but de sauver l'humanité de la soif ! En effet, à l'aube d'une crise planétaire pour la survie de l'Homme, différentes communautés doivent se satisfaire en eau par tous les moyens. Les moyens humains étant épuisés, il a fallu trouver une autre solution : **LES DRONES**. Pour ce faire, il en va de soi qu'il faut aller chercher l'eau dans des points de pompage situés aux quatre coins des territoires ennemis ! Le principe étant d'être le plus rapide, non seulement en termes de temps mais surtout en termes de distances. Il faut donc réussir à aller chercher de l'eau dans tous les points de pompage tout en minimisant la distance parcourue ! Mais il faut aussi penser que nous n'avons qu'un seul drone et que celui-ci dispose d'un réservoir de capacité limitée. Il faut donc revenir à la base à chaque fois que le drone n'a plus de place dans son réservoir sachant qu'il est impossible de ne pas prendre l'eau en totalité à un point de pompage. Il faut aussi satisfaire la contrainte suivante : il ne faut aller visiter qu'une et une seule fois *toutes* les bases. Pour éviter toutes sortes de problèmes, tous les points de pompage ont une capacité d'eau dans leur cuve inférieure à celle du réservoir du drone.
Il y a donc trois étapes pour pouvoir parvenir à nos fins :

1. Trouver tous les regroupements de points de pompage possible que l'on peut réunir sans dépasser la capacité du drone.
2. Calculer, pour chacun de ces regroupements, la distance minimum pour tous les visiter.
3. Partitionner les regroupements de telle sorte que l'on visite une et une seule fois un point d'eau et que la somme des distances parcourue pour ces regroupements soit minimale.

2 Composition du projet

2.1 Les fichiers

Dans ce projet, nous n'avons pas décidé de créer plusieurs classes mais bien plusieurs fichiers. En effet, au départ, tout était créé au sein d'un même fichier, il était donc préférable de le découper en plusieurs parties afin de respecter au mieux les étapes à suivre.
Le projet est séparé en cinq parties comme sur l'image ci-dessous pour observer les liens entre les différents fichiers :

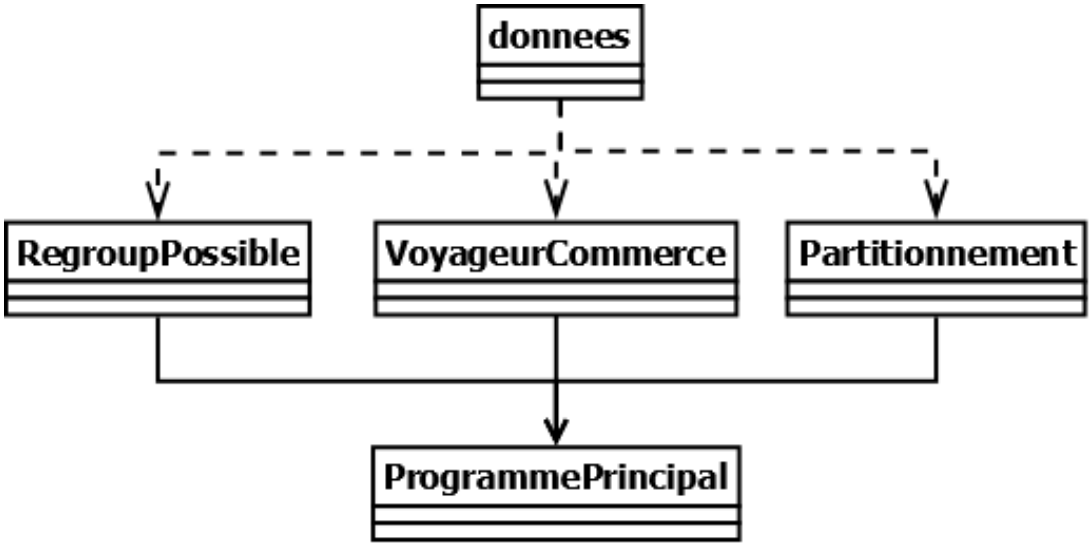

Figure 1: Liens entre les fichiers du projet

Voici à quoi correspond chacun de ces fichiers :

- le programme principal *main* : correspondant au fichier `projet_MONNEAU_GAILLARD.cpp`. Il fait appel à l'ensemble des autres fichiers permettant de résoudre le problème en trois étapes.
- le fichier `donnees.hpp` contient la structure de données recensant celles liées au nombre de lieux total (la base comprise), la capacité du réservoir du drone, la quantité d'eau à prélever à chaque point de pompage (base non comprise) et enfin la distance séparant chacun des points d'eau entre eux.
- les fichiers `regroupPossible.hpp` et `regroupPossible.cpp` permettent de résoudre la première étape du projet, c'est-à-dire de calculer tous les regroupements possibles permettant de satisfaire la capacité du drone.
- les fichiers `voyageurCommerce.hpp` et `voyageurCommerce.cpp` permettent de résoudre la seconde étape du problème. Grâce à la bibliothèque `GLPK`, le calcul de la plus courte distance pour parcourir l'ensemble des points d'eau de chaque regroupement sera aisé.
- les fichiers `partitionnement.hpp` et `partitionnement.cpp` permettent de résoudre le problème de partitionnement des tournées. Comme précédemment, grâce à la bibliothèque `GLPK`, savoir quels regroupements choisir, tout en minimisant la distance totale parcourue, sera un jeu d'enfant.

2.2 Les structures de données

Le projet étant fait en `C/C++` et manipulant énormément de données, il est préférable de bien choisir ses structures de données. En effet, les tableaux vont être de rigueur au sein des fichiers et plus particulièrement les `vector`. L'avantage de ces tableaux `C++` est qu'il n'y a aucune manipulation à effectuer pour libérer et allouer la mémoire. Ceci est produit automatiquement. Comparé au `C` où il faut réserver la mémoire avec

```
void* malloc(size_t nombreOctetsNecessaires);
void* calloc(size_t num_elements, size_t size);
```

puis la libérer avec

```
void free(void *ptr);
```

De plus, un nombre important de fonctionnalités sont déjà implantées au sein des `vector` ce qui permet de les rendre très manipulables. On peut connaître leur taille à tout moment grâce à la fonction `size()` contrairement aux tableaux C. On peut effacer très simplement tout le contenu d'un `vector` avec la fonction `clear()`, ...
Les `vector` sont donc très utiles et très rapides et nous permettent ainsi une conception algorithmique très efficace en termes de temps.

3 Les regroupements possibles

3.1 Principe

Comme décrit dans l'introduction, nous devons connaître tous les regroupements possibles que peut réaliser le drone sans que la capacité de son réservoir ne soit dépassée. Pour cela, nous ajoutons chaque point de pompage, qui formera un regroupement, sans vérification car la capacité en eau d'un point de pompage ne dépasse pas celle du réservoir du drone. Ensuite, pour chaque regroupement formé, nous essayons d'ajouter un nouveau point de pompage, si la capacité totale en eau des deux points d'eau ne dépasse pas celle du réservoir nous l'ajoutons et nous essayons d'ajouter un nouveau point d'eau, en reprenant le regroupement de taille un utilisé précédemment, sinon nous essayons directement

d'ajouter un autre point d'eau. Une fois tous les regroupements de taille deux établis, nous ajoutons tous les regroupements de trois points d'eau en essayant d'ajouter chaque point d'eau au regroupement de taille deux. Nous réitérons cette action tant que la taille des regroupements potentiels ne dépasse pas le nombre de points d'eau.

3.2 Description de l'algorithme

Pour obtenir tous les regroupements possibles, nous disposons donc d'une fonction appelée `regroupementPossible()` et qui prend en paramètre les données du problème. Nous avons deux variables importantes dans cette fonction :

- *result* est un vecteur de vecteur d'entiers, elle correspond à la structure de données qui recueillera tous les regroupements possibles
- *vec* est un simple vecteur d'entiers qui contiendra un regroupement à ajouter ou non, selon si la capacité du drone est suffisante pour recueillir l'eau de tous les points de pompage.

Pour décrire l'algorithme plus précisément, nous allons prendre un exemple avec quatre points de pompages, le point de pompage 1 ayant une capacité de 4 en eau, le 2 de 2, le 3 de 2 et le 4 de 1, et une capacité pour le réservoir du drone de 5.
La première étape de l'algorithme est d'ajouter tous les points d'eau de taille 1, donc nous prenons chaque point de pompage, nous l'ajoutons au vecteur *vec* et nous l'ajoutons à *result*, entre chaque ajout nous n'oublions pas de vider *vec*. Après cette étape, le contenu de *result* est le suivant :

0	1	2	3
[1]	[2]	[3]	[4]

La deuxième étape est d'essayer d'ajouter chaque regroupement de taille 2, 3, ..., la taille d'un regroupement ne peut être supérieure au nombre total de points d'eau, ici 4. Nous reprenons donc chaque regroupement créé et nous essayons d'ajouter un nouveau point d'eau. Pour cela, nous ajoutons une variable *début* et *fin* qui "pointeront" respectivement sur le début et la fin des regroupements créés précédemment. Pour le regroupement composé du point d'eau 1, nous essayons donc successivement d'ajouter les points d'eau 2, 3 et 4 en observant si la capacité du drone permet de récupérer l'eau de tous les points d'eau. Nous pouvons donc ajouter le regroupement composé de 1 et 4, car les regroupements composés de 1 et 2, et 1 et 3 ne permettent pas au drone de récupérer toute l'eau. Nous refaisons pareil pour le regroupement composé du point 2 mais en essayant d'ajouter les points 3 et 4 car le regroupement 1,2 correspond déjà au regroupement 2,1. A la fin de cette étape, le vecteur *result* est comme ci-dessous :

0 (debut)	1	2	3 (fin)	4	5	6	7
[1]	[2]	[3]	[4]	[1,4]	[2,3]	[2,4]	[3,4]

Maintenant *debut* et *fin* vont avoir les valeurs 4 et 8, et nous recommençons l'étape que nous venons de réaliser pour former les regroupements de taille 3, puis nous recommençons pour ceux de taille 4. Nous obtenons à la fin de l'algorithme le résultat suivant dans *result*.

0	1	2	3	4	5	6	7	8
[1]	[2]	[3]	[4]	[1,4]	[2,3]	[2,4]	[3,4]	[2,3,4]

Nous retournons à présent le vecteur *result* car tous les regroupements possibles ont été créés.

4 Le voyageur du commerce

4.1 Définir le problème

Le voyageur du commerce est un problème qui permet de résoudre le parcours de plusieurs points une et une seule fois avant de revenir au point de départ. Le fait est qu'il faut aussi minimiser la distance totale du parcours. C'est justement de cela dont nous avons besoin. En effet, maintenant que nous avons récupéré l'ensemble des regroupements possible que le drone peut effectuer en une seule fois sans dépasser la capacité de son réservoir, on peut modéliser ce problème. Il peut être représenté grâce à un graphe non orienté. Les sommets de ce graphe correspondent donc aux points de pompage à aller visiter et les arêtes représentent le chemin d'un point de pompage i à un autre point j. Ces arêtes ont un poids correspondant à la distance entre deux sommets.
Étant donné que nous avons affaire à des arêtes, faire le chemin du sommet 1 pour aller au sommet 2 est exactement la même chose que du sommet 2 pour aller au sommet 1. Il faut imaginer qu'au départ, nous avons un graphe complet (c'est-à-dire que pour tout sommet i, je peux rejoindre tout sommet j). Ensuite, c'est grâce aux différentes contraintes que nous allons énoncer que l'on peut étayer le graphe en supprimant des arêtes. Nous allons donc avoir un problème du voyageur du commerce à résoudre pour chaque regroupement que nous trouvé à la première étape.

4.2 Les variables de décision

Pour modéliser ce problème, nous avons besoin de variables. Étant donné, que nous avons un graphe complet, il en va de soi qu'il faut savoir si le drone passera par telle ou telle arête. Nous pouvons donc définir les variables :
x_{ij} $\forall i, j \in \{points\ de\ pompage\ du\ regroupement\ base\ comprise\}$.
Le regroupement correspond à un regroupement possible défini dans la première étape.
$x_{ij} = 1$ si l'arête du sommet i vers le sommet j existe, 0 sinon. On en déduit donc qu'il s'agit de variables binaires bornées entre 0 et 1 : $0 \leq x_{ij} \leq 1$.
Le nombre de variables dépend du nombre de points d'eau (la base comprise) intervenant dans le regroupement analysé pour le problème. En temps normal, on pourrait dire que nous avons $n * n$ variables avec $n = nombre\ de\ point\ d'eau$. Cependant, nous avons dit précédemment que x_{ij} était équivalent à x_{ji}. Prenons un exemple avec les points de pompage {0,1,3}. Nous aurions dans un premier temps les variables : $\begin{pmatrix} x_{00} & x_{01} & x_{03} \\ x_{10} & x_{11} & x_{13} \\ x_{30} & x_{31} & x_{33} \end{pmatrix}$ Les variables en rouge sont donc inutiles : $\begin{pmatrix} x_{00} & x_{01} & x_{03} \\ & x_{11} & x_{13} \\ & & x_{33} \end{pmatrix}$
Appelons t le nombre de points d'eau (la base comprise) à aller visiter. La formule pour connaître le nombre de variables est la suivante : $nbVar = \frac{t*(t+1)}{2}$
On pourrait même aller plus loin en disant que les variables du style x_{ii} sont inutiles étant donné qu'on ne peut visiter qu'une seule fois un point d'eau et qu'il serait absurde de revenir directement sur un point que nous venons tout juste de vider.

4.3 Les contraintes

Pour revenir à notre graphe initial complet, il faut spécifier que l'on peut arriver à chaque point d'eau puis en repartir. Le nombre de fois que doit apparaître la variable x_{ij} dans le problème doit donc être égal à 2. Pour remédier au problème de sous-tours de taille 1, c'est-à-dire de partir du point i pour revenir sur ce dernier, nous ajouterons ces contraintes directement. Nous pouvons d'ores et déjà connaître le nombre de contraintes associées au problème. Nous savons qu'il y en a une par point d'eau et nous venons de dire qu'il fallait ajouter les sous-tours de taille 1 pour chacun d'entre eux. Nous avons donc $nbContraintes = 2 * t$ avec t le nombre de points d'eau dans le regroupement avec la base comprise.

Algébriquement parlant, les contraintes sont modélisées de la façon suivante :

$$\sum_{i \in regroupement} x_{ij} = 2 \quad \forall j \in \{regroupement\}$$

Pour exprimer les contraintes liées aux sous-tours de taille 1, c'est assez simple. En effet, il faut que les variables x_{ii} soient égales à 0. Soit :

$$x_{ii} = 0, \quad i \in \{regroupement\}$$

4.4 La fonction objectif

Nous l'avons dit dès le départ, l'objectif est de minimiser la distance totale parcourue afin de visiter l'ensemble des points d'eau. Pour cette fonction objectif, nous avons donc besoin de la matrice qui est présente dans le fichier de données. Grâce à `GLPK`, nous serons à même de savoir quelles arêtes auront été choisies et donc quelles distances associées à celle-ci pour calculer la distance totale parcourue par le drone. Si nous devions exprimer cette fonction en français, cela serait : minimiser la distance totale parcourue. Mathématiquement, cela donne ceci :

$$min\ z = \sum_{i \in regroupement} \sum_{j \in regroupement} x_{ij} * c_{ij}$$

où c_{ij} correspond à la distance séparant le point i du point j.

4.5 La matrice creuse

Parlons maintenant un peu plus technique en évoquant la matrice creuse qu'il faut remplir afin de résoudre le problème avec la bibliothèque `GLPK` en `C/C++`. La matrice à une taille de $n * m$ avec $n =$ nombre de contraintes et $m =$ nombre de variables. Pour remplir cette matrice, nous avons besoin de trois paramètres, plus précisément de trois tableaux. Le premier correspond aux lignes de la matrice, c'est-à-dire le numéro de ligne sur lequel un 1 va apparaître. Le second paramètre correspond aux colonnes de la matrice. Et enfin le dernier désigne les valeurs que prend la matrice aux positions indiquées par les lignes et les colonnes. Le plus difficile dans tout cela, reste à identifier les valeurs que prendra le tableau lié aux colonnes. En effet, il faut trouver où se situeront les 1 dans la matrice en fonction des variables (représentant les colonnes). En fait, l'objectif de la matrice creuse est de représenter les contraintes. De ce fait, il faut identifier les variables impliquées dans la contrainte pour retrouver quelles colonnes seront marquées d'un 1.
Pour bien illustrer la matrice, prenons encore un exemple. Soit le regroupement $r = \{0, 1, 3, 5\}$. La matrice correspondante à ce problème de voyageur de commerce pour ce regroupement serait :

	x_{00}	x_{01}	x_{03}	x_{05}	x_{11}	x_{13}	x_{15}	x_{33}	x_{35}	x_{55}
1	1	1	1	1						
2		1			1	1	1			
3			1			1		1	1	
4				1			1		1	1
5	1									
6					1					
7								1		
8										1

Les quatre premières lignes correspondent aux contraintes liées aux degrés entrant et sortant de chaque sommet. Les quatre lignes suivantes correspondent aux sous-tours de taille 1.

4.6 Le problème des sous-tours

Le seul problème qu'il nous reste correspond aux sous-tours de taille supérieure à 1. Pour comprendre, il faut visualiser ce qu'est un sous-tour, prenons le regroupement : {0,1,2,3,4,5,6}. Voici à quoi pourrait ressembler deux sous-tours :

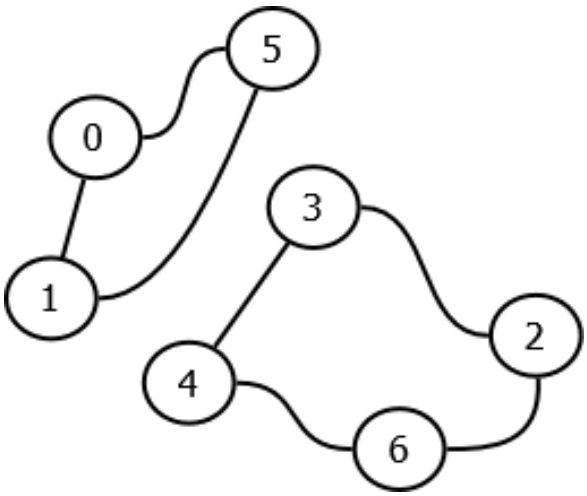

Figure 2: Deux sous-tours pour le regroupement {0,1,2,3,4,5,6}

Pour ce faire, pour un regroupement donné comportant au moins un sous-tour, il faut résoudre le problème tant qu'il en reste. Il faut donc identifier les sous-tours, les arêtes sont impliquées et les briser. Pour briser un sous-tour, il faut ajouter la contrainte suivante :

$$\sum_{i,j \in sousTours} x_{ij} \leq |sousTours| - 1$$

avec $sousTours$ correspondant à l'ensemble des variables inclus dans le sous-tour que l'on veut casser. Lorsque, pour un regroupement, on constate un sous-tour, il faut résoudre à nouveau le problème en incluant de nouvelles contraintes. Le nombre de lignes dans la matrice (correspondant aux contraintes) augmente. Il y a autant de nouvelles contraintes que de sous-tours. La dernière chose qu'il reste à faire est de sauvegarder ces nouvelles contraintes si jamais, après une deuxième tentative, la résolution du problème constate de nouveaux sous-tours et ainsi de suite.

5 Le partitionnement des tournées

5.1 Définir le problème

Maintenant que nous avons à la fois les regroupements possibles, ainsi que les distances minimales à parcourir pour chacun d'entre eux, il ne nous reste plus qu'à savoir quelles tournées seront établies en une seule fois. Le problème est simple, tous les points de pompage ne seront visités qu'une seule fois. Il suffit juste de partitionner les différents regroupements précédemment trouvés pour minimiser la distance totale parcourue sur l'ensemble des visites des points d'eau.

5.2 Les variables de décision

Ayant toutes les tournées réalisables, il nous faut choisir si le regroupement en question est envisageable ou non. On en déduit qu'il nous faut des variables de décision binaires permettant de dire si oui ou

non, nous gardons ce regroupement. Appelons x_i ces variables. Elles sont de types binaires et sont bornées supérieurement par 1. Elles sont bien entendues non négatives : $0 \leq x_i \leq 1$.
Le nombre de variables dépend du nombre de tournées possible, c'est-à-dire de ce que nous avons calculé lors de la première étape. Appelons T, le nombre de regroupements possibles satisfaisant la capacité du drone. On a donc $nbVar = T$.

5.3 Les contraintes

On a dit qu'on ne devait visiter qu'une seule fois un point d'eau. Cela veut dire que si l'on choisit un regroupement contenant le point de pompage p, on ne pourra plus choisir une autre tournée contenant ce point p. Nous devons donc répertorier dans quelles tournées (indicées de 1 à T) apparaît p. Une fois que nous avons récupéré l'ensemble des indices des tournées contenant p (appelons le **P**), nous pouvons en déduire les contraintes suivantes :

$$\sum_{i \in P} x_i = 1 \quad \forall p \in \{lieux\}$$

avec $lieux$, l'ensemble des points de pompage à aller visiter.

5.4 La fonction objectif

La fonction objectif reste à minimiser la distance parcourue par le drone tout en visitant tous les points de pompage. Elle dépend donc des distances minimales de chaque regroupement. Appelons-les l_i pour les regroupements d'indices i. Rappelons que le nombre de regroupements est identifié par T Elle se caractérise par :

$$min\ z = \sum_{i=1}^{T} x_i * l_i$$

6 Impact sur la taille des données

Ce projet permet de manipuler énormément de données. En effet, dans l'exemple, nous n'avions que 6 points d'eau au total. Cependant, lorsque nous nous trouvons face à un fichier de données où le nombre de points d'eau à visiter avoisine les 50, il y a de nombreuses possibilités de regroupements et donc d'importants calculs.

6.1 Ensemble des parties d'un ensemble

Imaginons que la capacité du drone soit infinie, le nombre de regroupements possible correspond donc à l'ensemble des parties de l'ensemble des points d'eau. On sait que le cardinal de l'ensemble des parties d'un ensemble est : $Card(\mathcal{P}(E)) = 2^{|E|}$. On a donc pour un ensemble de 50 éléments, $2^{50} =$ 1'125'899'906'842'624 éléments ! Pour une machine, cela commence à devenir très compliqué de calculer autant de résultats. Cette dimension doit donc entrer en compte pour la conception de l'algorithme des regroupements possibles. En effet, il en va de soi qu'il faut trouver des techniques d'élimination de possibilités afin de réduire au mieux le temps de calcul.
Nous concernant, pour le fichier test de données, nous avons un temps de calcul nul, ceci est instantané. Jusqu'à des données d'une taille de 35 lieux à visiter, le calcul de regroupements prend environ 1/100ème de seconde. Au-delà, on passe à environ 1/10ème de seconde pour des données de taille 40 puis à 1/4 de seconde pour 50 données. Ceci reste raisonnable; sachant que la capacité du drone n'est pas infinie et qu'elle ne permet de visiter que jusqu'à 5 ou 6 points d'eau à la fois.

6.2 Le solveur GLPK

Concernant les deux dernières étapes, c'est-à-dire le voyageur du commerce et le partitionnement des tournées, voyons la performance grâce à GLPK. Il faut savoir que l'on applique le voyageur du commerce sur chacun des regroupements répertoriés lors de la première étape. De plus, on l'applique de nouveau pour chaque sous-tour découvert. Pour le fichier de 50 données, nous arrivons d'un peu plus de 3,5 secondes. Cela reste largement raisonnable étant donné le nombre de sous-tours qu'il peut se créer. Pour donner un ordre d'idée, pour un nombre de lieux égal à 50, nous disposant d'un nombre de regroupements avoisinant les 44000.
Quant au problème lié au partitionnement des tournées, c'est ce qu'il y a de plus long dans l'ensemble du programme. En effet, on tourne autour des 15 secondes pour le fichier de 50 données.

7 Conclusion

En conclusion de ce projet, nous pouvons dire qu'il est riche et complet. En effet, en plus de résoudre un problème, il faut tout d'abord le décomposer pour obtenir des éléments qui seront la clé de la résolution. Il ne s'agit pas seulement d'obtenir un résultat pour chaque étape, mais bien d'obtenir et de comprendre l'ensemble des éléments qui permettront la résolution finale du problème. La mise en œuvre du langage C/C++ articulé autour de la bibliothèque GLPK est nécessaire comparé à certains algorithmes qui ralentiraient considérablement l'exécution du programme. Par exemple, au lieu d'utiliser GLPK pour résoudre le voyageur du commerce, on aurait pu écrire un algorithme mais sans avoir la certitude que celui-ci soit efficace sur d'importantes données.