L3 Informatique
X32I030 Recherche Opérationnelle

Université de Nantes
2022/2023

Feuille de travaux pratiques n^o 3
Aller sans retour ?

1 Contexte

Vous avez toujours voulu être un pionnier explorant des lieux inexplorés et cela va finalement être possible ! Grâce à votre suivi des bons plans, vous avez obtenu un aller simple vers la planète Mars pour un prix modique ! Les photos dans la publicité sont trop belles [1] et vous vous imaginez déjà visiter glorieusement ces superbes paysages. Malheureusement, vous n'avez pas fait attention aux détails dans votre achat impulsif, en particulier le point 5 : "ship refilled on mars using local resources". E. Musk (le vendeur) a fait le choix de fournir tous les détails techniques et juridiques sur twitter, mais il faut un compte twitter blue pour y avoir accès.

Heureusement, toutes les informations sont parfaitement expliquées dans une vidéo en cours de vol. Tout d'abord, le voyage dure 6 mois ! Cela implique qu'une grande partie de la nourriture et de l'oxygène disponibles seront consommés pendant le voyage. Ensuite, il n'y aura quasiment plus de carburant pour un vol retour. La partie touristique sur place s'annonce donc de courte durée avant une mort certaine, pour laquelle SpaceX décline toute responsabilité. Pour survivre, il faudra renoncer au tourisme, et récupérer des ressources à transformer, pour produire des aliments, de l'oxygène et du carburant pour un futur retour.

Pendant le vol, un passager a travaillé à partir d'images de Mars afin d'identifier le point d'atterrissage, la topologie des lieux, et des points d'intérêt environnants. On peut lui faire confiance, il est passé par le parcours VICO, et il semble certain du résultat de son algorithme de traitement d'images. Malheureusement, un problème linguistique vous empêche de prendre connaissance des données issues de ce traitement : vous ne parlez pas la même langue ! Ce passager est en effet chinois ! Heureusement, un autre passager passionné par les langues peut vous aider. D'ailleurs, s'il a choisi de faire ce voyage, c'est justement pour apprendre de nouvelles langues, éventuellement inconnues sur Terre. Il s'agit d'un diplômé du parcours ATAL, il est parfaitement fiable sur ce sujet, et on fera l'hypothèse qu'il n'y a pas de perte d'information dans sa traduction. Il reste à faire des prévisions réalistes sur la consommation de ressources et les restrictions éventuelles pour la survie. En se basant sur la consommation de l'équipage en vol, un diplômé du parcours DS a proposé un modèle probabiliste. Les besoins en oxygène apparaissant comme étant critiques d'après ce modèle, un sous-ensemble de points d'intérêt sera donc considéré de manière prioritaire. Afin de se rendre dans les points d'intérêt et d'y récupérer les ressources, des drônes dont l'autonomie est limitée sont disponibles et leur programmation est assuré par un diplômé du Master CORO (ECN). Un SGBD a été mis en place par un diplômé du parcours ALMA. Tout ce qu'il reste maintenant à faire, est de planifier les trajectoires des drônes vers les points d'intérêt, ainsi que leur retour à la base tout en minimisant le temps de trajet, car les drônes nécessitent également des ressources. Cette mission sera pour le diplômé du parcours ORO ! C'est quand même incroyable tous ces diplômés nantais dont les compétences sont complémentaires ! Toutes ces formations de qualité réunies dans une même ville, les étudiants nantais ont bien de la chance !

Il y a cependant un problème : personne ne trouve le diplômé du parcours ORO. Vérification faite : Il n'est pas du matin et il a loupé le décollage. Il ne reste donc que vous, (futur) diplômé de licence, pour vous charger de la partie optimisation. Un dernier détail : les ressources étant limitées, il a été décidé de larguer dans l'espace tous les passagers inutiles. En fait, ça a déjà commencé... Vous n'avez donc pas droit à l'échec !

Il est donc temps de prendre connaissance en détail de la tâche qui vous revient. Ce projet est un sous-problème du problème de planning dynamique de mission pour une flotte de drônes d'exploration [2]. Ce problème complexe

1. `https://www.spacex.com/human-spaceflight/mars/`
2. Ce problème est décrit dans de multiples références :

Brummit, B. & Stentz, A. (1996). Dynamic mission planning for multiple robots. *Proceedings of the IEEE international conference on robotics and automation,* April 1996.

R. Alyassi, M. Khonji, A. Karapetyan, S. Chau, K. Elbassioni, C. Tseng. Autonomous Recharging and Flight Mission Planning for Battery-operated Autonomous Drones. IEEE Transactions on Automation Science and Engineering, 2022.

se situe à l'intersection de plusieurs domaines scientifiques : robotique, sciences des données, recherche opérationnelle.

À chaque drône est affecté un sous-ensemble de points d'intérêt. Chaque drône part d'une base (permettant de l'entretenir et de recharger sa batterie) et va ensuite explorer les points d'intérêt lui étant affectés avant de rentrer à la base tout en cherchant à minimiser le temps total/la consommation énergétique nécessaire pour cela.

Le planning initial (affectation de sous-ensembles de point d'intérêt à des drônes, puis définition pour chaque drône de l'ordre d'exploration de ces points d'intérêt) est réalisé à partir d'estimations des temps nécessaires pour qu'un drône se rende de la base vers un point d'intérêt, ou d'un point d'intérêt vers un autre point d'intérêt. Ces estimations ont été obtenues à l'aide de données partielles, et leur fiabilité pourrait donc être remise en cause. En effet, les informations concernant les zones à parcourir seront complétées pendant les déplacements des drônes. Les images de ces zones vues sous d'autres angles, peuvent permettre de découvrir un relief différent, des obstacles... Les données sont donc complétées en cours de mission. Il est possible de se rendre compte qu'un déplacement d'un point d'intérêt vers un autre, initialement prévu pour un drône, soit finalement beaucoup trop long. Il devient alors judicieux de revoir les affectations des points d'intérêt qu'il reste à explorer, aux drônes en cours de mission (et ensuite les ordres d'exploration des points d'intérêt). C'est pourquoi on parle de planning dynamique d'une mission, et c'est dans ces modifications de planning que la flotte complète est considérée.

Pour ce projet, nous allons simplifier ce problème. Tout d'abord, nous travaillerons sans toutes les incertitudes sur les données. Les temps nécessaires pour se rendre d'un point d'intérêt à un autre seront donc connus de manière précise et définitive. Il n'y aura par conséquent pas de changement en ce qui concerne les affectations des points d'intérêt aux drônes. Chaque drône devra donc explorer un ensemble définitivement précisé de points d'intérêt, avant de rentrer dans sa base. Nous supposerons que l'affectation des points d'intérêt aux drônes est déjà réalisée et nous considérerons le sous-problème consistant à déterminer l'ordre de visite des points d'intérêt par un drône, de manière à minimiser le temps total d'exploration. Nous obtenons donc un problème de voyageur de commerce.

Le problème complet initialement posé est une variante du problème de voyageur de commerce multiple, auquel on ajoute de l'incertitude sur les données. La différence essentielle est qu'il y a plusieurs voyageurs partant de divers sites, et devant visiter de manière collaborative l'ensemble des villes.

La section 2 présentera des modélisations pour le problème du voyageur de commerce. La section 3 présentera ensuite des méthode de résolution. Finalement, la section 4 présentera le travail attendu.

2 Modélisations du problème

Il existe de nombreuses modélisations pour le problème du voyageur du commerce. Nous considérerons uniquement deux modélisations utilisant le paradigme de la programmation linéaire.

Le nombre de lieux à visiter (base, points d'intérêt) est noté n et le temps nécessaire pour aller du lieu i au lieu j est notée c_{ij} (où $i, j \in \{1, \ldots, n\}$). Les variables de décision sont données par

$$x_{ij} = \begin{cases} 1 & \text{si le drône se rend directement du lieu } i \text{ au lieu } j \\ 0 & \text{sinon} \end{cases}$$

où $i, j \in \{1, \ldots, n\}$.
Le modèle présenté en CM a été initialement proposé par Dantzig, Fulkerson et Johnson, et s'écrit de la façon suivante.

$$\begin{aligned}
\min\ z \quad = \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & & & \\
s.c. \quad & \sum_{j=1}^{n} x_{ij} & = & 1 & \forall i \in \{1,\ldots,n\} & (1) \\
& \sum_{i=1}^{n} x_{ij} & = & 1 & \forall j \in \{1,\ldots,n\} & (2) \\
& \sum_{i,j\in S} x_{ij} & \leq & |S|-1 & \forall S \text{ avec } |S| \leq n-1 & (3) \\
& x_{ij} \in \{0,1\} & & & \forall i,j \in \{1,\ldots,n\} &
\end{aligned}$$

Les regroupements (1)-(2) représentent les 2n contraintes du problème d'affectation. Les contraintes du regroupement (3) ont pour but d'interdire les sous-tours, et leur nombre est de l'ordre de 2^n. Dans la suite, nous préférerons utiliser le modèle suivant qui est une légère modification.

$$\begin{array}{rrlll}
\min\ z & = & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \\
s.c. & \displaystyle\sum_{j\in\{1,\ldots,n\}\setminus\{i\}} x_{ij} & = \quad 1 & \forall i \in \{1,\ldots,n\} & (1') \\
& \displaystyle\sum_{i\in\{1,\ldots,n\}\setminus\{j\}} x_{ij} & = \quad 1 & \forall j \in \{1,\ldots,n\} & (2') \\
& \displaystyle\sum_{i,j\in S} x_{ij} & \leq \quad |S|-1 & \forall S \text{ avec } 2 \leq |S| \leq n-1 & (3') \\
& x_{ij} \in \{0,1\} & & \forall i,j \in \{1,\ldots,n\} &
\end{array}$$

La principale différence est que les variables x_{ii} ($i \in \{1,\ldots,n\}$) ont disparu dans les regroupements $(1')$-$(2')$. Elles ne peuvent donc plus être fixées à 1 par ces contraintes. Par conséquent, il n'est plus nécessaire d'interdire les sous-tours de taille 1, d'où la modification obtenue dans le regroupement $(3')$. Cela ne supprime que n contraintes dans le regroupement $(3')$. Cela ne change donc pas l'ordre de grandeur du nombre de contraintes de ce modèle. Il reste impossible de saisir ce modèle directement dans un solveur, sauf pour de petites valeurs de n.

Remarque : les variables x_{ii} ($i \in \{1,\ldots,n\}$) n'ont plus aucun rôle dans ce modèle et pourraient être supprimées. Les garder simplifie cependant l'écriture du modèle.

Nous considérons maintenant un autre modèle faisant intervenir une notion de temps. Ce modèle a été initialement proposé par Miller, Tucker et Zemlin. Des variables de décision additionnelles sont considérées dans ce modèle.

$$t_j \geq 0 : \text{Date à laquelle la ville } j \text{ est visitée.}$$

où $j \in \{2,\ldots,n\}$. Il n'y a pas de variable t_1 puisque le lieu 1 est (arbitrairement) considéré comme le point de départ et le point d'arrivée du cycle. Lui donner une date de visite n'aurait donc pas de sens.

L'idée centrale est maintenant que si nous avons $x_{ij} = 1$ alors le lieu j est visité après le lieu i, nous devons donc avoir $t_j > t_i$. Une inégalité stricte n'est pas autorisée en programmation linéaire, mais comme l'unité de temps n'est pas spécifiée, nous pouvons simplement poser $t_j \geq t_i + 1$. Bien entendu, cette contrainte ne doit être respectée que si $x_{ij} = 1$. Nous obtenons donc la contrainte

$$t_j \geq t_i + 1 - M(1 - x_{ij})$$

où M est une constante prenant une "grande" valeur. En pratique, M peut être fixé à n puisque nous avons n dates consécutives à fixer et que ces dates peuvent tenir dans l'intervalle $[0, n-1]$. Nous obtenons donc finalement les contraintes suivantes :

$$t_j \geq t_i + 1 - n(1 - x_{ij}), \forall i,j \in \{2,\ldots,n\}.$$

Le modèle complet s'écrit donc de la façon suivante.

$$\begin{array}{rrlll}
\min\ z & = & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \\
s.c. & \displaystyle\sum_{j\in\{1,\ldots,n\}\setminus\{i\}} x_{ij} & = \quad 1 & \forall i \in \{1,\ldots,n\} & (1') \\
& \displaystyle\sum_{i\in\{1,\ldots,n\}\setminus\{j\}} x_{ij} & = \quad 1 & \forall j \in \{1,\ldots,n\} & (2') \\
& t_i - t_j + n.x_{ij} & \leq \quad n-1 & \forall i,j \in \{2,\ldots,n\} & (3'') \\
& x_{ij} \in \{0,1\} & & \forall i,j \in \{1,\ldots,n\} & \\
& 0 \leq t_j \leq n & & \forall j \in \{2,\ldots,n\} &
\end{array}$$

Dans ce modèle, l'ordre de grandeur du nombre de contraintes est n^2. Il est donc tout à fait possible d'en faire une saisie directe dans un solveur.

Remarque : les variables x_{ii} ($i \in \{1,\ldots,n\}$) n'ont ici aussi aucun rôle dans ce modèle. Nous pourrons cependant laisser les prétraitements du solveur supprimer ces variables.

3 Méthodes de résolution

Dans ce projet, nous considérerons deux méthodes de résolution exacte basées sur les deux modèles présentés dans la section précédente, et une méthode de résolution approchée basée sur un algorithme glouton.

3.1 Résolution exacte en utilisant le modèle Miller-Tucker-Zemlin

En utilisant le modèle Miller-Tucker-Zemlin, il est possible de résoudre un problème de voyageur de commerce en le saisissant directement dans un solveur. Il restera à tester la limite du solveur par rapport à la taille du problème.

3.2 Résolution exacte en utilisant le modèle Dantzig-Fulkerson-Johnson

Le modèle Dantzig-Fulkerson-Johnson ne peut par contre pas être résolu directement en utilisant un solveur. Nous ne pourrons en effet pas saisir la totalité des contraintes de ce modèle. Nous ne considérerons donc qu'un sous-ensemble de ces contraintes. Nous commencerons uniquement avec les contraintes des regroupements (1')-(2'). Nous résoudrons donc le modèle suivant.

$$
\begin{array}{rcll}
\min\ z & = & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & \\
s.c. & \displaystyle\sum_{j\in\{1,\dots,n\}\setminus\{i\}} x_{ij} = 1 & \forall i \in \{1,\dots,n\} & (1') \\
& \displaystyle\sum_{i\in\{1,\dots,n\}\setminus\{j\}} x_{ij} = 1 & \forall j \in \{1,\dots,n\} & (2') \\
& x_{ij} \in \{0,1\} & \forall i,j \in \{1,\dots,n\} &
\end{array}
$$

Ce modèle est juste un modèle de problème d'affectation, dans lequel les variables x_{ii} ($i \in \{1,\dots,n\}$) sont forcées à la valeur 0. Comme nous l'avons vu en cours, toute solution admissible de ce problème est décrit par une permutation (et inversement). Une permutation peut également être vue comme un produit de cycles disjoints. Par contre, une permutation ne décrit pas nécessairement une solution admissible du problème de voyageur de commerce. Il faut pour cela que le produit de cycles décrivant la permutation ne soit composé que d'un seul cycle.

Dans la suite, nous considérons pour l'exemple le distancier suivant.

	1	2	3	4	5	6	7
1	0	786	549	657	331	559	250
2	786	0	668	979	593	224	905
3	549	668	0	316	607	472	467
4	657	979	316	0	890	769	400
5	331	593	607	890	0	386	559
6	559	224	472	769	386	0	681
7	250	905	467	400	559	681	0

On peut remarquer que ce distancier est symétrique (il ne s'agit bien sûr pas d'une obligation en général, car il pourrait y avoir du relief ou des obstacles).

La résolution nous donne $x_{17} = x_{26} = x_{34} = x_{43} = x_{51} = x_{62} = x_{75} = 1$ avec $z = 2220$ (tous les autres x_{ij} sont nuls). Si nous voyons cette solution comme une permutation, nous obtenons

$$
\begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
7 & 6 & 4 & 3 & 1 & 2 & 5
\end{array}
$$

et comme un produit de cycles disjoints, nous obtenons

$$(175)(26)(34).$$

Nous n'avons pas la chance d'obtenir une solution composée d'un seul cycle. Nous aurons donc besoin de considérer des contraintes incluses dans le regroupement $(3')$.

Il est bien sûr impensable d'ajouter toutes les contraintes du regroupement $(3')$. Nous allons donc à chaque fois ajouter une seule contrainte de manière à casser un sous-tour et résoudre le problème obtenu. Nous espérons ainsi

avoir la chance d'obtenir une solution composée d'un seul cycle. En général, on préfère casser le plus petit cycle du produit.

Si on souhaite casser le cycle (26) du produit, nous devons uniquement ajouter la contrainte

$$x_{26} + x_{62} \leq 1.$$

En résolvant le problème avec cette contrainte additionnelle, on obtient la solution donnée par $x_{17} = x_{25} = x_{34} = x_{43} = x_{56} = x_{62} = x_{71} = 1$ avec $z = 2335$ (tous les autres x_{ij} sont nuls). Si nous voyons cette solution comme une permutation, nous obtenons

$$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 7 & 5 & 4 & 3 & 6 & 2 & 1 \end{array}$$

et comme un produit de cycles disjoints, nous obtenons

$$(17)(256)(34).$$

Nous n'avons pas encore une solution composée d'un seul cycle. Nous cassons alors le cycle (17) en ajoutant la contrainte

$$x_{17} + x_{71} \leq 1.$$

La résolution de ce nouveau problème nous donne alors $x_{15} = x_{23} = x_{34} = x_{47} = x_{56} = x_{62} = x_{71} = 1$ avec $z = 2575$ (tous les autres x_{ij} sont nuls). Si nous voyons cette solution comme une permutation, nous obtenons

$$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 5 & 3 & 4 & 7 & 6 & 2 & 1 \end{array}$$

soit le cycle (1562347) !

Affirmation : Nous avons donc ici obtenu une solution admissible et optimale pour cette instance du problème de voyageur de commerce.

De plus, seules deux contraintes du regroupement (3') ont été nécessaires pour cette résolution.

Nous résumons l'approche utilisée.

- Résoudre le problème en ne considérant que les regroupements de contraintes $(1')$ et $(2')$
- Répéter
  - Décomposer la solution obtenue en un produit de cycles disjoints
  - Si la solution obtenue est composée de plusieurs cycles alors ajouter au problème la contrainte cassant le plus petit sous-tour, puis résoudre le problème obtenu

  Jusqu'à l'obtention d'une solution composée d'un seul cycle

3.3 Résolution approchée par un algorithme glouton

Une méthode de résolution exacte peut demander un temps de résolution important, surtout si le problème est de grande taille. On peut envisager suivant le contexte, d'avoir recours à une méthode de résolution approchée. Définir une solution admissible pour le problème de voyageur de commerce est particulièrement simple, il suffit de définir un ordre de visite des lieux. Bien entendu, nous souhaitons de plus que la longueur du cycle soit aussi courte que possible. L'heuristique la plus simple pour cela est appelée l'heuristique de plus proche voisin. En partant d'un lieu arbitraire, on choisit de manière répétée le lieu le plus proche parmi les lieux qui n'ont pas encore été explorés. Une fois tous les lieux explorés, on revient au lieu initial.

Dans l'exemple de la section précédente, en partant du lieu d'indice 1, on se dirige séquentiellement vers les lieux d'indices 7, 4, 3, 6, 2, 5 avant de revenir au lieu 1. On obtient donc le cycle (1743625) avec $z = 2586$.

Il est intéressant de remarquer que la solution obtenue dépend du choix du lieu de départ.

4 Travail à effectuer

Une implémentation des méthodes de résolution présentées dans la section 3 est attendue en utilisant Julia/-JuMP/GLPK.

Plusieurs fichiers sont disponibles sur madoc dans l'archive `ProjetRO.zip` :
- Le fichier `Projet_NOM1_NOM2.jl` contenant un squelette à compléter,
- Un dossier `plat` contenant des instances numériques dont le distancier est symétrique,
- Un dossier `relief` contenant des instances numériques dont le distancier est asymétrique.

Les instances numériques sont des fichiers textes dont le format est donné par :
- La première ligne indique la taille du problème,
- Les lignes suivantes indiquent le distancier.

Le fichier des données de l'exemple est indiqué ci-dessous.

```
7
0 786 549 657 331 559 250
786 0 668 979 593 224 905
549 668 0 316 607 472 467
657 979 316 0 890 769 400
331 593 607 890 0 386 559
559 224 472 769 386 0 681
250 905 467 400 559 681 0
```

Un parseur pour ces fichiers de données est fourni dans le squelette de code. Les fonctions suivantes seront très utiles pour construire des contraintes que l'on va ensuite ajouter dans un modèle :
- `@expression(m::Model,expr::AffExpr)` : Cette macro retourne une expression (dépendant des variables de décision de `m`) qu'on pourra ensuite utiliser dans le modèle `m`, dans la définition de contraintes comme dans la fonction objectif. Cette macro est TRÈS sensible syntaxiquement. Le paramètre `expr` ne peut pas être une expression vide, ni même une variable de décision seule. Par exemple, `@expression(m,x)` ne retournera pas une expression utilisable alors que `@expression(m,1.0*x)` en retournera bien une.
- `add_to_expression!(aff::AffExpr, new_coeff::Float64, new_var::VariableRef)`: Cette fonction ajoute un terme dans l'expression affine passée en premier paramètre. Par exemple, l'appel à `add_to_expression!(aff,1.0,x2)` ajoute `1.0x2` dans l'expression `aff`.
- `@constraint(m::Model,con)` est la macro déjà utilisée dans les TPs précédents , sauf qu'ici, nous ne donnons pas de nom à la contrainte. Cela a une utilité si on veut utiliser cette macro dans une répétitive, puisqu'il est interdit de donner deux fois le même nom à une contrainte.

Exemple de construction d'une contrainte : on suppose que les variables `x1` et `x2` existent dans le modèle `m`.

`aff = @expression(m,1.0*x1)` crée une expression ne contenant que la variable `x1` et la stocke dans la variable `aff`.

`add_to_expression!(aff,1.0,x2)` ajoute `x2` dans l'expression `aff` qui contient maintenant `x1+x2`

`@constraint(m,aff <= 1)` ajoute la contrainte `x1+x2 <= 1` au modèle `m`.

La date limite de remise des projets est fixée au vendredi 31 mars à 17h. Le code source et le rapport devront être remis sur madoc dans une archive `.tar.gz` ou `.zip`. Le rapport devra impérativement inclure les contenus suivants.
- Une description et une justification des structures de données choisies pour réaliser les implémentations demandées.
- La preuve de l'affirmation qui apparaît en gras dans le texte dans la section 3.2. Plus précisément, en résolvant un problème ne contenant qu'un sous-ensemble des contraintes (3'), si on obtient une solution composée d'un seul cycle alors cette solution est optimale.
- Des pseudo-codes présentant la traduction d'une solution en permutation, puis en produit de cycles disjoints,
- Une analyse expérimentale pour les méthodes de résolution exacte : on considérera séparément les instances de la catégorie `plat` et `relief` et on étudiera l'évolution du temps CPU par rapport à la taille des instances. Dans le cas de la méthode basée sur le modèle Dantzig-Fulkerson-Johnson, on pourra aussi s'intéresser au nombre de contraintes qu'il est nécessaire d'ajouter pour achever la résolution. On pourra aussi

considérer séparément les instances des catégories `plat` et `relief`. Lesquelles sont les plus difficiles à résoudre ? Peut-on expliquer pourquoi ?

— Une analyse expérimentale de la méthode de résolution approchée (temps CPU, qualité de solution obtenue).

— Éventuellement, des améliorations pourront être proposées suite à ces analyses expérimentales. Si des étudiants finissent vite ce projet, les enseignants pourront faire des propositions.