Feuille de travaux pratiques n° 4
Mission planning pour une flotte de robots d'exploration

1 Contexte

Nous considérons dans ce projet un sous-problème du problème de planning dynamique de mission pour une flotte de robots d'exploration [1] . Ce problème complexe se situe à l'intersection de plusieurs domaines scientifiques : robotique, intelligence artificielle, recherche opérationnelle.

Un environnement en grande partie inconnu doit être exploré par une flotte de robots. Un ensemble de points d'intérêt étant identifiés, une mission consiste à en planifier l'exploration. À chaque robot est affecté un sous-ensemble de points d'intérêt. Chaque robot part d'une base (permettant de l'entretenir et de recharger sa batterie) et va ensuite explorer les points d'intérêts 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'intérêts à des robots, puis définition pour chaque robot de l'ordre d'exploration de ces points d'intérêt) est réalisé à partir d'estimations des temps nécessaires pour qu'un robot 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 robots. Les images de ces zones vues sous d'autres angles de vue, 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 robot, soit finalement beaucoup trop long. Il devient alors judicieux de revoir les affectations des points d'intérêt qu'il reste à explorer, aux robots en cours de mission (et ensuite les ordres d'exploration des points d'intérêts). 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 de quatre heures, nous allons simplifier ce problème. Tout d'abord, nous travaillerons sans toutes les incertitudes dans les données. Les temps nécessaires pour se rendre d'un point d'intérêt à un autre seront donc connues de manière précise et définitive. Il n'y aura par conséquent pas changement en ce qui concerne les affectations des points d'intérêt aux robots. Chaque robot devra donc explorer un ensemble définitivement précisé de points d'intérêts, avant de rentrer dans sa base. Nous supposerons que l'affectation des points d'intérêt aux robots est déjà réalisée et nous considérerons le sous-problème consistant à déterminer l'ordre de visite des points d'intérêts par un robot, de manière à minimiser le temps total d'exploration. Nous obtenons donc un problème de voyageur de commerce (souvent abrégé par TSP : Traveling salesman problem).

Le problème complet initialement posé est une variante du $mTSP$ (où le m signifie "multiple"), auquel on ajoute de l'incertitude sur les données. La différence essentielle entre le TSP et le $mTSP$ est qu'il y a plusieurs voyageurs partant de divers sites, et devant visiter de manière collaborative l'ensemble des villes.

1. Ce problème est décrit dans les références suivantes :
Brummit, B. & Stentz, A. (1996). Dynamic mission planning for multiple robots. *Proceedings of the IEEE international conference on robotics and automation,* April 1996.
Brummit, B. & Stentz, A. (1998). GRAMMPS : a generalized mission planner for multiple robots. *Proceedings of the IEEE international conference on robotics and automation,* May 1998

2 Modélisation du problème et difficultés

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

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

Le modèle s'écrit ensuite

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

Si on ignore l'ensemble de contraintes (3) (ayant pour but de casser les sous-tours), on obtient la modélisation du problème d'affectation linéaire. Ce dernier problème ne comptant que $2n$ contraintes, sa saisie ne saturera donc pas la mémoire de la machine utilisée pour le résoudre. Par contre, la saisie de la modélisation complète compte environ 2^n contraintes, elle devient donc inexploitable directement pour de larges valeurs de n.

Comme nous l'avons vu en cours, le problème d'affectation peut être vu comme un problème de permutation. 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.

Exemple : on considère 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 pas bien sûr pas d'une obligation en général, car il pourrait y avoir du relief ou des obstacles). Si nous ignorons l'ensemble de contraintes (3) et résolvons l'instance correspondante du problème d'affectation, nous obtenons nécessairement la solution donnée par $x_{11} = x_{22} = x_{33} = x_{44} = x_{55} = x_{66} = x_{77} = 1$ avec $z = 0$ (tous les autres x_{ij} sont nuls). Si nous voyons cette solution comme une permutation, nous obtenons

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

et comme un produit de cycles disjoints, nous obtenons

$$(1)(2)(3)(4)(5)(6)(7).$$

Sans surprise, cette solution est composée de 7 cycles et n'est donc pas admissible pour le problème de voyageur de commerce. Ce résultat s'obtiendra quelque soit l'instance considérée (car la distance de i vers i est nécessairement de 0). Nous pouvons donc immédiatement imposer que $x_{ii} = 0$ pour tout i. Cela ne nécessite en réalité pas l'ajout

d'une contrainte additionnelle, mais une simple modification des ensembles de contraintes (1) et (2). On obtient alors

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1, \ \ \forall i \in \{1, \ldots, n\} \ \ (1')$$

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1, \ \ \forall j \in \{1, \ldots, n\} \ \ (2')$$

L'ensemble de contraintes (3) peut être modifié en conséquence pour ne contenir que les contraintes cassant les sous-tours de taille ≥ 2, on obtient

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \ \ \forall S \text{ avec } 2 \leq |S| \leq n-1 \ \ (3')$$

Les ensembles de contraintes $(1')$, $(2')$ et $(3')$ sont équivalents aux ensembles de contraintes (1), (2) et (3). En ignorant l'ensemble de contraintes $(3')$ et en résolvant le problème correspondant, la solution obtenue ne contiendra plus de cycle de taille 1. Si nous somme chanceux, nous obtiendrons une permutation qui se décompose en un unique cycle, soit une solution admissible pour le problème de voyageur de commerce. **Affirmation :** Cette solution est alors également optimale pour ce problème.

La résolution avec l'instance de l'exemple en ignorant l'ensemble de contraintes $(3')$ 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 les contraintes incluses dans $(3')$.

3 Introduction progressive des contraintes cassant les sous-tours

Il est bien sûr impensable d'ajouter toutes les contraintes de l'ensemble $(3')$. 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 en ne considérant que deux contraintes de l'ensemble $(3')$.

Nous résumons l'approche utilisée.
- Résoudre le problème en ne considérant que les ensembles 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

4 Travail à effectuer

Une implémentation de cette méthode en faisant appel au solveur GLPK est demandée. Clairement, on ne souhaite pas ici "juste" résoudre un Programme Linéaire en variables binaires. Plusieurs résolutions successives sont en effet nécessaires et en fonction du résultat de chacune d'entre elles, une contrainte différente est ajoutée. Il est donc nécessaire de faire appel au solveur GLPK en tant que bibliothèque de fonctions dans un code C/C++.

Plusieurs fichiers sont disponibles sur madoc dans l'archive `ProjetRO.zip` :
- Le fichier `Projet_NOMS.c` 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
```

En plus des fonctions présentées en cours et utilisées dans le TP3, la fonction suivante sera utile :
- `int glp_add_rows(glp_prob *lp, int nrs);` : fonction ajoutant `nrs` contraintes au problème pointé par `*lp` et retournant le nouveau nombre total de contraintes.

La date limite de remise des projets est fixée au mercredi 1er avril à 18h. Le code source et le rapport devront être remis sur madoc dans une archive `.tar.gz` ou `.zip`. Le contenu du rapport devra impérativement décrire :
- La preuve de l'affirmation qui apparaît deux fois en gras dans le texte. 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.
- La correspondance entre les (double-)indices des variables du problème, et les indices utilisés dans GLPK,
- La récupération de la solution optimale retournée par GLPK et sa traduction en permutations, puis en produit de cycles disjoints,
- Une analyse expérimentale effectuée à partir des instances fournies (temps CPU, nombre de contraintes ajoutées pour résoudre le problème...). On considérera séparément les instances de la catégorie `plat` et `relief`. Lesquelles sont les plus difficiles à résoudre avec l'algorithme proposé, pourquoi ?
- Éventuellement, des améliorations pourront être proposées suite à cette analyse expérimentale.