Projet d'Optimisation Stochastique

Problème du voyageur de commerce

-

Document technique

Enseignant : Abdel LISSER

Groupe : 3

Membres du groupe : Adrien LAVILLONNIERE
Corentin MANSCOUR
Hien Minh NGUYEN

Rédigé le : 04/11/2018

Nombre de pages : 14

Table des matières

Introduction

Ce projet vise à proposer une solution au célèbre problème du voyageur de commerce : le voyageur doit passer par un ensemble de villes, avec la contrainte de ne passer qu'une seule fois par ville, de minimiser son temps de trajet et de retourner à la ville de départ.

L'énoncé du problème semble simple mais cache une grande complexité de calculs. En effet, un algorithme naïf aurait une complexité de O(n!) (*n* représentant le nombre de villes) et serait vite inapplicable pour un échantillon de plus de 15 villes environ. Il s'agit d'un des problèmes type appartenant aux problèmes NP, à savoir les problèmes n'étant pas résolvables en un temps polynomial, mais dont les solutions sont facilement vérifiables.

Nous allons donc implémenter un algorithme visant à résoudre ce problème de manière plus optimisée : le recuit simulé, qui sera détaillé par la suite.

L'objectif de ce document technique sera de détailler dans un premier temps le modèle mathématique du problème, puis de présenter le fonctionnement ainsi que les réglages de notre algorithme du recuit simulé. Enfin, nous présenterons comment résoudre la version déterministe et stochastique de ce problème grâce au logiciel CPLEX en prenant pour exemple les instances proposées dans le sujet.

Modèle mathématique

Telles que fournies dans le sujet, voici les données mathématiques du problème :

Nous allons utiliser le jeu de donné suivant :

- $G = (V, E)$un graphe orienté complet de n sommets
- c_{ij}le coût de l'arc (v_i, v_j)

$$x_{ij} = \begin{cases} 1 & \text{si et seulement si l'arc (i, j) est retenu dans le circuit,} \\ 0 & \text{sinon} \end{cases}$$

Le problème est également donné dans le sujet sous la forme du programme linéaire stochastique en variables binaires suivant :

Nous cherchons donc à minimiser le coût total du trajet. Cela se traduit par le programme linéaire suivant :

$$\min_x \{\sum_{i=1}^{n}\sum_{j=1}^{n} \bar{c}_{ij} x_{ij}\}$$

s.t.

$$\sum_{i\neq j, j=1}^{n} x_{ij} = 1, i = 1, \ldots, n \tag{1a}$$

$$\sum_{i\neq j, i=1}^{n} x_{ij} = 1, j = 1, \ldots, n \tag{1b}$$

$$\sum_{i|v_i\in S}\sum_{j|v_j\in S} x_{ij} \leq |S| - 1, S \subset \{v_1, \ldots, v_n\} \ \ et\ S \neq \emptyset \tag{1c}$$

$$\mathbb{P}\{\sum_{i=1}^{n}\sum_{j=1}^{n} \bar{c}_{ij} x_{ij} \leq Z\} \geq \alpha \tag{1d}$$

$$x_{ij} \in \{0, 1\},\ 1 \leq i, j \leq n \tag{1e}$$

Explication du modèle mathématique

Le problème du voyageur est reformulé en une fonction objectif et cinq contraintes.

Fonction objectif : elle nous indique que le but du problème est de minimiser la somme du coût de tous les arcs empruntés par le voyageur.

Contrainte (1a) : La somme de tous les coefficients x_{ij} correspondant à une ville doit valoir 1, et ce pour chaque ville. Dans le parcours du voyageur, une ville i ne peut donc posséder qu'un seul chemin se dirigeant vers une ville j. Cela signifie que l'on ne sort qu'une seule fois de chaque ville.

Contrainte (1b) : Idem : on ne rentre qu'une seule fois dans chaque ville.

Contrainte (1c) : Soit S un sous ensemble de ville de cardinalité |S|. Si il y a |S| arcs ou plus reliant les villes de ce sous-ensemble, alors cela veut dire que l'ensemble est un sous-tour de la solution totale, et donc que la solution n'est pas acceptable.

Contrainte (1d) : C'est la contrainte de probabilité. La solution donnée par CPLEX doit pouvoir être de qualité proche de la solution déterministe du problème (notée Z) au moins une certaine partie du temps (notée $\boldsymbol{\alpha}$).

Contrainte (1e) : assure le bon format des éléments manipulés dans le problème

Algorithme du recuit simulé

L'algorithme du recuit simulé est basé sur un principe métallurgique. C'est un procédé qui se déroule comme suit : un métal est porté à une certaine température et refroidi de manière contrôlée. Ce cycle peut être répété plusieurs fois et permet de stabiliser les cristaux à l'intérieur de la matière.

Cette technique a inspiré les mathématiciens qui en ont déduit une méthode permettant de trouver le minimum ou le maximum d'une fonction.

L'algorithme du recuit simulé consiste en l'exploration du voisinage d'une solution de la fonction objectif, en partant d'un point de départ X_0 et d'une température T_0. Afin de trouver l'extremum, il faut explorer le voisinage de X_0 en calculant des solutions voisines à celle-ci, que nous appellerons X_i.

Si la solution X_i est meilleure que la solution initiale X_0, alors on accepte cette nouvelle solution. Si au contraire la qualité de la solution est dégradée, il est quand même possible de retenir X_i. La décision de garder ou rejeter X_i dans ce cas là est prise aléatoirement grâce à la *distribution de Gibbs-Boltzmann* : $\exp(-\Delta Coût/T)$. On tire un nombre généré aléatoirement entre 0 et 1, et si celui-ci est inférieur à la probabilité de Gibbs-Boltzmann, alors on conserve X_i. Plus la température est élevée, plus la chance d'accepter un X qui dégrade la solution est élevée. On doit donc observer une certaine stabilisation au fur et à mesure que la température descend.

On effectue ce procédé un certain nombre de fois, nombre qui peut être fixé de plusieurs manières. Les itérations fonctionnent par cycle, par palier. A la fin de chaque palier, la température est diminuée d'un certain coefficient. L'algorithme s'arrête lorsque nous avons effectué un certain nombre de palier prédéfini tout en ayant un taux d'acceptation faible. Cette situation pourrait se produire lorsque la température est trop basse pour qu'une solution coûteuse soit acceptée, générant ainsi un taux d'acceptation faible.

Réglages nécessaires pour l'algorithme

Dans notre cas, plusieurs réglages seront à effectuer sur les différents paramètres de l'algorithme. L'algorithme complet est disponible à la page suivante.

Solution de base : comme le recuit simulé permet la détérioration de la solution initiale, il n'est pas nécessaire que celle-ci soit de bonne qualité pour que l'algorithme fonctionne efficacement. Nous avons donc choisi de calculer cette solution initiale :

- soit on parcours bêtement les villes dans l'ordre
- soit en choisissant une ville aléatoire à chaque itération
- soit à chaque ville, on prend l'arc le moins coûteux menant vers une ville pas encore explorée (algorithme naïf de recherche du **plus proche voisin**).

L'expérience nous montrera quelle est la meilleure solution. Nous pensons que l'algorithme du plus proche voisin sera le meilleur pour donner une solution initiale acceptable.

Température : Le **choix de la température initiale** ainsi que de son évolution est primordial au bon déroulé du recuit simulé. Une température trop élevée, par exemple, empêche l'algorithme d'affiner sa solution, la soumettant à trop de variations négatives. D'habitude, trouver une température initiale efficace, ainsi qu'une évolution pertinente de celle-ci, nécessite plusieurs tests de l'algorithme. La température doit être calculée de façon automatique : il faut donner une température initiale à l'algorithme, la diminuer si elle laisse passer trop de mouvements coûteux. Ensuite, on applique le **schéma de Kirkpatrick** : en fonction du taux d'acceptation, on change la température ou non. Si ce taux est grand (autour de 80%), on garde la température actuelle. Sinon, on double la température et on effectue une nouvelle itération.

Condition 1 : cette condition porte sur le **nombre de changements de palier** de température. Nous avons pensé pertinent de ne pas choisir une valeur fixe prédéterminée, mais plutôt d'arrêter de faire baisser la température lorsque aucune amélioration de la solution n'est constatée. On pose alors que si 10 paliers d'affilé ne permettent pas un taux d'acceptation de nouvelles solutions supérieur à 10%, alors on arrête de générer de nouveaux paliers de températures, et on termine l'algorithme. Ces valeurs sont temporaires, et seront sûrement modifiées après une première série de tests.

Condition 2 : Cette condition porte sur le **nombre d'itération par paliers**. Là aussi, des test sont nécessaires pour trouver une valeur efficace. Une méthode plausible serait de changer de palier lorsque l'évolution stagne. On fixe un taux d'acceptation minimal et un nombre d'itération maximum par palier. Quand le compteur d'itération atteint le maximum défini, si l'évolution est supérieur au taux, on recommence le palier. Sinon, on baisse la température[1].

Taux d'acceptation minimal : Celui-ci a été déterminé en fonction des tests que nous avons effectués au cours de l'élaboration du programme. Nous avons fixé ce taux à 5%, mais celui-ci pourra être modifié dans le programme par l'utilisateur.

Exploration du voisinage

Dans le déroulement de l'algorithme du recuit simulé, une étape importante est le choix du voisinage, autrement dit, le fait de trouver une solution voisine à la solution actuelle. Il existe un certain nombre de méthodes pour obtenir un voisin, de complexités et d'intérêts variables. Nous choisissons d'utiliser l'algorithme d'exploration du voisinage suivant :

2-Opt

D'après divers articles, il semble être possible d'utiliser l'algorithme 2-opt en conjonction du recuit simulé pour améliorer sa performance. Mais au vu du manque d'information que nous possédons sur la question, nous ne testerons probablement pas cette possibilité

Le graph du problème étant complet, il est possible d'aller n'importe où depuis n'importe quelle ville. On est donc assuré que ces changements dans l'ordre de parcours des villes produisent une nouvelle solution qui soit réalisable.

[1] http://idboard.net:10000/maths/2018/03/11/recuit-simule/

K-Opt

Après nous être intéressés au 2-Opt, nous nous sommes penchés sur l'algorithme du K-Opt[2] , où K représente un entier quelconque. Au lieu de chercher à permuter 2 arêtes à chaque itération, nous pourrions en permuter K. Une exécution d'un algorithme K-opt aura une complexité de $O(n^k)$. Plusieurs sources semblent nous indiquer que lorsque K est grand, le temps de calcul augmente, mais la qualité de la solution augmente également.

[2] http://akira.ruc.dk/~keld/research/LKH/KoptReport.pdf

Algorithme du recuit simulé

Initialisation

Calculer une solution initiale X_{actuel} : $X_{meilleur} \leftarrow X_{actuel}$
Fixer le nombre de paliers (condition 1) : seuilPalier ← valeur fixe, compteurPaliers ← 0
Fixer le nombre d'itérations par palier (condition 2) : nbIté ← valeur fixe
Initialisation de la température initiale : T0 ← valeur fixe
Taux ← taux d'acceptation, $Taux_{min}$ ← taux d'acceptation minimum fixe
nbMouv ← 0

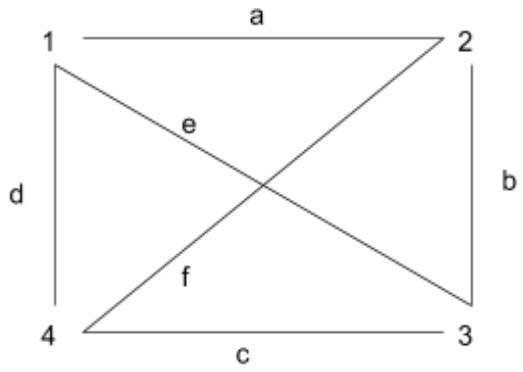
Début
| **faire :**
| | nbMouv ← 0
| | **pour** i de 1: nbIté **faire :** *//itérer à travers un palier*
| | | $X_i \leftarrow$ générerVoisin(X_{actuel})
| | | Calcul du coût : $\Delta \leftarrow$ distance(X_i) - distance(X_{actuel})
| | |
| | | **si** $\Delta < 0$ **alors :** *//on garde la solution*
| | | | $X_{actuel} \leftarrow X_i$
| | | | nbMouv ← nbMouv + 1
| | | | **si** distance(X_{actuel}) < distance($X_{meilleur}$) alors :
| | | | | $X_{meilleur} \leftarrow X_{actuel}$
| | | | | compteurPaliers ← 0
| | | | **finSi**
| | | **finSi**
| | | **sinon**
| | | | Tirer p dans [0, 1]
| | | | **si** p ≤ exp (Δ/T) **alors :**
| | | | | $X_{actuel} \leftarrow X_i$
| | | | | nbMouv ← nbMouv + 1
| | | | | **si** distance(X_{actuel}) < distance($X_{meilleur}$) alors :
| | | | | | $X_{meilleur} \leftarrow X_{actuel}$
| | | | | | compteurPaliers ← 0
| | | | | **finSi**
| | | | **finSi**
| | | **finSinon**
| | **finPour**
| |
| | Taux ← nbMouv / nbIté *//calcul du taux d'acceptation*
| | **si** Taux < $Taux_{min}$ alors :
| | | compteurPaliers ← compteurPaliers + 1
| | **finSi**
| | Temp ← Temp * coeffTemp *//fin du palier : baisse de température*
| |
| **tant que** compteurPaliers < seuilPalier
| **retourner** $X_{meilleur}$
Fin

Pour que cet algorithme fonctionne avec le problème du voyageur de commerce stochastique, il suffit de rajouter la contrainte SOCP lors de la recherche de solution voisine (voir plus bas pour les explications sur cette SOCP)

Résolution du problème avec CPLEX

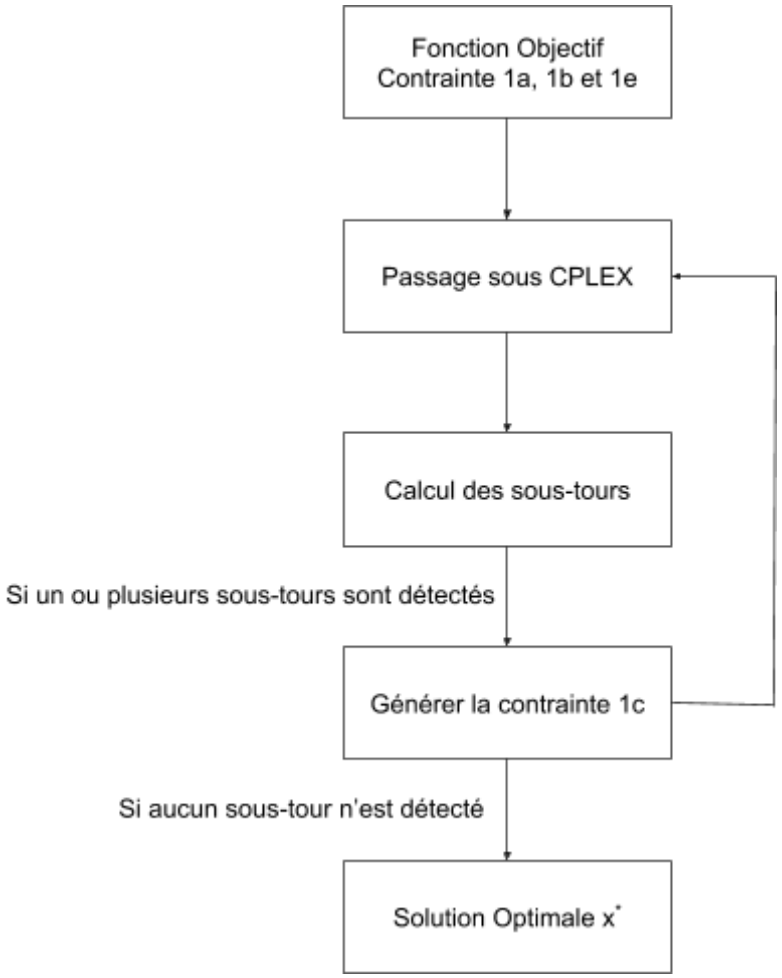
Résolution du problème pour le modèle déterministe

Afin d'utiliser CPLEX pour résoudre le problème selon le modèle déterministe, nous devons transformer la fonction objectif et les contraintes du programme linéaire originel en **équations déterministes** compréhensibles par le logiciel.



Pour cela, nous pouvons donner un exemple sous forme d'un problème à 4 noeuds dont voici une illustration. Les 4 sommets 1, 2, 3 et 4 représentent 4 villes et sont reliés par les arêtes a-f.

Il faut savoir que résoudre le problème décrit plus haut via CPLEX est impossible, car trop demandant en temps et en mémoire. En effet, il existe une contrainte de type (1c) pour chaque sous-tour possible du graphe. Pour éviter d'avoir à générer ces millions de contraintes (1c), il faut procéder de manière itérative. Il faut d'abord exécuter une version simplifiée du programme avec uniquement les contraintes (1a), (1b) et (1e) sous CPLEX, puis détecter les sous-tours présents dans la solution calculée, générer les contraintes (1c) associées, puis relancer le programme avec ces nouvelles contraintes.



Afin de générer la contrainte de sous-tours (1c), il faut :

1) Détecter un sous-tour dans la première solution calculée : si un sous-ensemble contient un nombre d'arêtes égale au nombre de ses villes, alors il existe un sous-tour dans ce sous-ensemble
2) Ajouter une contrainte à ce sous-ensemble pour que le nombre d'arêtes soit égal à (n-1), n étant le nombre de villes du sous-ensemble
3) Relancer CPLEX avec cette nouvelle contrainte

Le programme linéaire, sans les contraintes de sous-tours, peut se traduire par :

Fonction objectif : $c_{12}\,x_{12} + c_{13}\,x_{13} + c_{14}\,x_{14} + c_{24}\,x_{24} + c_{23}\,x_{23} + c_{34}\,x_{34}$

Contrainte (1a) :

$x_{12} + x_{13} + x_{14} = 1$
$x_{21} + x_{23} + x_{24} = 1$
$x_{31} + x_{32} + x_{34} = 1$
$x_{41} + x_{42} + x_{43} = 1$

Contrainte (1b) :

$x_{21} + x_{31} + x_{41} = 1$

$x_{12} + x_{32} + x_{42} = 1$
$x_{13} + x_{23} + x_{43} = 1$
$x_{14} + x_{24} + x_{34} = 1$

Résolution du problème pour le modèle stochastique

Pour pouvoir résoudre la version stochastique du problème en utilisant CPLEX, il faut d'abord transformer le problème.

On part de la contrainte en probabilité (1d)

$$\mathbb{P}\{\sum_{i=1}^{n}\sum_{j=1}^{n}\bar{c}_{ij}x_{ij} \leq Z\} \geq \alpha \qquad (1d)$$

On peut reformuler cela comme suit : $P_r\{ T^T(\boldsymbol{\omega})x <= Z \} >= \boldsymbol{\alpha}$

avec T($\boldsymbol{\omega}$) : vecteur de toutes les variables aléatoires représentant les coûts de chaque arc ;
x : vecteur de toutes les variables de décisions x_{ij} mises bout à bout.
Z : la valeur de la solution déterministe du problème du voyageur de commerce
$\boldsymbol{\alpha}$: flottant appartenant à [0 ; 1]

La première étape de notre transformation est de ramener toutes nos variables aléatoires à une loi normale centrée réduite. On a alors :

$$\mathbb{P}_r\{T^T(\omega)x \leq \mathbb{Z}\} \geq \alpha$$

$$\Leftrightarrow \quad \mathbb{P}_r\left\{\frac{T^T(\omega) - \mu^T x}{\sqrt{x^T V x}} \leq \frac{\mathbb{Z} - \mu^T x}{\sqrt{x^T V x}}\right\} \geq \alpha$$

Avec μ le vecteur de toutes les moyennes des variables aléatoires
et V la matrice des variances-covariances

$$\Leftrightarrow \quad F\left(\frac{\mathbb{Z} - \mu^T x}{\sqrt{x^T V x}}\right) \geq \alpha$$

$$\Leftrightarrow \quad \mathbb{Z} - \mu^T x \geq \sqrt{x^T V x}\, F^{-1}(\alpha)$$

$$\Leftrightarrow \quad \mu^T x + \sqrt{x^T V x}\, F^{-1}(\alpha) \leq \mathbb{Z}$$

Cette nouvelle expression est une SOCP : contrainte de second ordre. Contrairement à la contrainte de départ, celle-ci est traitable par CPLEX.

Enfin, il nous manque l'expression de $\mathbb{Z}$ pour lancer la résolution. Comme expliqué plus haut, Z est le coût total associé à la solution de la version déterministe du problème, majoré

de 25%. Pour convertir le problème stochastique en son équivalent déterministe, il suffit de remplacer chaque variable aléatoire par la moyenne de sa Loi Normale. Pour obtenir $\mathbb{Z}$, il ne reste alors plus qu'à résoudre ce nouveau problème comme décrit plus haut.

Pour faire résoudre la version stochastique du problème du voyageur de commerce par CPLEX, il suffit donc de d'abord résoudre le problème déterministe pour obtenir la valeur de Z, puis de relancer CPLEX sur l'instance stochastique à l'aide de cette nouvelle contrainte.

Pour finir, nous pouvons lancer un certain nombre simulations de ce processus (résolution du problème déterministe puis intégration des contraintes stochastiques), par exemple 100. Ces résultats nous permettront d'établir un graphe des solutions admissibles. On peut alors vérifier si au moins α% des solutions doivent être admissibles pour satisfaire la contrainte (1d).