RANDIMBIVOLOLONA Fabien **Groupe 7**
PERNIN Dylan
LAUPRETRE Julien
PAGET Jules

Projet de Programmation Stochastique

Rapport Technique

Introduction

Le but de ce projet est de proposer 2 solutions à ce fameux problème qu'est le voyageur de commerce, une directe utilisant le solveur CPLEX et une autre utilisant l'algorithme de Recuit simulé. Il s'agit d'un problème combinatoire NP-complet donc en O(n!) (problème Non Polynomial) ce qui signifie qu'au-delà d'un certain nombre de villes, le problème devient impossible à traiter directement. Ce qui fait qu'il est nécessaire d'utiliser des heuristiques.

Le modèle Mathématique

Le problème voyageur consiste à optimiser le chemin pris pour atteindre chaque sommet une seule fois, étant donné une liste de points (des villes) et des distances entre ces points (côut du chemin). C'est un problème qui semble simple mais qui en réalité, plus on agrandit la liste des points, plus la résolution du problème devient compliquée. On essaie ainsi de trouver une méthode pour s'approcher au mieux de la solution optimale du problème, difficile à avoir.

Pour cela, on représente la situation selon un graphe orienté complet de n sommets, où chaque arête représente le coût du trajet entre deux sommets (fixés dans le modèle déterministe, aléatoire dans le modèle stochastique).

Explication des contraintes du problème

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

Fonction objectif : Il s'agit de minimiser la somme du coût de tous les arcs empruntés par le voyageur dans le cadre d'un circuit hamiltonien (passer par tous les nœuds de manière unique).

Contrainte (1a) : pour tout i fixé, la somme sur j des xij = 1. Cela signifie que le voyageur ne sort qu'une seule fois de chaque ville j

Contrainte (1b) : pour tout j fixé, la somme sur i des xij = 1. Cela signifie que le voyageur ne rentre qu'une seule fois dans chaque ville j

Contrainte (1c) : Pour chaque sous-graphe S, le nombre de sommets atteints |S| doit être strictement supérieur aux nombres de chemins empruntés.

Contrainte (1d) : La contrainte 1d est appelée contrainte en probabilité. La solution donnée par CPLEX Z doit pouvoir être meilleure que la solution déterministe du problème (notée Z) au moins α fois sur cent. On souhaite pour notre système α = 25%.

Contrainte (1e) : Les variables x ne peuvent prendre que pour valeur 1 ou 0. C'est à dire que d'un chemin d'une ville à une autre est soit retenu, soit nul.

Approfondissement pour la contrainte 1d.

On souhaite transformer cette contrainte stochastique en contrainte compréhensible pour CPLEX.
Soit Z une solution déterministe au problème.
Soit une matrice A_1 : la matrice des coûts. aij = cij, le coût entre le sommet i et j.
et $\overline{A}_1$: la matrice des coûts moyens.
Soit x la matrice représentant les chemins.
Soit C1 la matrice de variance-covariance
On a donc d'après la contrainte 1d :

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\overline{c}_{ij}x_{ij} = A_1x$$

Ce qui donne :

$$P\{A_1x \leq Z\} \geq \alpha$$

Or on a :

$$P(A_1x \leq Z) = P\left(\frac{A_1x - \overline{A}_1x}{\sqrt{x^TC_1x}} \leq \frac{Z - \overline{A}_1x}{\sqrt{x^TC_1x}}\right)$$

donc A_1x suit une loi normale N($\overline{A}_1x$, $\sqrt{x^TC_1x}$) , donc d'après les propriétés sur la loi normale :

$\frac{A_1x - \overline{A}_1x}{\sqrt{x^TC_1x}}$ suit la loi centrée réduite .

d'où :

$$P(A_1x \leq Z) = \Phi\left(\frac{Z - \overline{A}_1x}{\sqrt{x^TC_1x}}\right) \geq \alpha$$

avec

$$\Phi\left(\frac{Z - \overline{A}_1x}{\sqrt{x^TC_1x}}\right) = P\left(X \leq \frac{Z - \overline{A}_1x}{\sqrt{x^TC_1x}}\right)$$

où X suit la loi normale centrée réduite.
Il faut alors traduire cette loi normale pour obtenir des contraintes compréhensibles par CPLEX.
On voit que cela dépend de Z : On doit donc avoir calculé la solution déterministe avant

$\overline{A}_1$: il faut donc un certain nombre d'exécution avec les contraintes stochastiques pour ensuite calculer les probabilités moyennes et vérifier que la condition est respectée.

Résolution du problème déterministe avec CPLEX

Nous devons transformer le problème mathématique en une fonction objective, et des contraintes dans le but de pouvoir utiliser CPLEX.

Soit un problème avec 4 villes 1,2,3,4.

Fonction objectif : c1_2 x1_2 + c1_3 x1_3 + c1_4 x1_4 + c2_4 x2_4 + c2_3 x2_3 + c3_4 x3_4 + c2_1 x2_1 + c3_1 x3_1 + c4_1 x4_1 + c4_2 x4_2 + c3_2 x3_2 + c4_3 x4_3

(Avec la symétrie xi_j et xj_i)

Contrainte 1a :

c1: x1_2 + x1_3 + x1_4 = 1
c2: x2_1 + x2_3 + x2_4 = 1
c3: x3_1 + x3_2 + x3_4 = 1
c4: x4_1 + x4_2 + x4_3 = 1

Contrainte 1b :

c5: x2_1 + x3_1 + x4_1 = 1
c6: x1_2 + x3_2 + x4_2 = 1
c7: x1_3 + x2_3 + x4_3 = 1
c8: x1_4 + x2_4 + x3_4 = 1

On ajoute les contraintes qui empêchent le voyageur de prendre un chemin dans le sens inverse (si le voyageur a pris le chemin 1_2 il ne peut pas prendre le 2_1)

c9: x1_2 + x2_1 <= 1
c10: x1_3 + x3_1 <= 1
c11: x1_4 + x4_1 <= 1
c12: x2_4 + x4_2 <= 1
c13: x2_3 + x3_2 <= 1
c14: x3_4 + x4_3 <= 1

Contrainte 1e :

0 <= x1_2 <= 1
0 <= x1_3 <= 1
0 <= x1_4 <= 1
0 <= x2_4 <= 1
0 <= x2_3 <= 1
0 <= x3_4 <= 1
0 <= x2_1 <= 1
0 <= x3_1 <= 1
0 <= x4_1 <= 1
0 <= x4_2 <= 1
0 <= x3_2 <= 1
0 <= x4_3 <= 1

Concept de l’algorithme du Recuit Simulé

L’algorithme du recuit simulé (Simulated Annealing en Anglais) est inspiré d’un principe thermodynamique notamment utilisé dans l’industrie métallurgique. Il permet de trouver le minimum (ou le maximum) d’une fonction en cherchant un équilibre énergétique.

Lorsqu'une barre de métal chauffée se refroidit, sa structure moléculaire qui au départ évolue beaucoup, va finir par se figer au fur et à mesure que la température chute, pour arriver à une état stable où l’énergie nécessaire à maintenir la structure est minimale.

Dans notre cas, l’énergie sera en fait la distance d’une arête entre 2 sommets (villes) du graph, et la structure moléculaire sera le circuit hamiltonien du voyageur.

L’idée est donc de partir d’un état initial (qui est un paramètre de l’algorithme) composé d’une solution, mais pas nécessairement bonne, du problème, donc dans notre cas un circuit hamiltonien quelconque, et d’une température quelconque fixée elle aussi.

Ensuite, en procédant par palier de température, on explore le voisinage de la solution, c'est-à-dire dans notre cas en permutant aléatoirement 2 sommets dans le circuit pour créer une solution voisine, puis en regardant comment la longueur totale du circuit a évoluée.

Si la solution voisine réduit la durée du trajet, elle est gardée et devient notre nouvelle solution, sinon il y a une probabilité de la garder quand même, ce critère de sélection est appelé Critère de Metropolis et il devient plus sélectif à mesure que la température diminue (il est issu de l’algorithme de Metropolis qu’on va expliquer sommairement juste après).

On réitère une certain nombre de fois par palier (ce nombre est aussi un paramètre) puis on passe à autre palier en diminuant la température (selon un coefficient de refroidissement, qui lui aussi est un paramètre). On continue à diminuer la température jusqu’à ce que le système soit “gelé”.

La condition d'arrêt qui stipule qu'un système est gelé dépend du taux d'acceptation des solutions de la part de l'algorithme. Pour chaque palier on compte le nombre de solutions acceptées, et le ratio nombre de solutions acceptées / nombre de permutations effectuées nous donne ce taux, souvent on se fixe un taux minimal et en dessous de ce taux on considère que le système n'évolue plus assez et que la solution doit donc être minimale.

L'algorithme du Recuit Simulé en détail

L'implémentation de Johnson et Al

C'est l'implémentation la plus classique de l'algorithme du recuit simulé dont on a expliqué le concept ci-dessus.
L'algorithme en pseudo-code s'écrit comme ceci (ses paramètres sont détaillés plus bas) :

• Générer une solution initiale **S0**
• S := S0
• T := **T0**
• Répéter
  – nb_moves := 0
  – Pour i := 1 à **nbr_iter_palier**
    • Engendrer un voisin S' de S
    • Calculer Δ = f(S') – f(S)
    • Si CritMetropolis(Δ, T), alors
      - S := S';
      - nb_moves := nb_moves + 1
  – **acceptance_rate** := nb_moves / nbr_iter_palier
  – T := T * **coeff_refroidissement**
• Jusqu'à <**condition_fin**>
• Retourner la meilleure configuration trouvée

source : UFRST Bourgogne

Cette implémentation applique en fait itérativement l'algorithme de Metropolis qu'on peut détailler ainsi :

L'algorithme de Metropolis

Dans l'algorithme de Metropolis, on part d'une configuration donnée, et on lui fait subir une modification aléatoire. Si cette modification fait diminuer la fonction objectif (ou énergie du système), elle est directement acceptée ;
Sinon, elle n'est acceptée qu'avec une probabilité égale à $exp(\Delta E/T)$ (avec E=énergie, et T=température), cette règle est appelée critère de Metropolis.
Dans notre cas on a vu que l'énergie est en fait la longueur du cycle hamiltonien, c'est-à-dire le chemin emprunté par le voyageur de commerce.

On remarque que la probabilité diminue avec la température car ce critère n'a de sens que lorsque ΔE <0, en effet si il est >0 alors le critère n'intervient pas car la solution est acceptée obligatoirement. On comprend donc que plus la température baisse et tend vers 0, plus la probabilité d'accepter une solution "mauvaise" (qui dégrade la solution actuelle) va diminuer car $exp(-\infty) \sim 0$.

Il est en fait une amélioration de l'algorithme 2-Opt, qui lui consiste à explorer le voisinage en permutant à chaque fois 2 sommets mais en prenant uniquement les solutions qui optimisent la fonction objectif. C'est une amélioration car l'algo 2-Opt peut tomber dans des minimum locaux desquels il ne peut s'extirper, car il ne peut jamais revenir en arrière, alors que le fait de pouvoir accepter des solutions "mauvaises" permet de pallier ce problème.

Paramétrage de l'algorithme

Solution initiale S0 : 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. De ce fait, nous avons pris une solution extrêmement naïve : on parcourt les villes dans l'ordre croissant de leur numéro d'identifiant.

Température initiale T0 : 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. Cependant la température initiale doit quand même être assez élevée puisqu'elle intervient dans le critère de Metropolis et qu'elle doit permettre de passer par des solutions détériorées.

Acceptance_rate : C'est le taux d'acceptation qui correspond au rapport du nombre de mouvements réellement exécutés / nombre d'itérations, qu'on compare à α le seuil du taux d'acceptation. (Un mouvement correspond à une solution acceptée durant un palier)

Coeff_refroidissement : Le cooling factor ou facteur de refroidissement correspond à la vitesse à laquelle on refroidit, c'est-à-dire à laquelle on progresse dans la recherche du plus court chemin. A chaque fin de palier, on multiplie la température actuelle par le facteur de refroidissement.

nbr_iter_palier : Borne le nombre d'itérations par palier et de ce fait à un impact plus important au début de l'algorithme.

Condition_fin : Un compteur sert à déterminer si l'algorithme stagne. Le compteur est fixé à zéro au début La recherche s'arrête quand le compteur atteint un certain seuil d'arrêt.
A la fin d'un palier, le compteur est
- incrémenté si le pourcentage d'acceptation est inférieur au seuil α.
- remis à zéro si la qualité de la meilleure solution a évolué au cours du palier

(*Remarque : avec une température très basse il devient très dur d'accepter des solutions qui n'améliorent pas le résultat final, du coup l'algo fini par terminer)*

Exemple d'exécution de l'algorithme du Recuit Simulé

On initialise les paramètres de départ :
La température initiale T=100°
Le coefficient de refroidissement coeff=2
Le nombre d'itération par palier, la matrice qui décrit le graphe des villes et le coût entre chaque ville : 3
On choisit une solution initiale : Par exemple avec l'algorithme du plus proche voisin naïf, un ensemble de 4 villes (1,2,3,4), d'où S0 = 4->1->2->3->4.
On enchaîne les paliers de température en modifiant la température (T = T * coeff), chaque palier comportant 3 essais. A chaque essai, on regarde les performances du système.

Premier palier de température :
Je commence j'ai fait 0 mouvement.
1er essai :
On échange par exemple 2 villes au hasard dans le chemin, par exemple 1 et 2. La nouvelle solution devient donc 4->2->1->3->4.
On calcule la distance de ce nouveau chemin et la compare.
On vérifie le critère de metropolis :
_si cette modification fait diminuer la fonction objectif, elle est directement acceptée.
_ Sinon, on tire un nombre au hasard, si ce nombre est supérieur ou égal à $exp(\Delta E/T)$ le changement est accepté.

Si on a accepté une solution, on dit qu'on a effectué un mouvement, et on part sur un nouvel essai.

Disons que les 2e et 3e essais ne sont pas acceptés par le système.

Fin du premier palier :
->On regarde le taux d'acceptance : alpha = nombre d'itérations d'un palier / nombre de mouvement (en gros solutions acceptés) si alpha < alphamax, on arrête l'algorithme
-> Sinon, je diminue la température de 10% donc maintenant T = 90°C

On réitère les paliers jusqu'à ce qu'on atteigne le taux d'acceptance :
On considère alors le système comme figé. On s'est alors suffisamment rapproché de la solution optimale.