CL03 – P2024 – Projet

1. Cahier des charges

Vous pouvez faire des groupes de 4 au maximum. Vous pouvez faire le projet à 1, 2 ou 3 si vous voulez, mais cela demande plus de travail par personne. Si vous ne trouvez pas de groupe, vous pouvez vous adresser aux étudiants pendant une séance de TD ou envoyer un mail à la liste de diffusion cl03@utt.fr. Il faut m'envoyer la composition de votre groupe par email pour le samedi 30 mars dernier délai.

Le projet consiste à résoudre un problème de distribution avec des heuristiques et des programmes linéaires. Les heuristiques peuvent être faites dans n'importe quel langage (VBA, Python, C, Java…). Les modèles doivent être résolus avec GUSEK, pas avec le solveur d'Excel. En effet, GUSEK est plus puissant : il est plus proche de l'écriture mathématique et son nombre de variables n'est pas limité.

Chaque groupe doit rendre un seul rapport, en Word ou PDF, en français ou en anglais. Pour chaque heuristique, le rapport doit donner l'explication des principales variables utilisées, un algorithme en pseudo-code et la solution du test à résoudre. Les modèles mathématiques doivent être tapés proprement avec l'éditeur d'équations de Word ou Latex, en expliquant brièvement le rôle des variables, les contraintes et l'objectif, et être suivis par la solution du test. L'important est de résoudre le test : le langage de programmation choisi et le solveur utilisé ne sont pas importants, on pourrait en utiliser d'autres. C'est pourquoi les codes des algorithmes dans le langage choisi et les codes GUSEK doivent être rendus dans des fichiers séparés : aucun code ne doit être mis dans le rapport.

Il faut m'envoyer vos fichiers dans une archive compressée pour le vendredi 21 juin. Une pénalité de 5 points sera appliquée en cas de non-respect du cahier des charges. Un logiciel anti-plagiat sera utilisé pour détecter les groupes qui se contentent de copier les autres. Tout projet pompé aura la note zéro.

2. Problème à résoudre

On considère la distribution *truckload* de fioul domestique entre des dépôts de carburants et des gros clients industriels. On connait le nombre de dépôts m, le nombre de clients n, le stock de fioul A_i de chaque dépôt i = 1, 2, …, m (en m^3), la demande de fioul B_j de chaque client j = 1, 2, …, n (en m^3) et les coûts de transport C_{ij} entre dépôts et clients (en euros/m^3). Chaque dépôt peut livrer plusieurs clients et chaque client peut être livré par plusieurs dépôts. L'objectif est de déterminer les quantités livrées par chaque dépôt à chaque client, pour satisfaire les demandes et minimiser le coût total de transport. Le stock total des dépôts est au moins égal à la demande totale, sinon il n'y a pas de solution.

Pour simplifier, les stocks et les demandes dans ce projet sont des entiers et les coûts sont des réels. Les entiers peuvent être grands, il faut donc utiliser des entiers longs dans les heuristiques. En VBA par exemple, il faut employer le type *Long*, au lieu du type *Integer* qui ne peut pas dépasser 32767.

Deux jeux d'essai ("instances") sont fournis : un petit *test-1* à 3 dépôts et 4 clients, pour déboguer et tester vos heuristiques et vos programmes linéaires, et un test plus gros *test-2* à 10 dépôts et 20 clients. Ils sont disponibles sous forme de fichier Excel dans le dossier du projet sur Moodle. Voici *test-1* qui va nous servir d'exemple. La partie encadrée est la matrice des coûts C.

Coûts C_{ij}	1	2	3	4	Stock A_i
1	21	36	43	20	18
2	60	30	50	43	5
3	18	10	48	72	7
Demande B_j	4	18	6	2	

3. Heuristiques

Pour les trois heuristiques à faire, il est conseillé d'utiliser des tableaux indicés à partir de 1 comme en maths et en GUSEK. Cela simplifie la programmation. En C, Java et Python les tableaux sont indicés à partir de zéro, mais en C (par exemple) il suffit de déclarer le tableau des stocks A par "int A[m+1]" : les éléments seront indicés de 0 à m et on n'utilisera pas l'élément d'indice 0.

a) Heuristique du coin nord-ouest (CNO)

Cette heuristique très simple remplit une matrice X de flots (quantités transportées, initialisées à zéro) à partir du coin supérieur gauche (le "coin nord-ouest") et calcule le coût total Z. On commence par copier les demandes dans un vecteur D et les stocks dans un vecteur S. C'est pour éviter de détruire les données, car les nombres dans D et S vont être modifiés.

Ensuite, à chaque itération, on considère une route $i \rightarrow j$, au début $i = j = 1$, et on l'utilise au maximum sans dépasser le stock du dépôt i et la demande du client j : $X_{ij} = \min(S_i, D_j)$. Puis on met à jour le stock S_i et la demande D_j en leur soustrayant X_{ij}, ce qui donne le stock restant pour le dépôt i et la demande restant à satisfaire pour le client j. Noter que le stock ou la demande vont s'annuler.

Si S_i devient nul, le dépôt i ne peut plus livrer et on passe à la ligne suivante ($i = i + 1$). Si D_j s'annule, la demande du client j est satisfaite et on passe à la colonne suivante ($j = j + 1$). Noter que si S_i et D_j s'annulent en même temps, alors i et j sont incrémentés. On répète cette itération jusqu'à ce que l'indice j sorte du tableau (toutes les demandes sont satisfaites). Le coût total de transport est égal à la somme des produits $C_{ij}X_{ij}$ sur l'ensemble des routes $i \rightarrow j$.

Pour déboguer votre code, voici la solution obtenue sur le petit exemple *test-1*. Le coût est 1142 avec 6 routes utilisées. Les flots non nuls forment une structure typique en escalier.

Flots C_{ij}	1	2	3	4
1	4	14	0	0
2	0	4	1	0
3	0	0	5	2

Travail à faire. Donnez un algorithme en pseudocode pour CNO et traduisez-le dans le langage choisi. Donnez la matrice des flots et le coût obtenu pour le grand problème *test-2*. Au lieu de donner la matrice complète des flots, vous pouvez donner la liste des X_{ij} non nuls, car ils sont minoritaires.

b) Heuristique du coût minimum (CMIN)

CNO donne souvent un mauvais résultat car elle ne tient pas compte des coûts. Mais elle a le mérite de fournir rapidement une solution réalisable. L'idée de CMIN est d'utiliser en priorité les routes les moins coûteuses. Au lieu de remplir X à partir du coin nord-ouest, elle calcule les flots X_{ij} dans l'ordre des coûts C_{ij} décroissants. À part cela, le calcul du X_{ij} à chaque itération et la mise à jour de S_i et D_j se font comme dans CNO. Noter qu'il est inutile de chercher le coût minimum dans une ligne i si S_i est devenu nul, et dans une colonne j si D_j est devenu nul : ces lignes et colonnes doivent être ignorées.

CMIN se termine quand toutes les demandes sont satisfaites. La solution obtenue peut changer selon la façon de choisir le C_{ij} minimal quand il y a plusieurs minimums. Pour que tout le monde trouve la même solution, il faut choisir le premier C_{ij} minimal dans l'ordre naturel de lecture de la matrice : ligne par ligne, puis colonne par colonne dans chaque ligne. Sur *test-1*, on trouve une solution de coût 818.

Travail à faire. Comme pour CNO.

c) Heuristique de Russell (HRUS)

Cette heuristique est prévue pour un problème équilibré, c'est-à-dire avec un stock total SA égal à la demande totale SB, ce qui est le cas de *test-1* et *test-2*. Si vous voulez résoudre d'autres instances mais non équilibrées, voici comment faire. Si $SA < SB$, on crée un dépôt fictif $m + 1$ de stock $SB - SA$, qui peut livrer tous les clients par des routes de coût nul. Si $SA > SB$, on crée un client fictif $n + 1$ de demande $SA - SB$, qui peut être livré par tous les dépôts via des arcs de coût nul. Dans la solution, il faudra ignorer les flots sur les routes partant du dépôt fictif ou arrivant au client fictif.

Voici la description d'une itération de HRUS. Les dépôts dont le stock est vide et les clients à demandes satisfaites sont ignorés, comme dans CNO et CMIN. Calculer le coût maximum U_i sur chaque ligne i et le coût maximum V_j sur chaque colonne j. Remplir une matrice Δ avec $\Delta_{ij} = C_{ij} - (U_i + V_j)$. Choisir la route $i \rightarrow j$ avec le Δ_{ij} minimal (le plus négatif). S'il y a plusieurs minimums, prendre le premier dans l'ordre de lecture de la matrice, pour que tout le monde trouve le même résultat. Calculer X_{ij} et mettre à jour les stocks et les demandes comme dans CNO et CMIN. Sur *test-1*, on trouve un coût de 910.

Travail à faire. Comme pour CNO et CMIN.

4. Résolution exacte

On va maintenant résoudre le problème de manière exacte, c'est-à-dire trouver une solution optimale. Comme les demandes et les besoins sont des nombres entiers, les flots seront automatiquement entiers : il n'est pas nécessaire de dire que les variables de flots sont entières dans GUSEK.

Pour résoudre les instances test-1 et test-2, on peut copier-coller le contenu des fichiers Excel directement dans la partie "data" de GUSEK. Attention, le format utilisé dans GUSEK est un peu différent, donc il y a quelques modifications à faire après le copier-coller. Par exemple, il fait insérer les indices devant chaque nombre dans les tableaux A et B.

Pour éviter de refaire ces modifications si vous passez de test-1 à test-2 et revenez à test-1. Je vous conseille de mettre les deux instances dans la parie "data" de votre modèle GUSEK et de mettre l'instance qui n'est pas résolue entre commentaires (/* et */, voir la notice GUSEK).

a) Résolution du problème de base

Donner un modèle mathématique générique (= bien paramétré, avec des tableaux pour les données et les variables). Ce modèle est indépendant du solveur utilisé. Puis traduire le modèle en GUSEK et vérifier son bon fonctionnement sur *test-1*. Résoudre ensuite *test-2*. Comparer le coût obtenu avec ceux des heuristiques.

b) Contraintes de sécurité

Dans la solution du problème de base, certains clients sont servis par un seul dépôt. La livraison peut être impossible en cas de grève au dépôt ou d'un bouchon sur la route (accident par exemple). Pour des raisons de sécurité, on veut donc que chaque client soit livré par au moins deux dépôts. Donner les modifications à faire par rapport au modèle mathématique précédent (inutile de redonner un modèle complet dans le rapport). Traduire le nouveau modèle en GUSEK et résoudre *test-2*.

c) Coûts fixes pour les véhicules

Question indépendante de la précédente, sans les contraintes de sécurité. Les livraisons sont faites par des camions avec une capacité $C1 = 42$ m^3 de fioul : si on transporte 85 m^3 sur une route $i \rightarrow j$, il faut 3 camions. On suppose qu'on peut utiliser autant de camions qu'on veut.

Chaque camion utilisé a un coût fixe $F1 = 300$ euros, qui s'ajoute aux coûts de transport des questions précédentes. On veut minimiser le coût total. Donner les modifications à faire par rapport au modèle mathématique de base (inutile de redonner un modèle complet dans le rapport). Traduire le nouveau modèle en GUSEK et résoudre *test-2*.

Maintenant on peut utiliser en plus des camions plus petits de capacité $C2 = 20$ m^3 et un coût fixe $F2 = 140$ euros. Que devient le coût de la solution avec les deux types de camions?

Indication. Pour trouver le nombre de camions de capacité $C1 = 42$ m3 sur une route $i \rightarrow j$ pour un flot X_{ij}, on pourrait écrire en GUSEK "floor (X[i,j] / C1)". La fonction *floor* (= plancher, c'est-à-dire la partie entière d'un nombre) existe en GUSEK, voir la liste des fonctions dans le document *gmpl.pdf*, page 14. Mais elle est applicable seulement à des données. On ne peut pas l'appliquer à une variable car elle est non linéaire. Or GUSEK ne peut résoudre que des programmes linéaires.

Il faut définir une variable <u>entière</u> N_{ij} pour le nombre de camions nécessaires sur la route et écrire $N_{ij} \geq X_{ij}/C1$, qui est bien une contrainte linéaire. Par exemple si $X_{ij} = 100$ et $C1 = 40$ m3, $\frac{X_{ij}}{C1} = 2{,}5$. Mais comme la variable N_{ij} doit être entière, elle va donner 3 <u>ou plus</u>. Donc la contrainte peut donner un nombre de camions supérieur au nombre minimum nécessaire. Mais comme on minimise le coût total dans la fonction-objectif, N_{ij} sera égal au nombre minimum de camions nécessaires dans la solution optimale.