Université d'Evry Val d'Essonne
L1 PC-SPI - Année universitaire 2013/2014

Modélisation et Simulation - MOS21

TP4 - Complexité et Optimisation combinatoire

Consignes :

- tout document (cours, internet...) autorisé,
- une partie de la note est liée à l'atteinte des objectifs durant la séance,
- l'autre partie porte sur l'évaluation du compte-rendu et des scripts qui seront transmis pour le début de séance suivante,
- le compte-rendu répondra aux questions posées et détaillera de manière concise et pertinente la démarche de modélisation,
- les scripts Matlab utilisés sont à transmettre en même temps que le compte-rendu,
- vos documents sont à adresser à gaetan.hello@ufrst.univ-evry.fr.

1/ Complexité algorithmique de la résolution d'un système linéaire par Matlab

La modélisation de nombreux problèmes de sciences physiques et de sciences de l'ingénieur repose sur la formulation de systèmes d'équations aux dérivées partielles. Ces systèmes sont ensuite traduits sous forme algébrique afin de pouvoir accéder simplement à une approximation de la solution par des moyens informatiques. La résolution du problème de départ se ramène alors à celle de :

$$A.x = b \tag{1}$$

où $A \in M_{n,n}(\mathbb{R})$ est une matrice connue, $x \in \mathbb{R}^n$ est le vecteur des inconnues, et $b \in \mathbb{R}^n$ est le vecteur second membre connu.

Pour résoudre ce type de problème fort courant, il existe des méthodes "simples" dont la complexité est en $O(n^3)$. Cela signifie que le temps d'execution de la méthode croit de manière cubique avec le nombre d'inconnues du système. Ainsi, si pour $n = n_1$ on a une durée d'exécution d_1, la durée d'exécution pour $n = 2n_1$ sera de l'ordre de $2^3 d_1 = 8d_1$.

Pour résoudre les systèmes linéaires, Matlab utilise des fonctions optimisées qui possèdent une meilleure complexité que les méthodes "simples". On se propose ici d'en estimer le degré de complexité.

1.1 En vous basant sur le script fourni *MOS21_ TP4_ EX1.m*, tracer la courbe donnant l'évolution de la durée de résolution pour $n \in \{1000, 1100, 1200, ..., 4000\}$. Tracer sur une nouvelle figure la même courbe en échelle logarithmique sur les 2 axes (ajouter pour cela les commandes *set(gca,'xscale','log') ;* et *set(gca,'yscale','log') ;* entre le *hold on* et le *hold off*).

1.2 Copier vos tableaux d'abscisses et d'ordonnées et coller les dans le tableur de votre choix. Tracer la courbe $duree = f(n)$ et ajouter une courbe de tendance de la forme $\alpha\, n^{\beta}$ dont vous afficherez l'équation. Quel degré de complexité trouvez-vous ? Commenter.

1.3 Relancer votre script une nouvelle fois et déterminer à nouveau le degré de complexité. Commenter.

2/ Problème d'optimisation combinatoire du "knapsack"

Au même titre que le problème du voyageur de commerce ("travelling salesman problem"), le problème du sac-à-dos ("knapsack problem") a fait l'objet de nombreuses recherches en raison de son large spectre d'applications tant théoriques que pratiques. Il consiste à maximiser la valeur du contenu d'un sac en le

remplissant par des objets issus d'une liste d'éléments de masse et valeur données tout en respectant la masse limite que peut supporter le sac. On notera ainsi :

- M_{max} la masse maximale que le sac peut supporter,
- n le nombre d'objets de la collection,
- m_i et v_i respectivement la masse et la valeur de l'objet d'indice i,
- $c_i \in \{0, 1\}$ la coefficient indiquant si l'objet i est inclus (1) ou non (0) dans le sac,
- $M = \{m_i\}_{1 \leq i \leq n}$ la liste des masses pour les différents objets,
- $V = \{v_i\}_{1 \leq i \leq n}$ la liste des valeurs pour les différents objets,
- $C = \{c_i\}_{1 \leq i \leq n}$ la liste des coefficients pour les différents objets (0 ou 1).

La fonction que l'on souhaite maximiser est alors la suivante (valeur cumulée des objets présents dans le sac) :

$$v(C) = \sum_{i=1}^{n} c_i \, .v_i \tag{2}$$

Le choix fait pour C doit respecter la contrainte d'admissibilité du chargement :

$$\sum_{i=1}^{n} c_i \, .m_i \leq M_{max} \tag{3}$$

On se propose ici de résoudre le problème avec :

$$\begin{aligned} n &= 6 \\ M_{max} &= 10 \\ M &= \{5, 2, 4, 3, 6, 1\} \\ V &= \{10, 11, 12, 13, 14, 15\} \end{aligned} \tag{4}$$

Votre travail consiste à compléter le script Matlab *MOS21_ TP4_ EX2.m* permettant de trouver la valeur de C maximisant (2) en respectant (3) pour les paramètres (4). La nature de l'algorithme est laissée à votre discrétion sachant qu'un algorithme de parcours exhaustif des combinaisons s'avère ici relativement simple à implémenter...