MLA TP 1

On suppose que vous avez déjà une installation d'un langage (`C++` , `python` , `julia`) capable d'appeler un solveur. Pour ce TP il faut pouvoir résoudre un PL et ajouter des coupes au fur et à mesure, comme en ECMA. Si cependant vous n'êtes pas à l'aise avec la solution utilisée en ECMA, je peux vous donner une machine virtuelle avec `C++` et `cplex` déjà installés et configurés. Vous pouvez installer cela même sur les machines physiques dans la salle de TP au CNAM, en suivant les indications ci dessous.
`http://cedric.cnam.fr/~porumbed/mla/vbox.html`

Exercice 1 Le modèle Benders par plans coupants
Cet exercice vous aide à résoudre le modèle du Slide 5. Au début, on peut commencer avec la plus petite instance du Slide 5 décrite ainsi :

```
int n=6;                          //number of vars including w
int f[] ={7, 2, 2, 7, 7, 1};//last entry= the coef of w
int c[] ={66, 5, 4, 3, 2};  //66 means infinity
int d   =3;
```

L'algorithme de Benders résout le modèle par plans coupants comme indiqué au Slide 8. Les variables de décision sont : y_1, y_2, y_3, y_4, y_5 et w. On peut considérer que toutes ces variables sont *entières*.

Pour ajouter ces coupes, vous pouvez utiliser un langage de votre choix, ex, celui utilisé en ECMA. Mais je suis obligé de me focalisé sur un seul langage par la suite ; je vais utiliser `C++` avec `cplex` ou `gurobi`. Si vous prenez cette voie, voici les points à suivre :
1. Télécharger le code `cedric.cnam.fr/~porumbed/mla/cp.zip`
2. Remarquer que l'objet `CutPlanesEngine` permet d'appeler une fonction qui résout le sous-problème de séparation
3. Écrire la fonction de séparation : il faut résoudre le sous-problème Benders indiqué au Slide 8 comme un PL. La coupe retournée dans les variables `a` et `rHand` est ajoutée automatiquement au modèle.
4. Afficher la solution optimale (les variables **y** et w).

Exercice 2 Le modèle de base sans décomposition

1. Écrire le modèle du slide 5 sans décomposition pour un nombre $2 \cdot n = 2 \cdot 5$ de variables de décision (y_1, $y_2, \ldots, y_5$ et x_1, $x_2, \ldots, x_5$). Vérifiez que vous obtenez les solutions qu'on a trouvées manuellement.
2. Modifier l'instance du problème en utilisant le code à droite pour passer à une instance de taille n pour une valeur de n de notre choix. Pour ce modèle de base, il faut charger l'instance avec `loadData(taille, false)`. Le dernier argument "`false`" veut dire simplement que le modèle de base ne nécessite pas une variable supplémentaire w.
3. Comparer l'efficacité du modèle Benders avec ce modèle de base avec plusieurs valeurs de n, sur plusieurs versions de `cplex` si possible.

```
int n,d;
int *f,*c;
void loadData(int nn,bool benders){
    n=nn;
    d=n/2;
    f = new int[n+1];//see below why n+1
    c = new int[n];
    f[0] = 7;
    c[0] = 8;
    for(int i=1;i<n;i++){
        f[i] = (f[i-1]*f[0])%159;
        c[i] = (c[i-1]*c[0])%61;
    }
    if(benders){//If we solve a
        f[n] = 1;//Benders model,
        n++;     //add a var w
    }
}
```