Le but du TP2 est continuer TP1 pour comparer plusieurs solutions en termes de vitesse. En pratique, **le temps de calcul d'un algorithme peut être moins robuste qu'on ne le pense**. J'ai déposé à `youtu.be/YuKnE6zH-J8` une vidéo qui montre que le temps de calcul peut être multiplié par 7 si on oublie d'appeler `cplex.end()` à la fin du sous-problème Benders, voir aussi un phénomène similaire à `https://youtu.be/9NE0rdLM3Zg`.

Exercice 3 Benders automatique Les versions de `cplex` à partir de 12.9 permettent de réaliser une décomposition de Benders automatiquement. Si vous déclarez les variables **y** comme des variables entières et les variables **x** comme des variables continues, il va jouer sur les variables **x** pour générer (probablement) la même décomposition que nous.

Utiliser le même code de l'exercice précédent sur le modèle de base, mais ajoutez :

```
cplex.setParam(IloCplex::Param::Benders::Strategy, IloCplex::BendersFull);
```

Exercice 4 Résolution ultra rapide du sous-problème Pour la décomposition de Benders manuelle, le principal goulot d'étranglement est la résolution du sous-problème par la PL. Nous avons vu (Slide 10) que si la solution **y** à séparer vérifie $\sum y_i = d$, on peut résoudre plus vite le sous-problème, sans faire appel à la PL. Modifier la procédure de séparation ainsi :

si $\sum y_i = d$ résoudre le sous-problème directement sans PL

si $\sum y_i > d$ résoudre le sous-problème comme précédemment avec la PL

Note : Pour installer une licence `gurobi`, il faut

1. télécharger et extraire l'archive `gurobi` à `http://cedric.cnam.fr/~porumbed/mla/`
2. demander une licence académique au serveur guroby sur
   `www.gurobi.com/downloads/end-user-license-agreement-academic/`

   ou

   demander à l'enseignant, qui est une source inépuisable de licences, c.à.d., tout est dans le fichier
   `cedric.cnam.fr/~porumbed/mla/gurobi.txt`
3. aller dans le dossier `bin` de votre installation `gurobi` et taper la commande `grbgetkey ...` récupérée grâce à une des deux solutions des points 1 ou 2 ci-dessus
4. utiliser le Makefile fourni dans l'archive `cedric.cnam.fr/~porumbed/mla/cp.zip` et vérifier si cela fonctionne. N'oubliez pas qu'avant l'exécution (Linux) il faut taper une commande comme celle-ci :
   `export LD_LIBRARY_PATH=/opt/gurobi811/linux64/lib/` .

Exercice 5

1. Analyser le code `gurobi` dans l'archive `cp.zip` ; la classe `CutPlanesEngine` est une « sur-couche » qui permet d'appeler `cplex` ou `gurobi` (sous le capot). Le fichier `main.cpp` peut résoudre le problème de départ sans connaître le solveur utilisé pour le problème maître. Vous devriez avoir un phénomène très similaire sous `julia` .
2. Si la condition $\sum y_i = d$ n'est pas satisfaite dans le sous-problème, il faut résoudre le sous-problème par la PL. Écrivez le modèle de résolution à l'aide de Gurobi. Vous pouvez vous inspirer du fichier `pl_std_via_gurobi.cpp` ; voir aussi le code en annexe.

Exercice 6 Écrire un fichier texte de maximum 1000 mots pour donner vos intuitions ou un aperçu sur l'efficacité des vos implémentations :

1. la décomposition de Benders manuelle (exercice 1)
2. le modèle de base sans décomposition (exercice 2)
3. la décomposition de Benders automatique (exercice 3)
4. la décomposition de Benders manuelle avec résolution sans PL du sous-problème via `cplex` (exercice 4)
5. la décomposition de Benders manuelle avec résolution sans PL du sous-problème via `gurobi` (exercice 5)
6. Dans mes tests, le modèle de base sans aucune décomposition est résolu en 0.6 secondes avec `cplex 12.6` et en 10 secondes avec `cplex 12.10`. Est-ce que vous pouvez confirmer un phénomène similaire si vous comparez votre version actuelle de `cplex` (pas forcement 12.10) avec la version `cplex 12.6` (disponible en-ligne) ?

Soumettez vos réponses à la dernière question (et le code source si vous voulez) grâce au lien suivant :

`http://cedric.cnam.fr/~porumbed/mla/soumettre.html`

ANNEXE : exemple de PL résolu en C++ avec Guroby

```
#include "gurobi_c++.h"
using namespace std;

int main()
{
  try {
    int n = 3;
    GRBEnv env = GRBEnv();
    GRBModel model = GRBModel(env);

    GRBVar* vars = new GRBVar[n];
    for(int i=0;i<n;i++)
        vars[i] = model.addVar(0,GRB_INFINITY, 0, GRB_CONTINUOUS, "x");

    GRBLinExpr expr = 0;
    for(int i=0;i<n;i++)
        expr += vars[i];
    model.setObjective(expr, GRB_MINIMIZE);


    GRBLinExpr constr = 0;
    for(int i=0;i<n;i++)
        constr += vars[i];
    model.addConstr(constr >= 4, "c0");

    model.optimize();

    model.addConstr(constr >= 5, "c0");

    model.optimize();

    if (model.get(GRB_IntAttr_SolCount) > 0) {
        for (int i = 0; i < n; i++){
            double s;
            s = vars[i].get(GRB_DoubleAttr_X);
            cout<<"vars["<<i<<"]="<<s<<endl;
        }
    }
    cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;

  } catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  } catch(...) {
    cout << "Exception during optimization" << endl;
  }

  return 0;
}
```