Objectifs

- ☞ Modélisation
- ☞ Méthode par séparation et évaluation : outils logiciels

Pour cette séance de travaux pratiques nous vous proposons, dans un premier temps, de découvrir une formulation dite *non compacte* pour le problème du voyageur de commerce. Puis, dans un second temps, nous vous proposerons de découvrir l'outil logiciel **`coin-cbc`** dédié à la résolution de problèmes en nombres entiers par séparation et évaluation (et coupe). Ce logiciel fait partie de la suite logiciels **`Coin-Or`**. Elle est dédiée à la résolution de problèmes d'optimisation divers.

Manipulations 1 — [Problème du voyageur de commerce]

L'objectif de cette partie est de comparer la qualité de deux formulations (modélisations) du problème du voyageur de commerce (**`PVC`**). La premier formulation (celle vue en cours), dite *compacte*, possède un nombre exponentiel de contraintes contrairement à la nouvelle formulation, dite *formulation étendue*, qui possède un nombre polynômial de contraintes.

Pour la modélisation, nous vous proposons d'utiliser le langage de modélisation **`ampl`**.

Soit $G = \langle V, E, \gamma \rangle$ un graphe simple orienté et valué. Le problème (**`PVC`**) consiste à déterminer un circuit hamiltonien de coût minimum dans le graphe G, c'est-à-dire, un circuit de poids minimum dans G et visitant une et une seule fois chaque sommet du graphe. Nous avons vu, en cours, une formulation de ce problème. Cette formulation a l'inconvénient d'avoir un nombre exponentiel de contraintes. Cependant, comme nous le verrons par la suite, elle a tout de même un avantage (lequel ? à vous de le découvrir !).

Comme exemple test nous considérerons le graphe de la figure 1 suivante (exemple traité en TD !). Vous noterez que ce graphe n'est pas orienté !

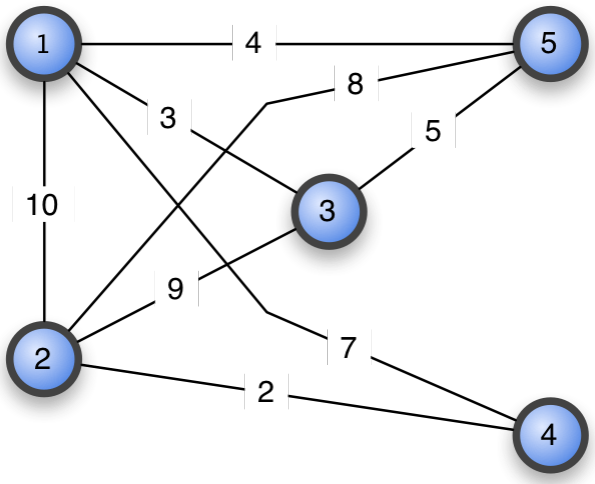

FIGURE 1 – Exemple test.

Partie A : Première formulation

Rappelons la première formulation du problème (**PVC**) vue en cours. Considérons les variables suivantes :

$$x_{uv} = \begin{cases} 1, & \text{si l'arc } uv \text{ est pris,} \\ 0, & \text{sinon} \end{cases} \quad , \forall uv \in E. \tag{1}$$

Nous avons vu qu'à l'aide des variables x le problème (**PVC**) pouvait être formulé comme suit :

$$\begin{aligned} \min \quad & \sum_{uv \in E} \gamma_{uv} x_{uv} \\ s.c. \quad & \\ & \sum_{v \in V : uv \in E} x_{uv} = 1, \forall u \in V, \\ & \sum_{v \in V : vu \in E} x_{vu} = 1, \forall u \in V, \\ & \sum_{u,v \in S : uv \in E} x_{vu} \leq |S| - 1, \ \forall \emptyset \subsetneq S \subsetneq V, \ |S| \geq 2, \\ & x_{uv} \in \{0,1\}, \ \forall uv \in E. \end{aligned} \tag{2}$$

Dans la formulation (2), les contraintes :

$$\begin{aligned} \sum_{v \in V : uv \in E} x_{uv} = 1, \forall u \in V, \\ \sum_{v \in V : vu \in E} x_{vu} = 1, \forall u \in V, \end{aligned}$$

imposent la *connexité* du parcours solution (un sous-graphe de G). Les *sous-tours* sont éliminés par les contraintes :

$$\sum_{u,v \in S : uv \in E} x_{vu} \leq |S| - 1, \ \forall \emptyset \subsetneq S \subsetneq V, \ |S| \geq 2.$$

1. Le modèle **ampl** de la formulation (2) est le suivant :

```
#---
#--- AMPL Model for the ATSP problem
#---
param N integer;
#-- Vertices
set Vertices := 1 .. N;
#-- Arcs
set Arcs within {Vertices, Vertices};
#-- Cost function
param Cost{Arcs};
#-- To generate subsets of Vertices
set Codes := 1 .. 2**N-1;
set Subset{id in Codes}:={u in Vertices: (id div 2**(u-1)) mod 2 = 1};
#-- Variables
var x{Arcs}>=0;  #-- A variable for each arc ...
#-- Objective functin
minimize Obj : sum{(u,v) in Arcs} Cost[u,v]*x[u,v];
#-- Constraints
#--- Assignment 1
subject to assiga{u in Vertices} :
 sum{v in Vertices : (u,v) in Arcs} x[u,v] = 1;
#--- Assignment 2
subject to assigb{u in Vertices} :
 sum{v in Vertices : (v,u) in Arcs} x[v,u] = 1;
#--- Subtour contraints
subject to subtour{id in Codes} :
 sum{u in Subset[id], v in Subset[id] : (u,v) in Arcs and
```

Listing 1 – Modèle **ampl**.

et le fichier d'une instance ressemble au fichier suivant :

```
#---
#--- ATSP instance
#---
data;
param N := 5;
param: Arcs: Cost :=
 1 2 10
 2 1 10
 1 3 3
 3 1 3
 1 4 7
 4 1 7
 1 5 4
 2 4 2
 4 2 2
 5 1 4
 2 5 8
 5 2 8
 3 2 9
 2 3 9
 3 5 5
 5 3 5;
```

Listing 2 – Modèle **ampl**.

Notez bien la syntaxe de chaque fichier ci-dessus !

2. Déterminer le circuit hamiltonien optimal dans le graphe de la figure 1.
3. Résoudre les instances dans le dossier **instances**. Notez à chaque fois la valeur de la relaxation continue.

Partie B : Seconde formulation

Dans cette partie nous vous proposons d'étudier une autre formulation du problème **PVC**. Considérons les variables suivantes, pour tout sommer u :

$$y_u = \begin{cases} 1, \text{ si } u = 1, \\ \text{longueur du chemin menant vers le sommet } u, \text{ sinon} \end{cases} . \tag{3}$$

Notez que les variables y sont définies relativement au sommet 1. Cela n'est pas restrictif car il suffit de renuméroter les sommets pour se ramener au cas considéré.

1. Vérifier que les contraintes suivantes sont bien satisfaites par tout circuit hamiltonien $\mathcal{C}$:

$$y_u + 1 \leq y_v + (n-1)(1 - x_{uv}), \forall uv \in E, \tag{4}$$
$$1 \leq y_u \leq n, \ \forall u \in V \setminus \{1\}. \tag{5}$$

2. Proposer une formulation du problème (**PVC**) dans laquelle figure un nombre polynomial de contraintes.
3. Proposer un modèle **ampl** pour la nouvelle formulation.
4. Déterminer, en utilisant la nouvelle formulation, le circuit hamiltonien optimal dans le graphe de la figure 1. Obtenez-vous une solution optimale de même valeur que celle obtenue dans la partie A ?
5. Résoudre la relaxation continue des instances dans le dossier **instances**. Notez à chaque fois la valeur de cette relaxation continue. Comparer vos résultats avec ceux de la partie A. Que remarquez-vous ? Que peut-on en déduire ?

Manipulations 2 [Optimiser avec Coin-Cbc]

L'objectif de cette partie est de vous introduire un nouvel outil logiciel pour résoudre des problèmes d'optimisation en nombres entiers. Il s'agit de **Coin-Cbc** [1].

1. **C**oin-or **B**ranch and **C**ut

Toute les informations relatives à **Coin-Cbc** sont accessibles à l'adresse `https://projects.coin-or.org/Cbc`. Nous vous recommandons de lire attentivement la section *User's guide* de la section *Documentation*. Elle contient une brève introduction à **Coin-Cbc**.

Bien qu'installé sur les machines de l'école, nous vous invitons à l'installer sur vos machines personnelles afin de mieux vous familiariser avec cet outil. Ainsi, vous aurez une meilleure idée sur l'implémentation de méthodes d'optimisation spécifiques en utilisant des outils logiciels tiers.

Des fichiers **.cpp** et un **Makefile** accompagnent cette partie. Ils sont accessibles sur le *Moodle*. Vous aurez, sans doute, à adapter le **Makefile** suivant votre installation de **Coin-Cbc**.

Pour les différentes parties ci-dessous, nous considèrerons le problème d'optimisation suivant :

$$\begin{array}{ll} \min & -x-2y \\ s.c. & \\ & -2x+2y \leq 3, \\ & 2x+2y \leq 9, \\ & 9x-4y \leq 21, \\ & x,y \in \mathbb{N}. \end{array} \tag{6}$$

Partie A : Optimiser avec `OsiClpSolver`

Dans cette première partie, vous utiliserez l'outil **coin-clp** pour résoudre un problème d'optimisation linéaire avec ou sans variables entières. Pour cela vous utiliserez l'interface **OsiClpSolverInterface** qui est le solver par défaut de **coin-cbc**. Toutefois, il est possible d'en utiliser d'autres : **glpk**, **cplex** .... La philosophie est la suivante, pour utiliser un solver **Xxx** il faut invoquer (si elle existe) **OsiXxxSolverInterface**. Un détail important, il faut préciser lors de l'installation de **coin-cbc** la liste des solvers que vous souhaitez *interfacer*.

1. Analysez le fichier **partie-a.cpp**.
2. Résoudre la relaxation continue du problème (6) puis le résoudre en considérant l'intégrité des variables.

Dans le fichier **partie-a.cpp** vous noterez l'utilisation des classes **CoinPackedVector** et **CoinPackedMatrix**. Ces deux classes sont nécessaires pour stocker l'instance générée aléatoirement.

Partie B : Optimiser avec `Coin-Cbc`

Dans cette deuxième partie, vous utiliserez l'outil **CbcModel** pour résoudre un problème d'optimisation linéaire avec ou sans des variables entières. La classe **CbcModel** offre des méthodes permettant, entre autres, de faciliter la gestion de l'arbre d'énumération, la gestion des coupes et des heuristiques.

La classe **CbcModel** *clone* toujours un solver (donc attention au problème de synchronisation entre le solver et son clone dans **CbcModel**).

1. Analyser le fichier **partie-b.cpp**.
2. Résoudre le problème (6) à l'aide de **Coin-Cbc**. Vous afficherez la solution optimale de la relaxation continue et la solution optimale du problème en variables entières.

Vous noterez que dans le fichier **partie-b.cpp** l'instance est lue à partir d'un fichier au format *mps*. Ce dernier est un des formats utilisés pour représenter des instances de problèmes d'optimisation (cf. Wikipedia pour plus de détails).

Comment feriez-vous pour résoudre à l'aide de **coin-cbc** une instance du problème du voyageur de commerce en utilisant la formulation étendue vue plus haut ?