Polytechnique de Montréal

TP2

MTH6601 : Méth. décomposition optimisation temps réel.

Enseignant : Issmail El Hallaoui.
Chargé de TP : Abderrahman Bani.

1er avril 2020

DIRECTIVES :

- Téléchargez le dossier **Lab.zip** sur le site moodle de cours.
- Vous pouvez travailler en équipes de 2 ou 3 au maximum.
- Inclure dans votre rapport, le code C++ que vous avez utilisé.
- Lors de la correction, il sera tenu compte de la clarté des démarches ainsi que la qualité de la présentation du rapport.

Ce TP décrit comment exécuter une optimisation PMNE (Programmation Mathématique en Nombres Entiers) parallèle en utilisant Open MPI (Message Passing Interface) comme protocole de transport pour gérer la communication entre le maître et les agents. Il étudie aussi un cas de la ré-optimisation pour le problème de partitionnement d'ensemble.

1 SPP problem

Le problème de partitionnement d'ensemble (SPP) est l'un des modèles fondamentaux dans le domaine des tournées de véhicules et des rotations d'équipages. Ce problème consiste à couvrir un ensemble de tâches (exactement une seule fois) avec des chemins de coût minimum. Une tâche peut être un vol d'avion, un segment de trajet de bus ou un client à visiter alors qu'un chemin représente l'horaire d'un pilote ou chauffeur d'autobus ou une tournée de véhicule. La formulation générique du SPP est comme suit :

$$\min_{x} \sum_{j \in \mathcal{N}} c_j x_j \tag{1}$$

$$\text{Sujet à : } \sum_{j \in \mathcal{N}} a_{ij} x_j = 1 \qquad \forall i \in \mathcal{T} \tag{2}$$

$$x_j \in \{0, 1\} \qquad \forall j \in \mathcal{N} \tag{3}$$

où $\mathcal{T} = \{1, ..., m\}$ est l'ensemble des tâches, $\mathcal{N} = \{1, ..., n\}$ est l'ensemble des chemins (colonnes) possibles. À chaque chemin $j \in \mathcal{N}$, une variable binaire x_j est associée ; x_j prend la valeur 1 si le chemin j de coût c_j fait partie de la solution et 0 sinon. Le coefficient a_{ij} prend la valeur 1 si le chemin $j \in \mathcal{N}$ couvre la tâche $i \in \mathcal{T}$ et 0 sinon. La fonction objectif (1) vise à minimiser le coût total. Les contraintes de partitionnement (2) indiquent que chaque tâche doit être couverte exactement une seule fois. Enfin, des contraintes d'intégralité (3) sont imposées aux variables.

2 Instances :

Un ensemble d'instances est disponible sur le site de moodle, il est décrit dans le tableau suivant :

Instance	Nombre de Tâches	Nombre de colonnes
S1.lp	26	771
S2.lp	712	90638
S3.lp	823	8905
vcs1200.pert1.mps	1200	133648

N.B. les instances de chaque partie sont dans le sous-répertoire associé à cette partie.

3 SSH

- Pour établir une connexion ssh à partir d'une machine Linux, MacOS X, il suffit de faire la commande : **ssh -X usager@ssh.gerad.ca**.
- Si vous travaillez sous Windows, vous devrez d'abord installer un logiciel. Par exemple, MobaXterm qui est disponible à l'adresse suivante : http://mobaxterm.mobatek.net/.

- Pour se connecter à une machine à l'intérieur du réseau de gerad, il suffit de taper la commande : **ssh nom-de-la-machine.**
- Une liste de machines est disponible :
  - gerad-1773, gerad-1780, gerad-1783, gerad-1792,
  - gerad-1650, gerad-1709, gerad-1724, gerad-1770,
- **N.B.** avant d'utiliser une machine, il faut se connecter à cette machine avec ssh, pour pouvoir l'utiliser par la suite.

4 Ré-optimisation :

Les perturbations en planification arrivent très souvent en pratique : fermeture d'un aéroport, bris d'un bus, absence d'un ou plusieurs chauffeurs. Ceci nécessite une ré-optimisation afin de réparer les plans opérationnels. Nous constatons que les solutions ré-optimisées s'écartent légèrement des solutions prévues. Par conséquent, de nombreuses tâches consécutives (vols, voyages en bus) sur les trajets initiaux resteront regroupées dans une solution optimale.

Pour simuler ces situations, une méthode a été développée pour perturber aléatoirement la solution optimale afin d'obtenir une solution perturbée avec un certain niveau de bonnes informations primales, similaire aux solutions généralement disponibles dans la pratique après perturbation des plans d'affectation des équipages. Nous considérons comme une mesure de bonne **information primale** le pourcentage de paires consécutives de tâches dans la solution initiale qui restent groupées dans la solution optimale.

Le processus de perturbation d'une solution consiste à sélectionner aléatoirement deux de ses colonnes et à les remplacer par deux colonnes couvrant les mêmes tâches que les deux colonnes initiales. Le processus de perturbation choisit deux tâches appartenant à deux chemins différents dans la solution (ensemble de chemins) et les relie pour créer un nouveau chemin.

Ce processus est répété jusqu'à ce que le nombre de colonnes inchangées (appartenant à une solution initiale) passe en dessous d'un certain pourcentage du nombre de colonnes de la solution. Pour les tests, nous avons fixé ce nombre à 20%, 35% et 50% (les solutions sont fournies dans l'ordre de perturbation).

De plus, les colonnes nouvellement générées sont ajoutées au problème et le coût maximum de toutes les colonnes du problème leurs est donné. Notez que les colonnes de la solution optimale ne sont pas retirées du problème.

1. Dans le dossier **Part1**, modifier au besoin le code et compiler le en utilisant le script **./run.sh**. Le script va lancer automatiquement les tests sur toutes les instances avec plusieurs solution perturbées. Analysez les résultats en décrivant l'effet de fournir une solution initiale avec une certaine information primale sur le processus de résolution de cplex. Est-ce que cplex profite de la solution fournie ou pas ? expliquer pourquoi.

2. Activez le **local branching** en utilisant le paramètre **IloCplex ::Param ::MIP ::Strategy ::LBHeur** de cplex dans le code et compiler le en utilisant le script **./run.sh**. Quel effet a ce changement sur le comportement de cplex ? (analysez le nombre de solutions entières trouvées).

3. Modifier le code et le script pour l'exécuter sur l'instance **vcs1200.pert1.mps**. Changer les paramètres de cplex pour trouver le plus rapidement possible et reporter les résultats ainsi que le code utilisée.

N.B. Les paramètres de cplex sont disponible dans le site `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.cplex.help/CPLEX/Parameters/topics/introListAlpha.html`

5 Optimisation classique :

Dans cette partie, on va lancer la version séquentielle de cplex ; dans le code, on fixe le nombre de threads à 1 en utilisant le paramètre **IloCplex ::Param ::Threads**.

1. Connectez vous sur une machine en utilisant la commande ssh, déplacez vous vers le dossier : **cd chemin-vers-le-dossier-Part2**.
2. Compiler le code en utilisant le script **./run.sh**. Le script va lancer automatiquement les tests sur toutes les instances.
3. Analysez la sortie de cplex pour sortir les informations suivantes pour chaque instance : la valeur de la fonction objectif, la valeur du GAP d'intégralité, la valeur optimale du LP et le nombre de nœuds de branchement.

6 PMNE distribuée de CPLEX

Principe (Source IBM Support) : L'algorithme d'optimisation PMNE distribuée de CPLEX est exécuté sur un unique maître associé à plusieurs agents. Le maître et les agents peuvent être des machines physiquement distinctes ou des machines virtuelles.

L'algorithme commence par pré-résoudre le PMNE sur le maître. Après, il envoie le modèle réduit à chacun des agents. Ensuite, il démarre la phase de montée en puissance. Au cours de cette phase, chaque agent mène ses propres recherches, fondées sur ses propres réglages, qui n'est pas forcément identiques avec ceux des autres agents. La phase de montée en puissance s'arrête lorsque le maître parvient à la conclusion qu'au moins un des agents a créé un arbre de recherche suffisamment grand. A ce stade, le maître désigne un vainqueur. Les réglages utilisés par le vainqueur au cours de la montée en puissance deviennent la base à partir de laquelle le maître détermine quels réglages appliquer dans la phase de recherche par séparation et évaluation distribuée. Sur chacun des agents autres que le vainqueur, l'arbre de recherche est supprimé. L'arbre de l'agent vainqueur est distribué à tous les agents, et c'est à ce moment que démarre la véritable séparation et évaluation parallèle distribuée. En d'autres termes, tous les agents travaillent à présent sur le même arbre de recherche et le maître coordonne la recherche dans cet arbre distribué.

1. Créez un fichier de configuration de machines virtuelles, **config.vmc** (dans le dossier **Part3**), afin de définir les agents disponibles. Voici un exemple d'un tel fichier valable décrit ici :

```
<?xml version="1.0"?>
<vmc>
  <machine name="gerad-1783">
    <transport type="process">
      <cmdline><item value="ssh"/><item value="gerad-1783"/>
        <item value="/home/ibm/cplex-studio/12.10.0.0/CPLEX_Studio/cplex/bin/
            x86-64_linux/cplex"/>
         <item value="-worker=process"/> <item value="-stdio"/>
         <item value="-libpath=/home/ibm/cplex-studio/12.10.0.0/CPLEX_Studio/
             cplex"/>
      </cmdline>
```

```
    </transport>
  </machine>

  <machine name="gerad-1792">
    <transport type="process">
      <cmdline> <item value="ssh"/> <item value="gerad-1792"/>
          <item value="/home/ibm/cplex-studio/12.10.0.0/CPLEX_Studio/cplex/bin/
              x86-64_linux/cplex"/>
          <item value="-worker=process"/> <item value="-stdio"/>
          <item value="-libpath=/home/ibm/cplex-studio/12.10.0.0/CPLEX_Studio/
              cplex"/>
      </cmdline></transport>
  </machine>

  <machine name="gerad-1773"> <transport type="process">
      <cmdline> <item value="ssh"/> <item value="gerad-1773"/>
          <item value="/home/ibm/cplex-studio/12.10.0.0/CPLEX_Studio/cplex/bin/
              x86-64_linux/cplex"/>
          <item value="-worker=process"/> <item value="-stdio"/>
          <item value="-libpath=/home/ibm/cplex-studio/12.10.0.0/CPLEX_Studio/
              cplex"/>
      </cmdline></transport>
    </machine>
</vmc>
```

2. Dans le dossier Part3, modifier au besoin le code et compiler le en utilisant le script **./run.sh**. Le script va lancer automatiquement les tests sur toutes les instances.
3. Analysez la sortie pour extraire la valeur de la fonction objectif, la valeur du GAP d'intégralité ainsi que la valeur optimale du LP et le nombre de nœuds de branchement.

7 Optimisation parallèle MPI :

L'application consiste à lancer trois méthodes en parallèle, une par machine(agent), chaque méthode correspond à lancer cplex avec des paramètres différents. la première méthode qui termine retournera le résultat de l'optimisation au maître.

1. Le dossier **Part4** contient les éléments suivants :
   - **instances** : un dossier qui contient les 3 instances.
   - **AppliMPI.cpp** : un fichier qui contient le code C++ pour utiliser MPI avec Cplex.
   - **makefile** : un fichier pour compiler le code avec la commande **make**.
   - **run.sh** : un script pour compiler le code et l'exécuter sur toutes les instances en utilisant plusieurs machines.
2. Modifiez la section des méthodes, trois en total, pour chaque méthode spécifiez des paramètres différents de branchement en utilisant les trois algorithmes disponible de résolution des programmes linéaires (LPs) dans le nœud racine et fils (Vous pouvez modifier d'autres paramètres pour trouver plus rapidement la solution). Exemple de configurations possibles :

Méthodes	Algorithme	Type de branchement
1	Barrier	Meilleure borne
2	Primal	Meilleure estimation
3	Dual	Parcours en profondeur

3. Allez au dossier **Part4** puis compilez le code avec la commande : **./run.sh**. Si la compilation est complété avec succès, le script va automatiquement lancer l'exécution de l'application sur toutes les instances.
4. Lorsque l'application termine son exécution, vous pouvez savoir quelle est la première méthode qui a réussi à résoudre le problème, consultez le fichier log de la méthode (**instancename-methodx.log**) pour sortir les informations suivantes : le temps, la valeur de la fonction objectif, la valeur du **GAP** d'intégralité et le nombre de nœuds de branchement. Joignez un tableau où vous spécifiez quelle méthode qui a résolu le problème en premier comme suit :

Instance	**Méthode qui a trouvé la solution en premier**
S1.lp	
S2.lp	
S3.lp	

8 Comparaison des résultats :

- Comparez et résumez les résultats dans le tableau suivant, en fournissant pour chaque approche : le temps, la valeur de la fonction objectif **Z**, la valeur du **GAP** d'intégralité et le nombre de nœuds de branchement.
  - **CP-C** : Cplex classique. **CP-D** : Cplex distribué.

	Temps (sec)			***Z***			***GAP(%)***			***#Noeuds***		
Instances	**CP-C**	**CP-D**	**MPI**	**CP-C**	**CP-D**	**MPI**	**CP-C**	**CP-D**	**MPI**	**CP-C**	**CP-D**	**MPI**
S1.lp												
S2.lp												
S3.lp												