Projet APG: Part1

1 Introduction et présentation du problème

Vous utiliserez le langage de votre choix parmi les langages usuels (C, C++, Java, Haskell, Ocaml, ...). Choisissez un langage avec lequel vous êtes à l'aise. Une partie de ce projet est à effectuer en utilisant GLPK, cela peut influencer votre choix.

Le but de ce projet est d'implémenter différents algorithmes pour le problème de déploiement de services sans limite de capacité. Dans ce problème il y a un ensemble de fournisseurs $1 \leq i \leq n$ et un ensemble de clients que nous indexeront par $1 \leq j \leq m$. Les fournisseurs ne sont pas nécessairement ouvert, c'est plutôt des endroits où on peut ouvrir un service. Le coût d'ouvrir le fournisseur i est dénoté par f_i. Un autre coût est celui de fournir un client, il dépend du fournisseur qui fournit ledit client. Si le fournisseur i fournit le client j, cela coûte $c_{i,j}$. Un fournisseur ne peut fournir un client que si il est ouvert. Le but est de trouver un ensemble de fournisseur à ouvrir et une association de chaque client à un fournisseur ouvert de manière à minimiser le coût total. Autrement dit on cherche à fournir touts les clients pour un coût minimum. On définit le problème formellement de la manière suivante :

Problème de déploiement de fournisseurs sans limite de capacité

Entrée: Un ensemble de fournisseur $\{1 \dots n\}$ avec des coûts d'ouverture $(f_i)_{i \in F}$
Un ensemble de clients $\{1 \dots m\}$ avec des coûts de connexion au fournisseurs $(c_{i,j})_{i \in F,\, j \in C}$

Sortie: Un sous-ensemble de fournisseurs $O \subseteq F$ à ouvrir qui minimise

$$\sum_{i \in O} f_i + \sum_{j \in C} \min_{i \in O} c_{i,j}$$

En fait une fois qu'on a choisit les fournisseurs à ouvrir la solution optimale est simple à calculer dans le sens où on connecte chaque client au fournisseur le plus proche. C'est le sens de la formule d'évaluation donnée ci-dessus.

2 Lecture des fichiers, fonction d'évaluation

Dans un premier temps vous programmerez un lecteur de fichier pour les fichiers proposés. Dans ce projet les entrée seront des fichiers texte toujours encodé de la même manière.

```
FILE: B1.1
50 100 0
1 4751 707 164 75 771 276 81 460 376...
```

La première ligne est le nom du fichier. La deuxième ligne contient le nombre de fournisseur ($n = 50$ en l'occurrence), le nombre de clients ($m = 100$). Il y a ensuite une ligne par fournisseur. Chacune de ces ligne commence par le numéro du fournisseur, ensuite vient le coût d'ouverture de ce fournisseur et après le coût de connexion de ce fournisseur à chacun des clients. Les fichiers à tester seront mis en ligne sur mon site. Ces fichiers sont des fichiers de benchmark classique. Pour une partie d'entre eux la solution optimale est donnée, cela vous permettra de comparer vos résultats.

La fonction d'évaluation doit être capable étant donné un sous-ensemble O et l'ensemble des coûts d'une instance de calculer:

$$\sum_{i \in O} f_i + \sum_{j \in C} \min_{i \in O} c_{i,j}$$

Vous devez programmer cette fonction, vous choisissez la structure de donnée que vous voulez. Choisir cette structure de donnée doit se faire en adéquation avec votre lecture de fichier. Dans la suite on notera $eval(O)$ la valeur de cette fonction.

3 Algorithme glouton

Le premier algorithme à implémenter est un algorithme glouton. C'est à dire que cet algorithme va ouvrir les fournisseurs qui nous font gagner le plus d'argent jusqu'à ce que l'ouverture d'un serveur ne soit plus rentable. A chaque étape l'algorithme va ouvrir le serveur tel que l'ouverture de se serveur améliore au plus la valeur de la solution donnée par *eval*. Si cette meilleur amélioration est négative, le nouveau fournisseur n'est pas ouvert et l'algorithme termine.

L'algorithme est décrit formellement de la manière suivante.

Algorithme 1 : Glouton

Données : $(f_i)_{i \in F}$ $(c_{i,j})_{i \in F,\, j \in C}$
$O \leftarrow \emptyset$;
tant que *Il existe* $i \notin O$ *tel que* $eval(O \cup \{i\}) < eval(O)$ **faire**
  Choisir i qui minimise $eval(O \cup \{i\})$;
  $O \leftarrow O \cup \{i\}$;
retourner O

Pour cet algorithme on considère que $eval(\emptyset) = +\infty$.

4 Programme Linéaire

Ce problème peut s'exprimer en programmation linéaire de la façon suivante :

$$\begin{array}{lll} \min & \left(\sum_{i\in F} f_i \times x_i + \sum_{i\in F, j\in C} c_{i,j} \times y_{i,j}\right) & \\ & y_{i,j} \leq x_i & \forall i \in F,\ j \in C \\ & \sum_{i\in F} y_{i,j} = 1 & \forall j \in C \\ & y_{i,j} \geq 0 & \forall i \in F,\ j \in C \\ & x_i \in \{0,1\} & \forall i \in F \end{array}$$

Les variables x_i représentent le fait que le fournisseur i est pris ou pas dans la solution (si $x_i = 1$ le fournisseur i est pris). Les variables $y_{i,j}$ représentent le fait que le client j obtient son service auprès du fournisseur i. Ainsi $y_{i,j}$ ne peut valoir 1 que si x_i vaut 1, les contraintes de la première ligne nous assurent que cette propriété est respectée. Les contraintes de la deuxième ligne nous assure que tous les clients sont connectés.

Vous devez utiliser GLPK pour résoudre ce programme linéaire avec un contrainte d'intégrité sur les variables $(x_i)_{i\in F}$. Résoudre les instances de cette manière vous permettra toujours de trouver la solution optimale, cependant sur certaines instances cela vous prendra beaucoup de temps N'hésitez pas à vous aider de votre TP d'ARO sur GLPK pour vous aider.

5 Relaxation et algorithme d'arrondi aléatoire

Une façon d'obtenir une réponse bien plus rapide est de résoudre le problème relâché, c'est-à-dire sans la contrainte d'intégrité. Cependant cela ne vous donnera pas nécessairement une solution réalisable. Une approche courante est d'utiliser cette relaxation et un peu de hasard pour trouver une solution réalisable de la manière suivante.

Algorithme 2 : Relaxation

$O \leftarrow \emptyset$;
Résoudre le programme linéaire relâché, soit $(x_i)_{i\in F}$ les valeurs des variables associées au fournisseurs;
pour chaque $i \in F$ **faire**
  Ajouter i à O avec une probabilité x_i;
retourner O

En résolvant le programme linéaire en utilisant le simplexe, les $(x_i)_{i\in F}$ ont une valeur comprise entre 0 et 1. C'est la probabilité avec laquelle l'algorithme va ouvrir le fournisseur. Pour obtenir une bonne solution il faudra lancer le tirage un bon nombre de fois et garder le meilleur tirage, mais il ne faut pas relancer le programme linéaire.

6 Un autre algorithme glouton

L'algorithme glouton suivant est un peu différent du précédent. Dans l'idée on va ouvrir un fournisseur de manière à satisfaire des clients pour un coût moyen minimum ou bien relier un client à un fournisseur déjà ouvert si le coût de connexion est inférieur au meilleur coût moyen. On définit le coût moyen du fournisseur i pour un ensemble de clients Y non connectés de la manière suivante, étant donné un ensemble de fournisseur déjà ouvert O et un ensemble de clients déjà couvert T :

$$\text{ratio}(i, Y) = (f_i + \sum_{j \in Y} c_{i,j} - \sum_{j \in T} (c(j, O) - c_{i,j})_+)/|Y|$$

Avec $c(j, O)$ représentant le coût de connexion du client j à l'ensemble O. La notation $(\ldots)_+$ ne considère ce qu'il y a entre les parenthèses seulement si c'est positif, cela revient à faire $\max(\ldots, 0)$.

Le ratio compte donc l'ouverture du fournisseur, le coût de connexion des nouveaux clients à ce fournisseur et les amélioration de connexion de clients déjà connectés que nous pourrions connecté à ce nouveau fournisseur. Le tout est divisé par le nombre de nouveaux clients connectés.

L'algorithme va ou bien choisir le fournisseur avec le meilleur ratio et connecter les clients correspondants, ou bien si un client non connecté a un coût de connexion à un fournisseur déjà ouvert inférieur à ce meilleur ratio, on va connecter ce client.

Voici un exemple d'exécution de l'algorithme sur l'instance décrite ci-dessous :

	Coût d'ouverture	Client 1	Client 2	Client 3
Fournisseur 1	5	1	4	8
Fournisseur 2	3	4	6	6

Table 1: Tableau de données

Pour le fournisseur 1 si il couvre juste le client 1 il aura un ratio $(5+1)/1 = 6$, si il couvre les clients 1&2 il aura un ration de $(5+1+4)/2 = 5$ et si il couvre les clients 1&2&3 il aura un ratio $(5+1+4+8)/3 = 6$. Ainsi le meilleur ratio est obtenu en couvrant les clients 1 et 2. Pour le fournisseur 2 le meilleur ratio est de couvrir les clients 1 2 et 3 pour obtenir un ratio de $(3+4+6+6)/3 = 6,333$. L'algorithme va donc choisir d'ouvrir le fournisseur 1 et d'y relier les clients 1 et 2. Il reste le client 3. Pour le fournisseur 2 le client 3 a un ratio de $(3+6)/1 = 9$ qui est inférieur au coût de connexion avec le fournisseur 1 qui est déjà ouvert. L'algorithme va donc choisir de relier le client 3 au fournisseur 1 déjà ouvert.

Pour des raisons de performances de l'algorithme on considère des coûts d'ouverture de fournisseur de $2 * f_i$ pour le calcul du ratio. C'est une façon de décourager l'ouverture de nouveaux fournisseurs.

L'algorithme peut être décrit formellement de la manière suivante :

Algorithme 3 : Glouton

Données : $(f_i)_{i\in F}$ $(c_{i,j})_{i\in F,\, j\in C}$
$S \leftarrow C$;
$O \leftarrow \emptyset$;
tant que $S \neq \emptyset$ **faire**
  $\alpha \leftarrow \min_{i\in O, j\in S} c_{i,j}$;
  $\beta \leftarrow \min_{i\notin O, Y\subset S}(2 * f_i + \sum_{j\in Y} c_{i,j} - \sum_{j\notin S}(c(j,O) - c_{i,j})_+)/|Y|$;
  si $\alpha \leq \beta$ **alors**
    Soit j le client qui réalise α;
    $S \leftarrow S - j$;
  sinon
    Soit i, Y le fournisseur et l'ensemble des clients qui réalisent β;
    $S \leftarrow S \setminus Y$;
    $O \leftarrow O \cup \{i\}$;
retourner O

Dans cet algorithme S est l'ensemble des clients non connectés et ainsi $\sum_{j\notin S}$ est une somme sur les clients connectés.

Ceci est une description de l'algorithme qui indique finalement peu de choses sur l'implémentation de celui-ci. Plus précisément il ne nous indique pas du tout comment calculer α et β rapidement. Entre autres β est calculé sur un ensemble exponentiel de sous-ensembles.

6.1 Calcul de β

Il va donc falloir comprendre comment calculer β efficacement.

- La partie $2 * f_i - - \sum_{j\notin S}(c(j,O) - c_{i,j})_+$ ne dépend pas de Y on peut la calculer assez rapidement dans un premier temps.

- Ensuite on veut des éléments de S qui dont la connexion coûte le moins cher. A chaque fois que nous rajoutons un élément j la somme augmente de $c_{i,j}$ et $|Y|$ augmente de 1. On a donc intérêt à ajouter les clients non couverts dont le coût de connexion est minimum.

- Donc d'un point de vue d'implémentation on va commencer par trier ces coûts de connexion. En fait on utilisera un tas dans lequel on va ajouter tous les coût de connexion à partir du fournisseur i.

- Ensuite on va ajouter à Y ces clients dans l'ordre croissant de coûts de connexion si ceux-ci font baisser le ratio moyen. Si ils font augmenter ce ratio, nous ne l'ajoutons pas et nous avons trouvé le meilleur ratio pour le fournisseur i. Il faut faire cela pour tous les fournisseurs.

6.2 Une structure de donnée adaptée aux algorithmes gloutons

Les algorithmes gloutons prennent toujours le meilleur élément selon un critère donné. Mais nous n'avons pas nécessairement envie de recalculer ce critère à chaque itération. Une façon intéressante de traiter ce problème est d'utiliser un tas comme structure de donnée.

Dans ce tas nous allons insérer tous les événements qui risquent de se produire, c'est à dire connecter un client ou bien ouvrir un fournisseur. La valeur associée à ces événements est le coût de sa meilleure connexion à O pour la connexion d'un client et la valeur de son meilleur ratio pour un fournisseur.

Algorithme 4 : Glouton 2

Données : $(f_i)_{i\in F}$ $(c_{i,j})_{i\in F,\, j\in C}$
$S \leftarrow C$;
$O \leftarrow \emptyset$;
pour chaque $i \in F$ **faire**
    Ajouter au tas $\beta(i, O)$;
tant que *Le tas T n'est pas vide* **faire**
    $x \leftarrow pop(T)$;
    si *x est un fournisseur i* **alors**
        si $x = \beta(i, O)$ **alors**
            $O \leftarrow O \cup i$;
            On retire de S les clients connectés par i pour ce ratio;
            On ajoute dans le tas les $(c_{i,j})_{j\in S}$;
        sinon
            On remet au tas $\beta(i, O)$ (la valeur est actualisée);
    sinon
        Dans ce cas x est un $c_{i,j}$;
        si $j \in S$ **alors**
            $S \leftarrow S - j$;
retourner O

Cet algorithme ne va actualiser les ratio du tas que quand il va les dépiler, il vérifie que le ratio est à jour, si il est à jour il ouvre le fournisseur, sinon il le remets dans le tas avec une valeur à jour.