Licence 3 Année 2014/2015

Algorithmique LI311

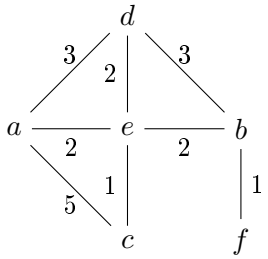
Mini-Projet
Arbre Steiner et conception de circuits électroniques

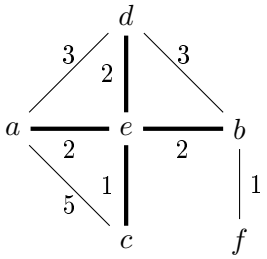
Description du problème

On considère un graphe non orienté $G = (S, A)$ et un sous-ensemble $N \subset S$, $N \neq \emptyset$, de sommets nommés *terminaux* de G. Les sommets de $S \setminus N$ sont appelés *sommets Steiner*. On appelle *arbre Steiner couvrant* T un arbre (S', A') avec $S' \subset S$ et $A' \subset A$ tel que $N \subset S'$. On considère une fonction c qui associe à chaque arête $a \in A$, un coût $c(a)$ positif ou nul. On s'intéresse ici à la recherche d'un arbre Steiner minimum, c'est-à-dire dont la somme des coûts des arêtes est minimum. Remarquons qu'un arbre Steiner peut contenir ou non des sommets Steiner, qui peuvent être ainsi vus comme des sommets optionnels.

De manière générale, déterminer un arbre Steiner minimum est un problème difficile, le but de ce projet est d'étudier différents aspects de ce problème et plusieurs algorithmes pour le résoudre de manière exacte ou approchée. Une analyse théorique et expérimentale devra être menée pour définir et vérifier expérimentalement l'efficacité des implémentations proposées.

Pour illustrer les définitions, ci-dessous à gauche un graphe $G_1 = (S_1, A_1)$ associé à l'ensemble des terminaux $N_1 = \{a, b, c, d\}$ et à droite un arbre Steiner minimum est indiqué par les arêtes en trait gras qui contient les 4 terminaux et un seul sommet Steiner.





Le problème de l'arbre Steiner intervient notamment comme une étape de la conception de circuits électroniques. Des graphes provenant de ce domaine seront utilisés comme instances pour effectuer des tests.

Première partie : analyse et applications

1 Conception de circuits électroniques

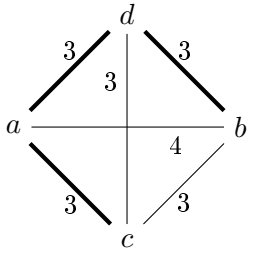
Question 1
Montrer qu'un arbre Steiner minimum d'un graphe G est un sous-graphe connexe de coût minimum de G contenant tous les terminaux.

Le problème de la recherche d'un arbre Steiner dans un graphe trouve une application naturelle dans la conception de circuits électroniques. La conception de circuits électroniques consiste à placer les composants et les pistes conductrices d'un circuit sur support, par exemple une carte. Parmi toutes les étapes nécessaires, l'une d'entre elles consiste à relier tous les composants (puces, condensateurs,...) aux bornes générales d'alimentation électrique du circuit, qui sont généralement placées dans un coin de la carte. La patte correspondant à la borne "+" (respectivement "-") de chacun des composants doit donc être reliée à la borne générale "+" (respectivement "-"). On appelle cette étape *le routage d'alimentation électrique* qui a lieu sur les différentes couches du circuit (ainsi deux des couches sont réservées à l'alimentation). Tous les emplacements de pistes ne sont pas utilisables : on doit donc choisir les pistes parmi un sous-ensemble connu de pistes possibles : ces pistes potentielles se croisent en différents *points* du circuits dont certains sont les pattes des bornes des composants et d'autres sont simplement des croisements entre deux pistes potentielles. On désire évidemment réduire le poids et le coût d'un circuit, on veut donc effectuer ce routage en utilisant une longueur totale minimale de pistes conductrices.

Question 2
Expliquer brièvement pourquoi déterminer le routage d'alimentation électrique d'un circuit électronique revient à rechercher deux arbres Steiner minimaux.

2 Propriétés du graphe des distances

Pour un sous-ensemble $S' \subset S$ de sommets de G, on considère le graphe complet $D(S')$, sur l'ensemble des sommets S', qui est appelé *graphe des distances*. On associe à ce graphe une valuation d des arêtes telle que $d(x, y)$ est la valeur d'une plus courte chaîne de x à y dans le graphe G.
Voici le graphe des distances $D(N_1)$ correspondant au graphe G_1 pour illustrer les définitions et un arbre couvrant minimum de ce graphe donné par les arêtes en trait gras.



Question 3
Montrer que, pour toute arête (x, y) de G, on a $d(x, y) \leq c(x, y)$.

Question 4
Montrer que la valuation d de $D(S')$ vérifie les inégalités triangulaires pour tout $S' \subset S$, c'est-à-dire que $d(x, z) \leq d(x, y) + d(d, z)$ pour tout triplet de sommets x, y, z.

Question 5
Etant donné un arbre Steiner T minimum de G et T_D un arbre Steiner minimum de $D(S)$, montrer que $c(T) = d(T_D)$ avec $c(T) = \sum_{(x,y)\in T} c(x, y)$ et $d(T_D) = \sum_{(x,y)\in T_D} d(x, y)$

Considérons alors l'algorithme *Transfert* suivant pour un sous-ensemble $S' \subset S$ contenant tous les terminaux (*i.e.* $N \subset S'$).

— Remplacer chaque arête $\{x, y\}$ de T_D par une plus courte chaîne de x à y dans G. On nomme H le graphe résultant de cette opération (une même arête n'apparaissant qu'au plus une fois dans H).
— Trouver un arbre couvrant minimum T dans H
— Ôter itérativement de T les sommets Steiner de degré 1.

Question 6
Montrer que l'algorithme *Transfert* utilisé pour l'ensemble S des sommets permet d'obtenir un arbre Steiner minimum T de G à partir de T_D.

3 Méthode exacte

Dans cette section, nous allons étudier une méthode exacte pour résoudre le problème de l'arbre Steiner.

Question 7
Est-il vrai que lorsque $N = S$, on peut trouver un arbre Steiner minimum avec un algorithme de complexité $O(n^3)$? Est-il vrai que lorsque $|N| = 2$, on peut trouver un arbre Steiner minimum avec un algorithme de complexité $O(n^3)$?

Question 8
Montrer qu'il existe un arbre Steiner minimal T_D de $D(S)$ pour lequel chaque sommet Steiner est de degré au moins 3.

Question 9
En déduire qu'il existe toujours un arbre Steiner minimum T_D dans $D(S)$ contenant au plus $|N| - 2$ sommets Steiner.
Indication : remarquer pour cela que la somme des degrés des sommets d'un graphe est égal à deux fois son nombre d'arêtes.

On peut déduire de la question précédente que l'algorithme suivant fournit un arbre Steiner minimum pour G.

- Enumérer tous les ensembles $S' \subset (S \setminus N)$ d'au plus $|N| - 2$ sommets Steiner
- Pour chacun de ses sous-ensembles S', déterminer un arbre couvrant minimum dans le graphe des distances $D(N \cup S')$.
- Retenir l'arbre couvrant minimum parmi tous.

- Par l'algorithme *transfert*, transformer cet arbre en un arbre Steiner de G de même coût.

Question 10
Donner la complexité de l'algorithme précédent dans le cas général. Que devient cette complexité quand il y a peu de terminaux, c'est-à-dire si $|N| = O(1)$.

Question 11
En supposant que pour un graphe de 500 sommets, il faut 0.01 seconde à un ordinateur pour calculer un arbre couvrant minimum, combien de temps durera l'algorithme précédent pour un graphe de 1000 sommets avec 500 sommets Steiner ? Que gagne-t-on avec un ordinateur 100 fois plus rapide ? Qu'en déduisez-vous ?

4 Méthode approchée

On s'intéresse dans cette section à une méthode approchée pour déterminer un arbre Steiner minimum.

Algorithme *Méthode approchée* :
— Construire le graphe des distances $D(N)$ limité aux sommets terminaux.
— Déterminer un arbre couvrant minimum T_D de $D(N)$.
— Remplacer chaque arête $\{x, y\}$ de T_D par une plus courte chaîne de x à y dans G. On nomme H le graphe résultant de cette opération (une même arête n'apparaissant qu'au plus une fois dans H).
— Trouver un arbre couvrant minimum T dans H.
— Ôter itérativement de T les sommets Steiner de degré 1.

Question 12
Quelle est la complexité de cet algorithme ?

Cet algorithme produit un arbre Steiner de G qui n'est pas systématiquement optimal. Néanmoins, il produit fréquemment de bonnes solutions. Dans le reste de cette section, nous allons voir qu'en fait les solutions produites par cet algorithme ont une garantie de performance.

Soit un arbre Steiner minimum $T^* = (S^*, A^*)$ de G. Prenons un sommet x quelconque de T^* et considérons une chaîne μ obtenue par la suite des arêtes partant de x et utilisant les arêtes (deux fois chacune exactement) de T^* pour visiter tous les sommets de T^* et revenir à x. On peut dire que μ "fait le tour" de l'arbre T^* pour revenir à x. Comme chaque arête est empruntée exactement deux fois dans μ, le coût est donc exactement $2c(T^*)$.

On va à présent montrer que le graphe des distances $D(N)$ limités aux sommets terminaux possède un arbre couvrant de coût au plus $2c(T^*)$. Pour cela, considérons μ' la chaîne obtenue en supprimant tous les sommets Steiner de μ. En remarquant que μ est une suite de chaînes de G reliant deux terminaux, on peut remarquer que $d(\mu') \leq d(\mu)$. En prenant alors un arbre couvrant parmi les arêtes de μ', on obtient donc un arbre Steiner de $D(S)$ de coût inférieur à $d(\mu')$. Donc il existe bien un arbre couvrant dans $D(N)$ avec un coût au plus de $2c(T^*)$.

D'autre part, on peut remarquer que l'algorithme produit un arbre Steiner T de G de coût au plus celui d'un arbre couvrant $D(N)$. On en déduit donc que l'algorithme garantit que l'arbre Steiner obtenu a un coût au plus de deux fois celui d'un arbre Steiner minimum.

Deuxième partie : implémentation de la méthode approchée

Dans cette deuxième partie, nous allons mettre en œuvre la méthode approchée pour construire des arbres Steiner pour un graphe G. Pour cela, voici quelques indications à suivre. Dans cette section, nous précisons comment doivent être implémentés différents outils nécessaires à la méthode approchée.

5 Manipulation des instances

5.1 Structure de données pour les graphes

On codera un graphe par un tableau de listes chaînées de voisins. Ce tableau sera indicé par les sommets du graphe. Pour un sommet donné, on indiquera par une valeur booléenne si le sommet est un terminal ou un sommet Steiner.

Il sera utile pour la suite du projet de disposer des fonctions suivantes :
- ajout d'une arête avec son coût,
- test de la présence ou non d'une arête dans le graphe,
- accès au coût d'une arête à partir de ses sommets extrémités.

Pour créer un graphe, vous pourrez tout d'abord créer un sommet sans arêtes allouées pour le nombre voulu de sommets. Ensuite, vous ajouterez une à une les arêtes dans la structure, en testant si une arête n'est pas déjà dans le graphe pour ne pas l'ajouter 2 fois.

Rappel : comme nous manipulons ici des graphes non orientés, pour chaque arête $\{s, t\}$, on doit indiquer que s est voisin de t et que t est voisin de s.

5.2 Instances

Pour tester au mieux la capacité de vos algorithmes dans des cas d'applications pratiques, vous l'utiliserez pour des instances réalistes issues de la conception de circuits intégrés. Nous vous proposons sur le site du module un lot d'instances (en anglais benchmark) issus du site TSPLIB. Il s'agit à l'origine de listes de coordonnées de points sur un plan.

Le nom des instances est composé des champs suivants :
- le nom de provenance de l'instance
- le nombre de sommets
- le pourcentage de terminaux parmi les sommets
- le coût maximal d'une arête (entre 0 et 1000)

- la densité (en pourcentage) du nombre d'arêtes selon la formule $\frac{2n}{n(n-1)}$
(Le dernier numéro existant sur certaines instances correspond à un code de génération aléatoire).

Les instances fournies respectent le format suivant :

nbsommets nbaretes	*Nombre de sommets et d'arêtes dans l'instance*
0 N_0 x_0 y_0	*Numéro d'un sommet (de 0 à nbsommets-1) suivi de*
1 N_1 x_1 y_1	N_i *est à 1 si sommet terminal et 0 si sommet Steiner*
2 N_2 x_2 y_2	*et de ses coordonnées* x_i *et* y_i
...	
1 4 $c(1,4)$	*numéro des extrémités d'une arête et coût correspondant*
1 7 $c(1,7)$	
...	

Noter que le sommet 0 est systématiquement un sommet terminal.

5.3 Visualisation des instances et des solutions

Votre logiciel doit permettre une visualisation des instances et des solutions en utilisant un procédé de votre choix. Beaucoup d'outils informatiques sont possibles (graphviz, xfig, ...). Attention, certains d'entre eux ne permettent pas la visualisation de graphes de grandes tailles. Nous vous fournissons sur le document accessible à l'url ci-dessous, une documentation simple pour visualiser des graphes en format postscript :
`http://www-desir.lip6.fr/~fouilhoux/documents/doc_postscript.pdf`

Il faudra développer une fonction de visualisation des solutions. Une telle fonction prendra en paramètre le graphe afin de représenter tous les sommets ainsi qu'une liste chaînée des arêtes correspondant à l'arbre solution.

Question 13
Décrire un ensemble de structures et de fonctions (librairies, classes,...) permettant de manipuler un graphe codé par tableau de listes d'adjacence, de lire une instance fournie selon le format indiqué ci-dessus et de visualiser un tel graphe et sa solution.

6 Algorithme de Dijkstra et de Prim

Afin d'implémenter les algorithmes de Dijkstra et de Prim avec une structure de données adéquate, il faut tout d'abord coder une structure de tas. Vous remarquerez que les implémentations des deux algorithmes sont fortement similaires.

6.1 Structure de tas

Pour les algorithmes de Dijkstra et de Prim présentés dans ce qui suit, nous utiliserons une structure de tas pour stocker des couples (x, l) où x est un sommet représenté par un entier et l est un nombre réel appelé priorité. La priorité d'un sommet dans le tas correspondra à l'évaluation minimale qui lui est associée.

Cette structure de tas K disposera ainsi des primitives suivantes :
- $intialiser(K)$ qui crée une structure K vide,
- $est\_vide(K)$ qui retourne vrai si K est vide et faux sinon,
- $ajouter(K, x, l)$ qui ajoute à K un élément (x, l),
- $supprime(K, x)$ qui supprime un élément (x, l) de K. Attention, cette primitive a pour hypothèse que x est déjà contenu dans K et qu'il est unique dans K,
- $recup\_min(K)$ qui supprime l'élément (x, l) de valeur l minimale dans K et qui retourne le sommet x.

Dans ce cas d'utilisation d'un tas, les éléments (x, l) du tas sont tels que x est associé à un nombre entre 1 et n (ou 0 et $n-1$) et que les éléments apparaissent au plus une fois dans le tas. Dans ce cas particulier de tas, proposer une modification à la structure classique de tas qui permet d'obtenir une complexité de la primitive *supprime* en $O(1)$.

6.2 Structure pour manipuler une solution des algorithmes de Dijkstra et Prim

On verra dans la section suivante qu'une solution étant un arbre, elle peut être codé par un tableau : en effet, si l'on choisit un sommet "racine" r quelconque d'un arbre, on peut le voir comme une arborescence enracinée en r, on peut définir le tableau T tel que $T[s]$ indique le père du sommet s dans l'arborescence et -1 sinon. Nous coderons les solutions de cette façon dans les algorithmes des sections suivantes.

Si vous voulez visualiser les solutions fournies par ces algorithmes, il est facile de déduire d'un tel tableau la liste chaînée des arêtes de la solution à fournir à votre fonction de visualisation : il s'agit des arêtes $(s, T[s])$ pour tout sommet avec $T[s]$ différent de -1.

6.3 Algorithme de Dijsktra

On précise ici des notations concernant l'algorithme de Dijkstra pour le calcul de l'arborescence des plus courtes chaînes dans un graphe à valuation positive. L'algorithme permet à partir d'un graphe $G = (S, A)$ et d'un sommet r de retourner deux tableaux d et $pred$ tels que
- $d[x]$ soit la valeur d'une plus courte chaîne de r à x
- $pred[x]$ indique le père de x dans une arborescence des plus courtes chaînes et $pred[r] = -1$.

Dans l'algorithme suivant, le tableau M, dit de marquage, sera tel que, pour un sommet x, $M[x]$ pourra prendre deux valeurs : 0 pour indiquer que le sommet n'a pas encore été rencontré ; 1 si le sommet x est dit "fermé" c'est-à-dire si un sommet a été rencontré et.

Voici une description de l'algorithme de Dijkstra utilisant un tas K.

entrée : graphe $G = (S, A)$ et sommet origine r
sortie : tableaux d et $pred$

```
Déclaration d'un tableau d'entiers M indicés sur les sommets
pred[x] ← −1 ∀x ∈ S
M[x] ← 0 ∀x ∈ S
initialise(K)
ajouter(K, r, 0)
d[r] ← 0
Tant que non est_vide(K) Faire
    x ← recup_min(K)
    Pour tout voisin y de x faire
        Si M[y] = 0 alors
            d[y] ← d[x] + c(x, y)
            pred[y] ← x
            ajouter(K, y, d[y])
            M[y] ← 1
        Sinon faire
            Si d(y) > d(x) + c(x, y) alors
                d(y) ← d(x) + c(x, y)
                pred[y] ← x
                supprime(K, y)
                ajouter(K, y, d[y])
            FinSi
        FinSi
    FinPour
FinTantque
```

6.4 Algorithme de Prim

On précise ici des notations concernant l'algorithme de Prim pour la recherche d'un arbre couvrant de coût minimal. On note r le sommet de plus petit indice de S. L'algorithme permet à partir d'un graphe $G = (S, A)$ de retourner un tableau $pred$ indiquant un arbre couvrant minimal, présenté comme enraciné au sommet r. Le tableau $pred$ est tel que $pred[r] = -1$ et $pred[x]$ indique le père de x pour tout sommet x différent de r.

Dans l'algorithme suivant, le tableau d sera tel que $d[x]$ pour un sommet $x \in S$ sera le coût d'une arête reliant x à l'arbre en construction. De même, le tableau M sera tel que, pour un sommet x, $M[x]$ pourra prendre trois valeurs : 0 pour indiquer que le sommet n'a pas encore été rencontré ; 1 si le sommet x est dit "ouvert", c'est-à-dire tel que tous ses voisins n'ont pas encore été traités ; ou 2 pour "fermé" si un sommet a été rencontré et n'est plus ouvert.

Voici une description de l'algorithme de Prim utilisant un tas K.

entrée : graphe $G = (S, A)$
sortie : tableau $pred$

Déclaration d'un tableau de réels d indicés sur les sommets
Déclaration d'un tableau d'entiers M indicés sur les sommets
$pred[x] \leftarrow -1 \ \forall x \in S$
$d[x] \leftarrow +\infty \ \forall x \in S$
$M[x] \leftarrow 0 \ \forall x \in S$
$initialise(K)$
$ajouter(K, r, 0)$
$d[r] \leftarrow 0$
$M[r] \leftarrow 1$
Tant que non $est\_vide(K)$ Faire
    $x \leftarrow recup\_min(K)$
    Pour tout voisin y de x faire
        Si $M[y] = 0$ alors
            $d[y] \leftarrow c(x, y)$
            $pred[y] \leftarrow x$
            $ajouter(K, y, d[y])$
            $M[y] \leftarrow 1$
        Sinon faire
            Si $M[y] \neq 2$ et $d(y) > c(x, y)$ alors
                $d(y) \leftarrow c(x, y)$
                $pred[y] \leftarrow x$
                $supprime(K, y)$
                $ajouter(K, y, d[y])$
            FinSi
        FinSi
    FinPour
    $M[x] \leftarrow 2$
FinTantque

Question 14
Implémenter et valider la structure de tas et les algorithmes de Dijsktra et de Prim. Pour cela, vous pouvez utiliser les valeurs suivantes :
- pour l'instance d198_20_400_30_9.gph :
une plus courte chaîne du sommet 0 au sommet 300 a pour coût : 515.614
l'arbre couvrant a pour coût : 2756.98
- pour l'instance d1291_10_400_0.2_6.gph :
une plus courte chaîne du sommet 0 au sommet 300 a pour coût : 542
l'arbre couvrant a pour coût : 13236.

7 Méthode approchée

On désire à présent implémenter la méthode approchée présentée en section 4. Pour cela, voici quelques indications et conseils.

7.1 Construction du graphe des distances $D(N)$

On peut facilement manipuler un tableau permettant de renuméroter les sommets terminaux : pour cela, une fois obtenu le nombre $nbterm$ des terminaux, on construit un tableau $Term$ tel que $Term[i]$ indique le numéro du $i^{\text{ème}}$ sommet du graphe de l'instance, $i \in \{0, ..., nbterm\}$.

Le graphe des instances peut être alors obtenu en utilisant itérativement l'algorithme de Dijkstra pour chaque paire de sommets terminaux. On définit le graphe $D(N)$ comme un graphe complet codé par liste d'adjacence avec $nbterm$ sommets et des coûts correspondant à une plus courte chaîne.

7.2 Construire le graphe H

Dans le graphe $D(N)$ obtenu dans la section précédente, on détermine un arbre couvrant par l'algorithme de Prim. La solution résultante est donc un tableau T indiquant un arbre sur $D(N)$.
On définit alors un graphe H vide de sommets et on va ajouter une à une ses arêtes de la façon suivante : Pour chaque arête $\{s, t\}$ de cet arbre, on calcule à nouveau l'arborescence des plus courtes chaînes partant de s, puis on en déduit une plus courte chaîne de s à t : pour cela, on utilise le tableau T en "remontant" l'arborescence de t à s de fils en père. On ajoute alors chaque arête de cette chaîne à H en testant si H ne contient pas déjà cette arête.

7.3 Ôter itérativement les sommets de degré 1 d'un graphe

On peut alors dérouler l'algorithme de Prim sur le graphe H, on récupère alors un arbre couvrant T_H sur une partie des sommets de l'instance d'origine. Notez que le codage de l'arborescence reste valide à ce moment-là.
On peut remarquer qu'il existe un algorithme simple basé sur les degrés des sommets du graphe. Pour cela on crée le vecteur Deg indicé sur les sommets et indiquant les degrés de chaque sommet dans l'arbre T_H. Ensuite, tant qu'il existe un sommet Steiner i avec $Deg[i] = 1$, on recherche son voisin j dans l'arbre T_H et on décrémente $Deg[j]$ de 1, ainsi que $Deg[i]$.
A la fin, on peut directement construire la liste chaînée des arêtes de l'arbre Steiner en conservant uniquement les arêtes de T_H ayant une valeur dans Deg strictement positive. Cette liste chaînée peut alors être directement affichée comme solution finale !

Question 15
Implémenter la méthode approchée et tester-là sur les instances fournies (la valeur obtenue dépendant de l'implémentation, nous ne pouvons pas vous fournir une valeur déterminée du coût de l'arbre Steiner à obtenir, néanmoins, un code-solution a fourni 9652 pour l'instance d1291_10_400_0.2_6.gph).

8 Mise en œuvre

8.1 Langage

Le choix du langage de programmation est libre.

9 Jeux d'essais et statistiques

Question 16
Pour les instances fournies, donner les valeurs des arbres Steiner obtenus. Donner la visualisation graphique de la solution d'une de ces instances.

Il n'est pas évident de tracer la courbe des temps d'exécution pour les instances de ce problème de l'arbre Steiner car sa complexité dépend de trois paramètres. Par contre, nous allons nous intéresser à la mesure de complexité expérimentale de l'algorithme de Prim.

Question 17
Pour l'ensemble des instances fournies, donner les temps CPU obtenus par l'algorithme de Prim. Déterminer un ensemble d'instances où le nombre d'arêtes est égal à k fois celui des sommets avec k faible. Donner alors une courbe de mesure de temps CPU pour ces instances. Est-ce que cette mesure correspond à la complexité attendue ?
Un lien utile pour réaliser des mesures CPU :
`www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2004/ue/aaea-2005fev/public/benchmarks.html`

Question 18
(Bonus)
Proposer des améliorations de la méthode approchée ou d'autres méthodes heuristiques pour déterminer des arbres Steiner de coûts faibles.

10 Organisation et dates

Binômage

Le travail est à effectuer par binôme.

Ramassage électronique

Les projets doivent être rendus le **4 Décembre 2014** au plus tard par mail à votre chargé de TD. Votre livraison sera constituée d'une archive `tar.gz` qui doit comporter un rapport décrivant votre implémentation et votre code ainsi que la description des tests de validation sur un benchmark d'instances.

Une **soutenance** est prévue lors de la dernière semaine de TD (semaine du 8 Décembre 2014).