Fiche Mini Projet

mmc
marc-michel.corsini@u-bordeaux.fr

18 janvier 2015

Le travail demandé représente une charge étudiant de l'ordre de 20h. Le projet étant réalisé par groupe de deux ou trois (exceptionnellement). Le travail à remettre est un rapport au format **PDF** et un code avec une page texte expliquant sa mise en œuvre.

1 Description

À partir d'un graphe non orienté, éventuellement pondéré dans $\mathbb{R}$. Il faudra partitionner le graphe en K classes au moyen de deux métaheuristiques vus en cours, de telle sorte que la somme des poids entre sommets n'appartenant pas à la même classe soit minimale. De plus il faudra s'assurer que les sommets du graphe soient répartis de manière **(à peu prés)** équitable.
Vous avez toute latitude pour définir, le nombre de classes et la notion d'« à peu prés équitable ». L'objectif du travail est de pouvoir comparer différentes approches de partitionnement, en fonction de la qualité de la solution trouvée, du temps de calcul, de la limite d'utilisation des méthodes en fonction de la taille du problème et éventuellement d'autres critères.
Dans un premier temps, vous vous restreindrez au partitionnement d'un graphe non modifiable. Pour mener à bien votre comparaison, il faudra implémenter les méthodes simples permettant de trouver la solution optimale afin de garantir que les méta-heursitiques implémentées soient cohérentes.

- Un algorithme d'énumération pouvant être explicite / implicite avec parcours BFS ou DFS ou Sac à dos ;
- Un algorithme glouton avec descente de gradient ;
- Ensuite la première métaheuristique sera le Recuit simulé.
- La seconde métaheuristique sera au choix :
  1. Méthode Tabou ;
  2. Réseau de Hopdield ;
  3. Algorithme génétique ;
  4. Agents ;

Vos algorithmes doivent être testés et validés, et doivent pouvoir traiter les exemples de graphes accessibles sur le site **Box.com** accessible via le lien sur la page **marcmichelcorsini/MIMSE**.

1.1 Structure fichiers fournis

Les fichiers de graphe à traiter sont basés sur une syntaxe simple, une ligne de commentaire débute par un dièse (#), la première ligne donne le nombre de sommets et le nombre d'arêtes. La seconde ligne le degré minimum et maximum du graphe, ensuite la liste des arêtes est un triplet de trois entiers, les deux extrémités et le poids de l'arête. Tous les fichiers fournis sont avec un poids de 1. Le dernier bloc donne pour chaque sommet son degré (sert uniquement à vérifier que tout s'est bien passé pendant la lecture)

```
# nbSommets nbAretes
10 30
# dmin dmax
```

```
3 7
# source but val
1 2 1
1 6 1
1 8 1
1 7 1
1 3 1
1 4 1
2 9 1
2 5 1
2 8 1
2 10 1
2 3 1
2 4 1
3 4 1
3 9 1
3 8 1
3 5 1
3 6 1
4 9 1
4 10 1
4 7 1
4 6 1
5 9 1
5 6 1
5 7 1
6 7 1
6 8 1
7 9 1
7 8 1
8 9 1
9 10 1
# sommet degre
1 6
2 7
3 7
4 7
5 5
6 6
7 6
8 6
9 7
10 3
```

2 fiches TP

Les fiches TP ci-dessous sont là pour vous donnez des pistes sur les différentes étapes, elles ne sont pas obligatoires mais certains points clefs seront utiles pour la construction de votre rapport final.

2.1 ficheTP01

Pour les premiers TP vous travaillerez sur les deux graphes suivants :

- $G_1 = (\{1, 2, 3, 4\}, \{12, 13, 23, 34, 44\})$
- $G_2 = (\{1, 2, 3\}, \{12, 13, 23\})$

Il vous faut réfléchir aux points suivants :

1. Définir la structure de stockage du graphe

2. Avoir deux variables G_1 et G_2 qui seront remplies avec les graphes correspondants
3. Définir la fonction que vous allez optimiser
4. Écrire la fonction qui, prend en paramètres une partition des sommets, un graphe, et renvoie en sortie la valeur de la fonction à optimiser.
5. Dans le cas où la valeur obtenue n'est pas le nombre d'arêtes, écrire une fonction qui, prend en paramètres une partition des sommets et un graphe, et renvoie en sortie le nombre d'arêtes entre les classes.
6. Définir, en français ce qu'est un mouvement élémentaire.

Travail à effectuer :

- La fonction à optimiser doit être implémentée
- Le codage de la partition effectuée
- La transformation élémentaire terminée et codée.

2.2 ficheTP02

On note FO la fonction objectif choisie. Les objectifs de cette séance sont :

- Décider si FO(s(P),G) se calcule directement ou à partir de FO(P,G). FO(,) est la fonction objectif ; P une partition de G ; s() un mouvement élémentaire.
- Implémenter une méthode énumérative qui cherche la solution optimale du partitionnement
- Comprendre les besoins pour la méthode du recuit simulé

2.2.1 Énumération

Énumérer c'est compter. Il suffit donc d'attribuer un entier à chaque partition possible du graphe. Si on considère un graphe à n sommets, que l'on veut partitionner en k classes, il existe k^n partitions possibles. Une idée simple consiste donc à noter les partitions sur n valeurs en base k.
Si vous avez choisi comme mouvement élémentaire « prendre un sommet au hasard et le changer de classe » toutes les partitions sont accessibles.
Si vous avez choisi comme mouvement élémentaire « prendre un sommet d'une classe et l'échanger avec un sommet d'une autre classe » toutes les partitions ne sont pas accessibles, il va falloir éliminer celles que vous ne pouvez pas atteindre.

2.2.2 Élimination

Nous séparons l'étude en 2 cas.

Deux classes

Dans le cas de deux classes, une astuce simple consiste à attribuer à une classe la valeur -1 et à l'autre la valeur 1. Cette transformation consiste à passer d'un codage binaire à un codage bipolaire grâce à la fonction f(p) = 2p -1.
Puis il suffit de faire la somme des digits de ce nouveau code. La partition sera dite accessible si la somme des digits vaut 0 (dans le cas où n est pair) +/- 1 sinon (on suppose que l'équitabilité est ici définie à 1 sommet près).
Dans le cas de G_2 les partitions possibles sont : 000, 001, 010, 011, 100 - les autres codages étant symétriques pourront être ignorés : après transformation bipolaire et sommation, on trouve les valeurs : -3, -1, -1, 1, -1 : il y a donc 4 partitions à examiner.

En résumé l'algorithme va énumérer les nombres de 0 à 2 puissance (n - 1) où n est le nombre sommets du graphe. Chaque nombre va être codé en bipolaire, si la somme respecte le critère alors, on lance l'évaluation de la fonction d'optimisation.

```
Input : G, n
Output : Partition et Score optimal

Best <- -1 // numero de la configuration optimale (entre 0 et 2^n - 1)
FOpt <- +infini // valeur de la fonction d'optimisation associee a Best

Pour i de 0 a 2^n - 1 Faire
 Calculer i en binaire
 Code <- Somme des digits en bipolaire
 Si Code = 0 ou Code = -1 ou Code = +1 Alors
    Calculer Foptim pour la partition consideree
    Si Foptim < FOpt Alors
       Best <- i
       FOpt <- Foptim
    Fsi
 Fsi
Fait
```

Cas général

Soit K le nombre de classes (K > 2), n le nombre de sommets. Il y a K^n partitions possibles. Chaque nombre de 0 à K^{n-1} va être codé sur n digits en base K. Du fait de l'équi-répartition supposée il y a dans chaque classe au plus ceil(n/K) sommets et au moins floor(n/K) sommets (ceil et floor étant respectivement les parties entières par valeur supérieure et inférieure). On va ensuite, associer K compteurs C0,...C(K-1) qui vont comptabiliser le nombre de digits 0 ... K-1 de la représentation en base K. Le test de validité va consister à contrôler que chaque compteur est compris entre les bornes min et max. L'exemple ci-après teste le cas, d'un graphe ayant 5 sommets que l'on veut partitionner en 3 classes. Le minimum est floor (5/3) = 1 ; le maximum ceil (5/3) = 2.

entier	ternaire	$C_0C_1C_2$	Décision
0	00000	5 0 0	Non
1	00001	4 1 0	Non
2	00002	4 0 1	Non
3	00010	4 1 0	Non
4	00011	3 2 0	Non
5	00012	3 1 1	Non
6	00020	4 0 1	Non
7	00021	3 1 1	Non
8	00022	3 0 2	Non
9	00100	4 1 0	Non
10	00101	3 2 0	Non
11	00102	3 1 1	Non
12	00110	3 2 0	Non
13	00111	2 3 0	Non
14	00112	2 2 1	**Oui**
15	00120	3 1 1	Non
16	00121	2 2 1	**Oui**
17	00122	2 1 2	**Oui**
...	...	...	...
2*(81+27+9+3+1)	22222	0 0 5	Non

Remarque : en fait il est inutile d'explorer au-delà de la répartition 0 5 0 correspondant au nombre ternaire 11111 qui vaut en décimal 81+27+9+3+1 soit 121. À vous de justifier cette allégation.

2.3 ficheTP03

Pour ce TP vous travaillerez sur les deux graphes suivants :

- $G_1 = (\{1,2,3,4,5,6\}, \{12,13,15,16,23,34,44,45,56\})$
- $G_2 = (\{1,\ldots,10\}, \{\{i,2i\},\{i,3i\},\{i,5i\}, \forall i \in X\})$

L'algorithme de descente de gradient est dépendant des conditions initiales. Cet algorithme trouve le minimum local de la fonction à optimiser accessible depuis la configuration initiale.
Le principe de l'algorithme de descente est le suivant :

```
Soit C_0 la configuration initiale
Soit m le mouvement élémentaire

i <- 0
fini <- faux
Tant que pas fini Faire
   Calculer fopt(C_i)
   Calculer V(C_i) =  C tel que C = m(C_i)
   Pour chaque C de V(C_i) Faire
      Calculer fopt(C)
   FPour
   Soit C_{i+1} de V(C_i) tel que f(C_{i+1}) = Min f(C)
   Si f(C_{i+1}) < f(C_i) Alors i <- i+1
   Sinon fini <- vrai
   Fsi
FTantque
Afficher f(C_i)
```

Les objectifs de cette séance sont :

- Etant donnée une configuration, être capable d'accéder à chaque configuration du voisinage.
- Mettre en place l'algorithme de descente de gradient pour une configuration initiale.
- Mettre en place le tirage aléatoire de la configuration initiale.
- Faire une boucle permettant d'obtenir le meilleur minimum local, le pire, la valeur moyenne. Le nombre d'itérations dépendra du nombre de sommets du graphe.

```
Input : G, n (nombre de sommets)

int f(n) // renvoie un nombre qui depend de n

Pour i de 1 a f(n) faire
 Choisir une configuration initiale
 Faire tourner l'algorithme de descente
Fait

Afficher le minimum
Afficher le maximum
Afficher la moyenne
```

2.4 ficheTP04

Les objectifs de cette séance sont :

- Transformer votre programme de telle sorte qu'il ne travaille que sur un graphe à la fois, cela a quelques conséquences telles que :
  - Il y a une variable pour la structure stockant le graphe
  - Il y a une variable pour chaque donnée pertinente relative au graphe, par exemple : nbSommets, nbAretes, Dmin, Dmax, ...

  - Cela peut modifier les signatures de vos méthodes.

- Étant donnée une configuration, être capable d'extraire aléatoirement une configuration de son voisinage.
- Être capable de calculer de manière automatique les paramètres utiles pour le Recuit Simulé tels que :
  - Temperature initiale : pour cela vous devez estimer la qualité de la configuration initiale par rapport à son voisinage.
  - Longueur de la chaîne de Markov, en théorie cette chaîne doit être infinie pour aboutir à l'équilibre thermodynamique à température constante. En pratique, il faut qu'elle soit suffisamment longue.
  - Définition du critère d'arrêt « système figé » : Température minimale atteinte, amélioration epsilonesque, durée maximale atteinte, ...
  - Mettre en place un chronomètre afin de mesurer la performance des différents algorithmes
  - Mettre en place une sortie des résultats afin de pouvoir analyser les performances dans votre rapport