Construction de graphes contraints

Sujet proposé par Gilles Schaeffer

Gilles.Schaeffer@lix.polytechnique.fr

URL de suivi : http://www.enseignement.polytechnique.fr/profs/informatique/Gilles.Schaeffer/INF431/projetX12.html

1 Préambule

Dans le contexte de la modélisation des réseaux (internet, sociaux, etc) se pose entre autre le problème de construire de bons modèles des réseaux réels : plutôt que de tester ou comparer les nouvelles solutions algorithmiques (algorithmes de routages ou d'exploration par exemple) sur un réseau réel, on souhaite commencer par simuler leur exécution sur un grand nombre de graphes tests. Ceci permet d'une part de limiter le coût de l'expérimentation, et d'autre part, de mieux comprendre le lien entre forme du réseau et efficacité des algorithmes. Cependant la question de la ressemblance de ces graphes tests aux réseaux réels qu'ils sont sensés modéliser est difficile : pour que les simulations aient un minimum de valeur prédictive, il faudrait construire des graphes qui satisfont un certain nombre de contraintes, sans pour autant être trop particulier.

En général la démarche adoptée est de se fixer une classe de graphe (disons les graphes connexes, ou bien les graphes qui peuvent se dessiner dans le plan sans croisement, ou bien les graphes d'attachement préférentiel obtenus en ajoutant incrémentalement des arêtes, etc) et de considérer des graphes à n sommets pris au hasard dans cette classe (uniformément, ou suivant une distribution de probabilité bien choisie). Le problème est de trouver des classes intéressantes de graphes pour lesquelles on sache construire efficacement de gros exemples representatifs : ainsi on entend beaucoup parler des graphes *petits mondes* mais la bonne spécification de cette famille de graphes n'est pas encore bien claire...

Le but de ce projet est d'étudier un tel problème de construction de graphes contraints. Il s'agit essentiellement d'un projet d'algorithmique de graphes, faisant appel à des structures de données et des algorithmes du type de ceux vu en cours (Chapitre 5).

2 Détail du sujet

2.1 Contraintes de degrés

Nous étudions dans ce projet un type classique de contraintes qui porte sur les degrés des sommets. Rappelons qu'un graphe est donné par un ensemble de sommet X et un ensemble d'arêtes $E \subset \mathcal{P}_2(X)$, où $\mathcal{P}_2(X)$ désigne l'ensemble des paires (non ordonnées) de sommets distincts. Le degré d'un sommet est le nombre d'arêtes qui lui sont incidentes, ou autrement dit, le nombre de ses voisins.

La classe de graphes auxquels on va s'intéresser est obtenue en fixant une distribution de degré : pour $d_1 \geq d_2 \geq \ldots \geq d_n \geq 1$, on note $\langle d_1, \ldots, d_n \rangle$ l'ensemble des graphes à n sommets ayant pour degré exactement $d_1, \ldots, d_n$. On note de plus $\langle d_1, \ldots, d_n \rangle_C$ le sous-ensemble de ces graphes qui sont connexes.

L'objectif du projet est de programmer un générateur qui, étant donnés les degrés $d_1, \ldots, d_n$, construit, lorsque c'est possible, un graphe de $\langle d_1, \ldots, d_n \rangle_C$.

2.2 Algorithmes

Un algorithme glouton Étant donnés $d_1 \geq \ldots \geq d_n$ on veut construire un graphe $\langle d_1, \ldots, d_n \rangle$ s'il en existe un. Pour cela on maintient un couple (G, δ) où G est un graphe à n sommets et δ est un tableau qui indique le nombre d'arêtes encore attendue sur chaque sommet :

1. au départ G est le graphe à n sommets sans arêtes et $\delta[i] = d_i$, pour tout $i = 1, \ldots, n$;
2. répéter pour $i = 1, \ldots, n$:
   (a) joindre le sommet i de G à chacun des $\delta[i]$ sommets suivants par une arête ;
   (b) mettre à jour δ : faire $\delta[i+j] := \delta[i+j] - 1$ pour $j = 1, \ldots, \delta[i]$, puis $\delta[i] := 0$;
   (c) si l'un des $\delta[j]$ devient négatif, produire un certificat du fait que $\langle d_1, \ldots, d_n \rangle = 0$;
   (d) sinon réordonner les entrées $\delta[i+1], \ldots, \delta[n]$ en ordre décroissant
3. le graphe G obtenu lorsque $\delta = [0, \ldots, 0]$ appartient à $\langle d_1, \ldots, d_n \rangle$.

Dans un premier temps vous devriez vous convaincre de la validité de l'algorithme : il faut pour cela expliquer en quoi consiste le certificat produit à l'étape (c).

L'algorithme précédent ne garantit pas la connexité du graphe construit : on obtient un graphe de $\langle d_1, \ldots, d_n \rangle$ mais pas nécessairement de $\langle d_1, \ldots, d_n \rangle_C$.

Échange d'arêtes et connexité Pour qu'on puisse espérer trouver un graphe connexe il faut que le nombre d'arêtes soit suffisant : en effet un graphe connexe à n sommets admet un arbre couvrant de ses n sommets, lequel est fait de $n-1$ arêtes. Ceci se traduit par la condition $\sum_i d_i \geq 2(n-1)$ nécessaire pour que $\langle d_1, \ldots, d_n \rangle_C$ soit non vide. Si cette condition est vérifiée et si on dispose d'un graphe de $\langle d_1, \ldots, d_n \rangle$ on peut alors le transformer en un graphe connexe en itérant la transformation locale suivante, appelée *flip* :

– soit G un graphe de $\langle d_1, \ldots, d_n \rangle$ avec $c \geq 2$ composantes connexes ;

– si $\sum_i d_i \geq 2(n-1)$ alors au moins une de ses composantes connexes contient au moins autant d'arêtes que de sommets et donc contient un cycle (le montrer par récurrence sur le nombre de composantes connexes) ; soit $a = (x, y)$ une arête de ce cycle ;
– soit $a' = (x', y')$ une arête prise dans une autre composante connexe du graphe ;
– remplacer a et a' par les arêtes (x, x') et (y, y') ; le nouveau graphe ainsi obtenu est toujours dans $\langle d_1, \ldots, d_n \rangle$ et il a $c - 1$ composantes connexes (le montrer).

Remarquons que cet algorithme nécessite de rechercher des cycles et des composantes connexes, ce qui se fait à l'aide des parcours vu en cours.

2.3 Énumération et graphes aléatoires

Une fois qu'on est capable d'engendrer au moins un graphe étant donné une suite de degré, on souhaite pouvoir les engendrer tous, ou en engendrer un au hasard. Dans les deux cas on va utiliser le fait que des flips du type définis à la section précédente permettent de passer d'un graphe de $\langle d_1, \ldots, d_n \rangle$ à n'importe quel autre.

Dans cette partie on ne s'occupera pas de la connexité. Pour énumérer tous les graphes de $\langle d_1, \ldots, d_n \rangle$ (on parle aussi de génération exhaustive), on va explorer à partir d'un graphe l'ensemble des graphes accessibles par flip de deux arêtes $a = (x, y)$ et $a' = (x', y')$ quelconque. On peut voir ce problème comme celui de l'exploration d'un (méta)graphe $\mathcal{G}$ dont l'ensemble des sommets est $\langle d_1, \ldots, d_n \rangle$ et les arêtes sont données par les flips. On admettra que ce graphe est connexe : le problème d'énumération revient alors à programmer un parcours du graphe $\mathcal{G}$ suivant votre méthode de parcours favorite.

Pour tirer un graphe aléatoire dans $\langle d_1, \ldots, d_n \rangle$ on effectue une marche aléatoire sur le $\mathcal{G}$: à partir du graphe initial engendré par l'algorithme de construction, on effectue une séquence flips aléatoires. Le nombre de flips effectués sera donné comme paramètre à l'algorithme : d'un côté plus on fait de flips plus l'algorithme est lent, d'un autre côté il faut faire suffisament de flips pour *oublier* la configuration initiale dont on est partie. (Le choix du nombre de flips nécessaires est lié à la convergence de la chaine de Markov associée, les étudiants portés pour la chose pourront consulter la litérature d'analyse d'algorithmes fournie sur la page de suivi.)

3 Extention aux contraintes de degré joints

Dans un second temps il est envisageable de raffiner la classe de graphes considérée en fixant non seulement les degrés mais aussi une matrice d'adjacence par degrés, qui indique, pour tout i, j, le nombre de couples de sommets (x, y) tels que x et y sont voisins et de degré i et j respectivement. Plus précisément, on se donne une partition de l'ensemble $V = \{1, \ldots, n\}$ des sommets en classes $V_1, \ldots, V_k$, ($V_i \cup V_j = \emptyset$ et $\bigcup_i V_i = V$) et une matrice symétrique $D = (d_{i,j})_{1 \leq i,j \leq k}$ et on cherche un graphe G tel que :
– le nombre total d'arêtes joignant des sommets de V_i et de V_j soit $d_{i,j}$ pour tout $i \neq j$.
– le nomobre total d'arêtes entre les sommets de V_i soit $d_{i,i}$
– les sommets de la classe V_i ont tous le même degré d_i, qui du coup est donné par la

relation

$$|V_i| \cdot d_i = 2d_{i,j} + \sum_{j \neq i} d_{i,j}$$

On note $\langle V, d, D \rangle$ l'ensemble des graphes satisfaisant ces conditions et $\langle V, d, D \rangle_C$ le sous-ensemble de ces graphes qui sont connexes.

L'intérêt de cette définition vient de l'observation expérimentale que pour de bon choix de matrice D les graphes pris au hasard dans $\langle V, d, D \rangle_C$ semblent ressembler plus aux graphes de réseaux réels que les graphes contraints seulement par les degrés, ou engendré par d'autres modèles classiques de graphes. Ce type d'affirmation est néanmoins toujours sujet à polémique : nous nous contenterons d'aborder le problème, déjà intéressant et difficile en soi, de construire des graphes de $\langle V, d, D \rangle_C$.

Pour traiter ce problème on utilisera l'algorithme de Amanatidis, Green et Mihail, dont on trouvera une description sur la page de suivi.

4 Travail demandé

Il est demandé de programmer au minimum le générateur de graphe connexe tel que décrit à la section 2.2, ainsi qu'un programme de test qui calcule des statistiques sur les graphes obtenus. Il est demandé de fournir les résultats de tests sur une série significative d'exemples, aussi gros que possible : une approche possible est de récuperer des graphes réels (cartes de portions d'internet, graphes de collaborations, ou autre), et de comparer leurs caractéristiques avec celle des graphes de même distribution de degré engendrés par votre algorithme.

4.1 Format d'entrée et sortie

Générateur La liste des degrés sera lue dans un fichier texte, tandis que les options (nom des fichiers d'entrée/sortie, connexité ou non du graphe demandé, version aléatoire ou non,...) seront prises en ligne de commande.

Le graph produit en sortie devra pouvoir être stoqué dans un fichier sous différents formats, incluants au minimum la matrice d'adjacence et le format graphml, décrit à l'adresse

`http://graphml.graphdraing.org/primer/graphml-primer.html`

Ce format est très riche mais les trois ou quatre balises principales sont suffisantes pour décrire un graphe.

Testeur Le testeur devra être capable de relire un graphe à l'un des formats produits par le générateur. En fonction des options prises en ligne de commande il calculera des statistiques sur le graphe, dont au minimum les deux suivantes :

- Le diamètre du graphe et le profil des sommets : pour chaque sommet le nombre de sommets à distance i pour tout i.
- Le nombre et la taille des composantes connexes.

4.2 Critères d'évaluation

En plus des critères standards d'évaluation du projet rappelés sur la page web des projets, on prendra soin d'expliquer les choix d'implantation et l'organisation des données adoptées pour gérer les graphes.

4.3 Extensions possibles

Il est bien entendu possible de programmer une visualistation des graphes produits mais il est plutôt conseillé d'utiliser un programme existant pour cela.

Une extension plus intéressante serait de traiter le problème évoqué aux sections 2.3 et 3. On trouvera pour cela des pistes de lecture sur la page de suivi.