Computing communities in large networks using random walks

Pascal Pons, Matthieu Latapy, 2005

(Présentation – 6 février 2006)

Plan

Communautés

- Graphes globalement clairsemés, mais **localement denses** $\rightarrow$ communautés
- Déterminer les communautés : décider d'une **partition** $\mathcal{P} = C_1, \ldots, C_k$ de V
- Une «bonne» structure en communautés : $\frac{|\text{arêtes internes aux } C_i|}{|\text{arêtes entre les } C_i|}$ doit être faible
- Graphes **connexes**, **non orientés**, arêtes **non pondérées**, où chaque nœud possède une arête vers lui-même (**boucle**)

Marche aléatoire

- Le graphe a une **matrice d'adjacence** A, où $A_{ij} \in \{0; 1\}$
- **Marche aléatoire** : départ d'un sommet donné, parcours du graphe en choisissant à chaque fois une arête au hasard
- Constatation initiale : une marche aléatoire dans un graphe tend à être «**piégée**» dans une composante fortement connexe
- **Probabilité de transition** :

$$P_{ij} = \frac{A_{ij}}{d(i)}$$

- Après un chemin de longueur t :

$$(P^t)_{ij} = P_{ij}^t$$

Marches aléatoires : 2 propriétés

Propriété (1)

$$\forall i \qquad \lim_{t \to \infty} P_{ij}^t = \frac{d(j)}{\sum_k d(k)}$$

Propriété (2)

$$\forall i \quad \forall j \qquad \frac{P_{ij}^t}{d(j)} = \frac{P_{ji}^t}{d(i)}$$

Distance entre nœuds

- La distance va dépendre de t ; doit être petite à l'intérieur des communautés, grande entre elles
- t : assez grande pour contenir des infos, mais pas trop à cause des deux propriétés précédentes
- P_{ij}^t est influencée par $d(j)$ (le marcheur)
- Deux nœuds i et j de la même communauté ont tendance à «voir» tous les autres nœuds de la même façon :

$$\forall k \qquad P_{ik}^t \simeq P_{kj}^t$$

Definition (distance entre i et j)

$$r_{ij} = \sqrt{\sum_{k=1}^{n} \frac{(P_{ik}^t - P_{jk}^t)^2}{d(k)}}$$

Distance entre communautés

Proba. d'aller depuis C ($i \in C$ choisi au hasard) vers j en t pas :

$$P^t_{Cj} = \frac{1}{|C|} \sum_{i \in C} P^t_{ij}$$

Definition (distance entre C_1 et C_2)

$$r_{C_1 C_2} = \sqrt{\sum_{k=1}^{n} \frac{(P^t_{C_1 k} - P^t_{C_2 k})^2}{d(k)}}$$

Remarque : $r_{ij} = r_{\{i\}\{j\}}$ et $r_{iC} = r_{\{i\}C}$

Principe de l'algorithme

Au début, $\mathcal{P}_1 = \{\{v\}, v \in V\}$ et on calcule toutes les distances entre nœuds voisins. Puis à l'étape k :

- choisir $\mathcal{C}_1$ et $\mathcal{C}_2$ de $\mathcal{P}_k$ (critère de distance détaillé ensuite)
- fusionner $\mathcal{C}_1 \cup \mathcal{C}_2 = \mathcal{C}_3$ et créer la nouvelle partition : $\mathcal{P}_{k+1} = (\mathcal{P}_k \backslash \{\mathcal{C}_1; \mathcal{C}_2\}) \cup \{\mathcal{C}_3\}$
- mettre à jour les distances entre communautés (en fait, adjacentes seulement)

Chaque étape définit une $\mathcal{P}_k$. Après $n-1$ étapes, on a $\mathcal{P}_n = \{V\}$.

Choix des communautés à fusionner

- Choix uniquement parmi les communautés adjacentes (rapidité, connexité)
- On cherche à minimiser la quantité globale

$$\sigma_k = \frac{1}{n} \sum_{C \in \mathcal{P}_k} \sum_{i \in C} {r_{iC}}^2 = \frac{1}{n} \sum_{C_i \in \mathcal{P}_k} \gamma_i$$

(bonne appartenance de chaque nœud à sa communauté)

- Pour cela on calcule

$$\begin{aligned} \Delta\sigma(C_1, C_2) &= \frac{1}{n}\Big(\sum_{i \in C_3} {r_{iC_3}}^2 - \sum_{i \in C_1} {r_{iC_1}}^2 - \sum_{i \in C_2} {r_{iC_2}}^2 \Big) \\ &= \frac{1}{n}(\gamma_3 - \gamma_1 - \gamma_2) \end{aligned}$$

à minimiser parmi les possibilités de fusion (au plus m)

Choix d'une partition

- Première méthode : maximiser la **modularité**

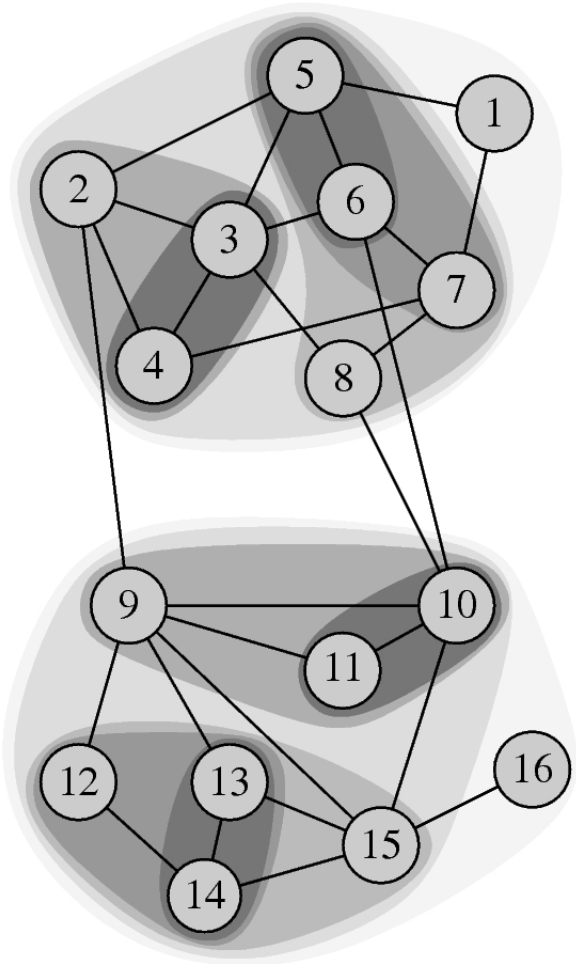
$$Q(\mathcal{P}) = \sum_{C \in \mathcal{P}} (e_c - {a_c}^2)$$

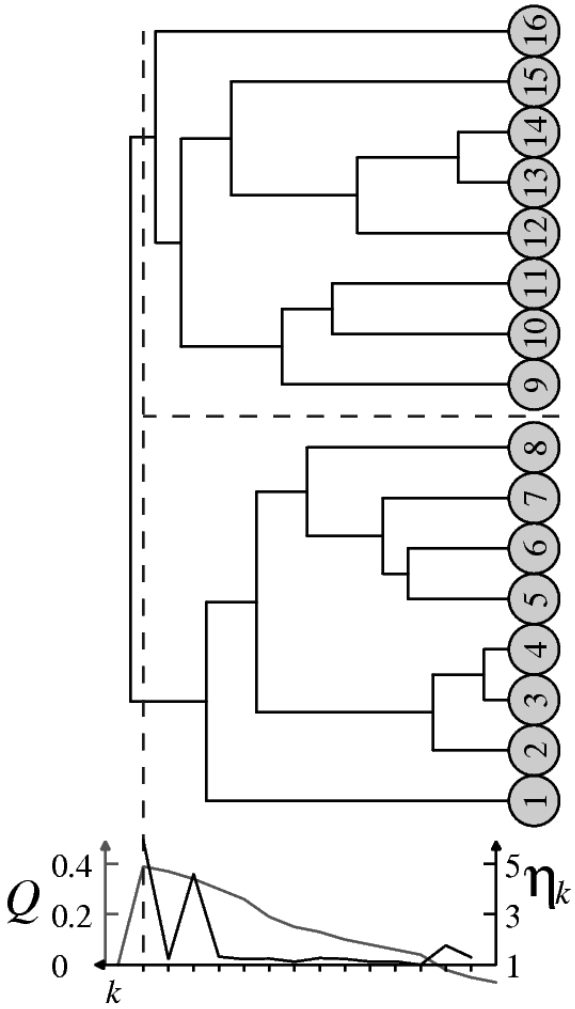
- Deuxième méthode : trouver les maxima de

$$\eta_k = \frac{\Delta\sigma_k}{\Delta\sigma_{k-1}}$$

(plus flexible, taille des communautés)

Choix d'une partition





Résultats

- Globalement bon pour le temps de calcul
- Excellent sur la «qualité» des communautés retrouvées (exemples artificiels mais aussi réels)
- Particulièrement adapté pour les grands graphes
- Réseaux : club de karaté (30), protéines (600), arXiv – citations (10 000), Internet (150 000)

Conclusion

- Meilleur compromis temps de calcul/qualité, pour les grand graphes. Contrepartie : besoin de beaucoup de mémoire
- Applicable aux graphes pondérés
- Souci du monde réel !
- Une piste pour les puces...
- Pas de vraie étude sur t ! (expérimentalement, $t = 5$ marche bien)
- Possible développement futur : communautés non disjointes

`http://www.manucornet.net/these/confs/`