minerLC

1 Introduction

minerLC est un programme qui identifie des patterns clos abstraits dans des données associées à un graphe de relations (orientées ou non) permettant ainsi de découvrir des connaissances locales.

2 Dossiers de *minerLC*

Un dossier *miner* contient l'ensemble des programmes, données et documentation associés à *minerLC*, dans les dossiers suivants:

- *doc* qui contient la documentation courante,
- *data* qui contient un exemple de données d'entrées (*mougel.ri* pour l'ancien format et *mougel.nri* pour le nouveau),
- *src* qui contient les programmes sources.

3 Compiler *minerLC*

Aller dans le dossier *minerLC* et taper *make minerlc* (le compilateur doit être compatible avec *c++11*), un executable *minerLC* est créé dans le répertoire *bin* sous *minerLC*. Pour supprimer les fichiers objets et l'exécutable taper *make clean*.

4 Lancer *minerLC*

Aller dans le répertoire *bin* et taper *minerLC* avec les options adéquates (voir ci-après). Par exemple :

```
minerLC mougel.nri
```

exécute *minerLC* à partir des données contenues dans le fichier *mougel.nri*.

Le nom du fichier contenant les données à traiter est l'unique paramètre obligatoire et doit être placé en premier. Les autres paramètres décrits ci-après sont optionnels et peuvent être mis dans n'importe quel ordre.

4.1 Paramètres généraux

- *-local* : pour chercher des clos locaux dans les composantes connexes du graphe courant (par défaut les clos recherchés sont globaux),
- *-min_sup n* : n détermine la fréquence minimale (sur les données) des clos recherchés (par défaut $n = 1$),
- *-lm x* : x determine le seuil de modularité locale. Chaque clos satisfera ce seuil et min_sup. Par ailleurs le programme sort 4 nombre : *lm* (nombre de clos satisfaisant la modularité locale), *lme* (nombre de clos satisfaisant une estimation de la modularité locale), *nec* nombre de clos qu'il est nécessaire d'explorer pour obtenir l'ensemble des clos solutions) et *f* (nombre de clos solutions les plus spécifiques par rapport au graphe d'exploration de minerLC).
- *-tg* : on recherche les 3-2 communautés à partir des composantes connexes du graphe des triangles (ce paramètre force implicitement le paramètre *-local*),
- *-eg* : (en plus de *tg*) on recherche les 3-2 communautés à partir des composantes connexes du graphe des arêtes (ce paramètre force implicitement le paramètre *-local*),
- *-clique k* : on recherche les 2-k communautés à partir des composantes connexes du graphe des arêtes, en ne conservant que les aretes impliquées dans une k-clique (ce paramètre force implicitement le paramètre *-local*).
- *-crown k* : on recherche les 2-k communautés à partir des composantes connexes du graphe des arêtes en ne conservant que les arêtes impliquées dans une k-couronne (ce paramètre force implicitement le paramètre *-local*).
- *-save_graphs* : sauve l'ensemble des graphes rencontrés pendant la recherche des clos sous format *.dot* (attention : cette option utilise le nom du fichier de données (paramètre 1) qui ne doit contenir ni chiffre, ni '.', ni aucun symbole comme '.', '*', '/', etc...)

5 *minerLC* et graphes non orientés

Si l'on n'utilise pas l'option *-o* le fichier de données fourni en entrée doit être non orienté.

5.1 Format des données d'entrée

Le fichier *mougel.ri* du dossier *data* contient un exemple de données d'entrées d'un graphe non orienté. Les relations sont décrites sous la forme d'une liste (redondante) d'adjacences :

```
# commentaire
A | B | C | D | E | F |
blues | folk | jazz | pop | rock
0 0,2                 // A (0) est décrit par {blues (0), jazz (2)}
1 2,3,4               // B (1) est décrit par {jazz (2), pop (3), rock (4)}
2
3 2,4
4 2,3,4
5 0,1,2
#
0 1,2                 // A(0)-B(1), A(0)-C(2)
1 0,3                 // B-A, B-D
2 0                   // C-A
3 1,5                 // D-B, D-F
4                      // E lié à aucun noeud
5 3                   // F-D
```

Les paramètres énumérés ci-après concernent les abstractions qui sont de 4 types : abstraction en triangle (t), en degré (d), nearStar (sd et sk), k-dense (k) et en composantes connexes minimales (c). Ces abstractions peuvent être composées et appliquées au niveau global ou/et local. Les abstractions dont les paramètres se termine par 1, comme t_1, s'appliquent aux niveaux global ET local, celles dont les paramètres se terminent par 2, par exemple $t2$, ne s'appliquent qu'au niveau local.

- *-d1 d* : on conserve les nœuds de degré supérieur ou égal à d,
- *-t1* : on conserve les nœuds impliqués dans un triangle,
- *-c1 n* : on conserve les nœuds qui font parti d'une composante connexe de taille au moins n,

- *-k1 n* : (k-dense) on conserve les arêtes (x, y) telles que x et y possèdent au moins $n - 2$ voisins communs,
- *-top1 α* : abstraction topologique ($0 < \alpha < 1$). Soit G le graphe initial. Soit un pattern p, G_p est le sous-graphe induit par p. L'abstraction topologique conserve tout sommet x de G_p si x, ou un voisin de x dans G_p, vérifie :
$$\frac{d_{G_p}(x) + 1}{d_G(x) + 1} \geq \alpha,$$
où $d_G(x)$ est le degré du sommet x dans G, et $d_{G_p}(x)$ est le degré de x dans G_p.
- *-sd1 d* : premier paramètre de nearStar : d est le degré minimal d'un hub,
- *-sk1 k* : second paramètre de nearStar : k est la distance maximale d'un noeud à un hub,
- *-d2 d* : on conserve les nœuds de degré supérieur ou égal à d,
- *-t2* : on conserve les nœuds impliqués dans un triangle,
- *-c2 n* : on conserve les nœuds qui font parti d'une composante connexe de taille au moins n,
- *-k2 n* : (k-dense) on conserve les arêtes (x, y) telle que x et y possèdent au moins n voisins communs,
- *-top2 α* : abstraction topologique ($0 < \alpha < 1$) (voir *-top1*),
- *-sd2 d* : premier paramètre de nearStar : d est le degré minimal d'un hub,
- *-sk2 k* : second paramètre de nearStar : k est la distance maximale d'un noeud à un hub.

Par défaut aucune abstraction n'est appliquée.

6 *minerLC* et graphes orientés

6.1 Format des données d'entrée

Le fichier *mougel_oriente.ri* du dossier *data* contient un exemple de données d'entrées d'un graphe orienté. Les relations sont décrites sous la forme d'une liste de successeurs.

```
# commentaire
A | B | C | D | E | F |
blues | folk | jazz | pop | rock
0 0,2             // A (0) est décrit par {blues (0), jazz (2)}
1 2,3,4           // B (1) est décrit par {jazz (2), pop (3), rock (4)}
2
2
3 2,4
4 2,3,4
5 0,1,2
#
0 1,2                 // A (0) ->B (1) , A (0) -> C (2)
1 3                   // B->D
2 0                   // C->A
3 5                   // D->F
4                     // E n'a pas de successeur
5 3                   // F->D
```

6.2 Paramètres spécifiques aux graphes orientés

- *-o* : precise que le graph est orienté,
- *-d1_in d* : on élimine les nœuds de degré entrant inférieur à d,
- *-d1_out d* : on élimine les nœuds de degré sortant inférieur à d,
- *-d1_simple d* : s'applique à un graphe orienté qu'on considérera comme un graphe (SIMPLE) non orienté, élimine les nœuds de degré inférieurs à d,
- *-sd_in1 d* : premier paramètre de nearStar in : d est le degré minimal d'un hub,
- *-sk_in1 k* : second paramètre de nearStar in : k est la distance maximale d'un noeud à un hub,
- *-sd_out1 d* : premier paramètre de nearStar out : d est le degré minimal d'un hub,
- *-sk_out1 k* : second paramètre de nearStar out : k est la distance maximale d'un noeud à un hub,

- *-hub1 h* : premier paramètre de *hub_authority* : h est le degré minimal d'un hub,
- *-aut1 a* : second paramètre de *hub_authority* : a est le degré minimal d'une autorité. L'abstraction *hub_autority* conserve les noeuds qui sont adjacents_in à au moins a hubs OU adjacents_out à au moins h autorités.
- *-hub_aut1 h a* version classique de *hub_authority*. ex) *-hub_aut1 3 2* où le degré minimal d'un hub est 3 et le degré minimal d'une autorité est 2.
- *-d2_in d* : on élimine les nœuds de degré entrant inférieur à d,
- *-d2_out d* : on élimine les nœuds de degré sortant inférieur à d,
- *-d2_simple d* : s'applique à un graphe orienté qu'on considérera comme un graphe (SIMPLE) non orienté, élimine les nœuds de degré inférieurs à d,
- *-sd_in2 d* : premier paramètre de nearStar in : d est le degré minimal d'un hub,
- *-sk_in2 k* : second paramètre de nearStar in : k est la distance maximale d'un noeud à un hub,
- *-sd_out2 d* : premier paramètre de nearStar out : d est le degré minimal d'un hub,
- *-sk_out2 k* : second paramètre de nearStar out : k est la distance maximale d'un noeud à un hub,
- *-hub2 h* : premier paramètre de *hub_autority* : h est le degré minimal d'un hub,
- *-auth a* : second paramètre de *hub_autority* : a est le degre minimal d'une autorité,
- *-hub_aut2 h a* version classique de *hub_authority*. ex) *-hub_aut2 3 2* où le degré minimal d'un hub est 3 et le degré minimal d'une autorité est 2.

Par défaut aucune abstraction n'est appliquée.

L'algorithme suivant est utilisé pour l'abstraction *hub_aut* version classique :

Algorithm 1 Algorithme hub_aut(G, h, a)

Input
- G /* graphe initial */
- h /* degré min hub */
- a /* degré min autorité */

Output
X /* sommets du graphe abstrait */

Begin
$X \leftarrow$ sommets(G) ;
$H \leftarrow \{x \mid dout(x) \geq h\}$;
$A \leftarrow \{x \mid din(x) \geq a\}$;
repeat
  $H_p \leftarrow H$; $A_p \leftarrow A$;
  $H \leftarrow \emptyset$; $A \leftarrow \emptyset$;
  for $x \in X$ **do**
    $dH(x) \leftarrow dout_{A_p}(x)$; /* degré out vers les nœuds de A_p */
    $dA(x) \leftarrow din_{H_p}(x)$; /* degré in depuis les nœuds de H_p */
    if $dH(x) < h$ and $dA(x) < a$ **then**
      retirer x de X ;
    else
      if $dH(x) >= h$ **then**
        ajouter x à H
      end if;
      if $dA(x) >= a$ **then**
        ajouter x à A
      end if
    end if
  end for
until $|A| \neq |A_p|$ or $|H| \neq |H_p|$
return X
End.

7 Exemples d'utilisation de *minerLC*

7.1 Version globale

Affiche les triplets $(c, e(c), e_1(c))$ où le clos c satisfait une abstraction. De plus, $e(c)$ est l'extension non abstraite de c et $e_1(c)$ sont extension abstraite. Par exemple

```
minerLC mougel.ri  -t1 -d1 5
```

affiche les triplets $(c, e(c), e_1(c))$ où c est abstrait selon une abstraction de triangles et de degrés supérieur ou égal à 5.

Pour un graphe orienté :

```
minerLC mougel_oriente.ri  -o d1_in 2 -d1_out 3
```

affiche les triplets $(c, e(c), e_1(c))$ où c est abstrait sur un graphe orienté (-o) selon une abstraction de degré entrant supérieur ou égal à 2 et de degré sortant supérieur ou égal à 3.

7.2 Version locale sans abstraction locale spécifique

Affiche les septuples (c, $e(c)$, $e_1(c)$, $e(l)$, $e_1(l)$, cc, l) où le clos c et le clos local l satisfont la même abstraction. De plus, $e(c)$ est l'extension non abstraite de c, $e_1(c)$ est l'extension abstraite de c, $e(l)$ est l'extension non abstraite de l, $e_1(l)$ l'extension abstraite de l et cc est la composante connexe de clos l. Ainsi :

```
minerLC mougel.ri  -local -t1 -c1 5
```

affiche les septuples (c, $e(c)$,$e_1(c)$, $e(l)$, $e_1(l)$, cc, l) où le clos global c et le clos local l satisfont une abstraction de triangles et de composante connexe supérieure ou égale à 5.

7.3 Version locale avec abstraction locale spécifique (a_2)

Affiche les septuples (c, $e(c)$, $e_1(c)$, $e(l)$, $e_1(l)$, cc, l), l où le clos c satisfait une abstraction a_1. Le clos local l est abstrait par l'abstraction a_1 ET par une abstraction a_2. $e_1(c)$ est l'extension abstraite (par a_1) de c, $e_1(l)$ est l'extension abstraite (par a_1) de l, et cc est le composante connexe de clos local l abstraite par a_1 et a_2. Par exemple :

```
minerLC mougel.ri  -local -t1 -d2 5
```

affiche les septuples (c, $e(c)$, $e_1(c)$, $e(l)$, $e_1(l)$, cc, l) où le clos global c est abstrait par une abstraction de triangles, et le clos local l est de plus abstrait par une abstraction de degré supérieure ou égale à 5.

7.4 Recherche des 3-2 communautés

Affiche les septuples (c, $e(c)$, $e_1(c)$, $e(l)$, $e_1(l)$, cc, l) où le clos c satisfait une abstraction a_1. Le clos local l est abstrait par l'abstraction a_1 ET par une abstraction a_2. Par ailleurs, l est le clos de la communauté associée à une composante connexe du graphe des triangles. La taille minimale des communautés est définie par le paramètre *min_sup*. Par exemple :

```
minerLC mougel.ri   -d1 4 -sd2 4 -sk2 1 -tg -min_sup 5
```

affiche les les septuples (c, $e(c)$, $e_1(c)$, $e(l)$, $e_1(l)$, cc, l) où le clos global c est abstrait par une abstraction de degrés supérieur ou égal à 4. Le clos local l satisfait l'abstraction de degré ET une abstraction nearStar de degré 4 et de distance 1. Enfin, l est associé à une communauté, issue d'une composante connexe du graphe des triangles, de taille supérieure ou égale à 5.

On peut aussi rechercher les 3 communautés en passant par le graphe des arêtes plutot que le graphe des triangles. il faut utiliser conjointement les options *-tg* et *-eg* :

```
minerLC mougel.ri   -d1 4 -sd2 4 -sk2 1 -tg -eg -min_sup 5
```

De manière générale, on n'a pas l'assurance que les communautés satisfont l'abstraction de degré, la vérification pouvant être couteuse. Il faut utiliser le paramètre *-ca* pour forcer cette vérification :

```
minerLC mougel.ri  -local -d1 4 -sd2 4 -sk2 1 -tg -min_sup 5 -ca
```