Université de Caen basse normandie
Laboratoire GREYC

FS^3: Méthode d'échantillonnage pour les K-plus fréquents sous-graphes

Stagiaire :
Khalil Youcef **LAGRAA**

Tuteur :
Albrecht **ZIMMERMANN**

Auteur :
Tanay Kumar Saha and Mohammad Al Hasan

Sommaire

1. **FS^3 (Fixed Size Subgraph Sampler).**
2. **Processus.**
3. **Approche.**
    a. **Transitions d'états.**
    b. **Fonction de score.**
4. **L'algorithme d'échantillonnage.**
5. **Manageur de chaîne.**
6. **L'algorithme de FS^3**
7. **Conclusion.**

FS : Fixed Size Subgraph Sampler

- Une méthode d'échantillonnage qui extrait une petite partie des sous-graphes qui sont **les plus fréquents**.
- Elle applique l'échantillonnage sur un espace de sous graphes avec une **taille fix prédéfinie**.
- Elle est basée sur les **chaînes de markov monte carlo**.
- Utilise un **manageur de chaîne** qui stock les K-plus fréquents graphes.

Processus

Le processus se compose de 2 étapes :

$G_i \in \mathcal{G}, \forall i = \{1 \dots n\}$ et la taille P.

- La première étape consiste à choisir un des graphes uniformément.

- Et La deuxième étape consiste à choisir les sous-graphes de taille P, en utilisant l**es chaîne de markov monte carlo**, cette étape s'exécute en plusieurs itérations et utilise un **manageur de chaîne** qui gère les K premiers ou plus haut position dans la chaîne.

Approche

- Dans une session d'échantillonnage, le nombre de graphes en entrée qui ont concluent le graphe g comme résultat d'échantillonnage est appelé **support$_a$(g)**. et donc on à support(g) est la valeur maximale de support$_a$(g).

- Pour dire qu'un graphe est fréquent si la valeur de son **extended-support est plus grande que les graphes de la même taille**.

- On appelle un un sous-graphe g Fréquent si une copie identique est échantillonnée depuis plusieurs graphes donnés en entrée dans les différentes itérations de l'échantillonnage.

$$\textbf{support}_a(g) <= \textbf{support}(g)$$

Approche : Transition d'états

L'espace d'échantillonnage de la marche de MCMC est l'ensemble de tous les P-sous graphes dans une base de graphe Gi.

Pour chaque temps :

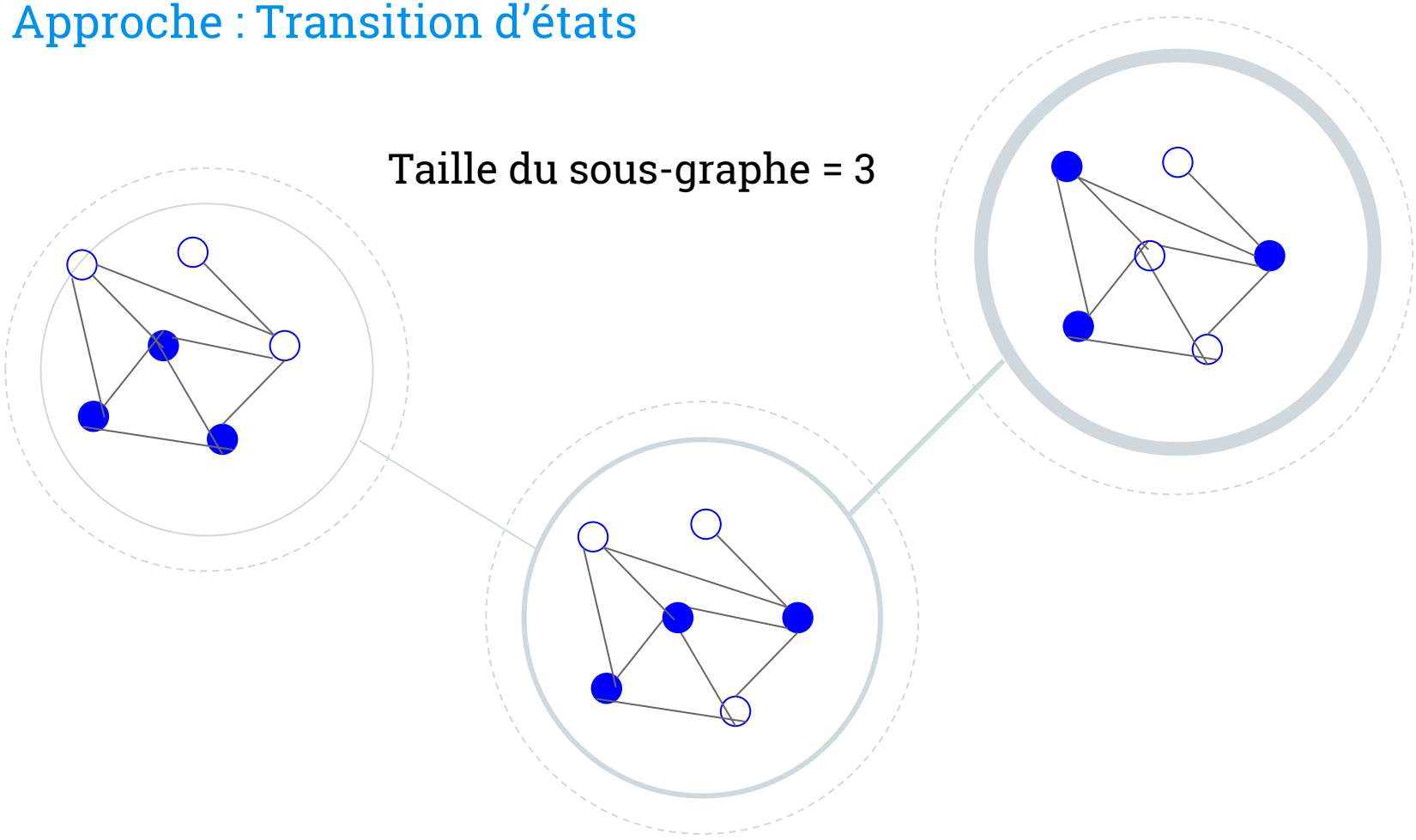
1. la marche aléatoire visite un des **p-sous graphes de Gi**.

2. Puis il localise tous **ses voisins** qui sont aussi des p-sous graphe.

3. choisit avec une **probabilité uniforme** le prochain voisin qui deviendra **le nouvel état de la chaîne en utilisant l'algorithme de MH.**

Approche : Transition d'états

Taille du sous-graphe = 3



Approche : Fonction de score

$$f : \Omega \rightarrow \mathbb{R}_+$$

Chaque graphe à son score

$$s_1(g) = \frac{1}{|E(g)|} \sum_{e \in E(g)} support(e)$$

La moyenne du support des arcs qui constitue le graphe g

$$s_2(g) = \left| \bigcap_{e \in E(g)} support(e) \right|$$

La cardinalité générée par l'intersection s=du support de l'ensemble de chaque arc de g

plus le support est grand, plus le score est grand.

L'algorithme d'échantillonnage

SAMPLEINDSUBGRAPH(G_i, p)

1 x = State saved at G_i
2 d_x = Neighbor-count of x
3 a_sup_x = score of graph x
4 **while** (a neighbor state y is not found)
5 y = a random neighbor of x
6 d_y = Possible neighbor of y
7 a_sup_y = score of graph y
8 $accp\_val = (d_x * a\_sup_y)/(d_y * a\_sup_x)$
9 $accp\_probablility = min(1, accp\_val)$
10 **if** $uniform(0,1) \leq accp\_probability$
11 **return** y

Entrée : le graphe et la taille p des sous-graphes

- Dans la ligne 1 on obtient un p-sous graphe et donc l'état de la chaîne de markov

- Dans la ligne 2 on compte le nombre de voisins de ce graphe c'est à dire les graphes adjacents

- Ligne 3 on calcule le score de la fonction s1 ou s2 du graphe

L'algorithme d'échantillonnage

SAMPLEINDSUBGRAPH(G_i, p)

1 x = State saved at G_i
2 d_x = Neighbor-count of x
3 a_sup_x = score of graph x
4 **while** (a neighbor state y is not found)
5 y = a random neighbor of x
6 d_y = Possible neighbor of y
7 a_sup_y = score of graph y
8 $accp\_val = (d_x * a\_sup_y)/(d_y * a\_sup_x)$
9 $accp\_probablility = min(1, accp\_val)$
10 **if** $uniform(0, 1) \leq accp\_probability$
11 **return** y

- Par la suite on choisis un voisin y uniformément.

- On compte le nombre de voisins de y.

- On calcule le score de fonction de y.

- On calcule la probabilité d'acceptation de transition de x à y

- L'algorithme décide si il transite de l'état x à l'état y.

Manageur de chaîne

- FS stocke les meilleurs graphes dans une chaîne de taille K
- La chaîne utilisée implémente la notion de priorité.

Graphe	Label	Expected-support ou support list	Score (s1 ou s2)	temps(la dernière itération)

L'algorithme FS^3

```
FS^3(G, p, mIter)
    G: Graph Database, p: Size of the subgraph
    mIter: Number of samples
 1  iter = 0, Q = ∅

 2  while iter ≤ mIter
 3      iter = iter + 1
 4      Select a graph G ∈ G uniformly
 5      h = SampleIndSubgraph(G, p)
 6      if Q.full = true and
                h.score() < Q.lowerHalfAvgScore()
 7          continue
 8      h.code = GenCanCode(h)
 9      if h ∈ Q
10          prevSupport = h.idset.size()
11          h.idset = h.idset ∪ G.id
12          if h.idset.size() > prevSupport
13              h.insertTime = iter
14      else
15          if Q.full = true
16              Q.evictLast()
17          h.idset = {G.id}
18          h.insertTime = iter
19          Q = Q ∪ {h}
20  return Q
```

Pour chaque itération on sélectionne le graphe et le sous-graphe échantillonné par l'algorithme précédent.

L'algorithme FS^3

```
FS^3(G, p, mIter)
    G: Graph Database, p: Size of the subgraph
    mIter: Number of samples
 1  iter = 0, Q = ∅

 2  while iter ≤ mIter
 3      iter = iter + 1
 4      Select a graph G ∈ G uniformly
 5      h = SampleIndSubgraph(G, p)
 6      if Q.full = true and
                h.score() < Q.lowerHalfAvgScore()
 7          continue
 8      h.code = GenCanCode(h)
 9      if h ∈ Q
10          prevSupport = h.idset.size()
11          h.idset = h.idset ∪ G.id
12          if h.idset.size() > prevSupport
13              h.insertTime = iter
14      else
15          if Q.full = true
16              Q.evictLast()
17          h.idset = {G.id}
18          h.insertTime = iter
19          Q = Q ∪ {h}
20  return Q
```

Si le score du graphe n'est pas assez grand alors on ne génère pas le code canonique qui est trop coûteux

L'algorithme FS^3

```
FS^3(G, p, mIter)
     G: Graph Database, p: Size of the subgraph
     mIter: Number of samples
 1   iter = 0, Q = ∅

 2   while iter ≤ mIter
 3       iter = iter + 1
 4       Select a graph G ∈ G uniformly
 5       h = SAMPLEINDSUBGRAPH(G, p)
 6       if Q.full = true and
                 h.score() < Q.lowerHalfAvgScore()

 7           continue
 8       h.code = GENCANCODE(h)
 9       if h ∈ Q
10           prevSupport = h.idset.size()
11           h.idset = h.idset ∪ G.id
12           if h.idset.size() > prevSupport
13               h.insertTime = iter
14       else
15           if Q.full = true
16               Q.evictLast()
17           h.idset = {G.id}
18           h.insertTime = iter
19           Q = Q ∪ {h}
20   return Q
```

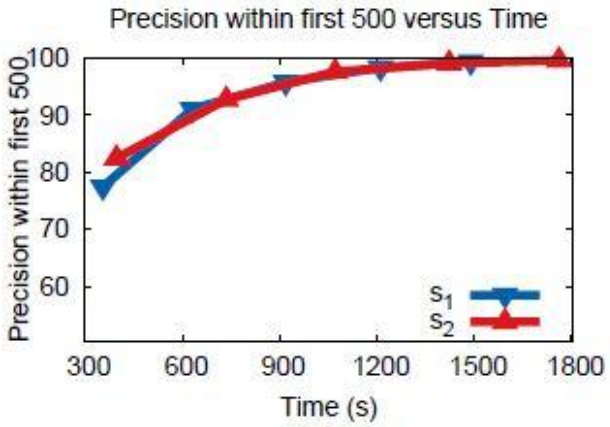
Sinon on génère le code canonique, si le graphe est déjà dans la chaîne alors on met à jour ses informations

L'algorithme FS^3

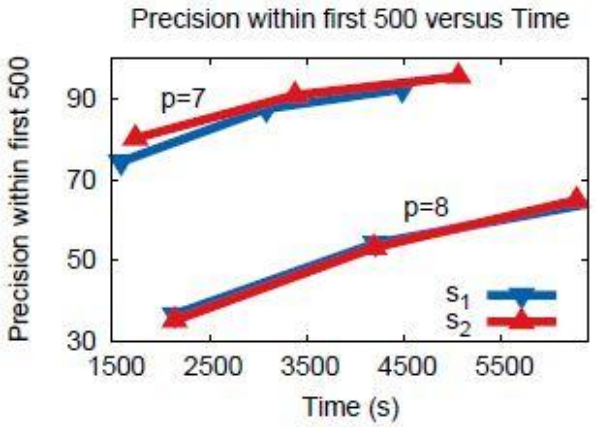
```
FS^3(G, p, mIter)
     G: Graph Database, p: Size of the subgraph
     mIter: Number of samples
 1   iter = 0, Q = ∅

 2   while iter ≤ mIter
 3       iter = iter + 1
 4       Select a graph G ∈ G uniformly
 5       h = SAMPLEINDSUBGRAPH(G, p)
 6       if Q.full = true and
                 h.score() < Q.lowerHalfAvgScore()

 7           continue
 8       h.code = GENCANCODE(h)
 9       if h ∈ Q
10           prevSupport = h.idset.size()
11           h.idset = h.idset ∪ G.id
12           if h.idset.size() > prevSupport
13               h.insertTime = iter
14       else
15           if Q.full = true
16               Q.evictLast()
17           h.idset = {G.id}
18           h.insertTime = iter
19           Q = Q ∪ {h}
20   return Q
```

Si le graphe n'est pas dans la chaîne alors on retire le dernier et on insère ce nouveau graphe et on met à jour ses informations

Expérimentation



(a) $p = 6$



(b) $p = 7, 8$

PS Dataset : **90 graphes | 67 Noeuds | 268 Arcs**

Résultat :
Pour 20 min d'exécution avec une taille de 6 nous avons : 99% de résultat.
Pour 1.4 heure avec une taille de 7 nous avons : 95 %.
Pour 1.8 heure avec une taille de 8 nous avons : 65%.

Conclusion

- FS^3 est une méthode d'échantillonnage qui permet d'extraire les graphes fréquents d'une taille donnée.

- Pour les données de grandes tailles, les autres algorithmes sont grandement limités alors que FS peut retourner un ensemble de graphes fréquents avec un temps acceptable.

Merci pour votre écoute!

Any questions?