Data Mining
Fouille de données

Motifs séquentiels fréquents

Arfaoui Ahlem

Les règles d'association vs Les motifs séquentiels

❖ **Règles d'association:**

- **Le temps n'a aucune importance**.
- On regarde seulement quels produits apparaissent ensemble dans le panier.

➢ **Exemple :**

- Dans un supermarché, on observe beaucoup de tickets de caisse.
- On remarque que très souvent, **les clients qui achètent du pain achètent aussi du fromage.**

➢ **Règle d'association : Pain → Fromage**

- si un client achète du pain, **alors il a de grandes chances** d'acheter aussi du fromage.
- Ici, **l'ordre d'achat n'est pas important** (le client peut acheter le pain avant ou après le fromage, ça ne change rien).

Les règles d'association vs Les motifs séquentiels

❖ **Motifs séquentiels:**

- **Le temps et l'ordre des achats sont importants**.
- On ne cherche plus seulement des co-occurrences, mais des **séquences dans le temps**.

➢ **Exemple :**

- On **observe le comportement des clients** au fil du temps.
- On remarque qu'un certain groupe de clients fait souvent les achats suivants :
  - ✓ **Jour 1 :** Pain
  - ✓ **Jour 2 :** Fromage
  - ✓ **Jour 3 :** chocolat

➢ **Motif séquentiel : Pain → Fromage → chocolat**

- beaucoup de clients achètent **d'abord** du pain, **puis** du fromage **plus tard**, **puis** du chocolat **ensuite** ➔ **l'ordre est crucial**.

Les règles d'association vs Les motifs séquentiels

❖ **Différence**

Aspect	Règles d'association	Motifs séquentiels
Temps / ordre	**Ignoré**	**Pris en compte**
Exemple	Pain → Fromage	Pain → Fromage → Chocolat
Objectif	Découvrir des produits souvent achetés ensemble	Découvrir des **séquences d'achats** qui se répètent dans le temps

- **Règles d'association = qui va avec quoi.**
- **Motifs séquentiels = qui vient après quoi.**

Qu'est-ce qu'une séquence ?

- Une **séquence** est une **suite d'éléments ordonnés** dans le temps.
- Chaque **élément** peut contenir **un ou plusieurs produits**.
- Les **éléments eux-mêmes sont ordonnés** (ils représentent des instants différents dans le temps).
- **À l'intérieur d'un élément**, les produits **ne sont pas ordonnés** (c'est comme un petit panier d'achats au même moment).

➢ **Exemple : < (ef) (ab) (df) c b >**

- Le **< ... >** = la séquence complète
- **(ef)** = un élément contenant les produits e et f achetés en même temps
- **(ab)** = un autre élément contenant a et b
- **(df)** = un autre élément contenant d et f
- **c et b** = éléments séparés contenant un seul produit

Qu’est-ce qu’une séquence ?

➔ Donc, la personne a acheté :

- e et f ensemble
- puis a et b ensemble
- puis d et f ensemble
- puis c seul
- puis b seul

Qu'est-ce qu'une sous-séquence ?

- Une **sous-séquence** est obtenue en **enlevant certains produits ou certains éléments**, tout en **gardant l'ordre des éléments restants**.

➢ **Exemple :**

- **Séquence : < a (a bc) ac (d c) f >**
- **Sous-séquence : < (bc) (dc) >**

➔ on garde **bc** comme un élément, puis **dc** plus tard.

➔ On a sauté d'autres produits, **mais on a gardé l'ordre global**.

➢ **Remarque:**

- **On peut supprimer des éléments entiers**.
- **On peut supprimer des produits à l'intérieur d'un élément.**
- Mais on ne peut pas **changer l'ordre** des éléments.

Exemple

SID1	**< (a c) b d >**
SID2	**< a (b c) (a b) >**
SID3	**< (a b) c >**
SID4	**< b (a c) >**

❖ **Motif fréquent : <(ab)>**

➢ **Interprétation :**

- un seul élément contenant **a** et **b** ensemble (ils doivent être dans le même élément).
- Vérifions dans chaque **SID** :
  - ✓ **SID1** < (a c) b d > : il y a (a c) mais pas d'élément (a b) → **non**.
  - ✓ **SID2** < a (b c) (a b) > : il y a **(a b)** → **oui**.
  - ✓ **SID3** < (a b) c > : il y a **(a b)** → **oui**.
  - ✓ **SID4** < b (a c) > : (a c) contient a mais pas b dans le même élément → **non**.
- Support = occurrences dans SID2 et SID3 = **2**.
- Comme support = 2 ≥ sup_min (2) ➔ **<(ab)> est fréquent.**

Exemple

SID1	**< (a c) b d >**
SID2	**< a (b c) (a b) >**
SID3	**< (a b) c >**
SID4	**< b (a c) >**

❖ **Motif non fréquent : <(ac) b>**

➢ **Interprétation :**

- un élément contenant **a** et **c** ensemble, puis **plus tard** (dans un élément ultérieur) un élément contenant **b**.
- Vérifions :
  - ✓ **SID1** < (a c) b d > : (a c) existe puis **après** il y a b → **oui** (correspond).
  - ✓ **SID2** < a (b c) (a b) > : il n'y a pas d'élément (a c) (il y a a seul ou (b c) ou (a b)) → **non**.
  - ✓ **SID3** < (a b) c > : pas d'élément (a c) → **non**.
  - ✓ **SID4** < b (a c) > : il y a (a c) mais **après** un b (l'ordre est b puis (a c)), or le motif exige (ac) **puis** b après → **non** (ordre non respecté).

Support = seulement SID1 = **1.➔** support = 1 < sup_min (2)**➔ <(ac) b> n'est pas fréquent.**

Base de séquences

- Comment transformer une **base de données transactionnelle** (type panier d'achats) en une **base de données séquentielle** (utile pour l'extraction de motifs séquentiels).

❖ **Base de données transactionnelle (Transaction Database):**

TID	Produit
1	a
2	b
3	c
4	a
5	b
6	b
7	c

Base de séquences

❖ **Base de données transactionnelle (Transaction Database):**

- On sait **quel client a acheté quel produit.**
- Mais on ne sait pas encore **l'ordre dans le temps.**

TID	Client	Produit
1	C1	a
2	C1	b
3	C1	c
4	C2	a
5	C2	b
6	C3	b
7	C3	c

Base de séquences

❖ **Base de données transactionnelle (Transaction Database):**

✓ **Ajouter une notion de temps:**

- Pour aller vers une base séquentielle, il faut savoir **quand** le client a acheté.
- On rajoute une **colonne "temps" ou "transaction ID" ordonné** :

TID	Client	Produit	Temps
1	C1	a	t1
2	C1	b	t2
3	C1	c	t3
4	C2	a	t1
5	C2	b	t2
6	C3	b	t1
7	C3	c	t2

Base de séquences

➢ **Regrouper par client → Base de séquences**

- On construit alors pour chaque **client** une **séquence ordonnée** :
  - ✓ **Client C1 :** < (a) (b) (c) > (a acheté d'abord a, puis b, puis c)
  - ✓ **Client C2 :** < (a) (b) >
  - ✓ **Client C3 :** < (b) (c) >

Base de séquences

❖ Cas où plusieurs produits sont achetés au même temps

Si un client achète **plusieurs produits à la même transaction**, on les regroupe dans le **même élément** :

➢ **Exemple :**

TID	Client	Produits	Temps
1	C1	a, c	**t1**
2	C1	b	**t2**
3	C1	d, f	**t3**

Séquence C1 : < (a c) (b) (d f) >

➔ **Transaction DB** → ligne par produit acheté

➔ Ajouter une **notion de temps** (ou numéro de transaction)

➔ Regrouper **par client** et **par temps** → obtenir une séquence d'achats

Recherche des motifs séquentiels

➢ **Exemple :**

- On observe les **achats des clients** dans un magasin.
- Chaque client fait plusieurs achats, parfois plusieurs articles en même temps (dans une même transaction).

➢ **Base de séquences :**

id client	Séquence
1	⟨ (a) (b c) (d) ⟩
2	⟨ (a b) (c) (d) ⟩
3	⟨ (b) (a c) (d) ⟩
4	⟨ (a) (c) (e) ⟩

Recherche des motifs séquentiels

id client	Séquence
1	⟨ (a) (b c) (d) ⟩
2	⟨ (a b) (c) (d) ⟩
3	⟨ (b) (a c) (d) ⟩
4	⟨ (a) (c) (e) ⟩

➢ **Exemple :**

- **Support minimal (min_sup) = 2**
- On cherche les **sous-séquences** qui apparaissent chez **au moins 2 clients**.

❖ **Étape 1 – motifs fréquents de taille 1**

- ⟨a⟩ apparaît chez clients 1,2,3,4 → support = 4 ➔**OK**
- ⟨b⟩ apparaît chez clients 1,2,3 → support = 3 ➔**OK**
- ⟨c⟩ apparaît chez clients 1,2,3,4 → support = 4 ➔**OK**
- ⟨d⟩ apparaît chez clients 1,2,3 → support = 3 ➔**OK**
- ⟨e⟩ apparaît seulement chez client 4 → support = 1 ➔**NON** (pas fréquent).

➔**Motifs fréquents : ⟨a⟩, ⟨b⟩, ⟨c⟩, ⟨d⟩**

Recherche des motifs séquentiels

id client	Séquence
1	⟨ (a) (b c) (d) ⟩
2	⟨ (a b) (c) (d) ⟩
3	⟨ (b) (a c) (d) ⟩
4	⟨ (a) (c) (e) ⟩

➢ **Exemple :**

❖ **Étape 2 – motifs fréquents de taille 2**

⟨a → c⟩ (a suivi de c) apparaît chez clients 1,2,3,4 → support = 4 ➔**OK**

⟨a → d⟩ apparaît chez clients 1,2,3 → support = 3 ➔**OK**

⟨b → d⟩ apparaît chez clients 1,2,3 → support = 3 ➔**OK**

⟨c → d⟩ apparaît chez clients 1,2,3 → support = 3 ➔**OK**

⟨a → e⟩ apparaît seulement chez client 4 → support = 1 ➔**NON**

➔**Motifs fréquents : ⟨a → c⟩, ⟨a → d⟩, ⟨b → d⟩, ⟨c → d⟩**

Recherche des motifs séquentiels

➢ **Exemple :**

❖ **Résultat :**

- Les **motifs séquentiels fréquents** sont par exemple :
  - ✓ **Taille 1 :** ⟨a⟩, ⟨b⟩, ⟨c⟩, ⟨d⟩
  - ✓ **Taille 2 :** ⟨a → c⟩, ⟨a → d⟩, ⟨b → d⟩, ⟨c → d⟩

➢ **Interprétation :**

- Cela veut dire que **beaucoup de clients qui achètent "a" achètent ensuite "c" ou "d"**.
- C'est une **tendance d'achat** qu'on peut utiliser pour :
  - ✓ faire du marketing (proposer c après a),
  - ✓ optimiser les rayons du magasin,
  - ✓ prédire le prochain achat d'un client.

Recherche des motifs séquentiels

❖ **Support avec occurrence unique:**

Quand on calcule le **support** d'un motif séquentiel, il faut savoir que :

➔ Une séquence (c'est-à-dire un client, un utilisateur, etc.) **ne compte qu'une seule fois** dans le support, même si le motif apparaît plusieurs fois dans cette séquence.

➢ **Exemple**

id	Séquence
1	⟨ (10) (20 30) (40) (20) ⟩
2	⟨ (20) (30) (40) ⟩

Motif = ⟨20⟩

✓ **Dans la séquence 1 :** l'item **20** apparaît **deux fois**, mais on compte **support = 1** (une seule occurrence par séquence).

✓ **Dans la séquence 2 :** l'item **20** apparaît une fois → support = 1.

➔**Donc Support(⟨20⟩) = 2 (deux séquences contiennent 20).**

Recherche des motifs séquentiels

❖ Inclusion (définition formelle)

- On dit que **S1 est une sous-séquence de S2** si on peut trouver les éléments de S1 dans S2 **dans le même ordre**, pas forcément consécutifs.

➢ Définition mathématique :

Soient

$$S1 = \langle a_1, a_2, \ldots, a_n \rangle$$
$$S2 = \langle b_1, b_2, \ldots, b_m \rangle$$

➔ $S1 \subseteq S2$ **si** il existe des indices $i_1 < i_2 < \cdots < i_n$ tels que $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n}$.

Recherche des motifs séquentiels

❖ **Inclusion (définition formelle)**

➢ **Exemple 1:**

$$S1 = \langle (a)(c) \rangle$$
$$S2 = \langle (ab)(cd)(e) \rangle$$

➔ Alors :

$$a \subseteq (ab)$$
$$c \subseteq (cd)$$

L'ordre est respecté ➔ $S1 \subseteq S2$.

➢ **Exemple 2:**

$$S1 = \langle (a)(e) \rangle$$
$$S2 = \langle (ab)(cd)(f) \rangle$$

➔ **Il n'y a pas de e dans S2 → donc pas d'inclusion.**

Recherche des motifs séquentiels

❖ **Inclusion (définition formelle)**

✓ **Le support** = nombre de séquences différentes qui contiennent le motif (une fois max par séquence).

✓ **L'inclusion** = vérifier si une séquence courte (motif candidat) existe dans une séquence plus longue (base de données).

➢ **Exercice:** Soit la séquence :

$$\boldsymbol{\beta} = \langle (\boldsymbol{ab})(\boldsymbol{c})(\boldsymbol{ad})(\boldsymbol{bcd})(\boldsymbol{f}) \rangle$$

Tester si les séquences suivantes $\boldsymbol{\alpha_i}$ sont des sous-séquences de $\boldsymbol{\beta}$:

- $\boldsymbol{\alpha_1} = \langle (a)(c)(d) \rangle$
- $\boldsymbol{\alpha_2} = \langle (bc)(f) \rangle$
- $\boldsymbol{\alpha_3} = \langle (ad)(b)(f) \rangle$
- $\boldsymbol{\alpha_4} = \langle (c)(a)(b) \rangle$
- $\boldsymbol{\alpha_5} = \langle (a)(f) \rangle$

Approches Apriori

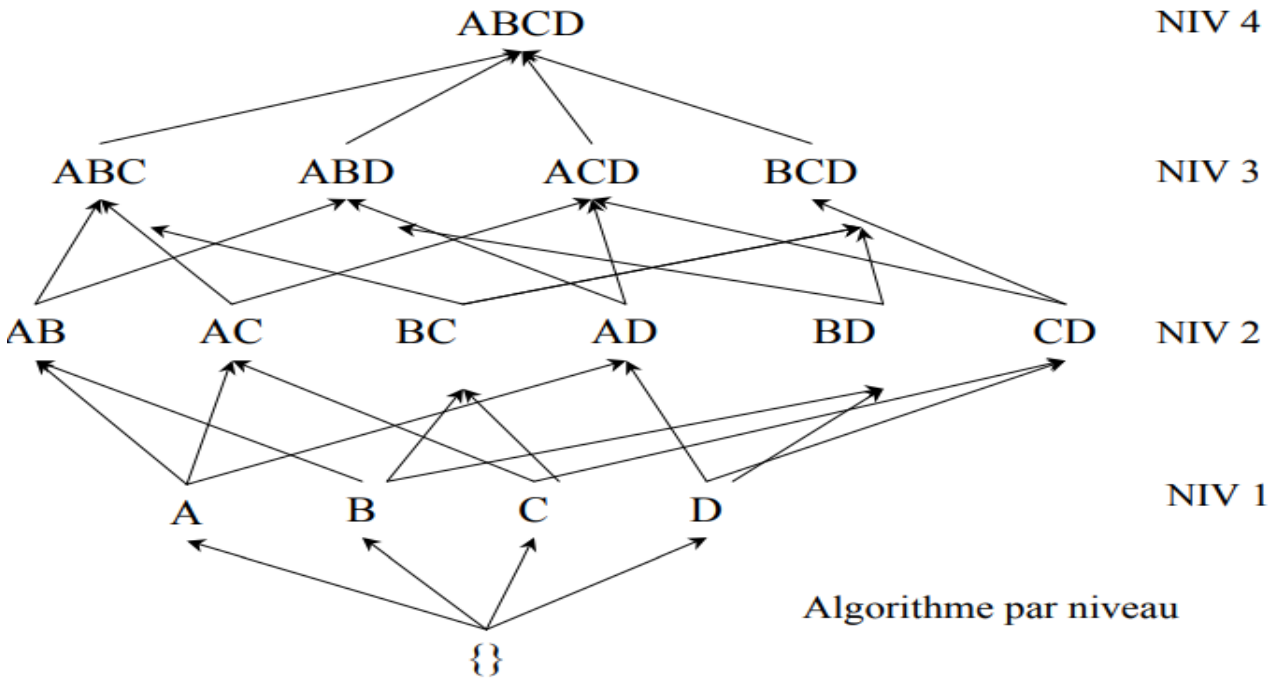
❖ **Comment on construit l'espace par niveaux (liaison avec ta structure)**

Niv 1 : A, B, C, D ← → L1 candidats de taille 1

Niv 2: AB, AC, BC, AD, BD, CD ← → C2 (toutes les paires possibles), après prune on garde L2

Niv 3: ABC, ABD, ACD, BCD ← → C3 générés par jointure de L2

Niv 4: ABCD ← → C4 etc

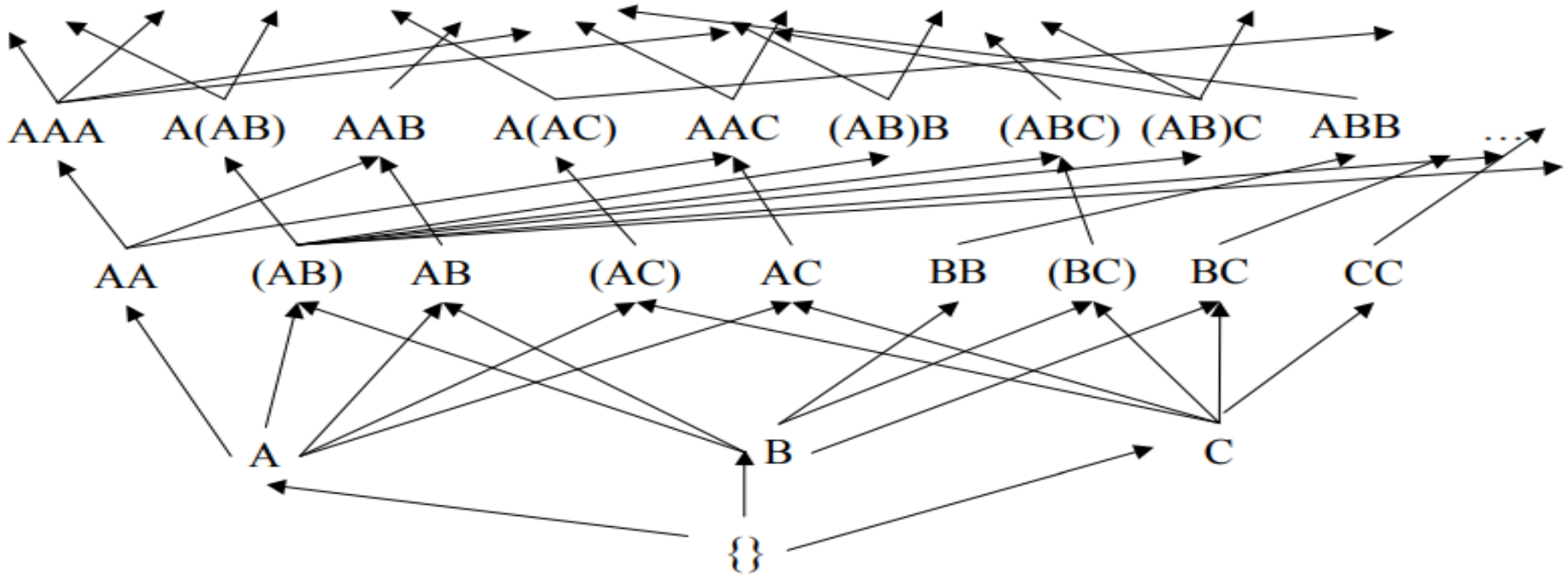


Algorithme par niveau

Approches Apriori

❖ Espace de recherche pour les motifs séquentiels

- L'**explosion combinatoire** est encore plus grande que pour Apriori simple, parce qu'on a deux dimensions :
- La **taille de la séquence** (nombre d'étapes dans l'ordre).
- La **taille des itemsets** dans chaque étape.



Approches Apriori

❖ **Propriété d'antimonotonie**

- **Si une séquence n'est pas fréquente, alors aucune de ses super-séquences ne peut être fréquente.**
- **Super-séquence** = une séquence qui contient la séquence initiale avec plus d'items ou plus d'étapes.

➢ **Exemple :**

Si ⟨(30,40,10)⟩ est **rare** (support < min), alors ⟨(30,40,10,50)⟩ ou ⟨(20)(30,40,10)⟩ le seront **forcément aussi**.

Intérêt : Réduire énormément l'espace de recherche.

- On ne scanne plus toute la base inutilement.
- L'algorithme se concentre seulement sur les séquences qui **peuvent potentiellement être fréquentes**.

➔ C'est exactement le principe de l'algorithme **Apriori** adapté aux séquences.

Approches Apriori: GSP(Generalized Sequential Patten)

❖ **Idée générale:**

- GSP cherche **toutes** les séquences fréquentes dans une base de séquences, en procédant **par niveaux** (longueur 1, 2, 3, …) comme Apriori.

- On utilise la **propriété d'antimonotonie** : si une séquence n'est pas fréquente, aucune de ses extensions (super-séquences) ne le sera.

Approches Apriori: GSP(Generalized Sequential Patten)

➢ **Exemple: Support_min= 2**

- Soit la base de transaction suivante:

Seq. ID	Séquence
10	⟨(bd)cb(ac)⟩
20	⟨(bf)(ce)b(fg)⟩
30	⟨(ah)(bf)abf⟩
40	⟨(be)(ce)d⟩
50	⟨a(bd)bcb(ade)⟩

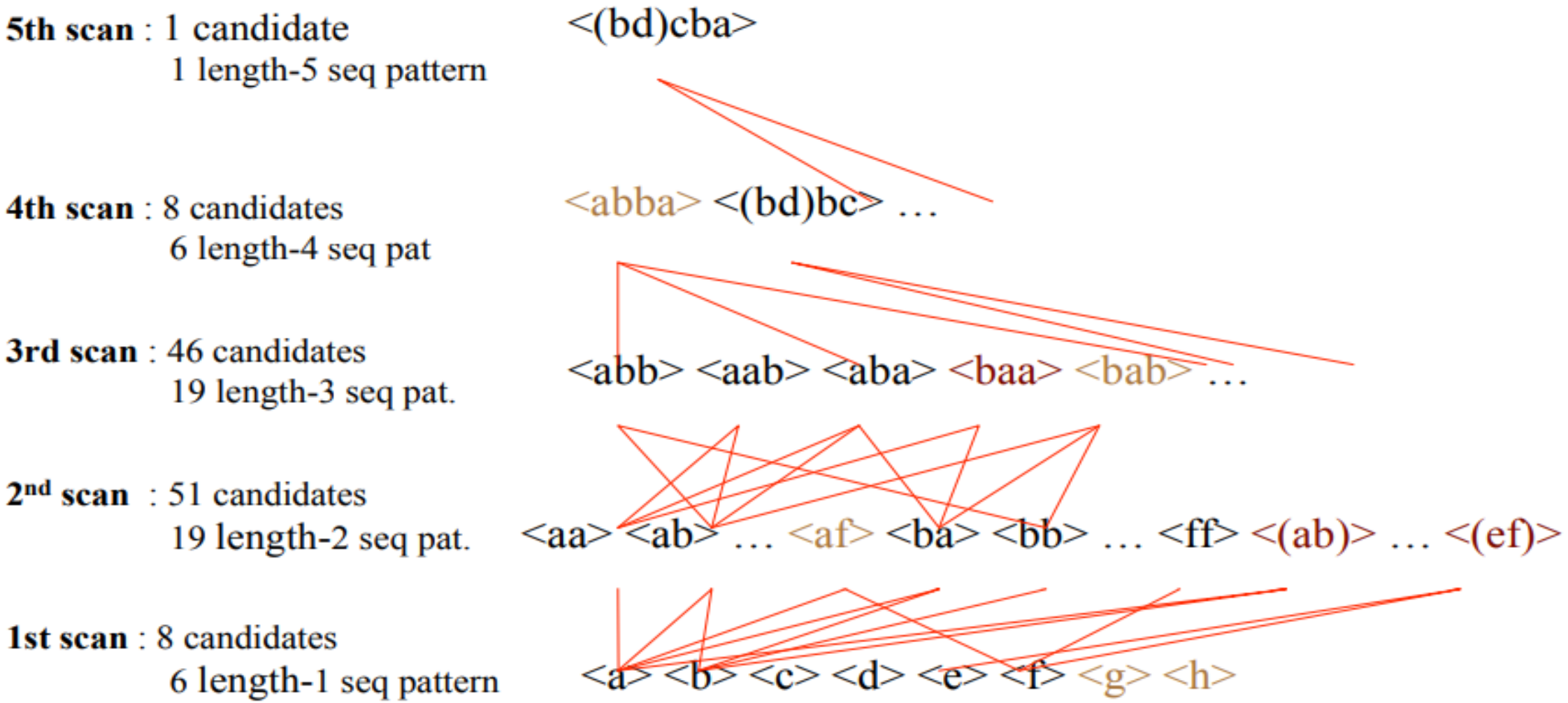
Séquence	Support
⟨a⟩	3
⟨b⟩	5
⟨c⟩	4
⟨d⟩	3
⟨e⟩	3
⟨f⟩	2
~~⟨g⟩~~	1
~~⟨h⟩~~	1

Candidats initiaux : toutes les séquences contenantes un item (**cardinal = 1**).

➔ <a >, <b >, <c >, <d >, <e > ,<f >, <g>, <h>

Approches Apriori: GSP(Generalized Sequential Patten)

- **Exemple:** On élimine les candidats dont le support est inférieur au **supmin**.



Approches Apriori: GSP(Generalized Sequential Patten)

- **Exemple:** Génération de candidats de **taille 2 .**

S-Extension

	⟨a⟩	⟨b⟩	⟨c⟩	⟨d⟩	⟨e⟩	⟨f⟩
⟨a⟩	⟨aa⟩	⟨ab⟩	⟨ac⟩	⟨ad⟩	⟨ae⟩	⟨af⟩
⟨b⟩	⟨ba⟩	⟨bb⟩	⟨bc⟩	⟨bd⟩	⟨be⟩	⟨bf⟩
⟨c⟩	⟨ca⟩	⟨cb⟩	⟨cc⟩	⟨cd⟩	⟨ce⟩	⟨cf⟩
⟨d⟩	⟨da⟩	⟨db⟩	⟨dc⟩	⟨dd⟩	⟨de⟩	⟨df⟩
⟨e⟩	⟨ea⟩	⟨eb⟩	⟨ec⟩	⟨ed⟩	⟨ee⟩	⟨ef⟩
⟨f⟩	⟨fa⟩	⟨fb⟩	⟨fc⟩	⟨fd⟩	⟨fe⟩	⟨ff⟩

	⟨a⟩	⟨b⟩	⟨c⟩	⟨d⟩	⟨e⟩	⟨f⟩
⟨a⟩		⟨(ab)⟩	⟨(ac)⟩	⟨(ad)⟩	⟨(ae)⟩	⟨(af)⟩
⟨b⟩			⟨(bc)⟩	⟨(bd)⟩	⟨(be)⟩	⟨(bf)⟩
⟨c⟩				⟨(cd)⟩	⟨(ce)⟩	⟨(cf)⟩
⟨d⟩					⟨(de)⟩	⟨(df)⟩
⟨e⟩						⟨(ef)⟩

I-Extension

Approches Apriori: GSP(Generalized Sequential Patten)

- **Exemple**

	⟨a⟩	⟨b⟩	⟨c⟩	⟨d⟩	⟨e⟩	⟨f⟩
⟨a⟩	**⟨aa⟩:2**	**⟨ab⟩:2**	⟨ac⟩:1	⟨ad⟩:1	⟨ae⟩:1	⟨af⟩:1
⟨b⟩	**⟨ba⟩:3**	**⟨bb⟩:4**	**⟨bc⟩:4**	**⟨bd⟩:2**	**⟨be⟩:3**	**⟨bf⟩:2**
⟨c⟩	**⟨ca⟩:2**	**⟨cb⟩:3**	⟨cc⟩:1	**⟨cd⟩:2**	⟨ce⟩:1	⟨cf⟩:1
⟨d⟩	**⟨da⟩:2**	**⟨db⟩:2**	**⟨de⟩:2**	⟨dd⟩:0	⟨de⟩:1	⟨df⟩:0
⟨e⟩	⟨ea⟩:0	⟨eb⟩:1	⟨ec⟩:0	⟨ed⟩:1	⟨ee⟩:1	⟨ef⟩:1
⟨f⟩	⟨fa⟩:1	**⟨fb⟩:2**	⟨fc⟩:1	⟨fd⟩:0	⟨fe⟩:1	**⟨ff⟩:2**

	⟨a⟩	⟨b⟩	⟨c⟩	⟨d⟩	⟨e⟩	⟨f⟩
⟨a⟩		⟨(ab)⟩:0	⟨(ac)⟩:1	⟨(ad)⟩:1	⟨(ae)⟩:1	⟨(af)⟩:0
⟨b⟩			⟨(bc)⟩:0	**⟨(bd)⟩:2**	⟨(be)⟩:1	**⟨(bf)⟩:2**
⟨c⟩				⟨(cd)⟩:0	⟨(ce)⟩:2	⟨(cf)⟩:0
⟨d⟩					⟨(de)⟩:1	⟨(df)⟩:0
⟨e⟩						⟨(ef)⟩:0
⟨f⟩						

Approches Apriori: GSP(Generalized Sequential Patten)

➢ **Exemple:**

- Après le calcul du support des candidats de taille 2, il nous reste 19 candidats :
  - 16 des motifs de la forme **<xy>**
  - 3 des motifs de la forme **<(xy)>**

- Pour générer les motifs de types <xyz> pour <aa>:
  - On concatène un autre **itemsetfréquent** qui commence par a pour formé <aaa> ou <aab>.

Approches Apriori: GSP(Generalized Sequential Patten)

- **Exemple:**

<aaa>:0, <aab>:0
<aba>:2, <abb>:2, <abc>:1,
<abd>:1, <abe>:1, <abf>:1
<baa>, <bab>
<bba>, <bbb>, <bbe>, <bbd>,
<bbe>, <bbf>
<bca>, <bcb>, <bcd>
<bda>, <bdb>, <bdc>
<bfb>, <bff>
<caa>, <cab>
<cba>, <cbb>, <cbc>, <cbd>,
<cbe>, <cbf>
<cda>, <cdb>, <cdc>
<daa>, <dab>
<dba>, <dbb>, <dbc>, <dbd>,
<dbe>, <dbf>
<cda>, <dcb>, <dcd>
<fba>, <fbb>, <fbc>, <fbd>,
<fbe>, <fbf>
<ffb>, <fff>

Approches Apriori: GSP(Generalized Sequential Patten)

- **Exemple:**

Pour générer les motifs de type <(xy)z> pour <(bd)>:

- On cherche les motifs qui **se termine avec b ou d** pour former des motifs comme : <a(bd)>, <b(bd)>, <c(bd)>, <d(bd)>, <f(bd)>
- On chercheles motifs qui **commence par b ou d** pour former des motifs comme: <(bd)a>, <(bd)b>, <(bd)c>, <(bd)d>, <(bd)e>, <(bd)f>

Algorithme GSP(Generalized Sequential Patten)

❖ Étape 0 — Préparation

- Choisir min_sup.
- Transformer la BD transactionnelle en **base de séquences** (par client / par période).
- Éventuellement supprimer les items qui n'apparaissent jamais (prétraitement).

❖ Étape 1 — Générer C1 et calculer F1

- **C1** = tous les candidats de longueur 1 : <(x)> pour chaque item x présent dans la base.
- Scanner la base une fois, compter pour chaque <(x)> le nombre de SIDs qui contiennent x.
- **F1** = { <(x)> | support ≥ min_sup } (les 1-séquences fréquentes).

Remarque : on compte **une seule fois par SID** même si x apparaît plusieurs fois dans la même séquence.

Algorithme GSP(Generalized Sequential Patten)

❖ Étape 2 — Boucle principale (k ← 1)

Répéter tant que Fk n'est pas vide :

➢ Générer candidats C(k+1) à partir de Fk

- **Join** : combiner des paires de séquences de Fk qui partagent les k−1 premiers éléments (ou qui peuvent être combinées par I/S-extension) pour créer des candidats de taille k+1.
- **Deux types d'extensions** à considérer pour chaque séquence fréquente de taille k :
  - **I-extension** : ajouter un item à l'intérieur du dernier élément.
  - **S-extension** : ajouter un nouvel élément contenant un item à la fin.

Exemple simple (L1={a,b}) → C2 contiendra au moins <(ab)> (I-ext) et <(a)(b)>, <(b)(a)> (S-ext).

Algorithme GSP(Generalized Sequential Patten)

❖ **Étape 2 — Boucle principale (k ← 1)**

➢ **Pruning (élimination)**

- Pour chaque candidat c ∈ C(k+1), vérifier que **tous** ses sous-séquences de taille k sont dans Fk.
- Si un des sous-séquences k-taille de c n'est pas fréquent → on supprime c. (principe d'antimonotonie)

➢ **Scanner la base**

- Pour chaque candidat restant c, scanner la base de séquences et tester **inclusion** : est-ce que c est une sous-séquence de la séquence S (règle d'inclusion que tu connais) ?
- Pour chaque SID, si c est présent on incrémente le compteur **une seule fois** (même si c apparaît plusieurs fois dans la même SID).

Algorithme GSP(Generalized Sequential Patten)

❖ Étape 2 — Boucle principale (k ← 1)

Construire F(k+1)

F(k+1) = { c ∈ C(k+1) | support(c) ≥ min_sup }.

k ← k+1 et recommencer.

❖ Étape 3 — Fin

- Quand Fk devient vide, on arrête.
- Résultat = ⋃k Fk (toutes les séquences fréquentes trouvées).

Algorithme GSP: Exercice

Une boutique en ligne enregistre les **séquences d'achats** de ses clients dans le temps.

- **Seuil de support minimal (min_sup) = 3**

SID (Client)	Séquence d'achats
1	⟨ {A}, {B, C}, {D} ⟩
2	⟨ {A, C}, {B}, {E} ⟩
3	⟨ {A, B}, {C}, {E} ⟩
4	⟨ {B}, {A, C}, {D} ⟩
5	⟨ {A}, {B}, {C, E} ⟩

Algorithme GSP(Generalized Sequential Patten)

❖ **Inconvénients de GSP:**

- **Grand nombre de scans de la base**
  - ➢ À chaque itération (taille 1, puis taille 2, puis taille 3…), l'algorithme doit **rescanner toute la base** pour calculer les supports.
- **Explosion combinatoire des candidats**
  - ➢ Le nombre de candidats générés croît **exponentiellement** avec la taille des séquences. Beaucoup de motifs candidats sont générés… mais très peu deviennent fréquents. ➔ Cela gaspille du temps et de la mémoire.
- **Mauvaise gestion des motifs longs**
  - ➢ GSP est efficace si les motifs fréquents sont **courts**. Si les motifs sont longs (plusieurs dizaines d'items dans l'ordre), l'algorithme génère trop de candidats intermédiaires.➔ Très lent et peu adapté aux séquences longues.
- **Pas optimal pour les très grandes bases**
  - ➢ GSP fonctionne sur des bases **de taille moyenne**.

Pattern-Growth-basedApproaches

❖ FreeSpan (Free-Pattern projected Sequential pattern mining)

- Introduit avant **PrefixSpan**.
- L'objectif : au lieu d'utiliser l'approche **Apriori (comme GSP**), on utilise une **projection de la base de séquences** pour éviter de scanner toute la base à chaque étape.

- **Comment ça marche ?**
  - ➢ On trouve d'abord les motifs fréquents de taille 1.
  - ➢ Pour chaque motif, on **projette la base de séquences** qui contient ce motif.
  - ➢ On explore récursivement dans cette base projetée.

Pattern-Growth-basedApproaches

❖ **FreeSpan (Free-Pattern projected Sequential pattern mining)**

➢ **Exemple:**

- Base : **min_sup = 2.**

SID10	**⟨ a (abc) (ac) d (cf) ⟩**
SID20	**⟨ (ad) c (bc) (ae) ⟩**
SID30	**⟨ (ef) (ab) (df) c b ⟩**
SID40	**⟨ e g (af) c b c ⟩**

Etape1:

- On commence par compter les items **1-séquence** (une seule fois par SID) :
- **Motifs fréquents de taille 1 :**
  - a:4, b:4, c:4, d:3, e:3, f:3
  - **g** éliminé (support = 1).

Etape 2 : Construire la f-list

C'est la **liste triée des items fréquents** par support décroissant :

- **f_list = [a, b, c, d, e, f]**

Pattern-Growth-basedApproaches

❖ **FreeSpan (Free-Pattern projected Sequential pattern mining)**

➢ **Exemple:**

- Base : **min_sup = 2.**

SID10	**⟨ a (abc) (ac) d (cf) ⟩**
SID20	**⟨ (ad) c (bc) (ae) ⟩**
SID30	**⟨ (ef) (ab) (df) c b ⟩**
SID40	**⟨ e g (af) c b c ⟩**

- FreeSpan va **diviser l'espace de recherche** en **6 projets** (un pour chaque item).

Étape 3 : Créer les bases projetées (Projected Databases)

L'objectif:

→ Pour chaque item fréquent, on regarde **les suffixes** de chaque séquence **après la première occurrence** de cet item.

→ Puis on cherche **de nouveaux motifs** dans ces suffixes.

Pattern-Growth-basedApproaches

- **FreeSpan (Free-Pattern projected Sequential pattern mining)**
  - **Exemple: Base projetée pour a**

On prend toutes la séquence contenante a, et on garde la **partie après a** :

SID	Séquence originale	Suffixe après a
10	⟨ a (abc) (ac) d (cf) ⟩	⟨ (abc) (ac) d (cf) ⟩
20	⟨ (ad) c (bc) (ae) ⟩	⟨ c (bc) (ae) ⟩
30	⟨ (ef) (ab) (df) c b ⟩	⟨ (df) c b ⟩
40	⟨ e g (af) c b c ⟩	⟨ c b c ⟩

Pattern-Growth-basedApproaches

❖ **FreeSpan (Free-Pattern projected Sequential pattern mining)**

➢ **Exemple:**

- **S-Extention**

On regarde la présence d'un item x **dans les suffixes** (au moins une fois par SID) :

- **b :** présent dans suffixes de SID10, SID20, SID30, SID40 → **support = 4**
- **c :** présent dans SID10, SID20, SID30, SID40 → **support = 4**
- **a :** présent dans SID10 et SID20 (a réapparaît plus loin) → **support = 2**
- **d :** présent dans SID10 et SID30 → **support = 2**
- **f :** présent dans SID10 et SID30 → **support = 2**
- **e :** seulement SID20 → support 1 (non fréquent)

➔ Donc, depuis a on a des motifs fréquents de taille 2 (séquentiels) :

⟨a⟩⟨b⟩ (support 4), ⟨a⟩⟨c⟩ (4), ⟨a⟩⟨a⟩ (2), ⟨a⟩⟨d⟩ (2), ⟨a⟩⟨f⟩ (2).

Pattern-Growth-basedApproaches

Inconvénient: le nombre de projections peut être encore grand → pas très optimal

❖ **PrefixSpan (Prefix-projected Sequential pattern mining)**

- Amélioration directe de FreeSpan.
- L'objectif : réduire encore le **nombre de projections**.
- Plutôt que de projeter la base entière à chaque extension, on ne garde que les séquences **suffixes à partir du préfixe courant** → donc moins de données à gérer.
- **Avantage :** beaucoup plus efficace que FreeSpan, considéré comme l'un des algorithmes les plus performants pour les motifs séquentiels.

Pattern-Growth-basedApproaches

❖ **PrefixSpan (Prefix-projected Sequential pattern mining)**

➢ **Exemple:**

- **Étape 1 : items fréquents de taille 1**

• On compte **une seule fois par SID** chaque item.

{a, b, c, d, e, f}

- **Étape 2 : Construire les suffixes pour chaque élément**

• PrefixSpan fonctionne en **prolongeant les motifs fréquents** en séquence.

• On regarde chaque élément fréquent comme un **préfixe** et on construit le **suffixe des séquences** après la première occurrence de ce préfixe.

Pattern-Growth-basedApproaches

❖ **PrefixSpan (Prefix-projected Sequential pattern mining)**

➢ **Exemple:**

- **Préfixe a**
  - ✓ **SID10 :** ⟨ {a,b,c}, {a,c}, {d}, {c,f} ⟩ → suffixe après premier a : ⟨ {b,c}, {a,c}, {d}, {c,f} ⟩
  - ✓ **SID20 :** ⟨ {a,d}, {c}, {b,c}, {a,e} ⟩ → suffixe : ⟨ {d}, {c}, {b,c}, {a,e} ⟩
  - ✓ **SID30 :** ⟨ {e,f}, {a,b}, {d,f}, {c}, {b} ⟩ → suffixe : ⟨ {b}, {d,f}, {c}, {b} ⟩
  - ✓ **SID40 :** ⟨ {e}, {g}, {a,f}, {c}, {b}, {c} ⟩ → suffixe : ⟨ {f}, {c}, {b}, {c} ⟩

Pattern-Growth-basedApproaches

❖ **PrefixSpan (Prefix-projected Sequential pattern mining)**

➢ **Exemple:**

▪ **Étape 3 : Étendre les motifs par PrefixSpan**

- Pour chaque préfixe fréquent, on regarde les items fréquents dans les suffixes pour créer des séquences plus longues.
- Calcul des éléments fréquents dans suffixe de a :
  - **b :** SID10, SID20, SID30, SID40 → **support = 4**
  - **c :** SID10, SID20, SID30, SID40 → **support = 4**
  - **d :** SID10, SID20, SID30 → **support = 3**
  - **f :** SID10, SID30, SID40 → **support = 3**
  - **a :** SID10, SID20, SID30 → **support = 3**
- Donc les **extensions fréquentes du motif a** : ⟨a → b⟩, ⟨a → c⟩, ⟨a → d⟩, ⟨a → f⟩, ⟨a → a⟩ (répétition possible dans PrefixSpan si permise)

Pattern-Growth-basedApproaches

❖ **PrefixSpan (Prefix-projected Sequential pattern mining)**

➢ **Exemple:**

- On répète le même processus pour chaque préfixe et pour chaque suffixe jusqu'à ce qu'il n'y ait plus d'extensions fréquentes.

- Pour générer les séquences de longueur 3, on répète la même procédure mais en prenant comme **préfixe les séquences de longueur 2 trouvées**.

Exemple: Trouver les "suffixes projetés" après ⟨a,b⟩

Pattern-Growth-basedApproaches

- **PrefixSpan (Prefix-projected Sequential pattern mining)**
  - **Exemple:**

SID	Séquence d'origine	Suffixe projeté après ⟨a,b⟩
10	⟨{a}, {a,b,c}, {a,c}, {d}, {c,f}⟩	⟨{a,c}, {d}, {c,f}⟩
20	⟨{a,d}, {c}, {b,c}, {a,e}⟩	⟨{a,e}⟩
30	⟨{e,f}, {a,b}, {d,f}, {c}, {b}⟩	⟨{d,f}, {c}, {b}⟩
40	⟨{e}, {g}, {a,f}, {c}, {b}, {c}⟩	⟨{c}⟩

Maintenant, on cherche les items fréquents dans ces suffixes projetés

On regarde tous les éléments qui apparaissent dans les suffixes après ⟨a,b⟩ :

Pattern-Growth-basedApproaches

❖ **PrefixSpan (Prefix-projected Sequential pattern mining)**

➢ **Exemple:**

- **c :** présent dans SID10, SID30, SID40 → **support = 3.**
- **a :** présent dans SID10, SID20 → **support = 2.**
- **d :** présent dans SID10, SID30 → **support = 2.**
- **f :** présent dans SID10, SID30 → **support = 2.**
- **b** : présent dans SID30 → **support = 1.**
- **e :** présent dans SID20 → **support = 1.**

Les éléments fréquents (support ≥ 2) sont : **{c, a, d, f}**

Pattern-Growth-basedApproaches

- **PrefixSpan (Prefix-projected Sequential pattern mining)**
  - **Exemple:**

Résultat:

On peut étendre le motif ⟨a,b⟩ avec ces éléments pour créer des motifs de longueur 3 :

- **⟨a, b, c⟩**
- **⟨a, b, a⟩**
- **⟨a, b, d⟩**
- **⟨a, b, f⟩**

➔ Ce sont les **motifs séquentiels fréquents de longueur 3 dérivés de ⟨a,b⟩.**