Skyline Multi-dimensionnelle sur des Données en Flux

Karim Alami, Sofian Maabout

Skyline Multi-dimensionnelle sur des Données en Flux

Karim Alami
LaBRI, Université de Bordeaux
Talence, France
karim.alami@u-bordeaux.fr

Sofian Maabout
LaBRI, Université de Bordeaux
Talence, France
maabout@u-bordeaux.fr

1 RÉSUMÉ

Dans le présent papier, nous proposons une structure d'indexation pour optimiser les requêtes skyline multi-dimensionnelles dans un contexte de données en flux. Quelques travaux ont traité le problème de la maintenance du skyline dans ce contexte, toutefois aucun n'a considéré le cas de skyline par rapport à un sous-ensemble de dimensions. Soient $D = \{D_1, \ldots, D_d\}$ un ensemble de dimensions, $T(id, D)$ un flux de données et ω la taille de la fenêtre de temps glissante, i.e., la durée de vie d'un enregistrement, notre structure répond à des requêtes de la forme $sky(T, X, k)$ skyline de T vis à vis du sous-espace X choisi et $k \leq \omega$ permet de restreindre la requête aux k flux les plus récents.

2 INTRODUCTION

L'opérateur skyline [1] est pertinent pour récupérer les meilleurs éléments d'un ensemble d'éléments, i.e., ceux qui ne sont pas dominés sur tous les critères.

Dans cet article, on traite le problème suivant : Etand donnés $D = \{D_1, \ldots, D_d\}$ un ensemble de dimensions et $T(Id, D_1, \ldots, D_d)$ une table continuellement étendue par un ensemble de données δ. On assume que tous les éléments ont une même *durée de vie* ω, qui dénote aussi la taille de la fenêtre de temps glissante. Maintenant, étant donné un sous-espace $X \subseteq \{D_1, \ldots, D_d\}$, on veut calculer le skyline par rapport à X en ne prenant en compte que les k flux les plus récents tel que $k \leq \omega$.

L'exemple suivant illustre le contexte que l'on étudie.

Example 1. *On considère l'ensemble de données dans le tableau 1, la dernière colonne représentant l'heure d'arrivée. On peut voir T comme une séquence de 5 transactions δ_i, dont chacune a inséré un ensemble d'éléments à l'instant i.*

Soit $\omega = 3$, notre système permet de répondre à une requête qui inclut les k flux les plus récents tel que $k \in \{1, 2, 3\}$.

Notez que pour chaque dimension, on assume que les valeurs les plus petites sont préférées.

Prenons le sous-espace AB et $k = 3$ alors $Sky(T, AB, 3) = \{r_5, r_6\}$. Aussi pour le sous-espace BC et $k = 2$ $Sky(T, BC, 3) = \{r_7, r_8\}$

A notre connaissance [4] a été le premier papier à traiter la problématique de skyline avec un flux de données. Cependant, aucun travail n'a considéré la requête par rapport à un sous-ensemble de dimension. D'un autre côté, les travaux qui traitent du problème de requêtes skyline multidimensionnelles, ne les traitent que dans un environnement où les données sont statiques. La structure qu'on propose dans le présent papier est basée sur la structure *NSC* présentée dans le papier [2].

Id	*A*	*B*	*C*	*heure arr.*
r_1	2	2	2	0
r_2	4	2	3	0
r_3	5	1	3	1
r_4	1	1	3	1
r_5	1	0	4	2
r_6	0	1	5	2
r_7	2	0	3	3
r_8	2	1	1	3
r_9	6	6	6	4

Table 1: T

3 STRUCTURE DE DONNÉES POUR LA REQUÊTE SKYLINE SUR DES DONNÉES EN FLUX

La solution qu'on propose se base sur la structure d'indexation présentée dans le papier [2] et qui consiste à : pour chaque enregistrement r, stocker les sous-espaces où r est dominé en forme de *paires de sous-espaces* en le comparant avec tous les autres enregistrements, e.g. en comparant r_2 à r_1 (voir table 1), on garde l'information $\langle AC|B\rangle$ tel que AC est le sous-espace où r_1 est strictement meilleur que r_2 et B est l'espace où ils sont égaux. En comparant un enregistrement r à tous les autres enregistrements de l'ensemble des données, on obtient un ensemble de paires $Paires(r)$. De cet ensemble, pour vérfier si r appartient au skyline par rapport à un sous-espace, il suffit de calculer la couverture de chaque paire dans $Paires(r)$ de la façon suivante : soit X, Y des sous-espaces de D, $couvert(\langle X|Y\rangle) = \{Z \in D | Z \subseteq XY \wedge X \cap Y \neq \emptyset\}$. Il est intéressant de noter que les paires d'un enregistrement sont organisées en liste d'ensembles de paires tel que les paires calculées avec un ensemble d'enregistrements arrivés au même moment sont rassemblées dans le même ensemble.

3.1 Traitement d'un Enregistrement à son Insertion

Dans cette section, nous expliquons comment nous calculons les paires d'un enregistrement récemment ajouté.

L'algorithme 1 décrit le calcul des paires d'un enregistrement à son arrivée. Pour un enregistrement r récemment ajouté, nous calculons une paire p avec chaque enregistrement r' actif, i.e. la différence entre l'heure d'arrivée $arr(r')$ et l'instant actuel t_c est plus petite que ω, puis la paire est placée dans $Paires(r, arr(r'))$ (ligne 3-5).

Algorithm 1: computePairs

Input: record r, T
Output: Paires(r)

```
Paires(r) ← ∅
begin
    foreach j ∈ [max(0, t_c − ω + 1), t_c] do
        foreach r' ∈ T ∧ (arr(r') = j) do
            Paires(r, j)∪ ← compare(r, r')
return Paires(r)
```

Compression de Paires(r). L'ensemble des paires $Paires(r)$ peut être réduit sans perte d'informations.

Example 2. *La table 2 représente les paires de r*5.

$Paires(r_5, 0)$	$Paires(r_5, 1)$	$Paires(r_5, 2)$
$\langle C\|\emptyset\rangle$	$\langle C\|\emptyset\rangle$	$\langle A\|\emptyset\rangle$
$\langle C\|\emptyset\rangle$	$\langle C\|A\rangle$	

Table 2: Paires de r_5

Observez que $couvert(\langle C|\emptyset\rangle) \subset couvert(\langle C|A\rangle)$ *dans* $Paires(r_5, 1)$. *Cela signifie que nous pouvons ignorer* $\langle C|\emptyset\rangle$ *à partir de* $Paires(r_5, 1)$ *et que nous ne perdons pas les informations de couverture de* $Paires(r_5, 1)$.

Aussi si $k = 1$, *nous calculons les ensembles couverts par les paires dans* $Paires(r_5, 2)$. *Si* $k = 2$, *nous calculons la couverture des paires dans* $(r_5, 1)$ *et* $Paires(r_5, 2)$. *De même, si* $k = 3$, *nous calculons la couverture des paires dans les trois ensembles de* $Pair(r_5)$. *Observez que* $couvert(Paires(r_5, 0)) \subset cover(Paires(r_5, 1))$ *par conséquent, le fait de supprimer tout le contenu de* $Paires(r_5, 0)$ *ne modifie pas le résultat d'une requête où* $k = 3$.

Ainsi les paires conservées sont $\langle C|A\rangle$ *dans* $Paires(r_5, 1)$ *et* $\langle A|\emptyset\rangle$ *dans* $Paires(r_5, 2)$

Nous formulons cette étape dans le problème suivant :

Problème 1. Etant donnée une liste d'ensemble de paires $Paires(r)$, $\forall j \in [max(0, arr(r) - \omega + 1), arr(r)]$, trouver $s(j) \subseteq Paires(r, j)$ t.q. $cover(s(j) \cup Paires(r, j + 1)) = cover(Paires(r, j) \cup Paires(r, j + 1))$ et $s(j)$ est de taille minimum.

Theorem 1. *Problème 1 est NP-difficile.*

3.2 Mise à Jour des Paires d'un Enregistrement

Dans cette section, nous expliquons la procéudre de mise à jour des paires d'un enregistrement r lorsque de nouveaux enregistrements sont ajoutés et que certains existants expirent.

Mise à jour des paires par rapport aux enregistrements expirés. A l'insertion de nouveaux enregistrements, la transaction la plus ancienne arrivée à $t_c - \omega$ est supprimée. Pour tous les enregistrements toujours actifs, les paires calculées avec les enregistrements expirés sont également supprimées. Ces paires sont situées dans $Paires(r, t_c - \omega)$.

Mise à jour des paires par rapport aux enregistrements récemment ajoutés. Pout tout enregistrement r toujours actif, on le compare avec les enregistrements récemment ajoutés et on stocke les paires dans l'ensemble des paires les plus récentes. S'en suit la procédure de compression des nouvelles paires.

4 RÉSULTATS EXPÉRIMENTAUX

Dans cette section, nous allons montrer quelques résultats expérimentaux qui démontrent les performances de notre solution à répondre aux requêtes skyline multi-dimensionnelles dans un contexte de données en flux. Nous comparons notre solution avec l'algorithme de l'état de l'art *BSkyTree*[3]. Faute de place, nous ne présentons que les résultats obtenus avec des données synthétiques indépendantes. Dans la figure 1 on évalue le nombre de requêtes qui peuvent être traitées entre deux transactions successives par *NSCt* ou *BSkyTree*. On considère un flux de données indépendantes avec 12 dimensions, une taille maximale de 10000 enregistrements et des transactions qui insèrent 1000 enregistrements à chaque cycle. On voit que globalement, *NSCt* évalue beaucoup plus de requêtes que *BSkyTree*($\approx \times 100$), même en prenant en compte le temps de mise à jour de *NSCt*. Aussi, l'écart se creuse en augmentant l'intervalle de temps entre deux transactions, ce qui montre que plus l'écart est grand, plus c'est intéressant d'utiliser *NSCt* dans un contexte de données en flux.

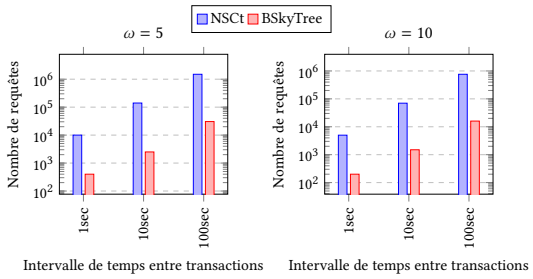

Figure 1: Nombre de requêtes évaluées avec des données indépendantes, $n = 10^4$, $d = 12$ et $|\delta| = 10^3$

5 CONCLUSION

Nous avons proposé une structure de données ainsi qu'une procédure efficace pour sa maintenance dans le but d'accélérer le temps d'évaluation d'une requête skyline multi-dimensionnelle sur des données en flux. À l'avenir, nous prévoyons d'étudier une version distribuée de notre solution.

RÉFÉRENCES

[1] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of ICDE conf.*, pages 421–430, 2001.
[2] N. Hanusse, P. Kamnang-Wanko, and S. Maabout. Computing and summarizing the negative skycube. In *Proc. of CIKM Conference*, pages 1733–1742, 2016.
[3] J. Lee and S. won Hwang. BSkyTree : scalable skyline computation using a balanced pivot selection. In *Proc. of EDBT conf.*, 2010.
[4] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *IEEE Transactions on Knowledge and Data Engineering*, 18(3) :377–391, 2006.