Maintenance Incrémentale du Skycube Négatif

Karim Alami, Sofian Maabout

Maintenance Incrémentale du Skycube Négatif

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

ABSTRACT

Soient $T(Id, D_1, \ldots, D_d)$ une table $X \subseteq \{D_1, \ldots, D_d\}$ un sous-ensemble de dimensions, ou sous-espace, $Sky(T, X)$ dénote le skyline de T vis à vis de X, i.e., l'ensemble des enregistrements de T qui ne sont pas dominés respectivement à X. Pour T, il y a $2^d - 1$ requêtes skylines possibles en fonction du X choisi. Pour optimiser toutes ces requêtes, une manière de procéder consiste à les pré-calculer toutes et les stocker. Ce calcul est coûteux en termes de temps et d'espace mémoire. Un travail antérieur a proposé le skycube négatif (NSC) qui est une structure qui stocke pour chaque enregistrement et d'une manière compressée, l'ensemble des sous-espaces où il est dominé. L'efficacité de cette structures en termes de temps de calcul, d'espace de stockage et d'optimisation des requêtes a déjà été établie. Dans le présent article, nous montrons comment mettre à jour le NSC sans avoir à le recalculer entièrement suite à une insertion/suppression d'un (ensemble de) tuple(s).

1 INTRODUCTION

Depuis son introduction à la communauté gestion de données dans [2], la requête skyline a suscité un large intérêt et différents aspects ont été traités dans la littérature, e.g., [1, 4–7]. Les requêtes skyline *multi-dimensionnelles* en représentent un cas particulier. Ici, on considère que les utilisateurs ont la possibilité de choisir la combinaison d'attributs sur lesquels l'évaluation de la requête va se baser. /

Exemple 1. *Soit la relation* Hotel *dont l'instance est illustrée dans Fig. 1. Les hôtels non dominés respectivement à $X = PDE$ sont h_1, h_2 et h_3. h_4 est dominé car, e.g., h_2 est meilleur que lui selon les trois critères P, D et E. Un utilisateur qui a suffisament de moyens, peut s'affranchir du critère P et ne considérer que D et E pour chercher les meilleurs hôtels alors qu'inversement, un étudiant qui a peu de moyens ne considèrera que le critère P. Dans ce dernier cas, h_3 est le meilleur alors que selon DS, h_2 et h_3 sont les meilleurs. La figure 2 montre les différents résultats en fonction des critères choisis : Noter que dans l'exemple ci-dessus, on cherche à minimiser le prix et la distance alors que l'on veut maximiser le nombre d'étoiles.*

Id	*(P)rix*	*(D)istance APlage*	*(E)oiles*
h_1	*100*	*200*	*4*
h_2	*120*	*150*	*2*
h_3	*80*	*220*	*4*
h_4	*250*	*200*	*1*

Figure 1: Hotel **relation**

Sous-espaces	*Skyline*	*Sous-espace*	*Skyline*
PDE	h_1, h_2, h_3	*PD*	h_2, h_3
PE	h_1, h_2, h_3	*DE*	h_1, h_2
P	h_3	*D*	h_2
E	h_1, h_3		

Figure 2: Le Skycube de la table Hotel

2 LE SKYCUBE NÉGATIF (NSC)

La structure NSC présentée dans [3] se base sur la remarque suivante : Pour un enregistrement t_1, rien qu'en le comparant à un autre tuple t_2, on déduit un *sous-ensemble* de sous-espaces où t_1 est dominé. En reprenant l'exemple 1 et en comparant h_4 à h_1, on obtient la *paire* $\langle PE|D\rangle$ signifiant que h_1 est meilleur que h_4 en P et E alors qu'ils sont égaux en D. De cette paire, on déduit que que h_4 ne fait partie d'aucun skyline relatif à un sous espace X tel que (i) $X \subseteq (PE \cup D)$ et (ii) $X \cap PE \neq \emptyset$.

En comparant donc chaque enregistrement t à tous les autres, on obtient un ensemble de paires, noté $Paires(t)$. Ce dernier encode tous les espaces où t est dominé. Ainsi, vérifier si $t \in Sky(T, X)$ revient à vérifier s'il n'existe pas $p = \langle Y|Z\rangle \in Paires(t)$ telle que $X \subseteq YZ$ et $X \cap Y \neq \emptyset$.

L'ensemble $Paires(t)$ peut être compressé en éliminant les paires superflues. Par exemple, soit $Paires(t) = \{p_1 : \langle AB|C\rangle, p_2 : \langle C|B\rangle, p_3 : \langle C|D\rangle\}$. On peut observer que p_2 peut être supprimée de $Paires(t)$ sans perte d'information. En effet, p_2 couvre $\{CB, C\}$. CB est couvert par p_1 alors que C est couvert par p_3.

Obtenir un sous-ensemble Q de $Paires(t)$ qui lui soit équivalent et de taille minimal a été montré NP-difficile [3] et un algorithme glouton avec garantie de l'approximation y a été proposé. Remarquer que la minimisation de $Paires(t)$ réduit à la fois l'espace de stockage de NSC et le temps de l'évaluation des requêtes.

3 MISE À JOUR DU NSC

Quand un enregistrement est ajouté ou supprimé, il faut mettre à jour le NSC correspondant. Nous commençons d'abord par rappeler la propriété suivante : pour obtenir $Paires(t)$ il suffit de comparer t non pas à *tous* les t' mais seulment à ceux qui sont dans le *topmost* skyline, i.e., le skyline respectivement à *toutes* les dimensions. Ainsi, on suppose que toute $p \in Paires(t)$ est obtenu en comparant t à un élément de *topmost*.

Nous présentons notre solution pour la prise en compte de la suppression et on abordera ensuite l'insertion.

3.1 Suppression

En se basant sur la propriété ci-dessus, on peut d'ores et déjà enoncer le fait suivant :

— Si l'on supprime t et $t \notin topmost$ alors pour chaque t', $Paires(t')$ n'a pas besoin d'être modifiée.

Dans le cas où $t \in topmost$, deux cas de figure peuvent se présenter pour t' selon que la paire obtenue en comparant t' à t, notons la $comp(t', t)$, soit dans $Paires(t')$ ou pas :

— Si $comp(t', t) \notin Paires(t')$ alors la suppression de t n'a aucun effet sur $Paires(t')$.

— Si par contre $comp(t', t) \in Paires(t')$ alors il faut mettre à jours cette dernière.

Dans le dernier cas, il se peut qu'il existe un $t'' \in topmost$ tel que $comp(t', t'') = comp(t', t)$. Dans ce cas aussi, $Paires(t')$ n'a pas à être modifiée. Pour identifier facilement cette situation, on ajoute à chaque paire un compteur indiquant le nombre de tuples du topmost qui ont permis de l'obtenir. Ainsi, lors de la suppression de t, on décrémente le compteur de $comp(t', t)$ et ce n'est que si ce dernier passe à 0 que la mise à jour de $Paires(t')$ devient nécessaire.

3.2 Insertion

Toujours en se basant sur la propriété du *Topmost*, on peut dire que :

— En insérant t, si $t \notin Topmost$, alors $Paires(t')$ n'a pas à être modifié quel que soit t'.

Si t est dans le nouveau *Topmost* alors non seulement il faut ajouter $comp(t', t)$ à $Paires(t)$ mais en plus t' peut dominer des éléments t'' de l'ancien *Topmost* et donc plus besoin de garder $comp(t', t'')$ dans $Paires(t'')$. Noter néanmoins que le fait de garder ces paires ne remet pas en cause la correction de NSC c'est juste dans un soucis de réduction de l'espace de stockage autant que possible. Il est important de noter ici que la suppression d'une paire obtenue suite à une comparison avec un t'' qui n'est plus dans le *Topmost* n'engendre pas la ré-introduction de nouvelles paires qui auraient été supprimées lors de la compression sémantique effectuée par l'algorithme glouton.

4 EVALUATION EXPÉRIMENTALE

Dans cette section, nous montrons quelques uns des résultats expérimentaux que nous obtenons avec notre méthode. Pour ce faire, et faute de place, nous ne présentons que les résultats obtenus avec des données synthétiques (anti-corrélées). Comme solution de base pour la mise à jour, nous avons sélectionné celle qui consiste à reconstruire le NSC à chaque mise à jour. Aussi, nous distinguons deux méthodes de mise à jour : (i) séquentielle, qui itère une même procédure sur un ensemble de tuples insérés/supprimés et (ii) batch, qui fait d'abord un taitement de l'ensemble des insertions/suppressions puis met à jour NSC .

On considère une table T avec un nombre de dimensions (attributs) $d = 14$ dont le NSC correspondant est déjà construit et on mesure le temps nécessaire pour prendre en compte l'insertion (resp. suppression) d'un ensemble de tuples Δ^+ (resp. $\Delta-$. On fait varier à la fois la taille de T ainsi que celle de Δ^+/Δ^-.

Ce que l'on peut observer à partir des Figures 3 et 4 et que la mise à jour en Batch est globalement 100 fois plus rapide que la reconstruction. On note aussi que quand le nombre de tuples insérés/supprimés est petit (une dizaine), la méthode séquentielle est la plus performante.

Nous avons effectué les mêmes expériences avec d'autres types de données (corrélées et indépendantes) et on arrive aux mêmes conclusions. Aussi, le facteur d a son importance car plus d est grand et plus le speed-up de notre approche par rapport à la reconstruction est grand.

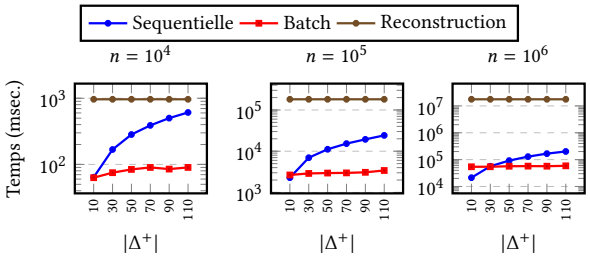

Figure 3: Insertion de Δ^+ en variant $n = |T|$

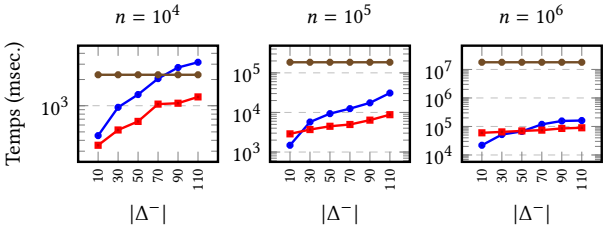

Figure 4: Suppression de Δ^- en variant $n = |T|$

5 CONCLUSION

Nous avons présenté dans cet article une solution pour la prise en compte efficace des mises à jours d'une relation afin de les répercuter sur la structure d'indexation NSC permettant de répondre efficacement aux requêtes skyline. Les expériences montrent que notre solution est largement plus rapide que celle sui consiste à reconstruire le NSC ce qui lui confère le caractère incrémental et qui rend NSC réellement utilisable.

RÉFÉRENCES

[1] K. S. Bøgh, S. Chester, and I. Assent. Skyalign : a portable, work-efficient skyline algorithm for multicore and GPU architectures. *VLDB Journal*, 25(6) :817–841, 2016.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of ICDE conf.*, pages 421–430, 2001.

[3] N. Hanusse, P. Kamnang-Wanko, and S. Maabout. Computing and summarizing the negative skycube. In *Proc. of CIKM Conference*, pages 1733–1742, 2016.

[4] K. Hose and A. Vlachou. A survey of skyline processing in highly distributed environments. *VLDB Journal*, 21(3) :359–384, 2012.

[5] J. Liu, L. Xiong, J. Pei, J. Luo, and H. Zhang. Finding pareto optimal groups : Group-based skyline. *PVLDB*, 8(13) :2086–2097, 2015.

[6] Y. Park, J. Min, and K. Shim. Efficient processing of skyline queries using mapreduce. *IEEE Trans. Knowl. Data Eng.*, 29(5) :1031–1044, 2017.

[7] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 15–26, 2007.