Indexation dynamique pour la maintenance du skyline dans les flux de données

Rui Liu*, Dominique H. Li*,**

*Département d'Informatique, Université de Tours
**Laboratoire d'Informatique Fondamentale et Appliquée de Tours
{liurui, dominique.li}@univ-tours.fr

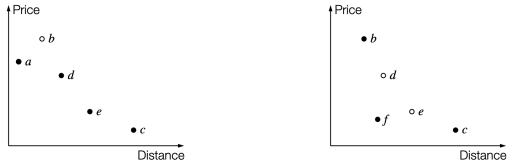
Résumé. Le calcul du skyline reçoit une attention intensive de la communauté des bases de données dont de nombreux algorithmes ont été développés au cours des deux dernières décennies. Cependant, la maintenance des skylines dans les flux de données est un défi car les mises à jour continues de skyline doivent tenir compte consécutivement de l'ajout de n-uplets entrants et la suppression des n-uplets expirés. Dans cet article, nous présentons RSS, une approche efficace basée sur l'indexation dynamique pour calculer des skylines dans les flux de données en fenêtre glissante. Notre analyse théorique prouve que la complexité temporelle de RSS est limitée par un sous-ensemble du skyline instantané ainsi que notre évaluation expérimentale montre l'efficacité de RSS sur les flux de données de haute et de faible dimension.

1 Introduction

Le calcul du skyline reçoit une attention intensive depuis la première introduction de la *requête skyline* (Borzsony *et al.* [2001]) qui vise à récupérer l'ensemble de n-uplets dominants dans des données multidimensionnelles, pour lequel de nombreux algorithmes efficaces ont été développés au cours des deux dernières décennies. Cependant, le problème de maintenance du skyline dans le contexte de flux de données est un vrai défi car il nécessite des mises à jour instantanées du skyline en concernant l'arrivée de nouvelles données et l'expiration de données trop précoces.

Dans cet article, nous présentons une approche efficace pour calculer les skylines dans les flux de données avec la *fenêtre glissante* (Patroumpas and Sellis [2006]). En général, le modèle de fenêtre glissante est basé soit sur le *comptage* qui couvre un nombre d'enregistrements (*n-uplets*) les plus récents à chaque instant, soit sur le temps (*temporelle*) qui est limité par un nombre d'unités de temps coïncidant avec les *timestamps* de données dans le flux. La figure 1 montre un exemple de maintenance du skyline en fenêtre glissante sur la relation prix-distance, où nous considérons les prix (axe Y) d'hôtels par rapport à leurs distances (X axe) vers un endroit, comme le centre-ville, la plage ou la gare, etc. Considérons une fenêtre de comptage $W = 5$ et l'ordre d'arrivée des enregistrements d'hôtel de a à f, alors il y a initialement 4 n-uplets skyline $\{a, c, d, e\}$ illustré dans la figure 1(a). Au moment où f arrive, le premier n-uplet a doit être éliminé afin de conserver la taille de la fenêtre. En conséquence, puisque a est le seul n-uplet qui domine b, b devient un n-uplet skyline lorsque a est écarté ; de plus, le

n-uplet entrant f domine les n-uplets skyline actuels d et c, donc d et c doivent être supprimés du skyline, et finalement le skyline à jour est donc $\{b, c, f\}$, comme le montre la figure 1(b).



(a) Skyline initial. (b) Skyline à jour.

FIG. 1 – *La maintenance du skyline.*

En fait, les mises à jour dynamiques montrées dans la figure 1 rendent la maintenance du skyline difficile sur les flux de données : (1) un n-uplet skyline expiré peut libérer des n-uplets dominés de sorte qu'ils deviennent des n-uplets skyline ; (2) un n-uplet entrant domine les n-uplets skyline existants de sorte que ces derniers doivent être supprimés du skyline. L'algorithme RSS (**R**ange **S**earch for **S**treams) présenté dans cet article est basé sur nos travaux récents (Liu and Li [2020a,c]), qui permet la maintenance efficace du *skyline en fenêtre glissante* dans les flux de données en utilisant l'indexation dynamique. Nos principales contributions comprennent :

— nous construisons une structure chaînée de tri pour maintenir un index dimensionnel dynamique qui dessert la fenêtre glissante ;
— nous montrons qu'avec l'indexation dynamique, les tests de dominance imposés par (1) et (2) ci-dessus sont limités à des sous-ensembles du skyline actuel dans la fenêtre glissante, qui rassure l'efficacité de notre approche.

Le reste de cet article est organisé comme suit. La section 2 brièvement introduit les travaux existants liés au calcul de skylines dans les flux de données. Dans la section 3, nous présentons l'approche RSS avec une analyse de complexité temporelle. Nous rapportons l'évaluation des performances de RSS dans la section 4 et nous concluons dans la section 5.

2 Travaux reliés

Le calcul du skyline est relativement peu étudié dans les flux de données en comparaison avec les bases de données conventionnelles. Basé sur l'algorithme BBS (Papadias *et al.* [2005]), Tao and Papadias [2006] ont proposé Lazy/Eager pour la maintenance de skylines dans les flux de données ; étant une amélioration de Lazy/Eager, Morse *et al.* [2007] ont proposé LookOut dont la structure de données se fond sur l'arbre quaternaire. Les deux approches sont limitées dans des données faibles dimensionnelles ($d \leq 6$) et donc ne sont pas comparables avec notre approche présentées dans cet article. L'approche la plus récente MSSD (Alami and Maabout [2020]) utilise une structure d'index NSCt (Negative SkyCube with timestamps) pour traiter les requêtes sous-espace de skyline dans les flux de données, qui est plus efficace que la méthode de l'état-de-l'art BSkyTree (Lee and Hwang [2014]) sur la maintenance de skylines. Nous montrons dans cet article que notre approche RSS est plus efficace que MSSD sur la mise à jour de n-uplets skyline.

3 Maintenance de skylines dans les flux de données

Soit t un n-uplet d-dimensionnel, on note $t[i]$ la valeur du n-uplet t sur la dimension i. Nous définissons l'*ordre de préférence* $\prec$ comme un ordre total sur toutes les dimensions, alors étant donné deux n-uplets t et u, on dit que $t[i]$ est mieux que $u[i]$ si $t[i] \prec u[i]$ et $t[i]$ est identique à $u[i]$ si $t[i] = u[i]$. On dit que $u[i]$ n'est pas pire que $t[i]$ si $t[i] \prec u[i]$ ou $t[i] = u[i]$, noté $t[i] \preceq u[i]$. Nous notons $t \prec u$ que t *domine* u si pour chaque dimension $1 \leq i \leq d$ nous avons $t[i] \preceq u[i]$ et pour au moins une dimension $1 \leq k \leq d$ nous avons $t[k] \prec u[k]$. On note $t \not\prec u$ pour que t *ne domine pas* u. Étant donnée une base de données d-dimensionnelles $\mathcal{D}$, un n-uplet $t \in \mathcal{D}$ est un *n-uplet skyline* si et seulement si $\nexists u \in \mathcal{D}$ tel que $u \prec t$. Le *skyline* $\mathcal{S}$ d'une base de données $\mathcal{D}$ est l'ensemble de n-uplets tel que $\mathcal{S} = \{t \in \mathcal{D} \mid \nexists u \in \mathcal{D}, u \prec t\}$. Pour tout couple de n-uplets skyline s et t, nous avons que s et t sont incomparables, noté $s \nsim t$.

Définition 1 (Index dimensionnel)**.** *Soit $\mathcal{D}$ une base de données d-dimensionnelles, l'index dimensionnel $\mathcal{I}$ de $\mathcal{D}$ est l'ensemble des d listes de blocs sur l'ordre de préférence $\prec$, chaque liste $I_i \in \mathcal{I}$, $1 \leq i \leq d$, est un sous-index par dimension. Chaque bloc dans I_i correspond à une valeur unique sur la dimension i et contient un ou plusieurs ID de n-uplet dans l'ordre lexicographique.*

Théorème 1. *Soit $\mathcal{I}$ l'index dimensionnel d'une base de données d-dimensionnelles, I_i est un sous-index arbitraire de $\mathcal{I}$ et t est un n-uplet skyline dans son bloc sur I_i. Alors, t est un n-uplet skyline s'il n'existe aucun n-uplet u tel que $u[i] \prec t[i]$, ou nous avons $s \not\prec t$ pour tous les n-uplets skyline s tels que $s[i] \prec t[i]$.*

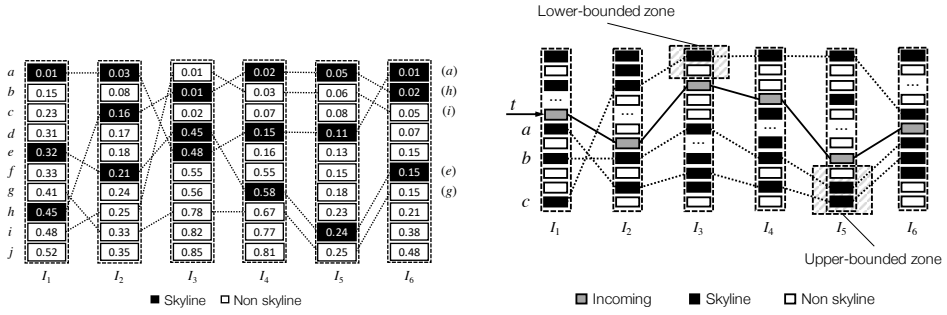


FIG. 2 – *(a) Un exemple d'index dimensionnel. (b) Détection de n-uplets skylines dans des zones bornées*

Prenons l'exemple de la figure 2(a) qui se compose de 6 sous-index $I_1, I_2, \ldots, I_6$ et 10 n-uplets. Pour simplifier notre exemple, nous nous intéressons aux n-uplets $\{a, e, g, h, i\}$ en supposant que les autres ne sont pas n-uplets skyline, où nous représentons les n-uplets par des lignes pointillées. Selon le théorème 1, a est un n-uplet skyline étant conclu à partir de I_1, I_2 ou I_6. Si nous nous concentrons uniquement sur I_3, nous avons que $h[3] = i[3]$ dont aucun n-uplet n'est pas meilleur que h et i dans cette dimension, donc $h \prec i$ et $i \prec h$ doivent être

testées à la fois pour déterminer si h et i sont des n-uplets skyline (en effet nous avons $h \prec i$ et $i \not\prec h$). Avec I_6 seulement, puisque 3 entrées consécutives contiennent la même valeur 0.15, ces 3 n-uplets doivent d'abord être comparés afin de filtrer les n-uplets skyline localement présents, e dans notre exemple, auquel le théorème 1 peut donc être appliqué.

Basé sur le théorème 1, nous proposons une nouvelle méthode pour la mise à jour d'index dimensionnel pour la maintenance efficace de skylines. Nous illustrons notre méthode avec l'exemple montré dans la figure 2(b) où nous supposons que toutes les valeurs sur n'importe quelle dimension sont uniques. Soit t un n-uplet entrant, nous localisons d'abord les entrées de t sur chaque sous-index par rapport à l'ordre de préférence $\prec$, comme ceux en grise. Ensuite, nous trouvons une dimension low, qui contient le nombre minimum de n-uplets skyline s tel que $s[low] \prec t[low]$, que nous appelons la dimension de *borne inférieure* (*lower-bounded dimension*) de t où une *zone inférieure* (*lower-bounded zone*) peut être détectée. Nous appelons l'ensemble des n-uplets skyline contenus dans la zone inférieure le *skyline inférieur* (*lower-bounded skyline*), auquel nous appliquons le théorème 1 pour tester si t est un n-uplet skyline. Dans cet exemple, le skyline inférieur est situé dans I_3 dont t est un n-uplet skyline. Le n-uplet entrant t peut évidemment dominer les n-uplets skyline existants, le théorème 1 peut également être appliqué pour les filtrer. En effet, nous trouvons une dimension up qui contient le nombre minimum de n-uplets skyline s tel que $t[up] \prec s[up]$, que nous appelons la dimension de *borne supérieure* (*upper-bounded dimension*) de t où une *zone supérieure* (*upper-bounded zone*) et le *skyline supérieur* (*upper-bounded skyline*) peuvent être détectés : pour chaque n-uplet s contenu dans le skyline supérieur, si $t \prec s$, alors s peut être supprimé du skyline. Dans notre exemple, les n-uplets a et b sont dominés par t, donc ils ne seront plus dans le skyline ; sinon, si t n'est pas un n-uplet skyline, aucune détection du skyline supérieur n'est nécessaire. Lors de la suppression d'un n-uplet skyline t, pour chaque n-uplet non skyline x contenu dans la zone inférieure de t telle que $t \prec x$, la zone inférieure doit être parcourue afin de tester si x est dominé par le skyline inférieur ; sinon, x sera un nouveau n-uplet skyline. Aucun test de dominance n'est requis lors de la suppression d'un n-uplet non skyline.

L'algorithme RSS (**R**ange **S**earch for **S**tream) est donc conçu de la description ci-dessus de tests de dominance.

1. Lorsque la fenêtre n'est pas encore remplie, effectuer un calcul incrémentiel du skyline par rapport aux n-uplets entrants.
2. Dès la fenêtre est remplie, un n-uplet entrant invoque la collection de n-uplets expirés par rapport au mode de la fenêtre, soit supprimer un seul n-uplet pour la fenêtre de comptage, soit supprimer un ensemble de n-uplets pour la fenêtre temporelle.
3. Pour chaque n-uplet skyline expiré, vérifier si des n-uplets dominés peuvent être libérés pour devenir de nouveaux n-uplets skyline.
4. Supprimer les n-uplets expirés de la fenêtre.
5. Ajouter le n-uplet entrant à la fenêtre en effectuant un calcul incrémentiel du skyline.

Afin de déterminer les dimensions de borne inférieure et de borne supérieure pour un n-uplet t, nous utilisons les formules d'estimation au lieu de compter les tailles exactes des skylines inférieur et supérieur :

$$I_{lower} = \arg\min_{I_i}(|\frac{v_t^i - \min(I_i)}{\max(I_i) - \min(I_i)}|) \quad \text{et} \quad I_{upper} = \arg\max_{I_i}(|\frac{v_t^i - \min(I_i)}{\max(I_i) - \min(I_i)}|).$$

L'estimation approximative de ces deux dimensions nécessite $\mathcal{O}(d)$, qui est un compromis entre la sélection de la meilleure dimension et le calcul dans le moins de temps. Dans une fenêtre glissante W sur un flux de données d-dimensionnelles, soit M la taille du skyline et supposons que le test de dominance nécessite $\mathcal{O}(d)$, nous avons le theorem 2 et le theorem 3.

Théorème 2. *Dans le pire cas, RSS insère un n-uplet skyline entrant dans le temps*

$$\mathcal{O}\Big(d\log|W| + 2(d+M) + \frac{M}{d}(d + \frac{(|W|-M)(d+M)}{d})\Big).$$

Démonstration. Pour insérer des entrées d'index dans les d sous-index de taille $|W|$ basés sur B-tree, le temps $\mathcal{O}(d\log|W|)$ est nécessaire. Ensuite, si t est un n-uplet skyline, le temps $\mathcal{O}(2d)$ est nécessaire pour trouver les dimensions de borne inférieure et de borne supérieure, où la taille des skylines inférieur et supérieur est à la fois M/d. Selon le théorème 1, M/d tests de dominance sont requis par t avec chaque skyline borné, donc le temps total est limité à $\mathcal{O}(2d+2M)$. Dans le pire cas, t domine tous les M/d n-uplets dans le skyline supérieur, puis pour chaque n-uplet skyline supérieur, le temps $\mathcal{O}(d)$ est nécessaire pour trouver la dimension borne supérieure, où les $(|W|-M)/d$ n-uplets non skyline sont dominés par chaque n-uplet skyline supérieur. Enfin, pour chacun de ces n-uplets non skyline, le temps $\mathcal{O}(d)$ est nécessaire pour trouver la dimension borne inférieure dont la taille bornée est M/d dans le pire des cas, c'est-à-dire que le temps $\mathcal{O}(d+M)$ est nécessaire pour tester les n-uplets skyline. □

Théorème 3. *Dans le pire des cas, RSS supprime un n-uplet skyline expiré dans le temps*

$$\mathcal{O}\Big(d + \frac{(|W|-M)(d+M)}{d} + d\log|W|\Big).$$

Démonstration. Pour supprimer un n-uplet skyline, le temps $\mathcal{O}(d)$ est requis pour trouver la dimension borne supérieure. Ensuite, comme indiqué dans la preuve du théorème 2, les $(|W|-M)/d$ n-uplets non skyline dans la zone supérieure doivent être testés dans le pire des cas, chaque test nécessite le temps $\mathcal{O}(d+M)$. Enfin, le temps $\mathcal{O}(d\log|W|)$ est nécessaire pour supprimer les d entrées des sous-indexes B-tree. □

Avec l'analyse ci-dessus, il est clair que RSS nécessite le temps $\mathcal{O}(d\log|W|+d+M)$ pour insérer un n-uplet entrant non skyline et $\mathcal{O}(d\log|W|)$ pour supprimer un n-uplet non skyline expiré. De plus, plus la dimensionnalité d de données est élevée, plus le facteur $(|W|-M)$ est petit. En effet, notre évaluation expérimentale confirme l'efficacité de RSS dans les flux de données de haute dimension.

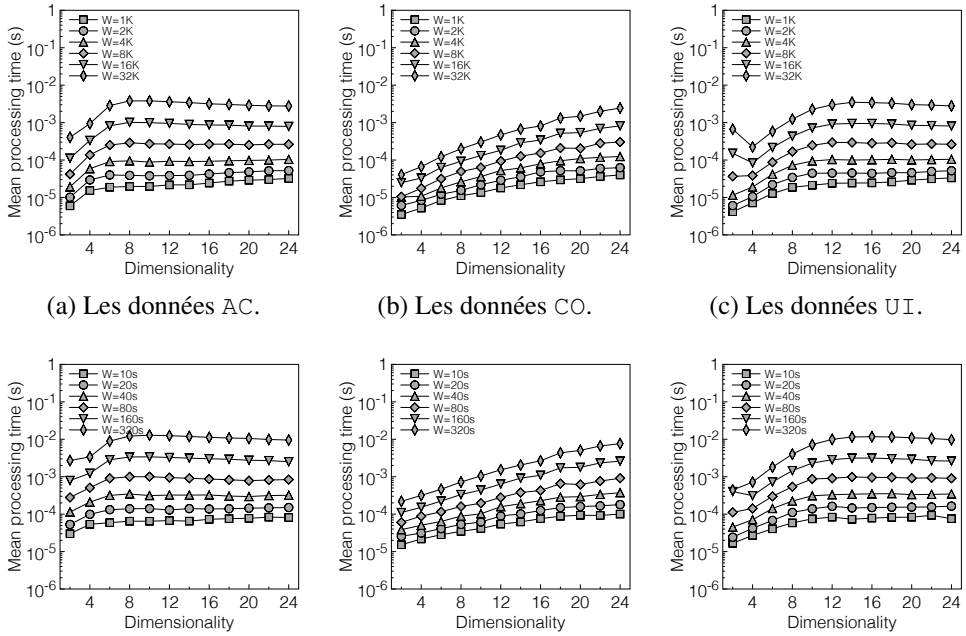
4 Évaluation expérimentale

Nous présentons notre évaluation expérimentale de RSS sur la fenêtre de comptage et la fenêtre temporelle sur les flux de données. Nous avons implémenté RSS en C++ [1] dont tous les exécutables sont compilés par LLVM Clang avec l'option `-O3`. Toutes les expériences ont été menées sur un processeur Intel Core i5 3,1 GHz avec 16 Go de RAM. En raison de la limitation

1. `https://github.com/skyline-sdi/sdi-rss`

de l'espace de cet article, nous ne présentons que des résultats essentiels [2]. Tous les résultats rapportés sont les performances moyennes sur 5 itérations.

Nous considérons les jeux de données synthétiques *anti-corrélées* (AC), *corrélées* (CO) et *indépendantes uniformes* (UI) décrits dans l'article de Borzsony *et al.* [2001]. Différentes tailles de fenêtre sont prises en compte dont $W \in \{1\text{K}, 2\text{K}, 4\text{K}, 8\text{K}, 16\text{K}, 32\text{K}\}$ n-uplets pour la fenêtre de comptage et $W \in \{10, 20, 40, 80, 160, 320\}$ secondes pour la fenêtre temporelle. En particulier, la vitesse d'arrivée du flux de données est uniformément randomisée entre 100 et 1000 n-uplets par seconde pour la fenêtre temporelle. Tous les résultats sont rapportés après le remplissage de la fenêtre glissante.



(a) Les données AC. (b) Les données CO. (c) Les données UI.

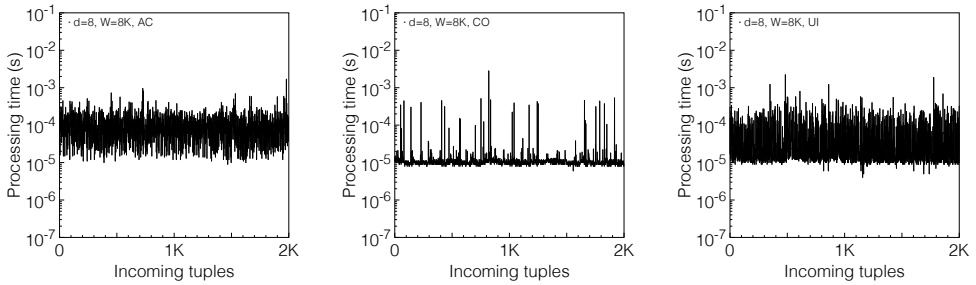
(d) Les données AC. (e) Les données CO. (f) Les données UI.

FIG. 3 – *Temps moyen de mise à jour du skyline pour la fenêtre de comptage (a, b, c) et la fenêtre temporelle (d, e, f) concernant la dimensionnalité sur les données synthétiques.*

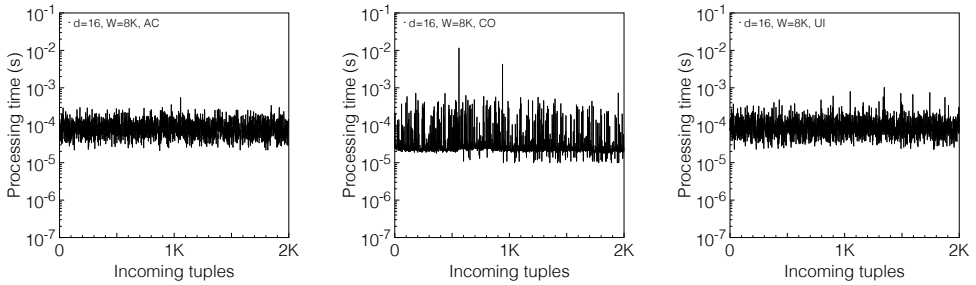
La figure 3 montre le temps de traitement moyen des données synthétiques en faisant varier la dimensionnalité de 2 à 24 par incréments de 2, où le temps de mise à jour par nouveau n-uplet entrant atteint 10^{-5} seconde. Nous notons qu'il y a des points irréguliers lors de $d = 2$ et $W \geq 8\text{K}$ dans la figure 3(c), où les mises à jour de skylines dépensent beaucoup plus de temps. Après avoir tracé ce phénomène dans toutes les courbes, nous constatons que le premier n-uplet expiré est un n-uplet skyline qui domine presque la moitié de n-uplet dans la fenêtre donc la mise à jour prend énorme de temps. Par exemple, pour $W = 32\text{K}$, le premier n-uplet expiré domine plus de 15 000 n-uplets dans la fenêtre dont la mise à jour dépasse 1 seconde, ainsi, le temps moyen pour la mise à jour est beaucoup affecté. Cependant, sauf pour le premier

2. D'autres résultats sont disponibles dans la version intégrale de cet article (Liu and Li [2020b]).

n-uplet expiré, la mise à jour de skylines pour tout nouveau n-uplet reste au niveau de 10^{-5} seconde en moyenne. La figure 4 montre les temps de mise à jour de skylines des données de 8 dimensions et de 16 dimensions dans une fenêtre de 8K n-uplets, les piques vérifient notre observation.



(a) Les données AC, $d = 8$. (b) Les données CO, $d = 8$. (c) Les données UI, $d = 8$.



(a) Les données AC, $d = 16$. (b) Les données CO, $d = 16$. (c) Les données UI, $d = 16$.

FIG. 4 – *Temps de mise à jour de skylines pour les nouveaux n-uplet dans la fenêtre $W = 8K$.*

La table 1 montre brièvement une comparaison du temps moyen passé sur les données indépendantes uniformes de 8 dimensions entre RSS et MSSD. Le temps moyen passé par RSS pour la mise à jour de skylines (100 nouveaux n-uplets) est plus court que le temps passé par MSSD (buffer = 100) pour la mise à jour de sa structure de données sans compter les requêtes sous-espace, toutefois les deux approches ne sont pas dans le même contexte d'applications.

	$W = 1$K	$W = 2$K	$W = 4$K	$W = 8$K	$W = 16$K	$W = 32$K
RSS	0,04 ms	0,06 ms	0,09 ms	0,22 ms	0,52 ms	1,36 ms
MSSD	0,28 ms	0,55 ms	1,13 ms	2,18 ms	4,20 ms	9,93 ms

TAB. 1 – *Comparaison entre RSS et MSSD.*

5 Conclusions

Dans cet article, nous présentons un algorithme efficace pour la maintenance de skylines dans les flux de données. Nous proposons une méthode d'indexation avec laquelle les tests de

dominance sont bornés à des sous-ensembles de skyline existant dans la fenêtre glissante. Nous avons développé l'algorithme RSS dont notre analyse théorique montre que la complexité temporelle de la mise à jour du skyline en fenêtre glissante est bornée. L'algorithme proposé a été évalué avec les fenêtres glissantes de comptage et temporelles et son efficacité est montrée. En tant que travail futur possible, nous nous intéressons à l'adaptation de RSS aux flux de données distribués.

Références

Karim Alami and Sofian Maabout. A framework for multidimensional skyline queries over streaming data. *Data & Knowledge Engineering*, page 101792, 2020.

Stephan Borzsony, Donald Kossmann, and Konrad Stocker. The Skyline operator. In *ICDE*, pages 421–430, 2001.

Jongwuk Lee and Seung-Won Hwang. Scalable skyline computation using a balanced pivot selection technique. *Information Systems*, 39 :1–21, 2014.

Rui Liu and Dominique Li. Calcul efficace du skyline basé sur l'indexation dimensionnelle. In *EGC*, pages 285–292, 2020.

Rui Liu and Dominique Li. Dynamic dimension indexing for efficient skyline maintenance on data streams. In *DASFAA*, pages 272–287, 2020.

Rui Liu and Dominique Li. Efficient skyline computation in high-dimensionality domains. In *EDBT*, pages 459–462, 2020.

Michael Morse, Jignesh M Patel, and William I Grosky. Efficient continuous skyline computation. *Information Sciences*, 177(17) :3411–3437, 2007.

Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, 30(1) :41–82, 2005.

Kostas Patroumpas and Timos Sellis. Window specification over data streams. In *EDBT*, pages 445–464, 2006.

Yufei Tao and Dimitris Papadias. Maintaining sliding window skylines on data streams. *IEEE Transactions on Knowledge and Data Engineering*, 18(03) :377–391, 2006.

Summary

The maintenance of skylines in data streams is challenging because of updating non stop adding of incoming tuples and removing of expired tuples. In this paper, we present a dynamic dimension indexing based approach RSS to skyline maintenance on data streams, which is efficient at both count-based and time-based sliding windows regardless the dimensionality of data. Our analysis shows that the time complexity of RSS is bounded by a subset of the skyline and our performance evaluation shows the efficiency of RSS.