Calcul efficace de skylines basé sur les requêtes de sous-ensemble

Dominique H. Li

Université de Tours, Laboratoire LIFAT
dominique.li@univ-tours.fr

Résumé. Le traitement des requêtes skyline est essentiel pour la communauté de bases de données, de nombreux algorithmes ont été conçus sur la réduction des tests de dominance entre les points multidimensionnels afin d'effectuer un calcul efficace dont l'incompatibilité entre les points a été considérée comme une propriété importante pour éviter les tests de dominance inutiles. Nous présentons une nouvelle approche basée sur les requêtes de sous-ensemble qui permet d'intégrer l'incomparabilité basée sur les sous-espaces aux algorithmes de skyline existants basés sur le tri et de réduire considérablement le nombre des tests de dominance afin d'accélérer les algorithmes existant dans les grands jeux de données. [1]

Résumé en Français de l'article "Subset Approach to Efficient Skyline Computation" à EDBT'23.

1 Introduction

Étant donné un ensemble de points multidimensionnels, l'*Opérateur Skyline* (Borzsony et al. (2001)) renvoie le *skyline* qui est l'ensemble de tous les points *non dominés*. Dans cette définition, un point p_i est dit *non dominé* s'il n'existe aucun autre point p_j tel que p_j soit meilleur que p_i dans toutes les dimensions par rapport à un ordre de préférence défini par l'utilisateur. Effectivement, le problème du calcul du skyline a reçu une attention intense par la communauté de bases de données.

Afin de résoudre efficacement ce problème, de nombreux algorithmes ont été conçus en se basant sur la réduction du nombre de tests de dominance, qui peuvent être classés en deux catégories : ceux basés sur le tri, tels que `BNL` (Borzsony et al. (2001)), `Index` (Tan et al. (2001)), `LESS` (Godfrey et al. (2007)), `SFS` (Chomicki et al. (2005)), `SaLSa` (Bartolini et al. (2008)) et `SDI` (Liu et Li (2020)), et ceux basés sur le partitionnement, tels que `D&C` (Borzsony et al. (2001)), `NN` (Kossmann et al. (2002)), `BBS` (Pei et al. (2005)), `LS` (Morse et al. (2007)), `OSPS` (Zhang et al. (2009)), `ZSearch` (Lee et al. (2010)) et `BSkyTree-P`/`BSkyTree-S` (Lee et Hwang (2014)). Essentiellement, l'efficacité d'un algorithme de skyline dépend fortement de la manière dont les tests de dominance sont réduits.

D'autre part, dans un espace de d dimensions, un sous-espace (Pei et al. (2005)) est un sous-ensemble de toutes les d dimensions. Soit $\mathcal{P}$ l'ensemble de points, $p \in \mathcal{P}$ un point skyline et

1. `https://openproceedings.org/2023/conf/edbt/paper-28.pdf`

$q \in \mathcal{P}$ n'importe quel autre point tel que $p \neq q$. Alors, si p ne domine pas q, il doit exister un sous-espace dans lequel, pour chaque dimension, la valeur de q est meilleure que celle de p ; autrement dit, q domine p dans ce sous-espace. Nous appelons un tel sous-espace un *sous-espace dominant*. Ainsi, si tous les points dominés par le point skyline p ont été éliminés, chaque point restant dans l'ensemble de points doit posséder un sous-espace dominant dans lequel il domine p. En effet, un tel point skyline p, également appelé *point pivot*, est beaucoup utilisé pour partitionner les points afin de réduire les tests de dominance.

Dans cet article, nous présentons une nouvelle approche d'indexation de skyline basée sur les sous-espaces pour gérer l'incomparabilité entre les points de test et les points skyline. Au lieu de partitionner l'ensemble de points, notre méthode indexe les points skyline en fonction des sous-espaces dominants pour réduire les tests de dominance. Nous montrons d'abord que le *sous-espace dominant* d'un point peut être fusionné selon plusieurs points pivots, ce que nous appelons le *sous-espace dominant maximal* ; ensuite, nous démontrons aussi que si un point p domine un point q, les sous-espaces dominants maximaux de p selon chaque point pivot doivent être un sur-ensemble de ceux de q. Sur la base des propriétés présentées, le schéma d'application de notre méthode peut être décrit comme suit :

1. trouver plusieurs points pivots pour générer le sous-espace dominant maximal pour chaque point non éliminé ;
2. exécuter un algorithme skyline avec les nouvelles actions suivantes :
   (a) une fois qu'un point skyline est déterminé, l'ajouter à la structure d'index proposée en fonction de son sous-espace dominant maximal ;
   (b) lors du test d'un point avec le skyline actuel, récupérer uniquement l'ensemble des points comparables à partir de la structure d'index proposée du sous-espace dominant maximal ;
3. retourner le skyline.

Notre méthode proposée ne dépend pas des algorithmes concrets de calcul du skyline, car elle est conçue comme un composant semblable à un conteneur qui permet de stocker les points skyline et de récupérer un nombre minimal de points skyline à comparer avec un point de test afin d'améliorer les algorithmes existants, qui présente un intérêt réel pour les industries de données. Le paradigme de notre méthode convient le mieux aux algorithmes de skyline basés sur le tri parce que les algorithmes basés sur le partitionnement ne tirent pas beaucoup d'avantages de notre méthode puisque les points sont déjà partitionnées et les tests de dominance sont limités aux partitions dont le double partitionnement entraîne des coûts supplémentaires.

Le principal résultat présenté dans cet article est que l'indexation du skyline peut être efficacement résolue par une requête de sous-ensemble. Le problème de requête de sous-ensemble est défini comme : étant donné un ensemble $\mathcal{X}$ de sous-ensembles distincts d'un univers D, pour tout ensemble de requête $Q \subseteq D$, retourner l'ensemble $\{Q' \in \mathcal{X} \mid Q \subset Q'\}$ qui contient tous les sur-ensembles de Q. En effet, étant donné un point à tester, son sous-espace dominant maximal peut être considéré comme un ensemble de requête. La tâche consiste à retourner tous les points skyline dont les sous-espaces dominants maximaux sont des sur-ensembles de l'ensemble de requête donc les tests de dominance requis peuvent être limités aux points skyline retournés.

Le reste de cet article est organisé comme suit. La section 2 définit les concepts nécessaires à la formalisation. Nous montrons dans la section 3 que le sous-espace dominant maximal de chaque point peut être fusionné à partir de plusieurs points pivots. Dans la section 4, nous présentons notre approche basée sur la requête de sous-ensemble pour indexer le skyline. La section 5 présente notre évaluation expérimentale et enfin nous concluons en section 6.

2 Préliminaires

Nous considérons un ensemble $\mathcal{P}$ (*nous ne redéfinissons plus la notation $\mathcal{P}$ dans le reste de cet article*) composé de N points de d dimensions, où N représente la *cardinalité* et d la *dimensionnalité* de $\mathcal{P}$. Soit $p \in \mathcal{P}$ un point, nous notons $p[i]$ la valeur de p sur la dimension i, où $1 \leq i \leq d$. Sans perte de généralité, l'*ordre de préférence* peut être défini comme la relation $<$ sur les valeurs dans chaque dimension pendant la modélisation du skyline. Étant donnés deux points p and q, $p[i]$ est considéré comme *meilleur que* $q[i]$ si $p[i] < q[i]$; $p[i]$ est *égal à* $q[i]$ si $p[i] = q[i]$; et $q[i]$ est *pas pire que* $p[i]$ si $p[i] \leq q[i]$.

Nous considérons que l'ordre de préférence $<$ est appliqué à toutes les dimensions, cela nous permet de traiter uniformément les relations entre les points pour calculer le skyline. Ainsi, un point p *domine* un point q, noté par $p \prec q$, si et seulement si dans chaque dimension i, nous avons $p[i] \leq q[i]$, et dans au moins une dimension k, nous avons $p[k] < q[k]$. Étant donné deux points p et q, nous notons $p \not\prec q$ pour signifier que p ne domine pas q ; nous notons $p \nsim q \iff (p \not\prec q) \wedge (q \not\prec p)$ pour signifier que p et q sont *incomparables*. Nous étendons également $\{\prec, \not\prec, \nsim\}$ à l'*ensemble de points* de la manière suivante : $p \prec X$ signifie $\forall q \in X$ tel que $p \prec q$; $X \prec p$ signifie $\exists q \in X$ tel que $q \prec p$; $X \not\prec p$ signifie $\nexists q \in X$ tel que $q \prec p$; et $p \nsim X$ ou $X \nsim p$ signifient $\forall q \in X$ tel que $p \nsim q$.

Un point p est un *point skyline* si et seulement s'il n'existe aucun point q tel que $q \prec p$. Le *skyline* $\mathcal{S}$ de $\mathcal{P}$ est l'ensemble complet des points skyline, tel que $\mathcal{S} = \{p \in \mathcal{P} \mid \nexists q \in \mathcal{P}, q \prec p\}$. Le calcul du skyline consiste à retrouver l'ensemble complet des points skyline à partir d'un ensemble de points multidimensionnels en fonction d'un ordre de préférence défini par l'utilisateur $\prec$.

Définition 1. *Soit $\mathcal{P}$ de d dimensions. L'ensemble $D = \{1, 2, \ldots, d\}$ est l'*espace *de $\mathcal{P}$. Tout sous-ensemble $D' \subseteq D$ est un* sous-espace *de $\mathcal{P}$.*

Définition 2. *Soient $p, q \in \mathcal{P}$ deux points dans l'espace de D, $D_{p \prec q}$ le sous-espace tel que $\forall i \in D_{p \prec q} \Rightarrow p[i] < q[i]$ et $\forall i \notin D_{p \prec q} \Rightarrow q[i] \leq p[i]$, alors $D_{p \prec q}$ est le* sous-espace dominant *de p par rapport à q.*

Selon la définition 2, nous avons : (1) $D_{p \prec q} = \emptyset \Rightarrow q \prec p$ ou $p = q$; (2) $D_{p \prec q} = D \Rightarrow p \prec q$.

(*En raison de la limitation du nombre de pages, toutes les preuves des lemmes et les analyses des complexités sont disponibles dans l'article complet publié dans EDBT'23.*)

Lemme 1. *Soit $p \in \mathcal{P}$ un point skyline et $q_1, q_2 \in \mathcal{P}$, $q_1, q_2 \neq p$, deux points arbitraires tels que $p \not\prec q_1$, $p \not\prec q_2$ et $q_1 \neq q_2$. $q_1 \nsim q_2$ si $D_{\{q_1 \prec p\}} \not\subseteq D_{\{q_2 \prec p\}}$ et $D_{\{q_2 \prec p\}} \not\subseteq D_{\{q_1 \prec p\}}$. C'est-à-dire, $|D_{\{q_1 \prec p\}} \cap D_{\{q_2 \prec p\}}| < \min(|D_{q_1 \prec p}|, |D_{q_2 \prec p}|) \Rightarrow q_1 \not\sim q_2$.*

Lemme 2. *Soit $p \in \mathcal{P}$ un point skyline et $q_1, q_2 \in \mathcal{P}$, $q_1, q_2 \neq p$, deux points arbitraires tels que $p \not\prec q_1$, $p \not\prec q_2$ et $q_1 \neq q_2$. $D_{q_1 \prec p} \not\supseteq D_{q_2 \prec p} \Rightarrow q_1 \not\prec q_2$.*

Le Lemme 2 montre une contrainte inévitable pour partitionner les points conformément au Lemme 1 : si un point p est un point skyline dans l'ensemble de points déterminé par le sous-espace D_x, alors p doit être comparé avec tous les points skyline dans l'ensemble de points déterminé par tout sous-espace $D_y \supset D_x$. Étant donné qu'un espace de d dimensions contient $2^d - 2$ sous-espaces (sans $\emptyset$ ni l'espace complet dans notre contexte), il est évident que les tests de dominance peuvent être efficacement réduits si tous les points peuvent être répartis dans le plus grand nombre possible de sous-espaces incomparables.

3 Union de sous-espaces

Selon le Lemme 1, si un point skyline $p \in \mathcal{P}$ dans un espace D dont $|D| = d$ est comparé à chaque autre point $q \in \mathcal{P}$, alors chaque point non éliminé q peut être attribué à un sous-espace dominant $D_{q \prec p}$, où $D_{q \prec p} \neq \emptyset$ et $D_{q \prec p} \neq D$. En fait, nous constatons que la distribution des points est déséquilibrée selon la taille du sous-espace (de 1 à $2^d - 2$), la plupart d'entre eux se situant dans des zones de petite taille, loin du niveau 2^d. Bien que des appels récursifs puissent être appliqués à chaque sous-espace pour trouver davantage de sous-espaces incomparables conformément au Lemme 1, le Lemme 2 limite la sortie immédiate des points skyline.

Nous proposons une méthode d'union de sous-espaces pour distribuer les points dans autant de sous-espaces que possible, où le terme *autant que possible* est contrôlé par un seuil donné qui affecte finalement le nombre de points pivots. Étant donné un point, le sous-espace dominant peut être fusionné à partir de plusieurs points pivots, c'est-à-dire un ensemble de points skyline, puisque tous les points pivots sont des points skyline, et nous appelons un tel sous-espace fusionné un *sous-espace dominant maximum*, où le terme *maximum* signifie le nombre maximal de dimensions dans lesquelles le point donné domine les points pivots.

Définition 3. *Soit S un ensemble de points skyline de $\mathcal{P}$ dans l'espace D. Pour un point $q \in \mathcal{P}$, l'union des sous-espaces dominants $D_{q \prec S} = \bigcup_{p \in S} D_{q \prec p}$ est le* sous-espace dominant maximum *de q.*

Avec la définition ci-dessus, nous avons les extensions suivantes du Lemme 1 et du Lemme 2, qui constituent la base de nos résultats.

Lemme 3. *Soient S un ensemble de points skyline de $\mathcal{P}$ et $q_1, q_2 \in \mathcal{P}$ deux points tels que $q_1, q_2 \notin S$, $S \not\prec q_1$, $S \not\prec q_2$, et $q_1 \neq q_2$. $q_1 \nsim q_2$ si $D_{q_1 \prec S} \not\subseteq D_{q_2 \prec S}$ et $D_{q_2 \prec S} \not\subseteq D_{q_1 \prec S}$.*

Lemme 4. *Soit S un ensemble de points skyline de $\mathcal{P}$ et $q_1, q_2 \in \mathcal{P}$ deux points tels que $q_1, q_2 \notin S$, $S \not\prec q_1$, $S \not\prec q_2$, et $q_1 \neq q_2$. $D_{q_1 \prec S} \not\supseteq D_{q_2 \prec S} \Rightarrow q_1 \not\prec q_2$.*

En pratique, il est difficile de déterminer le nombre de points pivots car une valeur optimale dépend de nombreux facteurs, y compris le nombre total de points skyline qui doit être considéré comme inconnu. Évidemment, trop peu de points pivots ne peuvent pas distribuer tous les points dans un grand nombre de sous-espaces, mais trop de points pivots ralentiront clairement notre méthode. En plus, nous utilisons un processus basé sur le tri pour sélectionner les points pivots et fusionner les sous-espaces dominants, le sous-espace dominant maximum étant

construit de manière itérative car chaque point skyline peut attribuer un sous-espace dominant à un point dont tous les points dominés seront éliminés.

À chaque itération, nous déterminons la variation du nombre de points pour chaque taille de sous-espace, c'est-à-dire le nombre de points dans le même sous-espace avec la même taille. Soit D_q le sous-espace dominant maximum d'un point q, alors, pour chaque point q, le sous-espace D_q peut être modifié par le point pivot actuel p en $D_q \cup D_{q \prec p}$. Ainsi, nous proposons une mesure heuristique, le *seuil de stabilité*, pour arrêter la fusion des sous-espaces dominants. Le seuil de stabilité est le nombre de tailles de sous-espaces qui ne changent pas lors des itérations, ce qui signifie qu'aucun nouveau point pivot n'est nécessaire pour continuer à modifier les sous-espaces dominants maximums. Pour les petits jeux de données, la sélection du seuil de stabilité est moins importante, car tout algorithme de skyline peut s'exécuter en peu de temps ; pour les grands jeux de données, le seuil de stabilité peut être testé à partir d'un échantillon aléatoire du jeu. Nous notons également que toute mesure peut être appliquée pour arrêter la fusion des sous-espaces dominants.

4 Requête de sous-ensemble pour l'indexation des skylines

Dans la section précédente, nous avons montré qu'un sous-espace dominant maximal peut être attribué à chaque point, ce principe peut être utilisé pour partitionner et indexer les points skyline afin de réduire les tests de dominance. Le Lemme 4 montre que $D_{q_1 \prec S} \not\supseteq D_{q_2 \prec S}$ est nécessaire pour déterminer si $q_1 \prec q_2$ lorsque q_1 et q_2 ont été attribués à un sous-espace généré à partir d'un ensemble de points skyline S.

Lemme 5. *Soit S un ensemble de points skyline de $\mathcal{P}$ et $D_{q \prec S}$ le sous-espace dominant superposé d'un point $q \in \mathcal{P}$, où tous les points dominés par S ont été éliminés. Soit p un point skyline dans $\mathcal{P}$, alors les tests de dominance pour déterminer si un point $q \in \mathcal{P}$ est un point skyline peuvent être effectués uniquement avec chaque point skyline p tel que $D_{p \prec S} \supseteq D_{q \prec S}$, sous la condition de pré-tri.*

Notre résultat montre que le partitionnement des points skyline basé sur les sous-espaces peut réduire considérablement les tests de dominance : un point à tester doit être comparé uniquement avec les points skyline ayant le sous-espace dominant maximal spécifié dans le Lemme 5. Par conséquent, si les points skyline peuvent être stockés et récupérés à l'aide d'un conteneur générique, ce conteneur peut alors être utilisé pour améliorer n'importe quel algorithme de skyline en réduisant le nombre total de tests de dominance. Ainsi, nous avons les énoncés de deux problèmes suivants.

1. Concevoir une structure de données avec :
   (a) chaque point skyline peut être stocké/indexé et partitionné par son sous-espace dominant maximal ;
   (b) étant donné un sous-espace D_q d'un point de test q, tous les points skyline avec un sous-espace $D' \supseteq D_q$ peuvent être retournés efficacement.
2. Étant donné un ensemble $\mathcal{X}$ de M sous-ensembles d'un univers D, pour tout ensemble de requête $Q \subseteq D$, retourner l'ensemble $\{Q' \in \mathcal{X} \mid Q \subset Q'\}$.

Soit $D_q^{\neg}$ le sous-espace dominant maximal inversé par rapport à D d'un point q, alors le problème 1 ci-dessus consiste à trouver tous les sous-ensembles de $D_q^{\neg}$ afin de récupérer les

points skyline associés comme le problème 2. En effet, de nombreuses études ont été faites au problème de la requête de sous-ensembles. Le résultat récent Charikar et al. (2002) montre que la requête de sous-ensembles peut être accomplie en un temps de $\mathcal{O}((dM)/c)$ dans notre contexte, où d est la dimensionnalité, M est le nombre de sous-espaces dominants maximal, et $c \leq M$ est une constante. Nous proposons une structure de type arbre préfixe basée sur la table de hachage pour résoudre ce problème de requête de sous-ensemble en un temps moyen de $\mathcal{O}(d/2)$ pour ajouter un point et en un temps moyen de $\mathcal{O}((d/2)^2)$ pour une requête, sachant que $d \ll M$.

Avec notre structure proposée, l'exécution en temps pour insérer un point peut se terminer en temps $\mathcal{O}(1)$ dans le meilleur des cas, en temps $\mathcal{O}(d-1)$ dans le pire des cas et en temps $\mathcal{O}(d/2)$ en moyenne ; la récupération d'un ensemble de points skyline comparables peut se terminer en temps $\mathcal{O}(1)$ dans le meilleur des cas, en temps $\mathcal{O}((d-1)(d-2)/2)$ dans le pire des cas et en temps $\mathcal{O}((d/2)(d/2-1)/2)$ en moyenne donc $\mathcal{O}((d/2)^2)$. Nous notons que la dimensionnalité d de l'ensemble de points est bien inférieure à la cardinalité N de l'ensemble de points, le temps moyen de requête de sous-ensemble de $\mathcal{O}((d/2)^2)$ peut donc être négligé en comparaison en temps de $\mathcal{O}(dN^2)$ ou de $\mathcal{O}(N \log^{(d-2)} N)$. Ainsi, la meilleure application de notre méthode est de renforcer les algorithmes de skyline basés sur le tri.

5 Évaluation Expérimentales

Nous avons appliqué notre méthode aux algorithmes référencés `SFS`, `SaLSa` et `SDI` sans modifier leurs conceptions originales, la fonction principale de notre méthode étant de stocker et de récupérer les points skyline. Le code [2] a été implémenté en C++ et tous les résultats sont basés sur le temps processeur moyen de 10 exécutions, et toutes les données ont été chargées dans la mémoire principale avant le début du comptage [3].

Méthode	2-D	4-D	6-D	8-D	12-D	16-D	24-D
`SFS` AC	× 2.26	—	× 1.85	× 4.21	× 5.23	× 11.15	× 29.77
`SFS` CO	× 10.16	× 10.23	× 7.49	× 4.70	× 1.57	—	—
`SFS` UI	× 6.81	× 3.22	× 1.20	× 1.82	× 4.15	× 2.57	× 41.87
`SaLSa` AC	× 1.83	—	—	× 1.55	× 1.86	× 3.01	× 23.90
`SaLSa` CO	× 12.25	× 7.45	× 6.92	× 4.77	× 2.01	—	× 1.14
`SaLSa` UI	× 6.87	× 2.62	× 1.22	× 1.51	× 1.78	—	× 20.63
`SDI` AC	× 3.07	—	× 4.75	× 5.21	× 3.53	× 4.93	× 5.50
`SDI` CO	× 15.44	× 23.08	× 34.25	× 27.89	× 13.23	× 4.60	× 1.19
`SDI` UI	× 15.40	× 7.41	× 3.18	× 2.92	× 9.79	× 1.65	× 6.13
Meilleur AC	14.46 ms	149.11 ms	5897.42 ms	5197.69 ms	93603.0 ms	98041.8 ms	9632.24 ms
Meilleur CO	2.01 ms	2.84 ms	3.95 ms	6.43 ms	20.32 ms	95.69 ms	892.26 ms
Meilleur UI	2.33 ms	9.69 ms	59.21 ms	337.97 ms	2447.71 ms	31524.0 ms	9872.25 ms

TAB. 1 – *Gain de performance sur le temps d'exécution sur les jeux de données synthétiques de 200K de types AC, CO et UI selon la dimensionnalité.*

En tant que résumé étendu, nous présentons uniquement les améliorations des algorithmes référencés ainsi que les meilleurs temps d'exécution sur les jeux de données synthétiques AC

2. `https://github.com/dominiquehli/skyline-subset`

3. L'auteur a bénéficié de l'utilisation du cluster au Centre de Calcul Scientifique en Région Centre-Val de Loire, France.

Méthode	200K	400K	600K	700K	800K	900K	1000K
SFS AC	× 4.21	× 7.35	× 4.92	× 9.66	× 9.59	× 8.89	× 7.24
SFS CO	× 4.70	× 4.88	× 5.81	× 5.79	× 6.22	× 3.68	× 5.31
SFS UI	× 1.82	× 2.31	× 2.32	× 1.88	× 2.27	× 1.60	× 1.87
SaLSa AC	× 1.55	× 6.94	× 3.32	× 5.34	× 5.02	× 3.93	× 3.02
SaLSa CO	× 4.77	× 7.05	× 6.25	× 6.85	× 6.81	× 6.68	× 6.90
SaLSa UI	× 1.51	× 1.78	× 1.82	× 1.54	× 1.49	× 1.84	× 1.91
SDI AC	× 5.21	× 7.17	× 8.42	× 6.72	× 7.31	× 5.50	× 4.85
SDI CO	× 27.89	× 28.78	× 30.78	× 30.31	× 28.71	× 23.44	× 29.02
SDI UI	× 2.92	× 3.91	× 5.38	× 4.05	× 3.92	× 4.04	× 3.93
Meilleur AC	5197.69 ms	11830.0 ms	23743.9 ms	37005.5 ms	43266.8 ms	70195.4 ms	94795.2 ms ms
Meilleur CO	6.39 ms	8.45 ms	14.60 ms	15.08 ms	17.57 ms	20.18 ms	21.30 ms
Meilleur UI	337.97 ms	481.07 ms	546.61 ms	898.93 ms	1049.89 ms	1082.65 ms	1220.89 ms

TAB. 2 – *Gain de performance sur le temps d'exécution sur les jeux de données synthétiques de 8-D de types AC, CO et UI selon la cardinalité.*

Méthode	House	NBA	Weather
SFS	×1.22 298.15 ms → 245.35 ms	×1.22 29.17 ms →23.84 ms	×1.0001 43279.4 ms → 43231.2 ms
SaLSa	– 211.2 5 ms → 249.89 ms	×1.07 26.01 ms → 24.24 ms	– 14349.4 ms → 26169.5 ms
SDI	×1.92 214.72 ms → 111.7 ms	×1.04 24.49 ms → 22.38 ms	×1.04 10653.4 ms → 10617.3 ms

TAB. 3 – *Gain de performance sur le temps d'exécution sur les jeux de données réels.*

(Anti-Corrélés), CO (Corrélés), UI (Uniformément Indépendants) et des jeux de données réels (House, NBA et Weather)[4]. Le gain de performance est calculé comme le ratio entre toute valeur obtenue par les algorithmes référencés sans optimisation et la valeur de ceux optimisé par notre méthode. S'il n'y a pas de gain de performance, nous le marquons par –. La table 1 montre les gains de performance en fonction de la cardinalité fixée à 200K en variant la dimensionnalité jusqu'à 24-D ; la table 2 montre les gains de performance en fonction de la dimensionnalité fixée à 200K en variant la cardinalité jusqu'à 1000K ; et la table 3 montre les gains de performance sur les jeux de données réels.

6 Conclusions

Dans cet article, nous présentons une approche de requête de sous-ensemble pour un calcul efficace du skyline, conçue pour améliorer les algorithmes de skyline basés sur le tri en tant que bibliothèque logicielle externe. Notre analyse théorique et nos résultats expérimentaux montrent que le calcul du skyline peut être amélioré par notre méthode d'union de sous-espaces et de requête de sous-ensemble. Les perspectives du travail présenté dans cet article incluent : (1) l'extension de la méthode proposée aux données AC et CO ; (2) la conception d'un modèle de coût visant à améliorer le seuil de stabilité pour déterminer le meilleur nombre de points pivots ; (3) l'adaptation de la méthode proposée à des données dynamiques, telles que les flux de données.

4. https://github.com/dominiquehli/skyline-datasets

Références

Bartolini, I., P. Ciaccia, et M. Patella (2008). Efficient sort-based skyline evaluation. *ACM Transactions on Database Systems (TODS) 33*(4), 1–49.

Borzsony, S., D. Kossmann, et K. Stocker (2001). The skyline operator. In *ICDE*, pp. 421–430.

Charikar, M., P. Indyk, et R. Panigrahy (2002). New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *International Colloquium on Automata, Languages, and Programming*, pp. 451–462. Springer.

Chomicki, J., P. Godfrey, J. Gryz, et D. Liang (2005). Skyline with presorting : Theory and optimizations. In *Intelligent Information Processing and Web Mining*, pp. 595–604. Springer.

Godfrey, P., R. Shipley, et J. Gryz (2007). Algorithms and analyses for maximal vector computation. *The VLDB Journal 16*(1), 5–28.

Kossmann, D., F. Ramsak, et S. Rost (2002). Shooting stars in the sky : An online algorithm for skyline queries. In *VLDB*, pp. 275–286.

Lee, J. et S.-w. Hwang (2014). Scalable skyline computation using a balanced pivot selection technique. *Information Systems 39*, 1–21.

Lee, K. C., W.-C. Lee, B. Zheng, H. Li, et Y. Tian (2010). Z-sky : an efficient skyline query processing framework based on z-order. *The VLDB Journal 19*(3), 333–362.

Liu, R. et D. Li (2020). Efficient skyline computation in high-dimensionality domains. In *EDBT*, pp. 459–462.

Morse, M., J. M. Patel, et H. V. Jagadish (2007). Efficient skyline computation over low-cardinality domains. In *VLDB*, pp. 267–278.

Pei, J., W. Jin, M. Ester, et Y. Tao (2005). Catching the best views of skyline : A semantic approach based on decisive subspaces. In *VLDB*, pp. 253–264.

Tan, K.-L., P.-K. Eng, B. C. Ooi, et al. (2001). Efficient progressive skyline computation. In *VLDB*, Volume 1, pp. 301–310.

Zhang, S., N. Mamoulis, et D. W. Cheung (2009). Scalable skyline computation using object-based space partitioning. In *SIGMOD*, pp. 483–494.

Summary

Skyline query processing is essential for the database community. Many algorithms have been designed to reduce dominance tests between multidimensional points to achieve efficient computation, where incomparability between points has been considered an important property to avoid unnecessary dominance tests. We present a new approach based on subset queries to integrate a pre-check of incomparability for sorting-based skyline algorithms, which can significantly reduce the number of dominance tests in order to accelerate existing algorithms in large datasets.