Calcul efficace du skyline basé sur l'indexation dimensionnelle

Rui Liu*, Dominique 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 vise à trouver l'ensemble exact des tuples dominants parmi un ensemble de données multidimensionnelles, pour lequel de nombreux algorithmes ont été développés lors des deux dernières décennies. Cependant, la plupart des algorithmes existants ont été conçus et optimisés pour les données de faible dimensionnalité. Dans cet article, nous proposons SDI, un algorithme progressif basé sur l'indexation dimensionnelle pour le calcul du skyline. Notre analyse théorique et nos expériences montrent que SDI surpasse les algorithmes de calcul du skyline de l'état-de-l'art tant sur des données de haute dimensionnalité que sur des données de faible dimensionnalité.

1 Introduction

Le calcul du skyline vise à trouver l'ensemble complet de tuples dominants à partir des données multidimensionnelles. La figure 1 illustre un exemple du skyline de la relation prix-distance, où l'on considère les prix (axe `Y`) d'hôtels en fonction de leurs distances (axe `X`) à un endroit (le centre-ville, la plage ou la gare). Dans cet exemple, 6 hôtels sont représentés sous forme de points 2D dans lesquels $\{a, c, f\}$ forment le skyline puisque leurs valeurs, tant en prix qu'en distance, ne sont pas moins meilleures que les autres (*meilleur* est synonyme de plus petit dans notre exemple). Par exemple, l'hôtel a est meilleur que l'hôtel b, l'hôtel f est meilleur que les hôtels $\{d, e\}$ et l'hôtel c est incomparable avec tous les autres hôtels.

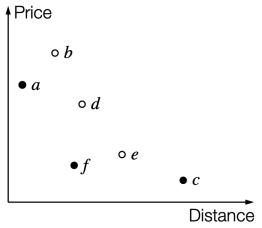


FIG. 1 – *Un example de skyline.*

Afin de réaliser les calculs du skyline, de nombreux algorithmes ont été développés au cours de plusieurs décennies. Cependant, la plupart des algorithmes existants ont été conçus et

optimisés pour les données de faible dimensionnalité en raison de la complexité temporelle en $\mathcal{O}(dN^2)$ dans le pire des cas [Godfrey *et al.*, 2005; Sheng and Tao, 2012], où d et N représentent respectivement la dimensionnalité et la cardinalité des données. Fondé sur une analyse théorique des précédentes méthodes, [Sheng and Tao, 2012] a proposé le premier algorithme qui calcule le skyline en temps $\mathcal{O}(N \log^{max(1,d-2)} N)$ dans le pire des cas, en utilisant des partitions dimensionnelles de données. L'algorithme BSkyTree [Lee and Hwang, 2014], basé sur ces techniques de partitionnement, nous sert de référence de l'état-de-l'art pour le calcul du skyline.

Dans cet article, nous présentons un algorithme progressif SDI (Sorted Dimension Index) pour le calcul du skyline, basé sur l'indexation dimensionnelle, qui est efficace dans les données de haute dimension ainsi que dans les données de faible dimension. En triant et en indexant toutes les dimensions, nous montrons qu'il est suffisant de comparer un tuple avec des tuples skyline existants sur une dimension arbitraire plutôt qu'avec l'ensemble complet de tous les tuples skyline dans les tests de dominance. Cette propriété réduit considérablement le nombre total de tests de dominance pendant le calcul du skyline, basé sur laquelle l'algorithme SDI est effectivement proposé. En effet, SDI crée d'abord un index constitué des sous-indexes triés par chaque dimension, puis parcourt tous les sous-indexes selon une stratégie de largeur en premier pour sortir immédiatement des tuples skyline. Notre analyse théorique montre que la complexité temporelle de SDI dans le pire des cas est meilleure que $\mathcal{O}(N \log^{max(1,d-2)} N)$ dans les domaines de haute dimension. Notre évaluation expérimentale montre en outre que SDI surpasse BSkyTree pour les données de haute dimension et est comparable à ce dernier pour les données de faible dimension.

Le reste de cet article est organisé comme suit. La section 2 présente la base de l'indexation des dimensions ainsi que l'algorithme SDI. Dans la section 3, nous montrons une analyse théorique sur le complexité temporelle de SDI, et ensuite la section 4 présente l'évaluation des performances de SDI sur des données synthétiques et réelles. Enfin, nous concluons à la section 5.

2 Algorithm SDI pour le calcul du skyline

Soit t un tuple d-dimensionnels, on note $t[i]$, où $1 \leq i \leq d$, la *valeur dimensionnelle* du tuple t sur la dimension i. Nous définissons l'*ordre de préférence* $\prec$ comme un ordre total sur toutes les dimensions, alors étant donné deux tuples t et u, on dit $t[i]$ est meilleur que $u[i]$ si $t[i] \prec u[i]$; $t[i]$ est identique à $u[i]$ si $t[i] = u[i]$; $u[i]$ n'est pas pire que $t[i]$ si $t[i] \prec u[i] \vee t[i] = u[i]$, noté $t[i] \preceq u[i]$. Nous avons $t[i] \prec u[i] \Rightarrow t[i] \preceq u[i]$. Nous notons $t \prec u$ que t *domine* u, si et seulement 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 $t[k] \prec u[k]$. Nous notons $t \not\prec u$ que t *ne domine pas* u, et $t \nsim u \iff t \not\prec u \wedge u \not\prec t$ que t et u sont *incomparables*. Afin de simplifier notre description formelle, nous étendons les relations $\{\prec, \not\prec, \preceq, \nsim\}$ à des ensembles de tuples tel que, par exemple, $t \prec X$ indique que le tuple t domine chaque tuple dans l'ensemble X, $t \nsim X$ indique que t est incomparable avec chaque tuple dans X, etc.

Étant donné une base de données d-dimensionnelles $\mathcal{D}$, un tuple $t \in \mathcal{D}$ est un *tuple skyline* si et seulement si $\nexists u \in \mathcal{D}$ tel que $u \prec t$. Le *skyline* de la base de données $\mathcal{D}$, noté $SKY(\mathcal{D})$, est l'ensemble complet de tous les tuples skyline, c'est-à-dire $\mathcal{SKY}(\mathcal{D}) = \{t \in \mathcal{D} \mid \nexists u \in \mathcal{D}, u \prec t\}$. En d'autres termes, le skyline d'une base de données est l'ensemble complet de

tous les tuples incomparables par rapport à un ordre de préférence $\prec$, de sorte que pour deux tuples skyline s et t, nous avons $s \nsim t$.

Definition 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 ordonnées 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 dimensionnelle unique sur la dimension i et contient un ou plusieurs ID de tuple triés par l'ordre lexicographique.*

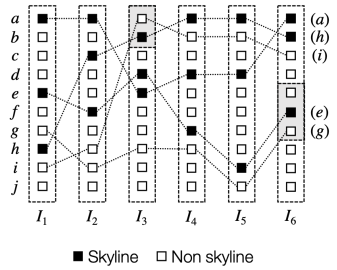


FIG. 2 – *Un exemple de l'index dimensionnel.*

Considérons l'index dimensionnel présenté dans la figure 2, qui consiste en 6 sous-indexes $I_1, I_2, \ldots, I_6$ et 10 tuples. Notez que pour simplifier notre illustration, nous ne concernons que 5 tuples $\{a, e, g, h, i\}$. Chaque unité indiquée dans un sous-index représente un ID de tuple unique, où les éléments noirs représentent les tuples skyline et les blancs représentent les tuples non-skyline (toutes les valeurs dimensionnelle étant triées, nous nous intéressons à l'ordre des tuples dans une dimension, mais pas à la valeur précise). Par défaut, chaque ID correspond à un *bloc* individuel, ce qui signifie qu'il n'y a pas de tuples avec des valeurs dimensionnelles identiques ; cependant, I_3 et I_6 contiennent deux blocs (en gris) comportant différents ID de tuple, ce qui signifie que deux tuples (h et i) ont la même valeur sur la dimension 3 et trois tuples (e, g et un autre non concerné) ont la même valeur sur la dimension 6.

Dans la figure 2, a est immédiatement un tuple skyline, car aucun tuple n'est meilleur que a au moins sur la dimension 1, aucun tuple ne peut donc le dominer. Dans notre exemple, nous avons $a \prec \{b, c, d\}$, donc si nous continuons avec I_1 pour vérifier tous les autres tuples, nous aurons l'observation suivante : puisque a est déjà un tuple skyline, il est impossible que $b \prec a$, donc une comparaison de dominance à sens unique $a \prec b$ est suffisante pour déterminer si b est un tuple skyline, c'est-à-dire, $a \prec b \Rightarrow b \notin SKY(\mathcal{D})$ ou $a \not\prec b \Rightarrow b \in SKY(\mathcal{D})$, où $SKY(\mathcal{D})$ est le skyline de la base de données exemple notée $\mathcal{D}$. De même, pour déterminer si c est un tuple skyline, puisque $a \in SKY(\mathcal{D})$ et $b \notin SKY(\mathcal{D})$, nous avons $a \prec c \Rightarrow c \notin SKY(\mathcal{D})$ ou $a \not\prec c \Rightarrow c \in SKY(\mathcal{D})$, il n'est donc pas du tout nécessaire de comparer b et c. C'est aussi le cas de d. Par conséquent, pour déterminer si e est un tuple skyline, il suffit de tester $a \prec e$, et le résultat indique que $a \not\prec e$, donc e peut être confirmé en tant que tuple skyline sans tester $e \prec a$.

En considérant le cas où il existe plusieurs ID de tuples dans un bloc, nous pouvons généraliser la procédure ci-dessus pour tout sous-index. Les *tuples skyline local* du bloc basés sur les tels blocs doivent être obtenus localement et doivent ensuite être comparés aux tuples

skyline du sous-index, tandis que pour les blocs ne contenant qu'un seul ID de tuple, le tuple est naturellement un tuple skyline local.

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

Démonstration. Si t apparaît dans le premier bloc du I_i, alors t est un tuple skyline, car aucun tuple n'est meilleur que t. Sinon, supposons que s soit un tuple skyline tel que $s[i] \prec t[i]$, si $s \prec t$, alors t n'est pas un tuple skyline puisque s domine t; si $s \not\prec t$, alors il existe au moins une dimension $k \neq i$ telle que $t[k] \prec s[k]$, donc aucun tuple dominé par s ne domine t. En conséquence, si pour tout s tel que $s[i] \prec t[i]$ nous avons $s \not\prec t$, alors aucun tuple dans la base de données ne domine t, donc t est incomparable à tout tuple skyline, donc t est un tuple skyline. Dans le cas $t[i] \prec s[i]$, alors $s \not\prec t$, il est donc inutile de comparer t avec s. □

Ainsi, Theorem 1 permet de déterminer un tuple skyline dans un sous-ensemble borné au lieu du skyline complet lors des tests de dominance. De plus, une fois que l'index dimensionnel a été construit, Theorem 1 permet également de basculer entre les sous-indexes, ce qui peut réduire considérablement le nombre total de tests de dominance et le temps pour calculer le skyline. Le principe simple de SDI est présenté dans Algorithme 1.

Algorithme 1 : SDI

```
Input : Dimension index I
Output : Skyline S
1  while true do
2      Get the best dimension I_best ∈ I with min(S_best)
3      while B ← Get next block from I_best do
4          if B = null then
5              return S
6          S_B ← Compute block skyline from B
7          foreach t ∈ S_B and t ∉ S do
8              if S_best ⊀ t then
9                  S_best ← S_best ∪ t
10                 S ← S ∪ t
11             if Found new skyline tuples then
12                 break
```

Soient $\mathcal{D}$ une base de données d-dimensionnelles et $p \in \mathcal{D}$ un tuple skyline. Une *ligne d'arrêt* (*stop line*) établie à partir de p, notée L_p, est un ensemble d'entités $\bigcup_{1 \leq i \leq d} p[i]$ sur chaque dimension. Pour tout tuple skyline p, tout tuple t tel que $p \prec t$ peut être élagué et le calcul peut être terminé en toute sécurité si SDI atteint p sur chaque dimension.

Theorem 2. *Etant donné une base de données d-dimensionnelles $\mathcal{D}$, prenons L_p comme la ligne d'arrêt par rapport à un tuple skyline p. En suivant une traversée de haut en bas sur*

toutes les dimensions, si tous les blocs contenant la ligne d'arrêt ont été atteints, l'ensemble complet de tous les tuples skyline a été extrait et le calcul peut être terminé.

Démonstration. Soit p un tuple skyline et $t \in SKY(\mathcal{D}) \setminus p$ un autre tuple skyline, nous avons (1) $t \nsim p$ ou (2) $t = p$ dans le cas de tuples en double, si t et p ont des valeurs dimensionnelles identiques. Soit $o_k(t)$ la position du bloc contenant tuple t sur la dimension k, alors dans le cas (1), $t \nsim p \Rightarrow \exists k, p[k] \prec t[k] \Rightarrow o_k(p) < o_k(t)$, c'est-à-dire si la traversée d'index passe la ligne d'arrêt L_p, le tuple t doit avoir été identifié au moins dans la dimension k ; dans le deuxième cas, nous avons $o_i(p) = o_i(t)$ pour toute dimension i. Dans les deux cas, si tous les blocs de lignes d'arrêt ont été traités, tous les tuples skyline ont été trouvés. □

En principe, tout tuple skyline peut être choisi pour être une ligne d'arrêt, cependant, différentes lignes d'arrêt se comportent différemment dans l'élagage des tuples dominé. Nous proposons une fonction optimale min_{stop} pour trouver la meilleure ligne d'arrêt L_p^*, définie comme :

$$L_p^* = min_{stop}(p) = \arg\min_p(max\{o_i(p)\}, \sum_{i=1}^{d} o_i(p)).$$

La fonction min_{stop} trie d'abord le décalage maximum, puis le décalage moyen dans toutes les dimensions, de sorte que le tuple skyline réduit constitue le meilleur tuple de ligne d'arrêt. Par conséquent, un tuple de ligne d'arrêt p mis à jour dynamiquement peut être conservé par $min_{stop}(p) < min_{stop}(t)$ pour tout nouveau tuple skyline t. Très différent de la notion de *point d'arrêt* (*stop point*) dans SaLSa [Bartolini *et al.*, 2008], dans notre approche, tout tuple peut effectivement arrêter l'exécution de SDI, et une ligne d'arrêt *best* est toujours disponible.

3 Analyse théorique

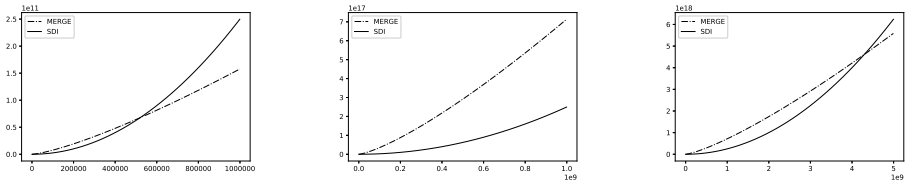
Dans notre analyse théorique de la complexité temporelle dans le pire des cas, nous notons d la dimensionnalité et N la cardinalité des données. Nous considérons le test de dominance entre un tuple skyline s et un tuple de test t comme $\mathcal{O}(d)$ pour des donnes de d dimensions, ce qui implique $(s \not\prec t)$ et $(t \not\prec s)$, alors le test de dominance dans SDI est sous $\mathcal{O}(d/2)$ puisque $(s \not\prec t)$ est suffisant. Nous notons également que la construction de l'index dimensionnel nécessite le temps $\mathcal{O}(dN \log N)$ par rapport aux algorithmes généraux de tri $\mathcal{O}(N \log N)$ sur toutes les d dimensions, donc l'analyse suivante concerne uniquement le temps de requête skyline. De plus, notre analyse ne concerne pas la ligne d'arrêt car elle n'a aucun sens dans le pire des cas.

Theorem 3. *Dans le pire des cas, tous les N tuples sont des tuples skyline. SDI calcule le skyline sous*

$$\mathcal{O}(dN \log N + \frac{N^2 - dN}{4}).$$

Démonstration. Dans le pire des cas, le nombre N/d de tuples en haut de chaque dimension doit être différent, sinon il doit y avoir des tuples dominés par d'autres. SDI teste donc N/d tuples sur chaque dimension car les tuples testés pourront être ignorés, c'est-à-dire pour chaque dimension, $\mathcal{O}((N/d)(N/d - 1)/2)$ tests sont requises dont chaque test nécessite le temps

$\mathcal{O}(d/2)$. Alors pour d dimensions, le temps de calcul de SDI est borné en $\mathcal{O}(((N/d)(N/d-1)/2)(d/2)d) = \mathcal{O}((N^2 - dN)/4)$. □



(a) $d = 6$, $N = 1M$. (b) $d = 8$, $N = 1000M$. (c) $d = 8$, $N = 5000M$.

FIG. 3 – *Simulation numérique pour étude de complexité dans le pire des cas.*

En comparaison avec la meilleure complexité temporelle $\mathcal{O}(N \log^{max(1,d-2} N)$ connue sur le calcul de skyline dans le pire des cas (Merge officieusement, [Sheng and Tao, 2012]), SDI fonctionne mieux dans des domaines de haute dimensionnalité en respectant une cardinalité raisonnable de données. Au fait, étant donné une dimensionnalité fixe $d > 2$, afin de déterminer la cardinalité des données N sur laquelle SDI est meilleur que Merge, l'équation suivante doit être résolue :

$$N \log^{d-2} N > dN \log N + \frac{N^2 - dN}{4}.$$

De toute évidence, l'équation ci-dessus appartient aux équations transcendantales qui n'ont pas de solutions de forme fermée. Notre simulation numérique, comme dans la figure 3, montre que dans le pire des cas, tandis que $d = 6$, SDI est meilleur que Merge jusqu'à $N \approx 550$K ; et tandis que $d = 8$, Merge bat SDI seulement si $N > 4000$M.

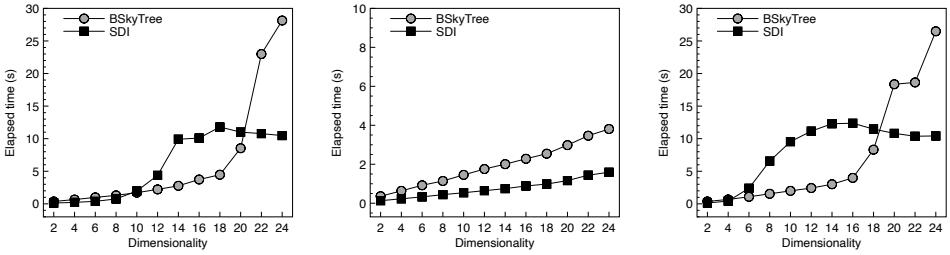
Notons qu'afin de simplifier notre analyse, nous ne discutons que le cas sans valeurs répétées sur chaque dimension, c'est-à-dire que chaque bloc d'index ne contient qu'un seul ID de tuple. En effet, si K valeurs répétées sont présentes dans un bloc, le temps $\mathcal{O}(dK^2)$ sera ajouté à un tel bloc si le plus simple algorithm BNL [Borzsony *et al.*, 2001] est appliqué pour calculer le skyline local du bloc.

4 Évaluation expérimentale

Dans cette section, nous rapportons notre évaluation expérimentale sur les performances de calcul du skyline de SDI [1] en comparaison avec BSkyTree [2]. Nous avons implémenté SDI en C++ et toutes les expériences ont été effectuées sur un processeur Intel Core i5 3,1 GHz avec 16 Go de RAM DDR3 1867 MHz sous le système d'exploitation macOS 10.14.5.

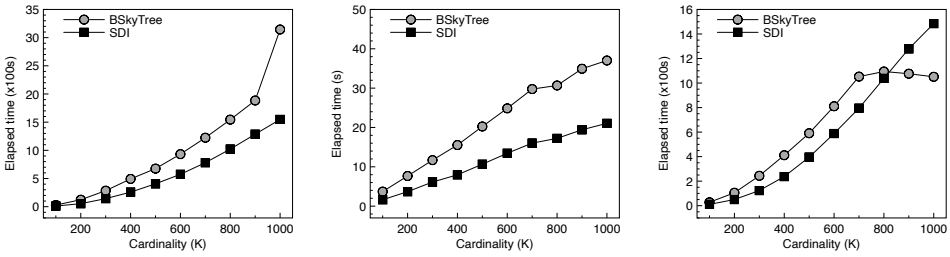
Nous considérons trois types de jeux de données synthétiques *uniform independent* (`UI`), *correlated* (`CO`) et *anti-correlated* (`AC`) générés par le Skyline Benchmark Data Generator [3] ; nous considérons en plus trois jeux de données réels `NBA`, `HOUSE` et `WEATHER` [Chester *et al.*, 2015]. En raison de l'espace limité de cet article, Nous rapportons uniquement le temps total d'exécution. Tous les résultats rapportés sont la performance moyenne sur 5 itérations.

1. `https://github.com/skyline-sdi/sdi-bench`
2. `https://github.com/sean-chester/SkyBench`
3. `http://pgfoundry.org/projects/randdataset`



(a) The UI dataset. (b) The CO dataset. (c) The AC dataset.

FIG. 4 – *Évaluation des performances sur l'effet de la dimensionnalité (N = 100K).*



(a) The UI dataset. (b) The CO dataset. (c) The AC dataset.

FIG. 5 – *Évaluation des performances sur l'effet de la cardinalité (d = 24).*

Les effets de **(1) dimensionnalité** et **(2) cardinalité** des données ont été évalués, comme montrés dans la figure 4 et la figure 5. Pour (1), la cardinalité est fixée à 100K ; pour (2), la dimensionnalité des données est fixée à 24. Nous remarquons que SDI surpasse BSkyTree en haute dimension ($d > 20$) qu'en faible dimension ($d < 10$) sur les jeux de type UI, et est moins performant que BSkyTree sur des donnes de type AC dans une large plage de dimensionnalité (cela confirme les résultats montrés par [Lee and Hwang, 2014]) du fait que l'effet de la ligne d'arrêt est très limité dans des données anti-corrélées. En revanche, grâce à la ligne d'arrêt, SDI surpasse systématiquement BSkyTree sur des données corrélées. En concernant les résultats sur les jeux de données réels montrés dans la table 1, SDI surpasse BSkyTree sur NBA et HOUSE, mais est beaucoup plus lent que BSkyTree sur WEATHER en raison du grand nombre de valeurs dimensionnelles répétées, où le facteur $\mathcal{O}(dK^2)$ (discuté dans la section 3) commence à ralentir SDI.

Dataset	d	N	$\|SKY\|$	SDI	BSkyTree
NBA	8	17264	1796	60ms	155ms
HOUSE	6	127931	5774	383ms	839ms
WEATHER	15	566268	26713	25132ms	11641ms

TAB. 1 – *Évaluation des performances sur les jeux de données réels.*

En résumé, notre évaluation montre les performances de SDI dans les domaines de haute

dimensionnalité et de faible dimensionalité, mais montre également sa limite par rapport à BSkyTree, en particulier dans les domaines de moyenne dimensionnalité.

5 Conclusion

Dans cet article, nous avons présenté un nouvel algorithme progressif SDI pour le calcul du skyline. Avec l'index dimensionnel, nous montrons que les tests de dominance requis par le calcul du skyline peuvent être bornés et tout tuple skyline peut être utilisé pour terminer le processus du calcul. Notre analyse théorique et notre évaluation montrent que SDI surpasse les algorithmes skyline de l'état-de-l'art tant sur des données de haute dimensionnalité que sur des données de faible dimensionnalité.. Notre future orientation de recherche comprend le calcul du skyline au contexte du Big Data y compris le calcul du skyline en sous-espaces.

Références

Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. Efficient sort-based skyline evaluation. *ACM Transactions on Database Systems*, 33(4) :31, 2008.

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

Sean Chester, Darius Šidlauskas, Ira Assent, and Kenneth S Bøgh. Scalable parallelization of skyline computation for multi-core processors. In *ICDE*, pages 1083–1094, 2015.

Parke Godfrey, Ryan Shipley, and Jarek Gryz. Maximal vector computation in large data sets. In *VLDB*, pages 229–240, 2005.

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

Cheng Sheng and Yufei Tao. Worst-case i/o-efficient skyline algorithms. *ACM Transactions on Database Systems*, 37(4) :26, 2012.

Summary

In this paper, we introduce a dimension indexing based algorithm SDI for efficient skyline computation. We show that the dominance tests required to determine a skyline tuple can be sufficiently bounded to a subset of current skyline and any skyline tuple can be used to terminate the computation. Our theoretical analysis and experimental evaluation shows that SDI outperforms the state-of-the-art skyline computation algorithms in high-dimensionality domains as well as in low-dimensionality domains.