Calculer et compresser le skycube négatif

Nicolas Hanusse, Patrick Kamnang Wanko, Sofian Maabout

Univ. Bordeaux - CNRS, LaBRI. France
{hanusse,pkamnang,maabout}@labri.fr

RÉSUMÉ. *Soit T une table avec D attributs. Le Skycube de T est l'ensemble de tous les skylines obtenus en considérant chacun des sous-ensembles de dimensions. Afin de réduire le temps de réponse des requêtes du skycube, les solutions de la littérature proposent soit des algorithmes efficaces pour le calcul du skycube complet, soit des techniques de compression du skycube dans le but de gagner en temps et en espace. Intuitivement, les précédents travaux avaient pour objectif de calculer ou de résumer l'information suivante : « pour chaque tuple t, la liste des skylines auxquels t appartient ». Dans ce travail, nous considérons l'information complémentaire, à savoir, « pour chaque tuple t, la liste des skylines auxquels t* n'appartient pas *». C'est pourquoi nous l'appelons le* Skycube Négatif. *En dépit de l'apparente équivalence entre ces deux informations, nos analyses et expérimentations montrent que ces deux points de vue ne conduisent pas à des comportements similaires des algorithmes associés. Plus spécifiquement, nos propositions montrent que (i) le résumé du* Skycube Négatif *peut être obtenu plus rapidement que les autre méthodes, (ii) en général, le* Skycube Négatif *requiert moins d'espace, (iii) les requêtes sont plus rapides suivant cette optique et (iv) le* Skycube Positif *peut être obtenu plus rapidement en calculant d'abord le* Skycube Négatif.

ABSTRACT. *Given a table T with D dimensions, the skycube of T is the union of all skylines obtained by considering each of the subsets of D (subspaces). The number of these skylines is exponential w.r.t D. To make the skycube practically useful, two lines of research have been pursued so far : the first one aims to propose efficient algorithms for computing it and the second one considers either that the skycube is too large to be computed in a reasonable time or it requires too much memory space to be stored. They therefore propose skycube summarization techniques to reduce time and space consumption. Intuitively, previous efforts have been devoted to compute or summarize the following information : "for every tuple t, list the skylines where t belongs to". In this paper, we consider the complementary statement, i.e., "for every tuple t, list the skylines where t* does not belong to*". This is what we call the* negative skycube. *Our proposal shows that (i) the negative summary can be obtained much faster than state of the art techniques for positive summaries, (ii) in general, it consumes less space, (iii) skyline queries evaluation using this summary are much faster and (iv) the positive skycube can be obtained much more rapidly than state of the art algorithms.*

MOTS-CLÉS : *requête skyline, algorithme, sous-espace, optimisation, k-dominant Skycube.*

KEYWORDS: *skyline query, algorithm, subspace, optimization, k-dominant.*

DOI:10.3166/ISI.22.3.9-33

Tableau 1. Les hôtels

Id	**(P)rix**	**(D)istance**	**(A)ire**	**(W)ifi**
h_1	100	10	20	No
h_2	10	100	20	Yes
h_3	100	10	25	Yes

1. Introduction

Les requêtes skyline sont un cas particulier de requêtes de préférence dont l'objectif est de retrouver un sous-ensemble de tuples qui ne sont pas moins bons que d'autres. Ce résultat s'appelle également *Optimum au sens de Pareto*. Dans un contexte multidimensionnel, l'utilisateur peut choisir un sous-ensemble d'attributs au regard desquels les tuples sont comparés. L'ensemble des requêtes skyline est appelé le skycube. En raison du nombre exponentiel de requêtes skyline, la matérialisation du skycube est chronophage et requiert beaucoup d'espace. Pour cette raison, deux directions de recherche ont été suivies depuis lors : d'une part, des algorithmes complexes pour accélerer le calcul du skycube ont été proposés, d'autre part, des techniques de compression (réduction de la taille) du skycube ont été définies pour réduire l'espace requis tout en conservant le temps de calcul acceptable. Considérant la réduction de taille, certains algorithmes supposent le skycube comme faisant partie de l'entrée tandis que d'autres présentent des procédures de compression partant de l'ensemble des données brutes.

Le présent travail se base sur l'observation suivante : affirmer qu'un tuple t fait partie du skyline nécessite qu'il soit comparé à l'ensemble des autres points. Cependant, la comparaison de t à un autre tuple t' nous permet de déduire un ensemble de sous-espaces pour lesquels t n'appartient pas au skyline. Partant de là, notre hypothèse de départ est alors que rechercher les sous-espaces au regard desquels t n'appartient pas au skyline peut se faire plus rapidement que le travail complémentaire. Une fois que cet ensemble de sous-espaces est déterminé, retrouver l'ensemble complémentaire est évident.

Sauvegarder pour chaque tuple t, l'ensemble des sous-espaces pour lesquels t n'appartient pas au skyline peut être impossible en pratique en raison de la grande quantité de mémoire qui doit être requise. C'est pourquoi, nous proposons une technique permettant de résumer cette information. L'exemple ci-dessous illustre notre approche.

EXEMPLE 1. — Considérons la liste d'hôtels présentée dans le tableau 1.

Nous avons une préférence pour les hôtels les moins coûteux, les plus près de la mer, présentant les chambres les plus grandes et disposant du Wifi. L'utilisateur a la possibilité de demander le skyline au regard de n'importe laquelle des $2^4 - 1$ requêtes correspondant aux sous-ensembles non vides de l'espace {**P**, **D**, **A**, **W**}. En comparant

les hôtels h_2 et h_1, nous déduisons que h_2 domine h_1 en chacun des sous-espaces $subspaces_2 = \{$**P**, **W**, **PW**, **AP**, **AW**, **APW**$\}$ [1]. Cependant, après cette comparaison, nous n'avons aucune information concernant les sous-espaces pour lesquels h_1 fait partie du skyline. Pour cela nous avons également besoin de comparer h_1 et h_3. Une fois la comparaison effectuée, nous déterminons un nouvel ensemble de sous-espaces dans lesquels h_1 est dominé c'est-à-dire $subspaces_3 =\{$**A**, **W**, **AW**, **AD**, **DW**, **AP**, **ADP**, **PW**, **ADW**, **APW**, **DPW**, **ADPW**$\}$. $subspaces_2 \cup subspaces_3$ est l'ensemble des sous-espaces dans lesquels h_1 est dominé. Son complémentaire est l'ensemble des sous-espaces pour lesquels h_1 fait partie du skyline c'est-à-dire $\{$**D**, **DP**$\}$. En pratique, le nombre de sous-espaces associés à un tuple peut être très grand. Nous proposons donc une technique permettant de résumer cet ensemble tout en conservant l'efficacité des requêtes. □

Nos contributions peuvent être résumées comme suit : (i) étant donné l'ensemble des sous-espaces au regard desquels un tuple n'appartient pas au skyline, nous présentons une structure de données appelée NSC qui résume le skycube négatif. (ii) Nous proposons une procédure prenant en entrée la table T et retourne le NSC correspondant. Enfin, (iii) nous présentons un ensemble de résultats d'expérimentations montrant les avantages et les limites de nos approches par rapport aux algorithmes de l'état de l'art.

Après une brève présentation des travaux se rapportant à cette problématique (section Travaux relatifs), nous définissons les principaux concepts et notations utilisés tout le long de cet article. Ensuite, nous présentons (section 3) la problématique du résumé du skyline négatif, nous élaborons les algorithmes permettant de résoudre ce problème. Enfin, quelques résultats d'expériences sont présentés (section 4).

Notations

Soient $T(Id, D_1, \ldots, D_d)$ une table et $\mathcal{D} = \{D_1, \ldots, D_d\}$ l'ensemble des dimensions de la table T. Chaque sous-ensemble non vide de $\mathcal{D}$ est un *sous-espace* (soit X un sous-ensemble de $\mathcal{D}$). Pour chaque dimension D_i, nous supposons établi un ordre total $<_i$ au sein de ses valeurs. Soient $v, v' \in D_i$, nous disons que v est *préféré* à v' ssi $v <_i v'$. Soient t et t' deux tuples de la table T, on dit que t est *dominé* par t' au regard du sous-espace X, noté $t' \prec_X t$, ssi $t'[D_i] \leq_i t[D_i]$ pour chaque $D_i \in X$ et il existe $D_j \in X$ tel que $t'[D_j] <_j t[D_j]$. Le skyline de T au regard du sous-espace X, noté $Sky_T(X)$ ou simplement $Sky(X)$, est l'ensemble des tuples de T qui ne sont pas dominés au regard de X. Le *skycube* de T est l'ensemble $\{Sky(X)|X \in 2^{\mathcal{D}}$ et $X \neq \emptyset\}$. Par conséquent, le skycube est composé de $2^d - 1$ skylines. Nous notons n le nombre de tuples appartenant à la table T (la taille de T).

Le tableau 2 résume les différents notations utilisées tout le long de cet article.

1. Nous verrons plus tard qu'il est facile d'obtenir cette information sans avoir à effectuer la comparaison au regard de chacun des sous-espaces.

Tableau 2. Notations

Notation	**Définition**
T, U	Instances de table (ensemble de tuples)
$\mathcal{D}$	Ensemble des dimensions
X, Y, Z	Sous-ensembles d'attributs (sous-espaces)
XY	Mis pour $X \cup Y$; X et Y sont des sous-espaces
$\|X\|$	Nombre d'attributs appartenant à X
$\langle X \| Y \rangle$	Paire de sous-espaces $X \neq \emptyset$ et Y
d	Nombre de dimensions ($d = \|\mathcal{D}\|$)
D_j	j^{eme} Dimension de T
n, m	Nombres de tuples
t, t', t_i, t_j	Tuples de T
$t_i[j]$	Projection du tuple t_i sur la j^e dimension
$t' \prec_X t$	t' domine t au regard de X
$Sky_T(X)$	Skyline de T au regard de X
$Topmost$	$Sky_T(\mathcal{D})$, Skyline de T au regard de $\mathcal{D}$
$\mathcal{DOM}(t)$	Sous-espaces dans lesquels t est dominé
$\mathcal{COMPARE}(t)$	Ensemble de paires de sous-espaces associés à t

EXEMPLE 2. — Le tableau 3 sert d'exemple tout le long de cet article. Dans cet exemple, l'utilisateur pourrait demander les *meilleurs* tuples au regard de chaque combinaison de dimensions $\{A, B, C, D\}$. Par exemple, $Sky(AB) = \{t_1, t_2\}$ et $Sky(ABCD) = \{t_2, t_3, t_4\}$.

Tableau 3. Une table de données T

Id	A	B	C	D
t_1	1	1	3	3
t_2	1	1	2	3
t_3	2	2	2	2
t_4	4	2	1	1
t_5	3	4	5	2
t_6	5	3	4	2

□

2. Travaux relatifs

Depuis son introduction à la communauté des chercheurs en bases de données par (Börzsönyi *et al.*, 2001), l'opérateur skyline fait l'objet d'un grand intérêt. Plusieurs al-

gorithmes l'implémentant ont été proposés dans la littérature, par exemple (Chomicki *et al*., 2003 ; Papadias *et al*., 2005 ; Bartolini *et al*., 2008). Au regard de nos lectures, BSkyTree (Lee, Hwang, 2010) est considéré comme étant l'algorithme le plus efficace pour une exécution séquentielle. Des algorithmes en parallèle ont également été proposés, par exemple (Chester *et al*., 2015). Certaines propositions considèrent des distributions particulières des données, par exemple, (Morse *et al*., 2007) proposent un algorithme très efficace dans le cas de dimensions présentant un nombre très faible de valeurs distinctes, tandis que (Shang, Kitsuregawa, 2013) présentent un algorithme ayant de bonnes performances lorsque les données sont anti-corrélées. D'autres travaux considèrent le problème de réduction de la taille du résultat d'une requête skyline en ne conservant que les tuples satisfaisant certaines contraintes, par exemple, les tuples appartenant à un nombre minimum de skyline de sous-espaces (Chan *et al*., 2006b) ou en sélectionnant les points skyline qui dominent un nombre maximal d'autres points (Lin *et al*., 2007).

Les travaux les plus proches du nôtre sont probablement *skyline group lattice* présenté dans (Pei *et al*., 2005 ; 2006) et le plus récent *compressed skycube* (CSC) dans (Xia *et al*., 2012). Ces deux travaux se proposent de résumer le skycube *positif*. Intuitivement, le concept de groupe skyline opère comme suit : chaque tuple est associé à un ensemble de paires $\langle S_1, S_2 \rangle$ où : S_1 est le *top* sous-espace X_t et S_2 est un ensemble de *bottom* sous-espaces $Y_{t_1}, \ldots, Y_{t_p}$. Ces sous-espaces sont choisis de telle façon que $t \in Sky(Z)$ pour tout $Z \subseteq X_t$ et il existe Y_{t_j} tel que $Y_{t_j} \subseteq Z$. Par exemple, supposons que la paire $\langle ABC, \{A, BC\} \rangle$ soit associée au tuple t. Alors nous pouvons déduire par exemple que $t \in Sky(AB)$. Le résumé consiste à associer chaque tuple au plus petit nombre possible de paires. Les requêtes skyline des sous-espaces telles que $Sky(Z)$ sont exécutées en parcourant pour chaque tuple, la liste qui lui est associée et en vérifiant que Z est bien compris entre un top sous-espace et un bottom sous-espace associé à ce dernier.

La technique de compression CSC quant à elle consiste à maintenir pour chaque sous-espace son *partial skyline* (skyline partiel). Plus précisement, un tuple t est ajouté au skyline partiel de Z ssi (i) $t \in Sky(Z)$ et il n'existe pas $Y \subset Z$ tel que $t \in Sky(Y)$. Lorsqu'une requête sur un espace $Sky(Z)$ est demandée, on procède à l'union de l'ensemble des skyline partiels des sous-espaces $Y \subseteq Z$ et une requête skyline classique est exécutée sur ce résultat intermédiaire. Dans (Xia *et al*., 2012), il est montré que CSC requiert moins d'espace que *skyline group*. Cependant, il nécessite encore l'exécution d'une requête skyline.

Dans (Raïssi *et al*., 2010), la structure *closed skycube* a été proposée. L'idée principale de cette technique est de regrouper les sous-espaces en classe d'équivalence au regard des résultats des requêtes skyline de façon à ne stocker qu'un seul exemplaire de l'ensemble des tuples. Même si cette solution présente un temps optimal de réponse aux requêtes, la recherche des classes d'équivalence requiert beaucoup de temps. D'autre part, la taille du *closed skycube* pourrait être égale à celle du skycube dans le cas où tous les skyline sont différents d'un espace à un autre.

Récemment, (Bøgh *et al.*, 2014) a proposé la structure Hashcube pour stocker l'ensemble du skycube. Intuitivement, cette méthode consiste à utiliser des vecteurs de bits pour encoder les sous-espaces pour lesquels les tuples appartiennent au skyline. Le temps de réponse de cette structure de données est impressionnant, en général il est seulement de 10% plus lent que le temps de réponse idéal. Par contre, il requiert beaucoup de mémoire, sa consommation de mémoire est de 10% la taille du skycube. Puisque la taille du skycube augmente de façon exponentielle avec le nombre de dimensions, la structure Hashcube sature rapidement la mémoire disponible. Plus important encore, les auteurs ne fournissent pas de procédure pour directement construire cette structure à partir des données. À la place, ils prennent en entrée le skycube, supposé construit, et encode celui-ci en la structure de données Hashcube; alors que la construction du skycube est une tâche requérant beaucoup de temps mais aussi de mémoire.

Dans le but de réduire la taille des requêtes skyline, la requête k-dominant skyline a été définie. Un tuple est k-dominé ($k > 0$) s'il existe un autre tuple et un sous-espace dont la taille est au moins k, dans lequel le premier est dominé par le second. Alors, le k-dominant skyline est l'ensemble de tuples qui ne sont pas k-dominés. Il existe plusieurs travaux autour des requêtes k-dominant skyline, certains d'entre-eux (Chan *et al.*, 2006a ; Siddique, Morimoto, 2009 ; 2010 ; Dong *et al.*, 2010 ; Siddique, Morimoto, 2012) présentent des algorithmes pour exécuter cette requête, mais seulement (Dong *et al.*, 2010) présentent une méthode pour optimiser le calcul de plusieurs requêtes. Ces dernière sont obtenues en faisant varier le paramètre k au sein du même espace de référence $\mathcal{D}$. Pour ce faire, ils proposent deux algorithmes calculant le k-dominant skyline cube qui est l'ensemble de tous les k-dominant skyline ($k = 1 \ldots d$). Le premier algorithme, appelé *Bottom-Up Algorithm* (en abrégé *BUA*), repose sur le résultat de $(k-1)$-dominant skyline pour calculer le k-dominant skyline; le second algorithme, appelé *Up-Bottom Algorithm* (en abrégé UBA), utilise le résultat du $(k+1)$-dominant skyline afin de calculer le k-dominant skyline. Aucun des précédents travaux ne fournit un algorithme optimisant le calcul de toutes les requêtes k-dominant skyline obtenues en faisant varier non seulement k mais aussi l'espace de référence. (Siddique, Morimoto, 2012), le plus récent, présente une technique pour calculer le k-dominant skyline reposant sur l'indexation (*Domination Power Index*). Les auteurs montrent, tout comme (Dong *et al.*, 2010), que leur algorithme est plus efficace que l'algorithme *TSA* élaboré par (Chan *et al.*, 2006a).

3. Le skycube négatif

Notre contribution se résume en la proposition d'une structure de données qui, pour chaque tuple t appartenant à la table T, résume l'ensemble des sous-espaces X tels que t *n'appartient pas à* $Sky(X)$. Ceci est motivé par l'observation suivante : pour un tuple t, alors qu'il est nécessaire de le comparer à tous les autres tuples pour savoir s'il appartient à un skyline $Sky(X)$, le comparer à un seul autre tuple t' permet déjà de déterminer une partie de l'ensemble des sous-espaces dans lesquels t est dominé,

c'est-à-dire, des sous-espaces pour lesquels il n'appartient pas aux skyline respectifs. Nous commençons par présenter quelques définitions préliminaires.

DÉFINITION 3 (Sous-espaces de dominance). — *Soit $t \in T$ et $X \subseteq \mathcal{D}$. X est un* espace de dominance *pour t ssi $t \notin Sky(X)$.*

DÉFINITION 4 (Skycube Négatif). — *Soit $t \in T$ et soit $DomSubspaces(t)$ l'ensemble des sous-espaces de dominance de t. Le* skycube négatif de T *est l'ensemble* $\{DomSubspaces(t) \mid t \in T\}$.

En d'autres termes, le skycube négatif maintient pour chaque tuple t, les sous-espaces dans lesquels t n'appartient pas à leur skyline respectif.

EXEMPLE 5. — À partir du tableau 3, il est facile de vérifier que $DomSubspaces(t_1) = \{ABCD,\ ABC,\ ACD,\ BCD,\ AC,\ BC,\ CD,\ C,\ D\}$. □

Clairement, si le skycube négatif de T est disponible, alors le calcul de tout skyline $Sky(X)$ est évident : pour chaque tuple t, $t \in Sky(X)$ ssi $X \notin DomSubspaces(t)$.

Donc, le skycube négatif de T apparaît comme le complémentaire de son skycube (positif).

À présent, nous montrons que le skycube négatif peut être calculé plus efficacement que la solution naïve consistant à calculer d'abord le skycube *positif* et ensuite de dériver le skycube négatif. Nous commençons par montrer que la comparaison de t et t' permet d'obtenir un ensemble de sous-espaces appartenant à $DomSubspaces(t)$.

La définition suivante part de l'idée selon laquelle une seule comparaison pourrait permettre de déterminer un certain nombre d'espaces dans lesquels un tuple est dominé. Par exemple, si à partir du tableau 3, nous comparons t_2 à t_5, nous observons que t_2 domine t_5 pour chacune des dimensions A, B et C. De ce fait, nous pouvons conjecturer que t_5 n'appartient à aucun skyline $Sky(X)$ tel que $X \subseteq ABC$. par conséquent, cette unique comparaison nous a permis de déterminer le statut *skyline* de t_5 par rapport aux sept sous-espaces non vides de ABC. Dans la suite, nous formalisons cette intuition en commençant par définir ce que signifie *comparer* deux tuples.

DÉFINITION 6. — *Soient $t, t' \in T$. Nous définissons la fonction de comparaison $\mathcal{COMPARE}$ comme suit : $\mathcal{COMPARE}(t, t') = \langle X|Y\rangle$ tel que X est l'ensemble des dimensions D_j telles que $t'[D_j] < t[D_j]$ et Y est l'ensemble des dimensions D_k pour lesquelles t et t' sont égaux*[2]*. Alors, $t[D_\ell] < t'[D_\ell]$ pour tout $D_\ell \in \mathcal{D} \setminus \mathcal{X} \cup \mathcal{Y}$.*

Pour simplifier la notation, nous utilisons $\mathcal{COMPARE}(t)$ pour désigner l'ensemble $\cup_{t' \in T}\{\mathcal{COMPARE}(t, t')\}$.

EXEMPLE 7. — Du tableau 3, nous avons $\mathcal{COMPARE}(t_5, t_6) = \langle BC|D\rangle$ parce que $t_6[B] < t_5[B]$ (3 vs. 4), $t_6[C] < t_5[C]$ (4 vs. 5) et les deux tuples sont égaux sur la dimension D (2). Nous avons $\mathcal{COMPARE}(t_6, t_5) = \langle A|D\rangle$. □

2. Des paires similaires sont définies dans (Bøgh *et al.*, 2014) mais ne sont pas utilisées de la même façon.

Évidemment, si $\mathcal{COMPARE}(t, t') = \langle X|Y\rangle$ alors $X \cap Y = \emptyset$.

DÉFINITION 8 (Couverture). — *Soit $\langle X|Y\rangle$ une paire d'espaces disjoints et soit Z un espace. On dit que la paire $\langle X|Y\rangle$* couvre *Z ssi $Z \subseteq XY$ et $Z \cap X \neq \emptyset$.*

EXEMPLE 9. — $p = \langle AC|B\rangle$ couvre les espaces A, AB, C, AC, BC et ABC. B n'est pas couvert par p car malgré le fait que $B \subseteq ACB$, on a $B \cap AC = \emptyset$. □

La proposition suivante montre que les espaces couverts par la paire obtenue en comparant t à t' sont précisemment les espaces dans lesquels t' domine t.

PROPOSITION 10. — *Soit $t, t' \in T$ et soit $\mathcal{COMPARE}(t, t') = \langle X|Y\rangle$. Alors, d'une part, t' domine t en chacun des espaces Z couverts par $\langle X|Y\rangle$, c'est-à-dire, $t \notin Sky(Z)$. D'autre part, t' ne domine t qu'en ces espaces.*

PREUVE 11. — Soit Z un sous-espace couvert par la paire $\langle X|Y\rangle$. Alors, il existe deux sous-espaces disjoints Z_1 et Z_2 tels que $Z_1 \cup Z_2 = Z$, $Z_1 \subseteq X$, $Z_2 \subseteq Y$ $\quad Z_1 \neq \emptyset$. $Z_1 \subseteq X$ implique que $t' \prec_{Z_1} t$ et $Z_2 \subseteq Y$ implique que $t'[Z_2] = t[Z_2]$ donc $t' \prec_{Z=Z_1 \cup Z_2} t$. ■

En fait, $\mathcal{COMPARE}(t, t')$ est un résumé de l'ensemble des espaces pour lesquels t est dominé par t'. Par conséquent, c'est un résumé d'une partie de l'ensemble des espaces pour lesquels t n'appartient pas aux skyline respectifs.

DÉFINITION 12. — *Soit E un ensemble de paires d'espaces. On note $\mathcal{COVER}(E)$ l'ensemble de tous les espaces couverts par au moins une des paires appartenant à E.*

EXEMPLE 13. — $\mathcal{COVER}(\{\langle A|B\rangle, \langle AD|\emptyset\rangle\}) = \{A, AB, D, AD\}$ □

DÉFINITION 14. — *$\mathcal{DOM}(t)$ dénote l'ensemble des espaces Z tels qu'il existe t' qui domine t au regard de l'espace Z.*

En d'autres termes, $\mathcal{DOM}(t)$ est l'ensemble de tous les sous-espaces de $\mathcal{D}$ pour lesquels t est dominé. Donc $2^{\mathcal{D}} \setminus \mathcal{DOM}(t, T)$ est l'ensemble des sous-espaces pour lesquels t appartient au skyline.

À ce niveau, il est possible de présenter un algorithme construisant le skycube négatif. Cette structure peut alors être utilisée pour répondre aux requêtes skyline.

L'algorithme 1 montre comment cette structure de données est construite. Son idée principale est simplement de comparer chaque paire de tuples (t, t') et d'ajouter $\mathcal{COMPARE}(t, t')$ à l'ensemble associé à t.

Dans le cas où $\mathcal{COMPARE}(t, t')$ est égal à la paire $\langle \mathcal{D}|\emptyset\rangle$ (voir ligne 5), cela signifie que t' domine t en toutes les dimensions. Par conséquent, t n'appartient à aucun skyline et l'ensemble qui lui est associé peut être réduit à une seule paire.

EXEMPLE 15. — Du tableau 3, la structure de données retournée par BUILDNSC est présentée dans la table 4. Notons que les paires $\langle X|Y\rangle$ pour lesquelles $X = \emptyset$ ne sont pas considérées parce qu'elles ne couvrent aucun espace (voir Proposition 10). □

Algorithme 1 : BUILDNSC

Input : Table T
Output : A data structure NSC summarizing the skycube.

```
1 begin
2     foreach t ∈ T do
3         NSC[t] ← ∅
4         foreach t' ∈ T do
5             Add COMPARE(t, t') to NSC[t]
6     return NSC
```

Tableau 4. NSC *relatif à* T

$Tuples$	$Paires$
t_1	$\langle C\|ABD\rangle, \langle CD\|\emptyset\rangle, \langle D\|\emptyset\rangle$
t_2	$\langle D\|C\rangle, \langle CD\|\emptyset\rangle, \langle D\|\emptyset\rangle$
t_3	$\langle AB\|\emptyset\rangle, \langle AB\|C\rangle, \langle CD\|B\rangle$
t_4	$\langle AB\|\emptyset\rangle, \langle A\|B\rangle, \langle A\|\emptyset\rangle$
t_5	$\langle ABC\|\emptyset\rangle, \langle ABC\|D\rangle, \langle BCD\|\emptyset\rangle, \langle BC\|D\rangle$
t_6	$\langle ABC\|\emptyset\rangle, \langle ABC\|D\rangle, \langle ABCD\|\emptyset\rangle, \langle A\|D\rangle$

L'algorithme 2 montre comment la structure de données précédente peut être utilisée pour exécuter une requête skyline $Sky(Z)$. Pour chaque tuple t, il scanne l'ensemble des paires qui lui sont associées. Si une paire couvrant Z est rencontrée, alors t n'appartient pas à $Sky(Z)$ sinon t est un point skyline de Z.

3.1. Réduction de la taille de NSC

Réduire la taille de NSC (de l'ordre de $O(n * min(n, 2^d))$) permet non seulement de réduire la consommation de mémoire mais en plus d'améliorer le temps de réponse des requêtes skyline. Alors, le problème que nous abordons ici consiste à 1) supprimer les tuples fallacieux, c'est-à-dire ceux qui n'appartiennent à aucun skyline et 2) pour chaque tuple restant, trouver un *ensemble minimal* de paires couvrant exactement les espaces couverts par $\mathcal{COMPARE}(t)$. Afin de donner une intuition à propos de ce procédé, considérons l'exemple suivant.

EXEMPLE 16. — Supposons $\langle \mathcal{D}|\emptyset\rangle \in \mathcal{COMPARE}(t)$. Puisque cette paire couvre tous les sous-espaces, nous concluons que t n'appartient à aucun skyline. Donc, t et l'ensemble de paires qui lui est associé peuvent être supprimés de NSC. □

EXEMPLE 17. — Soit $p = \langle\emptyset|Y\rangle$ une paire associée à t. Clairement, $\mathcal{COVER}(p) = \emptyset$. De ce fait, la paire p peut être retirée sans que cela n'affecte le résultat des requêtes skyline. □

Algorithme 2 : EVALUATESKYLINE

```
Input : NSC structure, subspace Z
Output : Sky(Z)
begin
    Sky(Z) ← ∅
    foreach t ∈ T do
        covered ← false
        foreach p ∈ NSC[t] do
            if p covers Z then
                covered ← true
                break
        if covered=false then
            Add t to Sky(Z)
    return Sky(Z)
```

EXEMPLE 18. — Soient $p_1 = \langle A|BC\rangle$ et $p_2 = \langle AB|C\rangle$ deux paires associées à t. $\mathcal{COVER}(\{p_1\}) = \{A, AB, ABC, AC\}$ et $\mathcal{COVER}(\{(p_2\}) = \{A, AB, ABC, AC, B, BC\}$. En raison de l'inclusion, p_1 peut être supprimé sans que cela n'affecte le résultat des requêtes skyline. □

Le test d'inclusion entre les paires peut être réalisé sans avoir besoin de générer l'ensemble des espaces que celles-ci couvrent. En effet, rappelons que le nombre de sous-espaces couverts est exponentiel avec la taille de la paire, ce qui rend le test d'inclusion coûteux à réaliser.

LEMME 19. — *Soient* $p_1 = \langle X_1|Y_1\rangle$ *et* $p_2 = \langle X_2|Y_2\rangle$. *Alors* $\mathcal{COVER}(\{p_1\}) \subseteq \mathcal{COVER}(\{p_2\})$ *ssi (i)* $X_1Y_1 \subseteq X_2Y_2$ *et (ii)* $X_1 \subseteq X_2$.

PREUVE 20. — Si $X_1Y_1 \not\subseteq X_2Y_2$ alors p_2 ne peut pas couvrir l'espace X_1Y_1 tandis que p_1 le couvre. Supposons à présent que la condition (i) est satisfaite mais pas la condition (ii). Alors, il existe une dimension $D_j \in X_1 \setminus X_2$. La paire p_1 couvre le sous-espace $Z = D_j$ alors que ce n'est pas le cas pour p_2. Donc la condition (ii) est nécessaire. Ce qui conclut la preuve. ■

L'égalité suivante est immédiate :

$$\mathcal{DOM}(t,T) = \mathcal{COVER}(\bigcup_{t' \in T} \mathcal{COMPARE}(t,t'))$$

Ainsi, matérialiser l'ensemble des paires $\bigcup_{t' \in T} \mathcal{COMPARE}(t,t')$ pour chaque tuple t nous permet de savoir si t appartient au skyline relatif à n'importe quel sous-espace $Z \in 2^{\mathcal{D}}$.

L'inconvénient de matérialiser l'ensemble $\bigcup_{t' \in T} \mathcal{COMPARE}(t, t')$ est le fait qu'il pourrait contenir de la redondance. Dans la section suivante, nous montrons comment réduire le nombre de paires tout en conservant les propriétés.

3.1.1. Approches pour la réduction de la taille de NSC

Le problème de compression de NSC consiste à trouver, pour chaque tuple t, un ensemble $\mathcal{P}$ de taille minimale tel que $\mathcal{COVER}(\mathcal{P}) = \mathcal{COVER}(\mathcal{COMPARE}(t))$.

Nous abordons ce problème selon deux approches :

– Nous considérons en entrée l'ensemble des paires associées à t et essayons de trouver un sous-ensemble minimal de $\mathcal{COMPARE}(t)$.

– Nous considérons en entrée les espaces dans lesquels t est dominé et déterminons un ensemble minimal de paires couvrant ces espaces.

Du point de vue résumé, la solution de la seconde approche est naturellement préférable puisqu'elle est de plus petite taille. Notons cependant que l'entrée de la première approche est de taille plus petite que celle de la seconde : les paires contre les espaces couverts par ces paires.

Notre problème pourrait être formalisé comme suit : étant donné un tuple t et l'ensemble de paires qui lui est associé $\mathcal{COMPARE}(t)$, comment déterminer un ensemble minimal de paires E^* tel que $\mathcal{COVER}(E^*) = \mathcal{COVER}(\mathcal{COMPARE}(t))$?

Ce problème peut également être reformulé de deux manières :

(i) le problème sous contrainte de ressource : $E^* \subseteq \mathcal{COMPARE}(t)$;

(ii) le problème sans contrainte : E^* n'est pas forcement un sous-ensemble de $\mathcal{COMPARE}(t)$.

La section suivante (section 3.1.2) aborde le problème sous-contrainte d'inclusion (3.1.1) tandis que la section 3.1.3 aborde le problème sans cette contrainte (3.1.1).

3.1.2. Minimisation de $\mathcal{COMPARE}(t)$

Soit E un ensemble de paires, l'objectif de cette partie est de trouver un autre ensemble de paires $E^* \subset E$ dont la cardinalité (nombre de paires) est la plus petite possible et tel que $\mathcal{COVER}(E^*) = \mathcal{COVER}(E)$

EXEMPLE 21. — Soit $p_1 = \langle AB|C\rangle$, $p_2 = \langle BC|A\rangle$ et $p_3 = \langle AC|\emptyset\rangle$ trois paires associées au tuple t. Il est facile d'observer qu'il n'y a pas d'inclusion entre les ensembles d'espaces couverts par ces paires. Mais, la paire p_3 peut être retirée. En effet, $\mathcal{COVER}(p_3) = \{A, AC, C\}$. A et AC sont couverts par p_1 tandis que C est couvert par p_2. Donc, la paire p_3 est redondante. □

Nous avons $\mathcal{COVER}(E^* = \{P_3\}) = \mathcal{COVER}(E)$, donc matérialiser E^* est moins coûteux que de matérialiser E pour deux raisons :

– E^* requiert moins d'espace que E;

– requêter E^* (vérifier qu'un espace Z est couvert) est plus rapide que requêter E.

Le problème abordé dans cette section peut alors être formalisé comme suit :

> **Le problème RSP :** Étant donné un tuple t et l'ensemble de paires qui lui est associé $\mathcal{COMPARE}(t)$. Réduire la taille de l'ensemble de paires $\mathcal{COMPARE}(t)$ (**RSP**) revient à trouver un sous-ensemble de paires $E \subseteq \mathcal{COMPARE}(t)$ de taille minimale tel que $\mathcal{COVER}(E) = \mathcal{COVER}(\mathcal{COMPARE}(t))$.

Le théorème suivant établit la difficulté du problème **RSP**.

THÉORÈME 22. — **RSP** *est NP-Difficile.*

PREUVE 23. — (esquisse) Évidemment, ce problème de minimisation est NP. La preuve qu'il est NP-Difficile est basée sur la réduction à partir du problème de l'ensemble minimal couvrant (minimal set cover, **MSC**). Étant donnée une instance de problème **MSC**, nous construisons une table T de tuples distincts t dans laquelle le nombre de dimensions d est égal au nombre d'éléments devant être couverts dans **MSC** et où le nombre de tuples n est égal au nombre initial d'ensembles dans **MSC**.

Nous établissons une correspondance entre E et les ensembles de ***MSC*** et montrons que toute solution de ***MSC*** est une solution de ***RSP***.

Soit $s = \{s_1, s_2, \ldots, s_n\}$ l'ensemble d'ensembles d'éléments, entrée de l'instance du problème **MSC**. Soit $t = (1, 1, \ldots, 1)$ un tuple ayant d valeurs. Pour chaque ensemble $s_j \in$ **MSC**, nous ajoutons à T un tuple t' tel que $t'[i] = 0$ ssi $i \in s_j$ sinon $t'[i] = 1$. Par conséquent, $\mathcal{COMPARE}(t, t') = \langle X|Y\rangle$ ssi $\exists s \in$ **MSC** tel que $X = s$.

Par exemple, soit $s = \{s_1 = \{1, 2\}; s_2 = \{2, 3\}; s_3 = \{1, 3\}\}$ l'instance de **MSC**. Le nombre d'éléments à couvrir est $d = 3$ et le nombre d'ensembles $n = 3$. Alors, nous obtenons la table T ayant $n + 1 = 4$ tuples (incluant t) et 3 dimensions. Ceci est illustré par le tableau suivant.

Id	**1**	**2**	**3**
t	1	1	1
t_1	0	0	1
t_2	1	0	0
t_3	0	1	0

Il y a une correspondance entre les $s_i \in s$ et les $p_i = Compare(t, t_i)$. Par exemple, $Compare(t, t_1) = \langle 12|3\rangle$ correspond à $s_1 = \{1, 2\}$. De ce fait, il peut être prouvé qu'un ensemble s_i appartient à la solution de l'instance de **MSC** ssi sa paire correspondante appartient à la solution du problème **RSP**. ■

Algorithme 3 : $ApproxMSP$

Input : Set of pairs: I
Output : Set of pairs: O

```
1 begin
2     O ← ∅
3     U ← COVER(I)
4     while U ≠ ∅ do
5         p ← arg max_{p'∈I} |Cov(p') \ COVER(O)|
6         O ← O ∪ {p}
7         U ← U \ Cov(p)
8     return O
```

Algorithme 4 : $listPairsTuple$, creates a reduced list of pairs associated to a tuple

Input : tuple: t, Set of tuples: $Topmost$
Output : Set of pairs: $listOut$

```
1 begin
2     listOut ← ∅
3     for each t' in Topmost do
4         add the pair COMPARE(t, t') to listOut
5     listOut ← ApproxMPC(listOut)
6     return listOut
```

Dans (Chvatal, 1979), un algorithme glouton approximatif en temps polynomial a été proposé pour résoudre le problème ***MSC***. Cet algorithme choisit à chaque étape l'ensemble couvrant le plus grand nombre d'éléments non encore couverts par les ensembles précédemment choisis. L'adaptation de cet algorithme pour résoudre le problème **RSP** est évidente et présentée à travers l'algorithme 3. Il s'agit dans notre cas de choisir, à chaque itération la paire couvrant le plus grand nombre d'espaces non encore couverts par les paires déjà choisies.

L'algorithme 4 montre comment créer une liste réduite de paires associées à un tuple. Cet algorithme appelle la fonction *ApproxMPC* qui est une adaptation de la proposition de (Chvatal, 1979) au problème **RSP**.

EXEMPLE 24. — Le résultat de la minimisation du tableau 4 utilisant l'algorithme 4 est présenté dans le tableau 5. Notons que le nombre de paires est passé de 20 à 8. □

Tableau 5. Liste des paires synthétisant les ensembles de dominance

Tuples	Listes des paires associées
t_1	$\langle C\|ABD\rangle, \langle CD\|\emptyset\rangle$
t_2	$\langle CD\|\emptyset\rangle$
t_3	$\langle AB\|C\rangle, \langle CD\|B\rangle$
t_4	$\langle AB\|\emptyset\rangle$
t_5	$\langle ABC\|D\rangle$
t_6	$\langle ABCD\|\emptyset\rangle$

3.1.3. Minimisation sans contrainte d'inclusion

Il est important de préciser que la solution exacte du problème **RSP** ne présente pas la garantie d'être le *plus petit* ensemble de paires codant l'ensemble des espaces dans lesquels t est dominé. Son idée maîtresse est juste de retirer les paires *redondantes*. Une autre manière d'aborder ce problème de minimisation est de considérer l'ensemble U des espaces couverts par l'ensemble des paires $\mathcal{P}$ associées au tuple t et d'essayer de dériver un ensemble minimal de paires $\mathcal{Q}$ qui couvre U. Notons que $\mathcal{Q}$ n'est pas forcement inclus dans $\mathcal{P}$. Nous illustrons la différence entre les deux approches à travers l'exemple suivant.

EXEMPLE 25. — Soit $\mathcal{P} = \{\langle AB|C\rangle, \langle BC|A\rangle\}$ l'ensemble des paires associées à t. $\mathcal{P}$ est minimal dans ce sens qu'il est imposible d'en retirer une paire quelconque tout en couvrant l'ensemble des espaces couverts par $\mathcal{P}$. Notons cependant que $\mathcal{Q} = \{\langle ABC|\emptyset\rangle\}$ couvre exactement les mêmes espaces que $\mathcal{P}$ et est de taille plus petite. □

Le problème MSP : Étant donnés un tuple t et l'ensemble qui lui est associé $\mathcal{DOM}(t)$ qui représente l'ensemble des espaces dans lesquels t est dominé. Le problème **MSP** consiste à trouver un ensemble minimal de paires E tel que $\mathcal{COVER}(E) = \mathcal{DOM}(t)$.

La difficulté que nous rencontrons à résoudre ce problème tient du fait que le skyline n'est pas monotone, c'est-à-dire $Y \subseteq X \not\Rightarrow Sky(Y) \subseteq Sky(X)$. La conséquence de cette propriété dans notre travail peut être présentée comme suit : soit $X \in \mathcal{DOM}(t)$. On pourrait être tenté de coder cette information par la paire $\langle X|\emptyset\rangle$. Cependant, cette paire couvre tous les sous-espaces de X alors qu'il est possible qu'il existe un sous-espace $Y \subseteq X$ tel que $t \in Sky(Y)$. Dans la suite, nous mettons en exergue quelques propriétés nous permettant d'accomplir notre tâche consistant à résumer cette information.

LEMME 26. — *Soit $t \in T$ et soient X, Y deux espaces. Si $t \notin Sky_T(XY)$ et $t \in Sky_T(X)$ alors $t \notin Sky_T(Y)$.*

PREUVE 27 (par l'absurde). — Supposons que $\underbrace{t \in Sky_T(X)}_{(1)}$, $\underbrace{t \in Sky_T(Y)}_{(2)}$ et $\underbrace{t \notin Sky_T(XY)}_{(3)}$

De (1) et (3), il existe $u \in Sky_T(X)$ tel que $\underbrace{u[X] = t[X]}_{(4)}$ et $\underbrace{u \prec_Y t}_{(5)}$

De (2) et (3), il existe $v \in Sky_T(Y)$ tel que $\underbrace{v[Y] = t[Y]}_{(6)}$ et $\underbrace{v \prec_X t}_{(7)}$

Nous obtenons une contradiction entre (1) et (7) car t est dominé au regard de X et t appartient au skyline de X. Nous obtenons également une contradiction entre (2) et (5) car t est dominé au regard de Y et t appartient au skyline de Y. ■

Ceci implique que, si $X \in \mathcal{DOM}(t)$ alors il existe au plus un fils X_1 de X [3] tel que $X_1 \notin \mathcal{DOM}(t)$.

Ce résultat peut être expliqué par le fait que pour tous X_1, X_2, deux sous-espaces distincts tels que $|X_1| = |X_2| = |X| - 1$, nous avons $X_1 \cup X_2 = X$. En appliquant la proposition 26, si $t \notin Sky_T(X)$ et $t \in Sky_T(X_1)$ alors $t \notin Sky_T(X_2)$; alors il existe au plus un fils X_1 de X tel que $t \in Sky_T(X_1)$.

En allant plus loin, nous pouvons généraliser en établissant le résultat suivant. Intuitivement, il déclare qu'il existe au plus un plus grand sous-espace Z de X tel que $t \in Sky(Z)$ tandis que $t \notin Sky(X)$. Formellement,

PROPOSITION 28. — *Soit* $X \in \mathcal{DOM}(t)$. *Alors l'ensemble* $\underset{Z \subset X, Z \notin \mathcal{DOM}(t)}{Argmax} |Z|$ *contient au plus un élément.*

Ceci signifie que si $t \notin Sky(X)$ alors il existe au plus un sous-espace $Z \subset X$ de taille maximale (en nombre d'attributs) tel que t appartient au skyline de Z.

PREUVE 29. — Supposons qu'il existe deux sous-espaces X_1 et X_2 de taille maximale tels que $X_1 \notin \mathcal{DOM}(t)$ et $X_2 \notin \mathcal{DOM}(t)$ alors $X_1 \cup X_2 \notin \mathcal{DOM}(t)$ (voir Proposition 26). De plus, puisque $X_1 \cup X_2 \subseteq X$ et $X_1 \neq X_2$ alors $|X_1 \cup X_2| > max(|X_1|, |X_2|)$. Il y a une contradiction car soit $X_1 \cup X_2 = X$ (mais $X \in \mathcal{DOM}(t)$), soit X_1 et X_2 n'appartiennent pas à la solution de (i). ■

De la proposition précédente, nous pouvons déduire le résultat suivant qui intuitivement montre que si Y est le plus grand sous-espace de X tel que $t \notin Sky(X)$ et $t \in Sky(Y)$ alors chaque $Z \subseteq X$ tel que $t \in Sky(Z)$ est nécessairement un sous-espace de Y. Formellement,

3. X_1 est un fils de X ssi $X_1 \subset X$ et $|X_1| = |X| - 1$.

PROPOSITION 30. — *Soit* $X \in \mathcal{DOM}(t)$, *Soit* $Y = \underset{Z \subset X, Z \notin \mathcal{DOM}(t)}{Argmax} |Z|$ *alors* $\forall Z \subset X$ *tel que* $Z \notin \mathcal{DOM}(t)$ *nous obtenons* $Z \subseteq Y$.

PREUVE 31 (par l'absurde). — Soit $W \subset X$ tel que $X \notin \mathcal{DOM}(t)$ et $W \nsubseteq Y$ alors $W \cup Y \subseteq X$ et $W \cup Y \notin \mathcal{DOM}(t)$ (car W et Y n'appartiennent pas à $\mathcal{DOM}(t)$, voir Proposition 26), alors il y a une contradiction car $|W \cup Y| > |Y|$, $Y \neq \underset{Z \subset X, Z \notin \mathcal{DOM}(t)}{Argmax} |Z|$. ■

Intuitivement, ce résultat établit que chaque sous-espace de $X \in \mathcal{DOM}(t)$ qui n'appartient pas à $\mathcal{DOM}(t)$ est inclus dans un autre espace (noté Y) pour lequel t appartient au skyline. Alors, chaque sous-espace appartenant à $2^X \setminus 2^Y$ appartient à $\mathcal{DOM}(t)$, de ce fait, $\langle X \setminus Y | Y \rangle$ est la paire couvrant à la fois X et le maximum de sous-espaces de X inclus dans $\mathcal{DOM}(t)$.

La propriété précédente peut être exploitée pour résoudre le problème **MSP** comme suit : soit $X \in \mathcal{DOM}(t)$. Alors X correspond à la paire $\langle X \setminus Y | Y \rangle$ telle que Y est le plus grand sous-espace de X tel que $t \in Sky(Y)$. Si chaque $X \in \mathcal{DOM}(t)$ est remplacé comme décrit précédemment, alors nous obtenons un ensemble de paires dont la minimisation consiste simplement à supprimer les paires p_i telles qu'il existe une paire p_j qui la couvre. Cette procédure est décrite dans l'algorithme 5.

PROPOSITION 32. — *La sortie de l'algorithme 5 couvre les espaces de* $\mathcal{DOM}(t)$ *et est calculée exactement en temps* $O(|\mathcal{DOM}(t)|^2)$. *De plus, cet ensemble est de taille minimale.*

PREUVE 33. — Soit $OutputSet$ la sortie de l'algorithme 5.

(1) nous montrons tout d'abord que $OutputSet$ couvre tous les espaces appartenant à $\mathcal{DOM}(t)$; et ne couvre que ceux-ci (double inclusion).

(a) nous commençons par montrer que $\mathcal{COVER}(p) \subseteq \mathcal{DOM}(t)$, $\forall p \in OutputSet$. Soit $\langle W | T \rangle$ une paire appartenant à $OutputSet$, T est (par construction) le plus grand sous-espace de WT qui n'appartient pas à $\mathcal{DOM}(t)$. Alors T inclut tout sous-espace de WT n'appartenant pas à $\mathcal{DOM}(t)$ (voir Proposition 30), alors 2^{WT} $2^T = \mathcal{COVER}(\langle W | T \rangle) \subseteq \mathcal{DOM}(t)$

(b) nous avons montré que pour chaque espace Z appartenant à $\mathcal{DOM}(t)$ il existe une paire de $OutputSet$ qui couvre Z. Chaque itération (voir cinquième ligne) de l'algorithme ajoute à $OutputSet$ une paire qui couvre un espace de $\mathcal{DOM}(t)$, non encore couvert par les paires ajoutées précédemment. Et l'algorithme a dans le pire cas une complexité en temps de $O(m^2)$ où m est la taille de l'entrée. Alors, l'algorithme s'arrête après un temps fini, lorsque tous les espaces appartenant à $\mathcal{DOM}(t)$ sont couverts par au moins une paire de $OutputSet$

(2) Ensuite, montrons que $OutputSet$ est de taille minimale. Supposons que lp soit une liste de paires couvrant exactement les espaces de $\mathcal{DOM}(t)$. Nous allons montrer qu'il existe une fonction injective f allant de $OutputSet$ vers lp ce qui reviendrait à dire que la taille de $OutputSet$ est plus petite (ou égale à) la taille de tout lp. Pour f, nous choisissons la fonction qui associe à chaque paire $\langle X | Y \rangle \in OutputSet$ la

Algorithme 5 : $MinimalSP$, minimal set of pairs covering $\mathcal{DOM}(t)$

```
Input : DOM(t), d
Output : Set of pairs: OutputSet
begin
    OutputSet ← ∅
    Marked ← ∅
    Sort elements of DOM(t) w.r.t. their size in the descending order
    for each X ∈ DOM(t) such that X ∉ Marked do
        Found ← false
        Level ← |X| − 1
        Y ← ∅
        while not(Found) and Level ≥ 1 do
            if ∃Z such that Z ⊂ X, |Z| = Level and Z ∉ DOM(t) then
                Found ← true
                Y ← Z
            else
                Level ← Level − 1
        Insert (X \ Y; Y) into OutputSet
        Insert X into Marked
        for each W ∈ DOM(t) such that W is covered by (X \ Y; Y) do
            Insert W into Marked
    return OutputSet
```

paire $\langle x|y\rangle \in lp$ telle que $\langle x|y\rangle$ couvre XY (s'il y a plusieurs paires satisfaisant cette condition, alors nous choisissons la plus petite dans l'ordre lexicographique).

(a) Montrons que la fonction f est définie pour toute paire $\langle X|Y\rangle \in OutputSet$. $\forall\langle X|Y\rangle \in OutputSet$, l'espace XY appartient à $\mathcal{DOM}(t)$ donc il existe au moins une paire $p \in lp$ telle que p couvre XY.

(b) Montrons enfin que la fonction f est injective; en d'autres termes, ceci est équivalent à montrer que pour tous $\langle X_1|Y_1\rangle$ et $\langle X_2|Y_2\rangle$, deux paires appartenant à $OutputSet$, il n'existe pas de paire $\langle x|y\rangle \in lp$ couvrant à la fois X_1Y_1 et X_2Y_2. Par l'absurde, supposons qu'il existe une telle paire $\langle x|y\rangle \in lp$; alors $\underbrace{X_1Y_1 \subseteq xy}_{(1)}$ et $\underbrace{X_1Y_1 \cap x \neq \emptyset}_{(2)}$ aussi bien que $\underbrace{X_2Y_2 \subseteq xy}_{(3)}$ et $\underbrace{X_2Y_2 \cap x \neq \emptyset}_{(4)}$ (1) et (3) impliquent que $X_1Y_1 \cup X_2Y_2 \subseteq xy$ et (2) et (4) impliquent que $(X_1Y_1 \cup X_2Y_2) \cap xy \neq \emptyset$ alors $X_1Y_1 \cup X_2Y_2 \in \mathcal{DOM}(t)$. Donc, il existe aussi une paire $\langle X|Y\rangle \in OutputSet$ couvrant $X_1Y_1 \cup X_2Y_2$ ce qui implique que $X \cap (X_1Y_1 \cup X_2Y_2) \neq \emptyset$, en d'autres termes $X \cap X_1Y_1 \neq \emptyset$ ou $X \cap X_2Y_2 \neq \emptyset$. Donc, $\langle X|Y\rangle$ couvre $\langle X_1|Y_1\rangle$ ou $\langle X_2|Y_2\rangle$ ce qui est absurde car, par construction de l'algorithme, 5 ne peut pas ajouter à $OutputSet$

la paire $\langle X_1|Y_1\rangle$ (resp. $\langle X_1|Y_1\rangle$) si l'espace X_1Y_1 (resp. X_2Y_2) est déjà couvert par une autre paire ajoutée ($\langle X|Y\rangle$).

■

3.1.4. Comparer les tuples au Topmost suffit

Dans cette partie, nous montrons que pour construire notre structure de données, il suffit de comparer les tuples à ceux appartenant au em topmost skyline c'est-à-dire $Sky(\mathcal{D})$, au lieu de les comparer à l'ensemble des tuples de la table T. Ceci est particulièrement intéressant lorsque la taille du topmost skyline est petite par rapport à n.

THÉORÈME 34. — *Soit* $E = \bigcup_{t' \in Sky(\mathcal{D})} \mathcal{COMPARE}(t, t')$.

Alors $\mathcal{COVER}(E) = \mathcal{COVER}(\mathcal{COMPARE}(t))$.

PREUVE 35. — On sait que

$$\mathcal{COMPARE}(t) = \bigcup_{t' \in Topmost} \mathcal{COMPARE}(t, t') \cup \bigcup_{t' \notin Topmost} \mathcal{COMPARE}(t, t')$$

or $\bigcup_{t' \in Topmost} \mathcal{COMPARE}(t, t') = E$. En raison de la distributivité de l'opérateur $\mathcal{COVER}$ sur l'*union* nous pouvons écrire

$$\mathcal{COVER}(\mathcal{COMPARE}(t)) = \mathcal{COVER}(E) \cup \mathcal{COVER}\left(\bigcup_{t' \notin Topmost} \mathcal{COMPARE}(t, t')\right)$$

Il suffit alors de prouver que

$$\mathcal{COVER}\left(\bigcup_{t' \notin Topmost} \mathcal{COMPARE}(t, t')\right) \subset \mathcal{COVER}(E)$$

En d'autres termes, on pourrait juste montrer que pour tout $t' \notin Topmost$,

$$\mathcal{COVER}(\mathcal{COMPARE}(t, t')) \subseteq \mathcal{COVER}(E)$$

Soit t' un tuple de T tel que $t' \notin Topmost$. par définition du skyline, il existe un tuple $u \in Topmost$ tel que u domine t', c'est-à-dire, $u \prec_{\mathcal{D}} t'$. Soit $\langle X_1|Y_1\rangle = \mathcal{COMPARE}(t, u)$ et $\langle X_2|Y_2\rangle = \mathcal{COMPARE}(t, t')$. Pour chaque sous-espace Z couvert par $\langle X_2|Y_2\rangle$, nous avons (*i*) $t' \prec_Z t$. De plus, $u \prec_{\mathcal{D}} t'$ implique que (*ii*) $u \preceq_Z t'$ (car $Z \subseteq \mathcal{D}$).
De (*i*) et (*ii*) on a $u \prec_Z t$; donc Z est couvert par $\langle X_1|Y_1\rangle$. Tout espace couvert par $\langle X_2|Y_2\rangle$ est aussi couvert par $\langle X_1|Y_1\rangle$

Par conséquent, pour tout $t' \notin Topmost$, la paire $\mathcal{COMPARE}(t, t')$ est *redondante* dans la liste de paires associées au tuple t. ■

Tableau 6. Données réelles

Données	d	n
NBA	17	20493
MBL	18	92797
IPUMS	10	75836

EXEMPLE 36. — De notre exemple précédent, puisque le topmost skyline est constitué des tuples t_2, t_3 et t_4, les tuples seront uniquement comparés à ceux-ci. Par exemple, la liste de paires associée au tuple t_1 est $\{\langle C|ABD\rangle, \langle CD|\emptyset\rangle\}$ dont la taille est 2 en considérant le $Topmost$ au lieu d'un ensemble de 3 paires si on avait comparé t_1 à l'ensemble des tuples. □

En plus du gain d'espace dû à la réduction du nombre de comparaisons pour chaque tuple, cette optimisation rend l'algorithme glouton utilisé pour résoudre le problème **RSP** plus rapide en raison de la réduction de la taille de l'entrée.

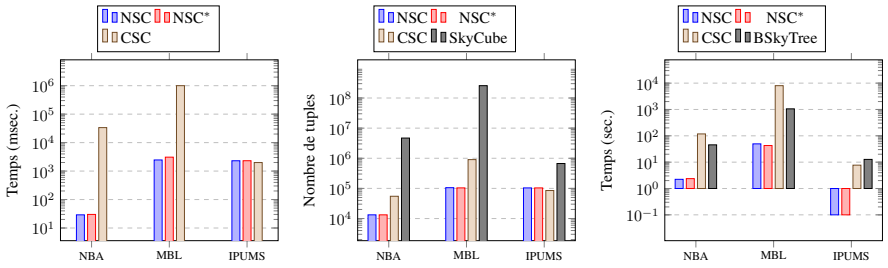
4. Évaluations expérimentales

Dans le but de comparer notre proposition aux algorithmes de l'état de l'art, nous avons implémenté nos solutions aussi bien que CSC (Xia *et al.*, 2012). Nous avons utilisé les implémentations de BSkyTree (Lee, Hwang, 2010), Hashcube (Bøgh *et al.*, 2014) et QSkyCube (Lee, Hwang, 2014) fournies par leurs auteurs respectifs. Tous les programmes ont été écrits en C++. Nous avons utilisé une machine équipée de deux processeurs hexa-core et 24 Go sous Linux. Dans cette section, nous dénotons par NSC la structure skycube négatif réduite à l'aide de l'algorithme approximatif répondant au problème **RSP** et nous notons NSC* la structure utilisant la solution du problème **MSP**. Clairement, NSC* devrait être de plus petite taille que NSC.

Pour les expérimentations, nous utilisons des données synthétiques obtenues suivant le procédé décrit par (Börzsönyi *et al.*, 2001). Nous utilisons également un ensemble de données réelles régulièrement exploitées au sein de la communauté skyline. Nous considérons trois critères de comparaison : le temps de construction, la quantité de mémoire requise et le temps des requêtes. En ce qui concerne le dernier critère et pour éviter tout biais, nous considérons le temps nécessaire pour exécuter *l'ensemble* des $2^d - 1$ requêtes skyline.

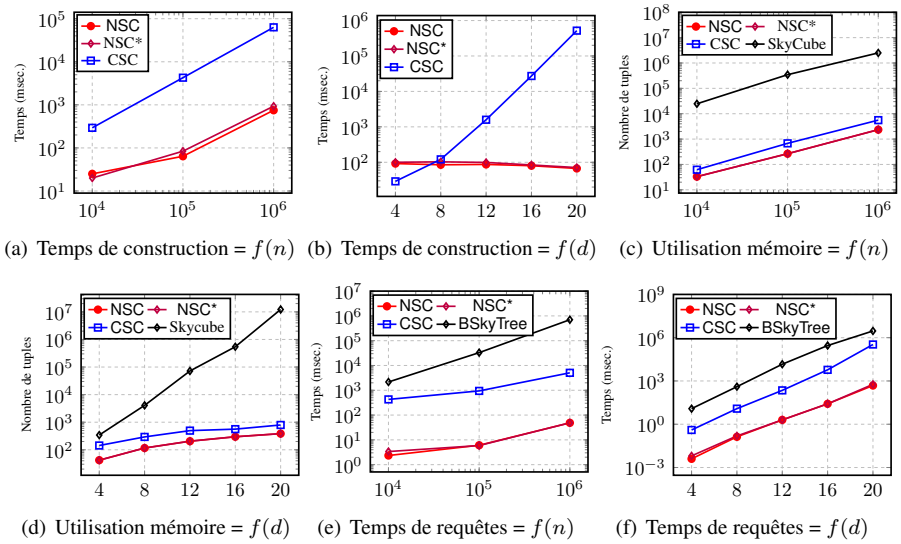
4.1. Données réelles

Ces données sont décrites dans le tableau 6. Les résultats obtenus présentés dans la figure 1 montrent que NSC surpasse CSC pour tous les critères : NSC est 100 fois plus rapide pour la construction de la structure. En ce qui concerne l'utilisation de la mémoire NSC ne requiert pas plus d'espace mémoire que CSC, par exemple,



(a) Temps de construction (b) Utilisation de la mémoire (c) Temps des requêtes

Figure 1. Données réelles



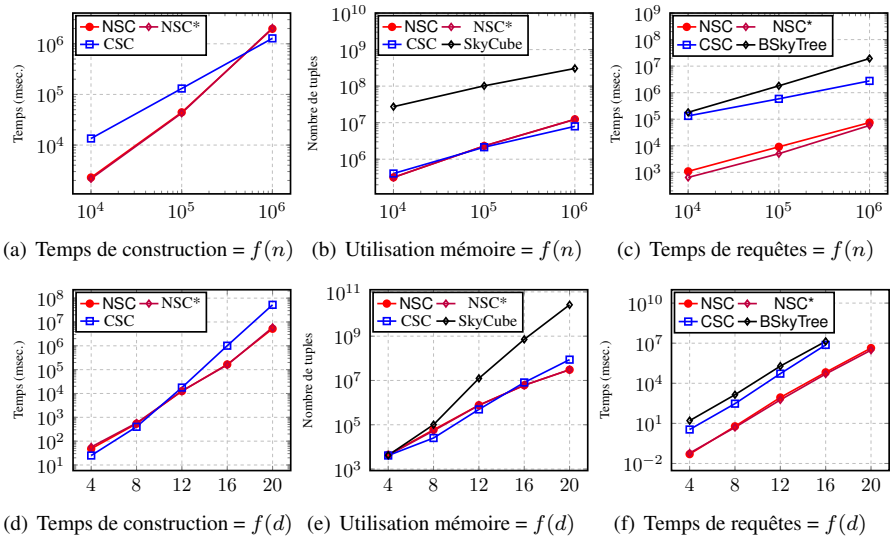
(a) Temps de construction = $f(n)$ (b) Temps de construction = $f(d)$ (c) Utilisation mémoire = $f(n)$

(d) Utilisation mémoire = $f(d)$ (e) Temps de requêtes = $f(n)$ (f) Temps de requêtes = $f(d)$

Figure 2. Données synthétiques corrélées ($k = 100$)

pour les données IPUMS, les deux algorithmes demandent presque la même quantité de mémoire, ce qui représente environ 10% de la mémoire requise par le skycube. Ce qui donne une première idée sur le *taux* de compression en fonction du nombre de dimensions : $d = 10$ pour IPUMS contre respectivement 17 et 18 pour NBA et MBL.

Nous constatons, au regard du temps de réponse, que pour un grand nombre de dimensions (NBA and MBL), CSC a de pires performances que BSkyTree, une solution ne faisant aucun pré-calcul. Notons que la même observation avec des données synthétiques a été soulignée dans (Bøgh *et al.*, 2014). Il est important de préciser que



(a) Temps de construction = $f(n)$ (b) Utilisation mémoire = $f(n)$ (c) Temps de requêtes = $f(n)$

(d) Temps de construction = $f(d)$ (e) Utilisation mémoire = $f(d)$ (f) Temps de requêtes = $f(d)$

Figure 3. Données synthétiques indépendantes

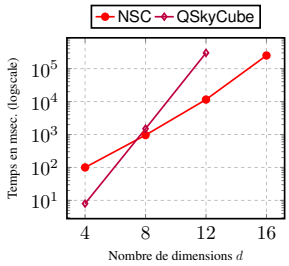


Figure 4. Construction du Skycube: NSC *vs* QSkyCube

NSC et NSC* ont des comportement similaires quel que soit le critère retenu. Leurs courbes respectives sont presque toujours confondues.

4.2. Données synthétiques

Nous générons trois types de données synthétiques : corrélées, indépendantes et anti-corrélées suivant l'outil fournit par (Börzsönyi *et al.*, 2001). Pour chacun d'eux, nous comparons NSC et NSC* à CSC en faisant varier le nombre de tuples n dans l'ensemble $\{10^4, 10^5, 10^6\}$ et en maintenant le nombre dimensions d égal à 14. Nous

faisons ensuite varier le nombre de dimensions d dans l'ensemble $\{4, 8, 12, 16\}$, tout en conservant le nombre de tuples n égal à 10^5.

Pour les données corrélées (figure 2), nous observons que lorsque d augmente, NSC surpasse largement CSC au regard des temps de construction et de requêtes (voir figures 2b et 2f). Au regard de n, les deux méthodes semblent atteindre des temps de construction et de requêtes presque linéaires (voir figures 2a et 2e). En termes d'utilisation de la mémoire, les deux algorithmes sont comparables même si NSC est légèrement plus petit. les résultats obtenus avec les données indépendantes sont présentés dans la figure 3. Les mêmes remarques tiennent pour les données indépendantes (figure 3) et les données anti-corrélées (figure 5). Cependant, il est important de préciser que lorsque $n = 10^6$ et $d = 20$ (figure 5b), CSC n'avait pas terminé la construction de la structure de données après 36 heures tandis que NSC a été construite en environ 3 heures. À partir de la structure de données construite, les $2^{20} - 1$ requêtes ont été exécutées en environ une heure. CSC pourrait alors être plus rapidement construite à partir du skycube obtenu de cette façon qu'en utilisant la version originale de l'algorithme fournie dans (Xia *et al.*, 2012). Ceci revient à dire que, utiliser un algorithme naïf pour construire CSC en construisant en premier le skycube et ensuite en déterminant pour chaque tuple les plus petits espaces pour lesquels ils appartiennent au skyline est plus rapide que l'algorithme complexe et inefficace proposé.

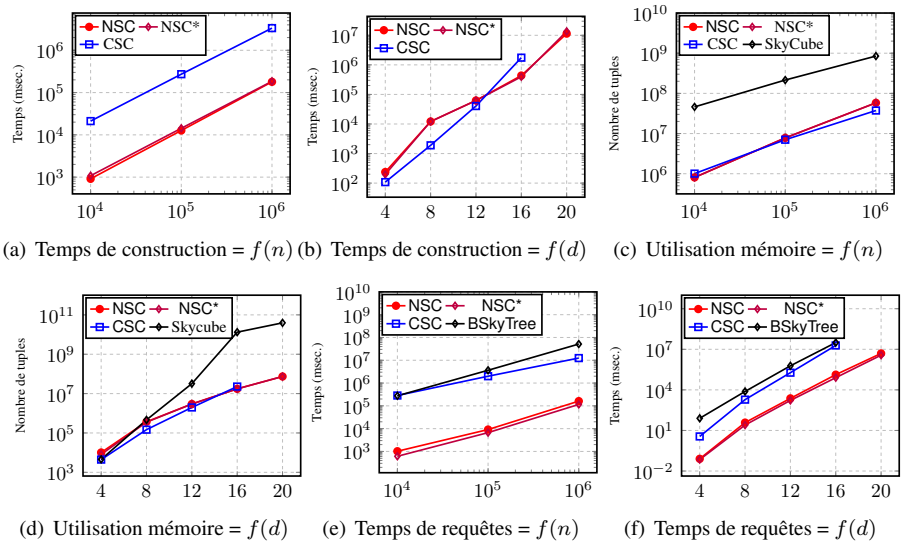
Il est cependant intéressant de remarquer que, pour de petites valeurs de d (figures 3e et 5d, $d = 4$), non seulement CSC requiert moins d'espace mémoire que NSC mais c'est aussi le cas pour l'ensemble du skycube. En d'autres termes, le nombre de skylines auxquels les tuples n'appartiennent pas est plus élevé que le nombre de skylines auxquels les tuples appartiennent. Par ailleurs, nous remarquons que CSC a presque la même taille que le skycube. Ceci signifie que dans de telles situations (d petit), il n'est pas recommandé d'utiliser NSC et encore moins CSC : le skycube entier est la meilleure option.

4.3. Comparaison à Hashcube

Dans cette partie, nous comparons NSC à la structure de données Hashcube proposée dans (Bøgh *et al.*, 2014). Nous avons utilisé l'implémentation de Hashcube fournie par les auteurs [4]. Il est important de préciser que l'implémentation de Hashcube a besoin que l'ensemble du skycube soit calculé puisque c'est ce dernier qu'elle prend en entrée. De ce fait, nous ne pouvons pas considérer le temps de construction. Avec les données NBA, NSC garde en mémoire 13259 tuples tandis que Hashcube stocke 541316. D'autre part, répondre à l'ensemble des $2^{17} - 1$ requêtes requiert 30 millisecondes à Hashcube contre 2.1 secondes pour NSC.

Dans le but d'éprouver la structure de données Hashcube, nous avons généré un jeu de données synthétiques présentant des dimensions indépendantes. Ce jeu de don-

4. Pour les besoins de comparaison, les auteurs ont également implémenté la technique CSC. Nous ne l'avons pas utilisée parce qu'elle était plus lente que la notre.



(a) Temps de construction = $f(n)$ (b) Temps de construction = $f(d)$ (c) Utilisation mémoire = $f(n)$

(d) Utilisation mémoire = $f(d)$ (e) Temps de requêtes = $f(n)$ (f) Temps de requêtes = $f(d)$

Figure 5. Anticorrelated Synthetic Data ($k = 100$)

nées contient 10^7 tuples et 16 dimensions. Son skycube requiert de stocker $20 * 10^9$ tuples. Nous avons essayé de construire la structure de donnée Hashcube correspondante mais les 24Go de mémoire disponibles n'ont pas été suffisants pour la conserver. Pour le même jeu de données, NSC utilise moins de 12Go pour stocker environ $4.5{*}10^7$ tuples. Par conséquent, lorsque le skycube est *trop grand*, ce qui arrive lorsque le nombre de dimensions est élevé, Hashcube ne peut pas être entièrement chargé en mémoire. Ce que nous pouvons tirer de cette expérimentation est que : on pourrait choisir Hashcube lorsque le nombre de dimensions est relativement petit. Dans de telles situations, il est préférable d'entièrement matérialiser le skycube, sinon NSC doit alors être choisi.

4.4. Construction du skycube

Dans cette section, nous comparons notre contribution à QSkyCube (Lee, Hwang, 2014) récemment proposé pour le calcul du skycube complet. La figure 4 montre certains résultats que nous avons obtenus. Nous avons généré des données synthétiques indépendantes fixant le nombre de tuples n à 10^5 et faisant varier le nombre de dimensions d dans l'ensemble $\{4, 8, 12, 16\}$. Il est important de souligner le fait que pour NSC, nous avons reporté le temps d'exécution total, c'est-à-dire le temps de construction de la structure et celui de l'exécution des $2^d - 1$ requêtes. Comme cela peut être observé, lorsque d augmente, c'est-à-dire $d > 8$, NSC devient plus rapide. Il est intéressant de noter que lorsque d atteint 16, QSkyCube a été arrêté car ayant saturé toute la mémoire disponible. Cette expérimentation montre que 1) notre proposition

est compétitive pour le calcul du skycube (particulièrement) lorsque d devient grand, et 2) notre proposition peut être considérée comme la première étape pour construire Hashcube puisque ce dernier requiert que le skycube soit entièrement construit.

5. Conclusion

Nous avons présenté la structure skycube négatif et fourni les algorithmes permettant de la réduire. L'idée maîtresse de cette structure est de stocker pour chaque tuple un résumé de l'ensemble des sous-espaces pour lesquels *il n'appartient pas au skyline*. Ceci est en contradiction avec les anciens travaux où l'objectif était de résumer l'ensemble des sous-espaces pour lesquels le tuple *appartient* au skyline. Nos expérimentations ont montré que NSC est particulièrement efficace en termes de temps de construction, d'utilisation de la mémoire et de temps de réponse. Dans les précédents travaux, soit les algorithmes demandaient beaucoup de temps pour la construction, soit la structure de données produite était trop grande pour tenir en mémoire. Nos expérimentations montrent également que cette structure est plus rapide pour construire le skycube que les algorithmes de l'état de l'art spécialement conçus dans cet objectif.

Une autre propriété intéressante est sa faculté à pouvoir répondre efficacement à une autre forme de requête qui est la requête k-dominant skyline. Dans les travaux futurs, nous prévoyons de proposer une solution à la maintenance incrémentale de la structure NSC. Cette possibilité est le principal avantage de la structure Compressed Skycube (CSC) bien qu'elle souffre, comme nous l'avons vu, de l'inefficacité de l'exécution des requêtes. Nous envisageons également d'adapter notre structure à la représentation binaire Hashcube. Ceci aura pour effet de réduire encore plus nos temps de réponse aux requêtes.

Remerciements

Nous remercions les rapporteurs anonymes ainsi que l'éditeur pour leurs suggestions qui nous ont permis de bien améliorer la présentation de ce travail. Ce travail a été partiellement réalisé dans le cadre des projets PETASKY *et* SPEED DATA *financés respectivement par l'initiative* MASTODONS *du CNRS et le Programme d'investissements d'avenir (PIA).*

Bibliographie

Bartolini I., Ciaccia P., Patella M. (2008). Efficient sort-based skyline evaluation. *ACM Trans. Database Syst.*, vol. 33, n° 4.

Bøgh K. S., Chester S., Sidlauskas D., Assent I. (2014). Hashcube: A data structure for space and query efficient skycube compression. In *Proceedings of CIKM conference.*

Börzsönyi S., Kossmann D., Stocker K. (2001). The skyline operator. In *Proc. of ICDE conf.*

Chan C. Y., Jagadish H. V., Tan K., Tung A. K. H., Zhang Z. (2006a). Finding k-dominant skylines in high dimensional space. In *Proceedings of SIGMOD international conference.*

Chan C. Y., Jagadish H. V., Tan K., Tung A. K. H., Zhang Z. (2006b). On high dimensional skylines. In *Proceedings of EDBT conference.*

Chester S., Sidlauskas D., Assent I., Bøgh K. S. (2015). Scalable parallelization of skyline computation for multi-core processors. In *Proceedings of ICDE conference.*

Chomicki J., Godfrey P., Gryz J., Liang D. (2003). Skyline with presorting. In *Proc. of ICDE conference.*

Chvatal V. (1979). A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, vol. 4, n° 3, p. 233–235. Consulté sur http://dx.doi.org/10.2307/3689577

Dong L., Cui X., Wang Z., Cheng S. (2010). Finding k-dominant skyline cube based on sharing-strategy. In *Proceedings of FSKD conference.*

Lee J., Hwang S. won. (2010). BSkyTree: scalable skyline computation using a balanced pivot selection. In *Proc. of EDBT conf.*

Lee J., Hwang S. won. (2014). Toward efficient multidimensional subspace skyline computation. *VLDB Journal*, vol. 23, n° 1, p. 129-145.

Lin X., Yuan Y., Zhang Q., Zhang Y. (2007). Selecting stars: The k most representative skyline operator. In *Proceedings of ICDE conference.*

Morse M. D., Patel J. M., Jagadish H. V. (2007). Efficient skyline computation over low-cardinality domains. In *Proceedings of VLDB conf.*

Papadias D., Tao Y., Fu G., Seeger B. (2005). Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, vol. 30, n° 1.

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

Pei J., Yuan Y., Lin X., Jin W., Ester M., Liu Q. *et al*. (2006). Towards multidimensional subspace skyline analysis. *ACM TODS*, vol. 31, n° 4, p. 1335-1381.

Raïssi C., Pei J., Kister T. (2010). Computing closed skycubes. *Proc. of VLDB conf.*.

Shang H., Kitsuregawa M. (2013). Skyline operator on anti-correlated distributions. *PVLDB*, vol. 6, n° 9, p. 649–660.

Siddique A., Morimoto Y. (2009). K-dominant skyline computation by using sort-filtering method. In *Proceedings of PAKDD conference.*

Siddique A., Morimoto Y. (2010). k-dominant and extended k-dominant skyline computation by using statistics. *International Journal on Computer Science and Engineering*, n° 5, p. 1934.

Siddique A., Morimoto Y. (2012). Efficient k-dominant skyline computation for high dimensional space with domination power index. *Journal of Computers*, vol. 7, n° 3, p. 608 - 615.

Xia T., Zhang D., Fang Z., Chen C. X., Wang J. (2012). Online subspace skyline query processing using the compressed skycube. *ACM TODS*, vol. 37, n° 2.