République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Licence en Informatique

Thème

Recherche des skyline à base de l'algorithme bitmap

Réalisé par :

- **M elle. Berrouigat khadidja**
- **M elle. Bensaha fatima**

Présenté le 27/05/ 2015 devant la commission d'examination composée de MM.

- *Mr. Hadjila Fethellah* *(Encadreur)*
- *Mr .Benziane Mohamed* *(Examinateur)*
- *Mr.Merzoug Mohamed* *(Examinateur)*

Année universitaire: 2014-2015

Table des matières

Liste des tables

Liste des figures

Résumé :

Le but de ce travail est la conception et l'implémentation d'algorithme de recherche de skylines.

L'operateur de Skyline est un moyen très important dans l'informatique décisionnelle, en effet, il permet de retourner un ensemble de points intéressants à partir d'un énorme espace de données. Dans ce travail nous proposons une approche de recherche de skylines basée sur une structure d'index appelée Bitmap. Cette dernière nécessite un cout de construction relativement faible par rapport au nombre de points, en plus elle garantit une complexité temporelle presque constante pour la vérification d'un point.

Mots Clés : requêtes Skylines, Algorithmes avec index, Algorithmes sans index, Bitmap

Introduction Générale

1- Contexte du travail :

Dans un contexte décisionnel, certaines requêtes ne renvoient aucun résultat. Dans ces requêtes, l'utilisateur recherche les tuples pour lesquels les valeurs de certains critères sont optimales. C'est le caractère « multicritère » de ces interrogations qui les rend généralement infructueuses. En effet, tel tuple peut être optimal pour un critère mais pas pour un autre, il est alors éliminé du résultat alors qu'il aurait pu être pertinent pour l'utilisateur .Afin d'apporter une réponse adéquate au type de requêtes décrites, l'opérateur skylines a été introduit. Il considère l'ensemble des critères d'une recherche comme étant de préférences et extrait les tuples globalement optimaux pour cet ensemble de préférences. Ainsi plutôt que de rechercher une hypothétique solution idéale, il extrait les candidats les plus proches possibles des souhaits de l'utilisateur. Son principe général s'appuie sur la notion de dominance.

2-Problématique :

L'objectif de ce travail est de répondre à la problématique de recherche des éléments skylines. En effet certain algorithmes offerts sont peu efficaces en termes de temps d'exécution, et de ce fait, l'utilisateur préfère toujours de mettre en œuvre des approches plus rapides et plus progressives.

3-Contribution :

Dans le cadre de ce mémoire, nous proposons une approche qui se base l'algorithme « bitmap ».

En informatique, un bitmap est une structure de données, utilisée dans plusieurs applications. On l'appelle aussi tableau de bits ou de bitmap index.

Le terme bitmap vient de la terminologie de la programmation informatique, ce qui signifie tout simplement une carte de bits

Notre projet consiste à définir formellement le problème de la recherche des Skylines, nous donnerons un aperçu des différents algorithmes pour calculer les points Skylines ainsi que l'approche « bitmap »

4-Plan de mémoire :

Le reste de ce mémoire est organisé comme suit :

Introduction générale

Chapitre1 : Des définitions sur le concept de Skyline ainsi que les deux grandes familles d'algorithmes avec leurs avantages et leurs inconvénients.

Chapitre2 : il présente notre prototype, et en particulier la conception et l'expérimentation de l'algorithme Bitmap.

Chapitre1:

Le concept skyline

1-Introduction :

Les requêtes skylines constituent un puissant outil d'analyse de données multidimensionnelles et d'aide à la décision. Lorsque les dimensions sont conflictuelles, les requêtes skylines retournent les meilleurs compromis associés à ces dimensions. De nombreux travaux se sont intéressés à l'extraction de points skylines dans le contexte de bases de données multidimensionnelles, ce travail présente une méthode qui accélère la recherche des skyline en se basant sur l'index bitmap.

2-Définition de Skyline :

Skyline est une opération importante dans de nombreuses applications de retourner un ensemble de points intéressants d'un potentiel énorme espace de données. Compte tenu d'une table, l'opération trouve toutes les lignes qui ne sont pas dominés par d'autres tuples.

3- L'operateur Skyline :

L'opérateur skyline été proposé par **[2]**. Avant définir de manière formelle l'opérateur skyline, il est important de bien situer le contexte dans lequel la problématique se pose. En effet, l'opérateur ne s'applique pas à n'importe quelle relation. Pour qu'il puisse effectuer les comparaisons nécessaires, il faut que les différents domaines sur lesquels sont définis les attributs, critères de choix de l'utilisateur, soient ordonnés de façon totale. Par simplicité, dans la suite du document, les attributs seront systématiquement codés sous forme de valeurs numériques.

Les tuples de nos relations pouvant être utilisées par l'opérateur skylines sont de la forme $t= (a_1, a_2,\ldots, a_k, c_1, c_2,\ldots, c_1)$ où les ***ai*** sont les attributs inutiles pour le skylines alors que les ***ci*** sont les critères sur lesquels l'utilisateur se fonde pour porter son choix.

3.1-Exemple : La relation illustrée par la table 3.1 est typique pour l'utilisation du skyline. Elle répertorie différents Smartphones. Les attributs classiques sont ici Propriétaire et les critères de choix pour trouver le « meilleur Smartphones » sont :

- le Prix de d'achat en dollars

-l'autonomie (appel) en heurs

-Résolution caméra en Megapixel

-la taille de l'écran˜

-le poids en grams

Nom	Prix	Autonomie (appel)	Résolution caméra	Taille de l'écran	Poids
Apple iphone 5	665	8H	8 Megapixel	4˜	112 Grams
Samsung galaxy S 3	550	22 H	8 Megapixel	4.8˜	133 Grams
Nokia limia 920	449	17 H	8 .7Megapixel	4.5˜	185 Grams
LG NEXUS 4	229	15H	8Megapixel	4.7˜	139Grams
Samsung Galaxy Note 2	650	35H	8Megapixel	5.5˜	180Grams
Blackberry Torch 9850	500	7H	5Megapixel	3.7˜	135Grams

Table 3.1 : Différents Smartphones et les valeurs de différents critères. [6]

3.2- Second exemple: Reprenons l'exemple du maquettiste. Celui-ci vient d'acheter 400 unités de balsa et 200 unités d'acajou. Il veut utiliser ces bois pour fabriquer des avions et des voitures miniatures et les vendre. Chacune de ces deux figurines nécessite les deux bois dans des proportions différentes.

Son premier objectif est de maximiser ses bénéfices. Toutefois, le maquettiste vient d'acquérir une machine permettant de faire une grosse partie du travail `a sa place. 70% du travail pour réaliser un avion peut être réalisé par la machine. Pour la voiture, seul 40% du travail peut être réalisé par la machine. Le maquettiste veut bien sûr maximiser le travail effectué par la machine.

Jouet	**B**alsa	**A**cajou	**P**rix de vente	**P**roportion par machine
Avion	6	1	4	70%
Voiture	2	4	7	40%

Table3.2- : Exemple de maquettiste.

4-Définition et propriétés :

Nous avons décrit les conditions d'application de l'opérateur skyline, nous en donnons à présent une définition formelle et la solution de calcul qui en découle via une requête SQL.

Notation	***Description***
C	*Ensemble de critères*
t, u	*Tuples de l'ensemble C*
r	*Relation smartphones*
S	*Ensemble de données*
d	*Ensemble des dimensions de S*
p, q	*Points de l'ensemble S*

Table 4.1- : Récapitulatif des notations.

4.1-Définition 1 (**Relation de dominance stricte**) - Soit $\boldsymbol{C = \{c_1, c_2, \ldots, c_d\}}$ l'ensemble des critères sur lesquels va porter l'opérateur skyline. Sans perte de généralité, nous considérons qu'ils doivent tous être minimisés. Soit deux tuples ***t*** et ***u*** à comparer, la relation de dominance suivant l'ensemble de critères ***C*** est définie comme suit :

$$\boldsymbol{t >_c u \iff t.c_1 < u.c_1 \text{ et } t.c_2 < u.c_2 \text{ et } \ldots \text{ et } t.c_d < u.dc_d}$$ **[15]**

Lorsque $t \geq_c u$ et $\exists\ t.c_i \neq u.c_i$ on parle d'une relation de dominance stricte, noté $t >_c u$.

4.2-Définition 3 (Dominance faible) : Un vecteur objectif $u = \{u_1, ..., up\}$ domine faiblement un autre vecteur objectif $v = \{v_1, ...v_p\}$ si et seulement si : $i \{1, ...p\}$,

$u_{ni} \leq v_i$

Ainsi dans le cadre d'une recherche multicritère les tuples dominés par d'autres (au moins un) ne sont pas pertinents et sont éliminés du résultat par l'opérateur skyline que nous définissons au debut.

4.3-Définition 4 (**L'opérateur skyline**) : A partir d'une relation ***r***, l'opérateur skyline renvoie l'ensemble des tuples qui ne sont dominés par aucun autre, suivant l'ensemble de critères *C* :

$$SKY_c\ (r) = \{t \in r \mid \nexists\ u \in r \setminus t, u >_c t\}$$ **[15]**.

4.4-Exemple : Avec notre relation exemple (cf. Table 3.1) et les critères suivants *C :{Autonomie; Prix}*, SKY_c (Smartphones) = $\{t_1\}$ car le tuple t_1 domine tous les autres. Il est donc le meilleur Smartphone possible pour l'utilisateur.

Si les tuples sont considérés comme des points dans un espace d -dimensionnel, leurs coordonnées sont alors les valeurs respectives des attributs critères du skyline. Le problème se ramène alors à celui du Maximal Vector Computation **[13, 11].**

4.5-Exemple : La figure 4.1 représente la projection de la relation exemple (cf. Table 3.1) sur les critères *C :{Autonomie; Prix}.*

En traduisant directement la définition précédente en SQL, nous pouvons calculer simplement le résultat de l'opérateur skyline. Cette requête doit réaliser les opérations suivantes :

1. comparer chaque tuple de ***r*** avec tous les autres. Une auto-jointure est donc nécessaire avec une imbrication d'un bloc corrélé ;

2. chaque tuple examiné ***t*** doit être écarté du résultat s'il existe un autre tuple ***u*** tel que ***u > t*** selon les critères définis par l'utilisateur.

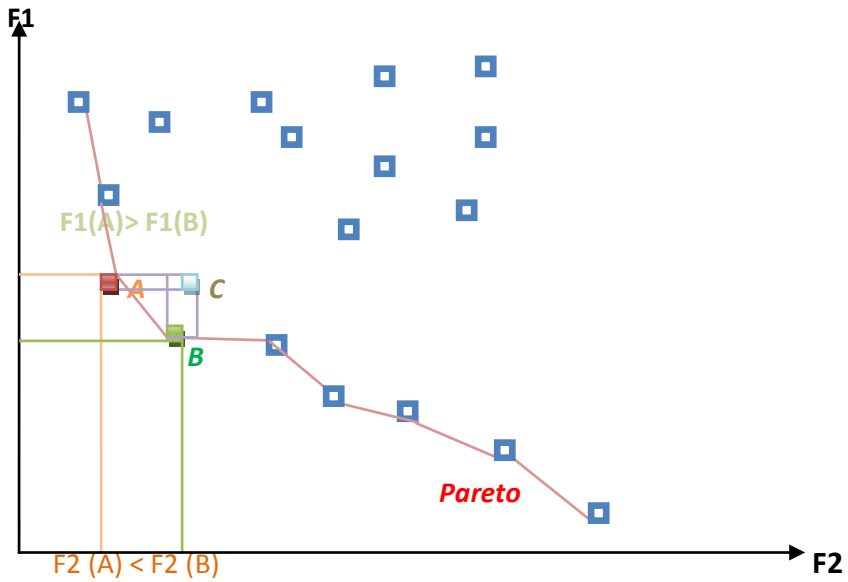

Figure 4.1-Exemple des points skyline

De plus, il est important de préciser que nous souhaitons que le calcul du skyline reste le plus général possible. Nous ne prenons pas en compte certains cas particuliers qui sont exploitables pour calculer efficacement le skyline. Il s'agit par exemple de calculs sur des attributs critères
dont les domaines ont tous de faibles cardinalités **[12]** ou bien de skyline portant sur une ou deux dimensions, cas traités dans **[1].**
Le travail présenté dans ce paragraphe porte sur l'operateur skyline, les conditions dans lesquelles il peut être appliqué et les algorithmes de mise en œuvre. Une typologie de ces algorithmes est dressée et des critères d'évaluation examinés. Pour ces critères, nous avons retenu :
- la progressivité qui vise à rendre accessibles les résultats le plus rapidement possible afin de conforter l'utilisateur dans ses choix.
- le caractère universel de l'algorithme qui ne doit pas tirer profit de telle ou telle spécificité.
- les limitations techniques, en particulier le passage à l'échelle.

Nous avons identifié deux familles d'algorithmes, avec ou sans méthode d'indexation, implémentant l'opérateur skyline. Nous avons étudié les algorithmes les plus pertinents par rapport aux critères d'évaluation retenus et, le cas échéant, les structures de données sous-jacentes.

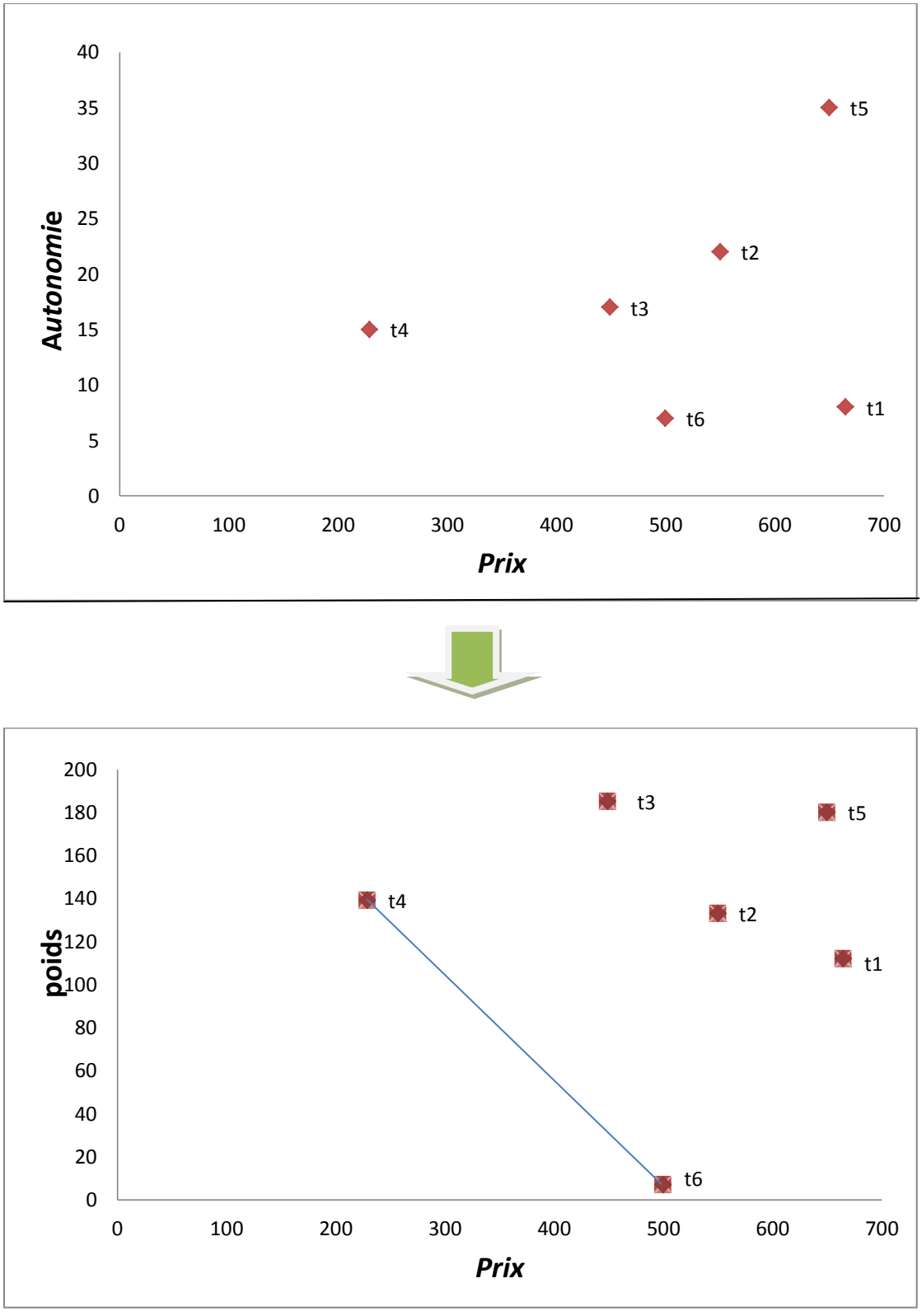

Figure 4.2 - Représentation géométrique de la projection de la relation Smartphones sur deux axes.

5- Skylines vs points ordinaire :

Nous nous intéressons dans ce qui suit aux critères que nous avons utilisés pour évaluer et comparer ces algorithmes.

5.1-Les critères d'évaluation :

Avant de présenter différents algorithmes, on doit pouvoir les comparer sur des critères identiques afin de mettre en évidence leurs forces et leurs faiblesses. **[10]** en proposent une liste sur laquelle nous nous appuyons sans la suivre strictement car certains points sont contradictoires et d'autres peu pertinents dans le cadre fixé. Voici la liste des critères choisis :

-Universalité

L'algorithme doit utiliser des structures standard facilement intégrables au cœur d'un SGBD.

Ce critère a trait à l'efficacité mais surtout au passage à l'échelle des algorithmes. Il s'agit de donner les limites, par exemple au nombre de critères du skyline et/ou à la taille de l'entrée, au delà desquelles l'algorithme n'est plus utilisable.

-Préférences

Le calcul des skylines intègre les préférences utilisateurs.

- Progressivité

Les volumes des bases de données sont souvent énormes. Un simple parcours de la base peut prendre un temps non négligeable. C'est pourquoi nous souhaitons que nos algorithmes calculent et affichent les résultats trouvés le plus rapidement possible, au fur et à mesure qu'ils sont découverts. Idéalement, les premiers résultats devraient être envoyés quasiment instantanément.

- Correction

Tous les points retournés sont des skylines.

- Complétude

Tous les points du skylines sont retournés à la fin.

6-Les algorithmes de recherche de skylines

Les algorithmes qui vont être détaillés maintenant peuvent se classer en deux catégories. Les premiers n'utilisent pas de structure d'indexation particulière et n'ont donc pas besoin d'un pré-traitement pour fonctionner. Les seconds regroupent tous ceux qui sont basés sur des index.

Ils les exploitent pour améliorer l'efficacité du calcul et modifier leur comportement, vis-à-vis par exemple de leur progressivité. La contrepartie est toutefois qu'il faut

disposer de l'index nécessaire, ou bien le cas échéant de devoir le calculer, avant de pouvoir d'utiliser ces algorithmes.

6.1-Algorithme de recherche dans un espace complet :

Nous constatons deux familles d'algorithmes selon qu'ils utilisent ou non les méthodes d'indexation. Les premiers n'utilisent pas de structure de données particulière, et donc la recherche des skylines se fait en analysant entièrement la base de données au moins une seule fois. Les seconds algorithmes regroupent ceux qui exploitent des index pour améliorer l'efficacité de calcul et d'accélérer davantage les requêtes skylines, en diminuant par exemple le nombre de tests de dominance.

6.1.1-Les algorithmes sans index :

6.1.1.1-L'algorithme Block-Nested-Loop « BNL » : L'idée directrice de cet algorithme **[1],** extension de l'algorithme Nested-Loops **[2,7]**, est de limiter le nombre d'E/S effectuées en chargeant en mémoire un ensemble de tuples candidats au Skyline appelé « fenêtre ». Les accès disque sont faits par bloc, les comparaisons ont lieu directement en mémoire centrale et sont donc bien plus rapides. Cependant, cette fenêtre a une taille limitée. De plus, comme la taille du Skyline est du même ordre de grandeur que la relation d'entrée, il est vraisemblable que la fenêtre ne puisse pas accueillir tous les candidats.

C'est pour cela que l'algorithme gère, en plus de la fenêtre, un fichier temporaire stocké en mémoire secondaire dans lequel sont écrits tous les candidats non considérés faute de place. Ils seront traités lors d'une prochaine itération.

Examinons les différents cas de figure qui se présentent lors de la comparaison d'un tuple *t* avec ceux de la fenêtre, trois cas de figure s'expliquent :

1. *t* est dominé par un tuple de la fenêtre

t est alors directement écarté et ne sera plus pris en compte pour le reste du calcul : il est dominé et ne fera donc jamais partie du Skyline.

2. *t* domine un ou plusieurs tuples de la fenêtre

Les tuples de la fenêtre dominés par *t* sont écartés et ne seront plus pris en compte pour les itérations futures. *t* est inséré dans la fenêtre sans problème, puisqu'il y a au moins une place libre.

3. *t* est incomparable avec l'ensemble des tuples de la fenêtre

C'est le cas le plus problématique : ***t*** doit être ajouté à la fenêtre mais il n'a éliminé aucun point sur son passage et il se peut que celle-ci soit pleine. Si ce n'est pas le cas, ***t*** est inseré normalement à la fenêtre. Sinon, ***t*** est mis de côté dans le fichier temporaire et sera examiné à nouveau au cours de la prochaine itération de l'algorithme.
D'prés ce dernier cas de figure, nous pouvons conclure que l'algorithme BNL peut nécessiter un grand nombre d'itération avant que le skyline final soit calculé.

Cet algorithme fonctionne particulièrement bien si le Skyline est petit et qu'il n'y a pas beaucoup de candidats à la fois : le fichier temporaire n'est alors pas utilisé puisqu'il y a toujours de la place dans la fenêtre. L'algorithme se termine ainsi après une seule itération, avec sa meilleure complexité *O(n)*, avec *n* le nombre de tuples de l'entrée à examiner. Inversement, dans le pire des cas, sa complexité est quadratique $O(n^2)$. Il reste cependant toujours bien plus efficace que la requête SQL puisqu'il limite considérablement les E/S, la fenêtre résidant en mémoire centrale.

6.1.1.2-L'algorithme Divide-and-Conquer « DC » :

La méthode DC est développée par **[1].** Pour appréhender le fondement principal algorithme, on modélise le problème de l'opérateur skyline de manière géométrique. Les tuples à examiner sont alors considérés

Comme des points, dont les coordonnées sont les valeurs numériques des attributs-critères du skyline.
L'idée de base de cette méthode est de travailler de manière récursive en divisant l'espace en des sous-espaces plus petits jusqu'à rendre le calcul du skyline trivial. Puis, il faut reconstruire le résultat final en réunissant tous les skylines partiels. Voici les trois étapes principales de l'algorithme :

1. **Calculer la médiane**
2. **Calculer les Skylines**
3. **Fusionner les Skylines**

La complexité de cet algorithme **[13,11]** est de l'ordre de $O(n. Log\ n)^{d-2}) + O(n .log\ n)$ (avec d le nombre de dimensions sur lesquelles porte le skyline et n le nombre de tuples en entrée). Ce résultat est meilleur que la complexité en $O(n^2)$ de la requête

SQL et de BNL. Cet algorithme se comporte donc en général bien mieux que BNL lorsque que le résultat est de grande taille et qu'il ne tient pas en mémoire.

6.1.1.3-L'algorithme Sort First Skyline(SFS) :

Cet algorithme est une version améliorée de l'algorithme BNL. Proposé par les auteurs de **[4]**. Il ordonne toutes les données en entrée à l'aide d'une fonction de score (monotone) correspondant aux préférences des utilisateurs (ex. somme des valeurs des dimensions,….etc.).En ordonnant l'ensemble des points, il assure qu'aucun de ces derniers ne peut être dominé par un point ayant un score inférieur au sien, et cela grâce à la propriété de préservation de la relation de dominance avec tout fonction monotone.

Par conséquent, chaque point chargé en mémoire centrale peut immédiatement être retourné comme point Skyline. Le nombre de passes nécessaires à effectuer sur le disque pour lire les données est alors égal à la taille de l'ensemble Skyline final divisé par la taille de la mémoire centrale (en terme de nombre de point pouvant être chargé à la fois). L'algorithme SFS permet donc de retourner les points Skylines de manière progressive et diminue le nombre de comparaisons entre les points.

7-**Les algorithmes avec index :**

Plusieurs types d'indexation ont été exploités pour mettre au point des algorithmes de calcul du Skyline. Nous allons brièvement présenter les deux grandes familles d'index :

7.1- Les index « classiques »

Il s'agit des index couramment utilisés dans les bases de données, tels que les **B-Trees** ou bien les index Bitmap. Deux algorithmes se basent sur ces index: le premier nommé Index et le second Bitmap, présentés dans **[14].** Ils ne sont pas étudiés dans paragraphe car ils présentent de nombreux désavantages. Le plus important d'entre eux est que l'index pré-calculé est adapté pour une demande particulière de Skyline. Ainsi, si l'utilisateur souhaite formuler plusieurs requêtes Skyline, il sera nécessaire de calculer un index spécifique pour chacune d'elles.

7.2- Les index « spatiaux »

Comme nous l'avons vu avec l'algorithme Divide and Conquer, appréhender le problème du calcul du Skyline dans sous son aspect géométrique permet d'exploiter certaines propriétés pour optimiser le temps de réponse des algorithmes. Développés dans le contexte des bases de données géographiques, les index spatiaux, permettent d'exploiter pleinement certaines de ces propriétés.

Nous allons dans ce paragraphe étudié les index spatiaux les plus utilisés pour le calcul du Skyline : les R-Trees **[8].**

Les R-Trees : Il s'agit d'une structure de données conçue pour indexer spatialement des objets. Cette structure est particulièrement performante lorsque le nombre de dimensions n'est pas trop important, c'est typiquement le cas pour les requêtes Skyline. En effet si le nombre de critères est trop important, le nombre de résultats de l'opérateur devenant très grand, l'interprétation de ces résultats devient plus délicate. Si l'utilisateur souhaite malgré cela prendre en compte un grand nombre de critère, d'autres structures d'indexation seront plus adaptées, comme par exemple les X-Trees **[3],** les SH-Trees **[5],** . . .

7.3-L'algorithme Bitmap :

L'algorithme Bitmap convertit chaque point p de l'ensemble de données en un vecteur de bits, représentant le nombre de point ayant une plus petite coordonnée que p dans chaque dimension. La longueur du vecteur de bits est déterminée par le nombre de valeurs distinctes sur toutes les dimensions. Les points Skylines sont alors obtenus en utilisant uniquement des opérations binaires sur les vecteurs à bits, permettant une optimisation considérable des temps de calcul des Skylines. De plus, l'algorithme Bitmap retourne les points Skylines au fur et à mesure de leurs calculs (i.e. c'est un algorithme progressif). Cependant, l'algorithme Bitmap présent un certain nombre de désavantages qui limitent considérablement son application. En effet, la consommation mémoire due à la conversion des points en structure Bitmap peut s'avérer prohibitive pour l'algorithme en présence d'un grand nombre de valeurs distinctes sur les dimensions. Autrement dit, l'algorithme n'est efficace qu'en présence de dimensions à domaine réduit de valeurs. Bitmap gère aussi de manière

inefficace les mises à jour. En effet, chaque ajout, suppression ou modification d'une dimension ou d'un point implique le recalcule de tous les vecteurs de bits.

Id	Coordinat	Bitmap représentation
A	**(1,9)**	**(1111111111,1100000000)**
B	**(2,10)**	**(1111111110,1000000000)**
C	**(4,8)**	**(1111111000,1110000000)**
D	**(6,7)**	**(1111100000,1111000000)**
E	**(9,10)**	**(1100000000,1000000000)**

Table 7.1- : Exemple de l'algorithme de Bitmap.

A(4,8)

A_Y=1011011111111

A_X&A_Y=1000000000000

Le premier dans le résultat indiquent les points dominante A.

7.4-L'algorithme Index :

L'indice approche organise un ensemble de dimension ***d*** des points dans listes d tels qu'un point p = (p_1, p_2, ..., p_d) est attribué à la i-ème liste ($1 \leq i \leq d$), si et seulement si, sa coordonnée pi sur l'axe ième est le minimum parmi toutes les dimensions, ou formellement: $p_i \leq p_j$ pour tout $j \neq i$. Les points de chaque liste sont triés par ordre croissant de leur minimum coordonner (Minc, pour faire court) et indexé par un B-tree. Un groupe de points pour chaque liste, représente l'ensemble des points ayant la même valeur sur la dimension correspondante. Bien que cette technique peut revenir rapidement les points Skylines en haut des listes, l'ordre dans lequel ces points sont extraits est fixé sans prise en compte des préférences définies par l'utilisateur **[10].**

La liste1		La liste 2	
a(1,9)	Min c=1	k(9,1)	Min c=1
b(2,10)	Min c=2	i(3,2), m(6,2)	Min c=2
c(4,8)	Min c=4	h(4,3), n(8,3)	Min c=3
g(5,6)	Min c=5	l(10,4)	Min c=4
d(6,7)	Min c=6	f(7,5)	Min c=5
e(9,10)	Min c=7		

Table 7.2- : Exemple de l'algorithme d'index.

7.5-L'algorithme Nearest Neighbor « NN » :

L'algorithme « NN » a été la première méthode développée pour résoudre le problème Skyline en exploitant les index spatiaux. Ce dernier est basé sur les requêtes de plus proche voisinage, il utilise une constatation géométrique très simple (on considère sans perte de généralité que les points intéressants sont ceux qui sont proches de l'origine du repère et qu'on utilise une distance de Manhattan) : La zone spatiale que domine un tuple $i = (c_1, c_2, \ldots, c_d)$ est la zone $[c_1, \infty) [c_2, \infty) \ldots [c_d, \infty)$. Nous la nommerons par la suite la région de dominance d'un tuple. L'algorithme NN a été proposé dans **[10]**. L'algorithme commence en cherchant le point le plus proche de l'origine du repère. Ce dernier fait forcément partie du Skyline puisqu'aucun autre point n'a pu le dominer. À chaque étape, il met à jour une liste de zones qu'il devra explorer nommée « to-do liste ». Cette liste est maintenue en mémoire principale tout au long de l'algorithme. Après la découverte du premier point, il ajoute les zones non dominées dans la liste.

7.6-Exemple : La figure 7.1 illustre le concept de région de dominance en considérant le tuple t_1 dans un exemple de logement avec les deux critères : prix et autonomie. Tous les tuples contenus dans la zone grisée sont dominés par t_1.

Lors de la découverte d'un nouveau point, l'algorithme utilise la susdite propriété, pour éliminer des régions de l'espace à ne plus explorer. Ainsi, il suffit de chercher uniquement dans les régions susceptibles de contenir d'autres points candidats.

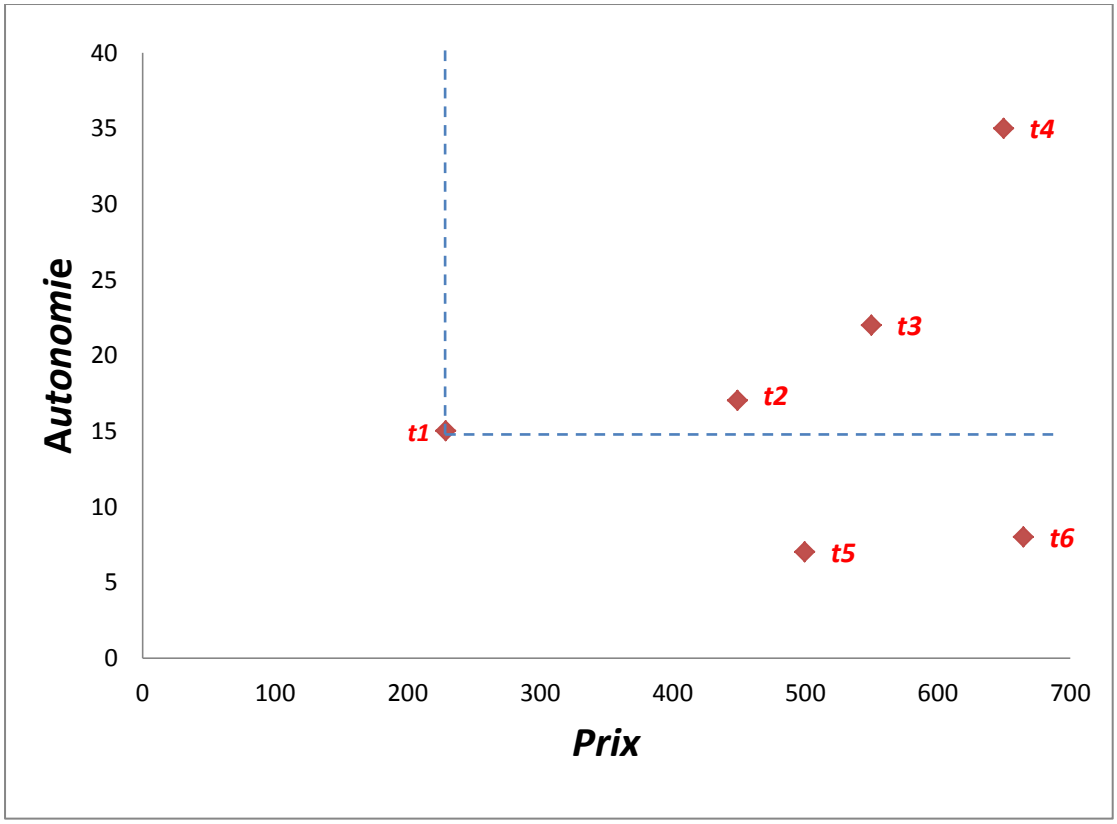

Figure7.1- : Illustration de l'algorithme NN.

Il va maintenant explorer les régions de la to-do liste, tant que celle-ci n'est pas vide. Aussi, remarque-t-on qu'une zone explorée vide ne partitionnera pas l'espace à son tour et en conséquence, n'agrandira pas la to-do liste. Ce n'est que la découverte d'un nouveau point qui partitionne l'espace.

En général, pour un espace *d*-dimensionnel, chaque nouveau point Skyline découvert ***p*** est à l'origine de ***d*** appels récursifs de l'algorithme, une nouvelle zone de recherche étant ajoutée pour chaque coordonnée de ***p***. Plusieurs problèmes se posent. Tout d'abord, pour $\mathbf{d \geq 3}$, certaines zones

de l'espace peuvent être recherchées de multiples fois.

Pour conclure, NN est un bon algorithme pour les Skylines de faible dimension. Toutefois il a de très sérieux défauts qui l'empêchent d'être exploitable dés que le nombre de dimension dépasse **4**.

7.7-L'algorithme Branch-and-Bound Skyline « BBS »:

Le dernier algorithme ***BBS***, est développé dans le même esprit que NN puisque lui aussi est basé sur des recherches de plus proches voisins. En effet, celui-ci part de la racine du R-Tree et l'explore en descendant seulement dans les branches qui lui sont utiles. C'est pour cela que l'on dit qu'il est optimal en nombre d'E/S **[14]**. Sa recherche des plus proches voisins ressemble à l'algorithme
best-first nearest neighbor **[9].**

Pour parvenir à ses fins, l'algorithme utilise un tas qui va contenir les entrées (feuilles/nœuds intermédiaires) de l'arbre pris en considération, triées selon la distance de leur MBR par rapport à l'origine du repère. À chaque fois le premier nœud est examiné et ses entrées non dominées par des points Skyline déjà trouvés sont ajoutées au tas. Ensuite, l'algorithme retire l'entrée ayant la plus petite valeur MBR du tas et insère ses nœuds fils comme entrées dans ce dernier. Lorsque l'entrée courante est un point non dominé, il est directement ajouté au Skyline et tous les nœuds/ sous-arbres qu'ils dominent sont définitivement élagués du R-Tree .l'algorithme s'arrête quand le tas est vide.

Prouve que l'algorithme ***BBS*** est correct et optimal vis à vis des E/S. Il constitue la méthode de calcul la plus efficace que nous ayons étudiée pour le calcul du Skyline. Elle sera donc appropriée s'il s'agit de calculer de multiples Skylines.

8-Comparaison :

Dans la section précédente, nous avons étudié les meilleurs représentants des deux grandes familles d'algorithmes (avec et sans index) remédiant à ce problème d'efficacité. Pour pouvoir opérer des choix entre ces différents algorithmes, dans ce qui suit, nous avons pris en considération plusieurs critères d'évaluation (un tableau récapitulatif), notamment la progressivité, incontournables pour notre contexte de travail.

Dans cette comparaison les algorithmes qui ne sont pas inclues sont les algorithmes de calcul de skyline dans des sous espaces car ses méthodes se basent sur les algorithmes de calculs de skyline dans un espace complet a fin d'extraire leur skyline.

Nous constatons que touts les algorithmes vu précédemment respectent correctement et parfaitement les critères correction, complétude et universalité en retournant le skyline exact.

On remarque que les algorithmes utilisant les index spatiaux sont plus efficaces que les autres concernant le critère d'efficacité, pour le critère de progressivité , le comportement progressif implique un pré-traitement des données en entrées, c'est-à-dire l'utilisation d'index (NN, BBS, et Bitmap) ou le tri des données (SFS). Ces pré-traitements peuvent être réutilisés par toutes les requêtes ultérieures de manière dynamique avec la structure correspondante.

Les algorithmes NN et BBS utilisent des structures d'index dynamique (R-Tree), pouvant être mises à jour de manière incrételmentale, ce ci est pour la gestion des mises à jour des données, par contre la maintenance utilisé par l'algorithme SFS peuvent être améliorées a l'aide d'un B-Tree. En revanche l'algorithme bitmap n'est adapté qu'à des ensembles de données statiques car une insertion / suppression d'un point peut engendrer la modification de la représentation bitmap de nombreux autre points. Pour le critère de préférences , a l'exception de NN, qui peut intégrer des préférences utilisateurs en personnalisant la distance utilisée dans la fonction de score , aucun des autres algorithmes ne prend en charge ou n'intègre de préférence utilisateurs.

Et pour le dernier critère d'universalité, nous constatons qu'il est respecté pour tous les algorithmes.

Algorithme	**Efficacité**	**Progressivité**	**Préférences**	**Correction**	**Complétude**	**Universalité**
BNL	Non	Non	Non	Oui	Oui	Oui
D&C	Non	Non	Non	Oui	Oui	Oui
SFS	Non	Oui	Non	Oui	Oui	Oui
Bitmap	Non	Oui	Non	Oui	Oui	Oui
NN	Oui	Oui	Oui	Oui	Oui	Oui
BBS	Oui	Oui	Non	Oui	Oui	Oui

Table 8.1- : Comparaison des algorithmes de calcul de Skyline dans un espace complet [15].

9-Conclusion :

Dans ce chapitre, nous avons décrit les algorithmes de recherche de skyline, ainsi que les différents critères qui permettent leur évaluation, dans le chapitre suivant nous présentons une approche particulière dénommée **Bitmap**.

Chapitr2 :
l’algorithme bitmap

1-Introduction :

Les approches de calcul de skyline ont été publiées en détail dans les grandes conférences de base de données y compris SIGMOD, VLDB et ICDE. Et pour cela, nous considérons un algorithme performent « Bitmap » qui se base sur la transformation des points en vecteurs de bits et toute l'information forme la structure de bitmap.

2-L'utilisation de l'algorithme bitmap :

Les bases de données qui font l'objet de l'extraction de connaissances sont de plus en plus volumineuses. Les algorithmes classiques de fouille de données peuvent alors être poussés à leurs limites. Intégrer les méthodes de fouille de données au cœur des Systèmes de Gestion de Bases de Données (SGBD) est une solution qui peut repousser ces limites. Pour cela en propose un algorithme efficace qui se base sur « bitmap ». La structure de bitmap reflète l'ordre des enregistrements de données dans toutes les dimensions, ignorons leurs grandeurs. Bitmap prend en charge le calcul d'horizon progressive depuis un point peut être retourné à l'utilisateur immédiatement une fois qu'il est identifié comme un membre de skyline.

L'utilisation de bitmap est justifiée. En effet, l'algorithme bitmap est muni de propriété très intéressante, que ce soit au niveau de leur stockage, de leur possibilité en matière de comptage, ou encore des opérations "bit à bit".
La requête peut se complexer un peu dans le cas où il y a des conditions sur plusieurs données. Auquel cas, avant d'effectuer un comptage, il faut transcrire cette multiple condition en utilisant l'opérateur 'AND', pour combiner les différents bitmaps concernés. Nous allons donc détailler l'opérateur 'AND' qui permet de répondre à ce type de requêtes, sans qu'il n'y ait de retour sur les données brutes.
Alors, nous utilisons l'opérateur 'AND' ou bien le 'ET' logique pour construire le bitmap résultat.

3-conception de l'approche Bitmap :

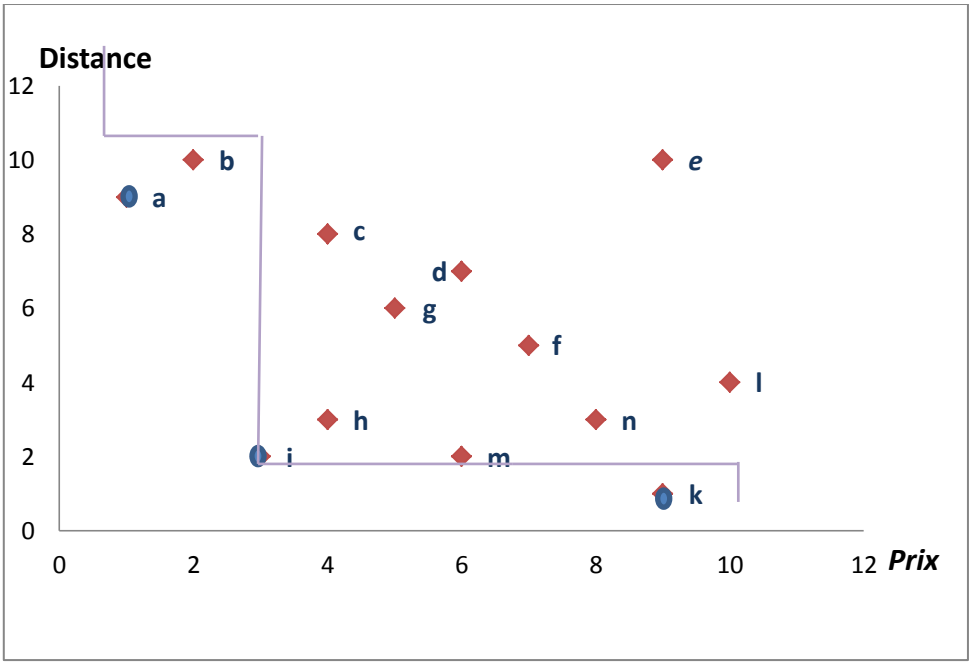

Figure 3.1- : Exemple des données skyline.

L'algorithme étudié code dans un bitmap toutes les informations nécessaires pour décider si un point est dans le Skyline. Un point de données ($p = p_1, p_2, ..., p_d$), où **d** est le nombre de dimensions, est mappé à un vecteur de **m** bits, où **m** est le nombre total de valeurs distinctes sur toutes les dimensions. Soit $\mathbf{k_i}$ le nombre total de valeurs distinctes sur la ième dimension (à savoir, $m= \Sigma_{i=1\sim d} k_i$). Dans la figure 1.1, par exemple, ils ya

$\mathbf{k_1 = k_2 = 10}$ valeurs distinctes sur les axes x-, y-dimensions et **m = 20**.

Supposons que $\mathbf{p_i}$ est le plus petit nombre de $j_i$éme sur l'axe de i; puis, il

$\mathbf{a_2}$ **(= 9)** est le 9-ième le plus petit sur l'axe des **y**, les premiers 10-9 + 1 = 2 bits de sa représentation sont **1**, tandis que les autres sont égaux à **0**.

id	cordonné	représentation bitmap
a	(1,9)	(1111111111, 1100000000)
b	(2,10)	(1111111110, 1000000000)
c	(4,8)	(1111111000, 1110000000)
d	(6,7)	(1111100000, 1111000000)
e	(9 ,10)	(1100000000, 1000000000)
f	(7,5)	(1111000000, 1111110000)
g	(5,6)	(1111110000, 1111100000)
h	(4,3)	(1111111000, 1111111100)
i	(3,2)	(1111111100, 1111111110)
k	(9,1)	(1100000000, 1111111111)
l	(10,4)	(1000000000, 1111111000)
m	(6,2)	(1111100000, 1111111110)
n	(8,3)	(1110000000, 1111111100)

Table 3.1- : Exemple de l'algorithme Bitmap.

Considérons maintenant que nous voulons décider si un point, par exemple, avec **c** représentation bitmap **(1111111000, 1110000000),** appartient à la skyline. Les bits les plus significatifs dont la valeur est **1**, sont les **4éme** et le **8éme**, dimensions **x** et **y**, respectivement. L'algorithme crée deux bits cordes, $\mathbf{C_X}$ **= 1110000110000** et $\mathbf{C_Y}$ **= 0011011111111**, par juxtaposant les bits correspondants (par exemple, 4éme et 8éme) de chaque point.

Dans la table 3.1, ces bits cordes (indiqués en couleur) contiennent **13 bits** (un à partir de chaque objet, commençant par **a** et terminant par **n**). Les 1 dans le résultat de $\mathbf{C_x}$ **&** $\mathbf{C_y}$ **= 0010000110000**, indiquer les points qui dominent **c**, à savoir, **c**, **h** et **i**. De toute évidence, s'il ya plus d'un seul **1**, le point considéré est pas dans la skyline[2]. Les mêmes opérations sont répétées pour chaque point dans le jeu de données, pour

5-Outils et méthode de développement :

Pour expliquer l'implémentation de notre application, nous allons tout d'abord présenter les outils utilisés

5.1-Java :

Notre application est développée à base de langage de programmation Java version 1.6.

Java est un langage de programmation informatique orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy , présenté officiellement le 23 mai 1995 au *SunWorld*. Nous avons préféré le choix de ce langage pour notre application à cause de sa portabilité ainsi que sa performance

5.2-NetBeans :

Notre application est écrite en NetBeans version 6.8. NetBeans est l'environnement de développement intégré, placé en open source par Sun en juin 2000 sous licence CDDL et GPLv2.En plus de Java NetBeans permet également de supporter différents autres langage comme python, C, C++, Java Script, XML, PHP, HTML.de façon native ainsi que bien d'autres (comme python ou Ruby) par l'ajout de greffons. Il comprend toute les caractéristiques d'un IDE moderne .les développeurs peuvent utiliser des applications complexes plus rapidement et de concevoir des autres qui résisteront a l'épreuve du temps. Pour cela le choix de NetBeans reste important et très utilisables.

6- Expérimentation :

Afin d'évaluer la performance de notre application nous avons mené une expérience. L'objectif de cette application est de démontrer comment on peut obtenir des points skyline à base de l'algorithme Bitmap. Nous avons développé notre prototype sous NetBeans IDE de Sun Microsystems, la machine d'expérimentation possède la caractéristique suivante :

Hp version Invent 630.

- Le système d'exploitation est Windows 7(32bit).
- Processeur Intel (R) Core (TM) 3CPU M380.
- 2 Giga de RAM.

7-Prototype :(Les fenêtres) :

Dans cette section nous présentons les interfaces graphiques de notre prototype.

7.1-La création de la base de données.

Nous avons créé des boutons définis comme suit :

Ok : récupère la dimension et le nombre d'exemple.

Random : pour générer la base d'exemple aléatoire (afficher les points graphiquement).

Maj bitmap : réaliser de la base Bitmap.

Skyline : pour la recherche des points skyline (application du ET logique entre les deux colonnes (si le resultat contient un 1 alors le point n'est pas un skyline sinon c'est un skyline).

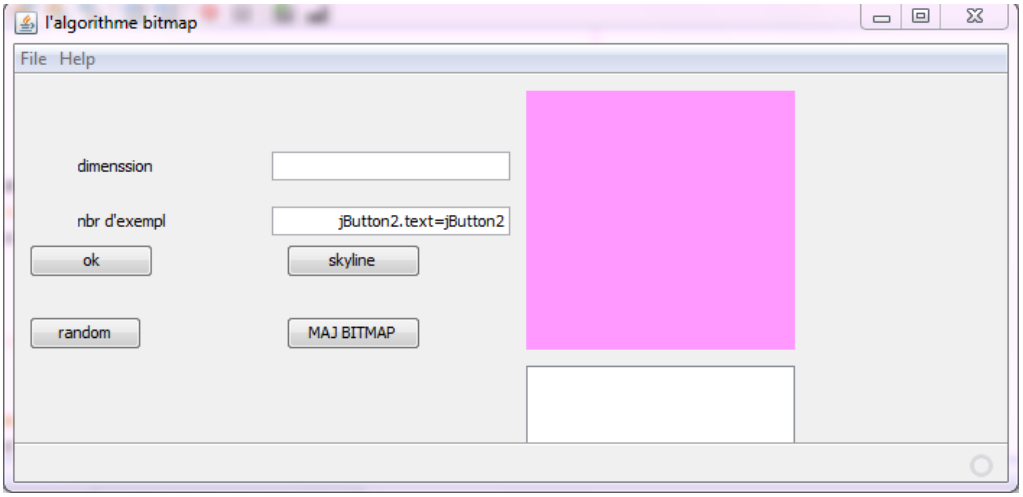

Figure 7.1- : La base de données.

7.2- :L'exécution :

1- On a entré le nombre de dimension(2) et le nombre d'exemple(10)

2-on exécuter random :

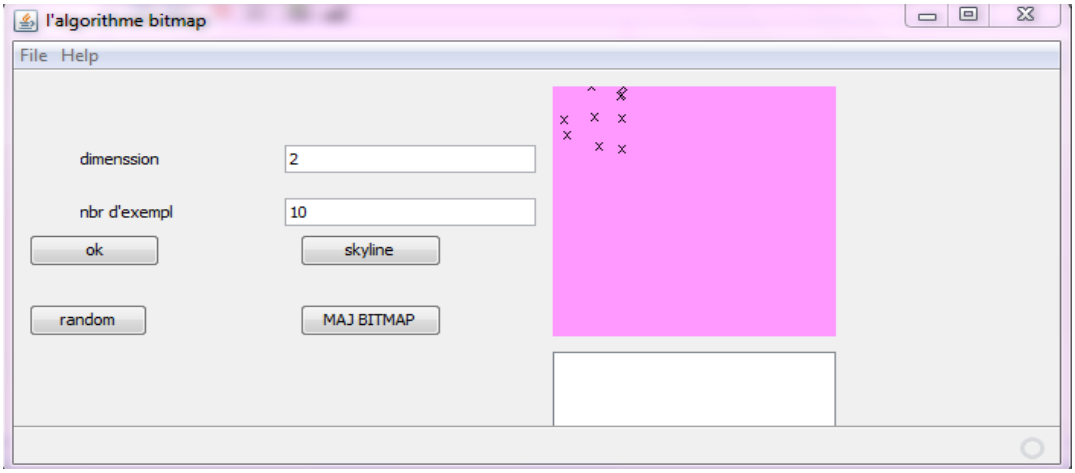

Figure 7.2- : l'exécution de random.

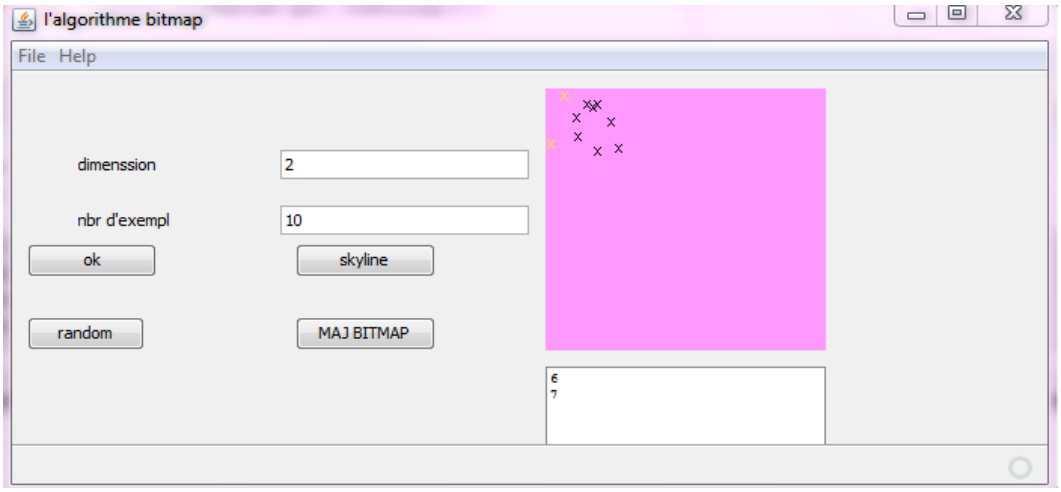

Figure 7.3- : le résultat de l'exécution de Skyline.

Résultat : les points 6 et 7 se sont des pointes skylines, elles sont colorées graphiquement

Afin de savoir le temps d'exécution de chaque nombre d'exemple, on exécute random, MAJBITMAP et Skyline et on reporte les résultats dans le tableau suivant :

dimension	Nombre d'exemple	Temps de calcul de bitmap (ms)	Temps d'exécution(Skyline) (ms)
2	10	0,66	10
2	20	10	20
2	30	10	10
……	……	……	……
2	100	30	80

Table 7.1- : l'évaluation de l'application

7.2.1-Discutions :

On remarque le temps de vérification d'optimalité d'un point est presque constant, et il est toujours inférieur ou égal à 1 milli.sec. En plus de calcul des skylines est linéaire par rapport au nombre de points.

Nous constatons aussi que le temps de calcul de la structure bitmap, est toujours acceptable, dans le sens ou l'augmentation du nombre d'exemple (ou points), par un facteur multiplicatif K engendre une légère augmentation en termes de temps d'exécution, par exemple une base d'exemple ayant 10 points nécessite 0.66 mill.sec alors qu'une base d'exemples ayant 30 points nécessite seulement 10 milli.sec.

8-Conclusion :

Au cours de ce chapitre, nous avons présenté un algorithme qui se base sur la transformation des points en vecteurs de bits et qui permet de retourner les points Skylines de manière progressive et dans un temps presque fixé.

Conclusion générale

Dans ce mémoire, nous avons proposé un tour d'horizon de l'analyse des bases de données avec l'operateur Skyline. Ce dernier permet à l'utilisateur d'extraire les solutions satisfaisant au mieux un ensemble de critères spécifiés. Après une description du contexte d'utilisation de cet operateur, nous avons étudié sa définition, ainsi qu'une solution « bases de données » sous forme d'une requête SQL.

Aussi nous avons étudie les meilleurs représentants des deux grandes familles d'algorithmes (avec et sans index) de calcul de **Skyline.**

Nous avons également conçu et implémenté l'algorithme de **Bitmap.**

Nous avons pris en considération plusieurs critères d'évaluation, notamment la progressivité, incontournables pour notre contexte de travail.

Comme perspectives à ce projet nous proposons l'approche **SkyCube**, cette structure est l'ensemble de tous les **Skylines** dans tous les sous-espaces non vides possibles.

References Bibliographiques:

[1] Borzsonyi, S., Kossmann, D. et Stocker, K. The skyline operator. In ICDE2001 (2001), pages 421-430,(2001).

[2] Berchtold, S., Bohm, C., Keim, D. A. et Kriegel, H.-P. (1997). A cost model for nearest neighbor search in high-dimensional data space. In Mendelzon et Ozsoyoglu (1997), pages 78-86. Chairman-Mendelzon, Alberto and Chairman-Özsoyoglu, Z. Meral.1

[3] Berchtold, S., Keim, D. A. et Kriegel, H.-P. (1996). The x-tree : An index structure for high-dimensional data. In Vijayaraman et al. (1996), pages 28-39.1

[4] Chomicki, J., Godfrey, P., Gryz, J. et Liang, D.Skyline with presorting: Theory and optimizations. In Proc of Intelligent Information Systems, pages 595-604.Spring Berlin /Heidelberg, (2005).

[5] Dang, T. K. (2006). The sh-tree: A novel and exible super hybrid index structure for similarity search on multidimensional data. IJCSA, 3(1):1-25.1

[6] Find The Best. Best Smartphones comparison - reviews and rating—findthebest, 2013. [Online; accessed 9-February-2013].

[7] Grust, T., Koger, J., Gluche, D., Heuer, A. et Scholl, M. H. (1997). Query evaluation in croque - calculus and algebra coincide. In Small et al. pages 84-100, (1997).1

[8] Guttman, A. (1984). R-trees : A dynamic index structure for spatial searching. In, Yormark (1984), pages 47-57.1

[9] Hjaltason, G. R. et Samet, H. Distance browsing in spatial databases. ACM TODS,Trans. Database Syst , 24(2):265-318, (1999).

[10] Kossmann, D., Ramsak, F. et Rost, S. (2002). Shooting stars in the sky : An online algorithm for skyline queries. In Very Large Data Bases, pages 275-286, (2002).

[11] Kung, H. T., Luccio, F. et Preparata, F. P. On finding the maxima of a set of vectors. Journal of the ACM , 22(4):469-476, (1975).

[12] Morse, M. D., Patel, J. M. et Jagadish, H. V. (2007).Efficient skyline computation over low-cardinality domains. In Proceedings of the 33rd international conference on Very large data bases,pages 267-278, (2007).

[13] Preparata , F. P., Shamos, M. I. Computational Geometry -An Introduction . Springer (1985).

[14] Papadias, D., Tao, Y., Fu, G. et Seeger, B. (2003). An optimal and progressive algorithm for skyline queries. In Proc of the 2003 ACM SIGMOD international conference on Management of data, pages 467-478, (2003).

[15] Sébastien NEDJAR, Cubes Émergents pour l'analyse des renversements de tendances dans les bases de données multidimensionnelles, thèse de doctorat-, Université de Marseille-France2009).

[16] Tan,K., Eng, P. Ooi, B.Efficient Progressive Skyline Computation. VLDB, 2001.