Université Abou Bekr Belkaid
Tlemcen Algérie
جامعة أبي بكر بلقايد
تلمسان الجزائر

**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 Skylines à base de l'algorithme Divide & Conquer

**Réaliser par:**

- **Kandouci Abdessamad.**
- **Hamimed Yazid**.

*Présenté le 26 Mai 2015 devant la commission d'examination composée de MM.*

- ***Mr F. Hadjila*** ***(Encadreur)***
- ***Mme N . I L E S*** ***(Examinateur)***
- ***Mme D. Berramdane*** ***(Examinateur)***

**Année universitaire: 2014-2015**

# *Remerciement*

*On remercie dieu le tout puissant de nous avoir donné la santé et la volonté d'entamer et de terminer ce mémoire.*

*Tout d'abord, ce travail ne serait pas aussi riche et n'aurait pas pu avoir le jour sans l'aide de l'encadrement de Mr. Haadjila Fethalla, on le remercie pour la qualité de son encadrement exceptionnel, pour sa patience, sa rigueur, son extrême gentillesse et sa disponibilité durant notre préparation de ce mémoire.*

*Nos remerciement s'adresse également à tous les enseignants du département d'informatique qui ont fait tout leur possible.*

*Aux membres du jury :*

- *Mme D.Iles*
- *Mme D.Berramdane*

*Qu'ils soient remerciés de nous avoir fait l'honneur de juger notre travail…*

*Nos profonds remerciements vont également à toutes les personnes qui nous ont aidés et soutenue de prés ou de loin*

# *Dédicaces*

*Le prophète Mohammed dit:*
*((De fil de chercher un moyen par lequel Dieu est le moyen facile de faire Paradise))*

*A mes parents, mes frères Amine, Abderrahmane et Fethallah pour leur présence et soutien moral.*

*A mon binôme Yazid et toute la famille Hmimed.*

*A mes amies dont j'ai bien aime leur présence dans ma vie Houssam,Ibrahim et Boumediène, pour leur conseilles, aides et leur encouragement*

*Abdessamad*

# *Dédicaces*

*Je dédie ce modeste travail à celle qui m'a donné la vie, le symbole de tendresse, qui s'est sacrifiée pour mon bonheur et ma réussite, à ma Mère …*

*A mon père , école de mon enfance, qui a été mon ombre durant toutes les années des études, et qui a veillé tout au long de ma vie à m'encourager, à me donner l'aide et à me protéger.*

*Que dieu les gardes et les protège.*

*A mon frère Ahmed et ma sœur Faiza.*

*A mon binôme abdessamd et toute sa famille.*

*A tout mes amies qui j'ai passé avec eux des bon moments durent ces 3 ans.*

*Yazid*

# Table des matières:

# Liste des tables



# Liste des figures

**Résumé:**

Le but de ce sujet est l'étude et la proposition d'algorithmes de requêtes multi-dimensionnelles de type "Skyline" en base de données.

Dans ce travail, nous proposons un algorithme heuristique basé sur le paradigme diviser et regner (Divide & Conquer), L'idée de base consiste à de couper l'espace des points en plusieurs parties (par défaut 4), ensuite on applique en parallèle un algorithme de type brute-force pour chaque segment, les resultsats partiels vont subir une autre compétition pour retenir les skylines finaux.

***Mots Clés:*** requêtes Skylines, Algorithmes avec index, Algorithmes sans index, Divide & Conquer.

# Introduction générale

**I.Contexte du travail:**

Actuellement on remarque que L'extraction et la mise en évidence de données pertinentes est une tâche difficile, en particulier lorsque les utilisateurs sont confrontés à de gros volumes de données pouvant être comparées selon de nombreux critères et que la majorité des requêtes employées par les décideurs ont une complexité très élevée en terme de calcul et nécessite par fois des mécanismes d'optimisation.

Dans ce contexte la communauté de recherche a introduit le concept de ''Skyline'' qui représente une solution Pareto optimale aux problèmes d'optimisation multicritères (ou multi-objectifs).

Dans les années 60, ce problème était connu comme le problème de la recherche des points adminissibles ou des vecteurs maximaux. Ces premiers algorithmes se sont montrés inefficaces dans le contexte des grandes bases de données, contenant de nombreux points et de nombreux critères. Les auteurs de furent les premiers à proposer des solutions dans ce contexte en intégrant un opérateur Skyline étendent le langage SQL. Désormais de nombreux outils d'aide à la décision intègrent des méthodes de recherche de Skyline et permettent de formuler des requêtes multicritères et d'extraire les points globalement optimaux pour l'ensemble des critères considérés.

**II .Problématique:**

L'objectif de ce travail est de répondre à la problématique de recherches des éléments Skylines. En effet certains algorithmes offerts sont peu efficaces en termes de temps d'exécution, et il est toujours préférable de mettre en œuvre des approches plus rapide.

**III .Contribution:**

Dans le cadre de ce mémoire, nous proposons une approche qui se base sur l'algorithme d'extraction de Skylines intitulé « Divide & Conquer » .Ce dernier est répandu dans divers domaines d'applications tel que les bases de données, les services web….

Notre projet consiste à définir formellement le problème de la recherche des Skylines, nous donnerons un aperçu des différentes méthodes d'extraction de points Skylines ainsi que l'approche Divide & Conquer.

***IV. Plan du mémoire:***

Le reste de ce mémoire est organisé comme suit:

- Chapitre1: il introduit le concept de skyline.

- Chapitre2: il présente la conception et l'implémentation de l'approche développée.

# Chapitre1: le concept de skyline

## I.Introduction:

Les requêtes Skylines, constituent un paradigme élégant et sophistiqué pour les requêtes à préférences dans les bases de données multidimensionnelles. Elles ont été largement Étudiées dans les communautés des bases de données et de l'intelligence artificielle. Dans un espace multidimensionnel où une préférence est définie pour chaque dimension, Les requêtes Skylines retournent les points qui sont meilleurs ou égaux sur toutes les dimensions et strictement meilleurs sur au moins une dimension.

## II.Ordres de préférence et Skyline:

Les notations utilisées dans cette section sont récapitulées dans la table **II.2** Les différentes définitions sont illustrées à partir de l'exemple de la table **II.1** qui décrit des propositions de Pc Portables selon les dimensions suivantes : le prix, la fréquence de processeur et la capacité de la RAM. Nous allons illustrer les définitions de dominance, de Skyline et de relation de préférence sur cet exemple :

| Package ID | Prix (Pr) | Fréquence(Fr) | Capacité(Cp) |
|---|---|---|---|
| A | 28000 | 1,8 | 4 |
| B | 33000 | 2,7 | 2 |
| C | 20000 | 2,0 | 4 |
| D | 35000 | 1,9 | 6 |
| E | 15000 | 1,1 | 3 |
| F | 23000 | 2,4 | 4 |
| G | 39000 | 2,5 | 2 |

**II.1 –Table Base ensemble des Pc Portables**

Soit D = $\{d_1,\ldots, d_n\}$ un espace à n dimensions, E un ensemble de points définis dans l'espace D et p, q deux points de l'ensemble E. On note par p ($d_i$) la valeur de p sur la dimension $d_i$ ne préférence sur le domaine de valeurs de la dimension $d_i$, notée par $\mathcal{P}$ $d_i$, est défini par un ordre partiel $\leq d_i$. Elle désigne l'ensemble des preferences binaires $\mathcal{P}$ $d_i = \{(u, v) | u \leq di\}$ avec (u, v) un couple de valeurs. Dans la suite, nous utilisons les deux notations pour P $d_i$.

| *Notation* | *Description* |
|---|---|
| ***E*** | ***Ensemble de données*** |
| ***\|E\|*** | ***Cardinal de l'ensemble E*** |
| ***D*** | ***Ensemble des dimensions de E*** |
| ***\|D\|*** | ***Cardinal de D*** |
| $D_i$ | ***Une dimension de l'espace D*** $(1 \leq i \leq \|D\|)$ |
| ***D'*** | ***Sous espace de D : D' inclus D*** |
| ***p, q*** | ***Points de l'ensemble E*** |
| $p(d_i)$ | ***Valeur du point p sur la dimension*** $d_i$ |
| $dom\ (d_i)$ | ***Dominance de valeurs de*** $d_i$ |
| P | ***Ensemble de préférences sur D'*** |
| P $d_i$ | ***Préférence sur*** $d_i$ |

**II.2– Table Récapitulatif des notations**

### II.1 Définition 1 (Relation de préférence):

Une relation de préférence sur le domaine de valeurs d'une dimension $d_i$ est définie par un ordre partiel noté $\leq_{di}$. Pour deux valeurs $p(d_i)$ et $q(d_i)$ du domaine de valeurs de $d_i$, nous écrivons $p(d_i) \leq_{di} q(d_i)$ si la valeur $p(d_i)$ est préférée à la valeur $q(d_i)$.**[1]**

### II.2 Définition 2:

Étant donné une relation de préférence $\leq_{di}$, les relations correspondantes de préférence stricte, indifférence et incomparabilité sont définies comme suit:

– $p <$ di q (p est strictement préféré à q sur la dimension d i) $\Leftrightarrow p \leq$ di q $\wedge \neg(q \leq$ di p)

– p = di q (p est indifférent (i.e. également préféré) à q sur la dimension di) $\Leftrightarrow$

$p \leq$ di q $\wedge$ q $\leq$ di p

– $p \leq$ di q (p est préféré à q sur la dimension di) $\Leftrightarrow$ p = di q $\vee$ p < di q

– $p \sim$ di q (p est incomparable à q sur la dimension di) $\Leftrightarrow \neg$ (p $\leq$ di q) q $\wedge \neg$ (q $\leq$ di p) **[1]**

### II.2.1 E x e m p l e :

L'ensemble de données E décrit dans la table **II.1** contient 7 points décrits par 3 dimensions Prix (Pr), la fréquence (Fr) et la capacité (Cp). Les valeurs des deux dimensions fréquence et capacité, sont totalement ordonnées par la relation d'ordre $\geq$ indiquant le plus grand de deux nombres réels. Cette preference spécifie qu'on préfère les Pc ayant un fréquence (resp. les capacités) le plus élevé (ex. B (Fr) $>_{Fr}$ G (Pr). Les valeurs de la dimension prix sont ordonnées selon les Pc moin cher.

Pour chaque dimension $d_i$ dans l'espace D, une préférence est définie. Dans ce qui suit, $\boldsymbol{P} = U_{i=1}^{|D|} \boldsymbol{P}\, \boldsymbol{d_i}$ désigne l'ensemble des préférences associées à l'espace D. L'absence de préférence sur une dimension $d_i$ (i.e. incomparabilité entre les valeurs de d i) est notée par $\mathscr{P}\, d_i = \emptyset$.Dans le cas d'une indifférence ou d'une égalité (i.e. points ayant des valeurs identiques sur une dimension) entre deux points, ils peuvent être considéré comme équivalents et donc regroupés dans des classes d'équivalence. Il est aisé de constater qu'avec l'utilisation des classes d'équivalence, la relation de préférence devient un ordre partiel strict sur une dimension.

Nous présentons maintenant les concepts et les définitions nécessaires à la définition formelle des Skylines.

**II.3 Définition 3 (Relation de dominance):**

P domine q sur D, noté par $p \prec_D q$, si p est strictement préféré ou égal à q sur toutes les dimensions de D et p est préféré à q sur au moins une dimension $\forall\ di \in D,\ P(\boldsymbol{d_i}) \leq \boldsymbol{d_i}$ $\boldsymbol{q\ (d_i)} \wedge \exists\ \boldsymbol{d_i} \in D,\ \boldsymbol{p\ (d_i)} < \boldsymbol{q\ (d_i)}$ Pour plus de lisibilité, $p \prec_D q$ sera simplement noté $p \prec q$. **[1]**

**II.3.1 Exemple:**

Soit un client à la recherche d'un Pc qui est moins cher et possède un bon fréquence.

Dans ce cas, Pc B domine le Pc G ($G \prec B$) since G (Pr) $<_{Pr}$ B (Pr),

G (Fr) $<_{Fr}$ B (Fr) et B (Cp) $=_{Cp}$ G (Cp).

De plus, Pc B ne domine pas le Pc F ($F \not\prec B$), car F (Fr) $<_{Fr}$ B (Fr) etB (Pr) $<_{Pr}$ F(Pr).

L'ensemble des Skylines, ou simplement le Skyline, d'un ensemble de données décrit dans un espace de dimensions contient des points de cet ensemble de données qui ne sont dominées par aucun autre point de ce même ensemble.

**II.4 Définition 4 (Skyline):**

Le Skyline de l'ensemble de données E sur l'espace de dimensions D, avec P l'ensemble des préférences associées à D, est l'ensemble des points qui ne sont dominés par aucun autre point dans E :

$Sky(D, E)_P = \{p \in E | (\forall q \in E, \neg (q \prec_D p))\}$. Si $P = \emptyset$, $Sky(D, E)_P = E$. **[1]**

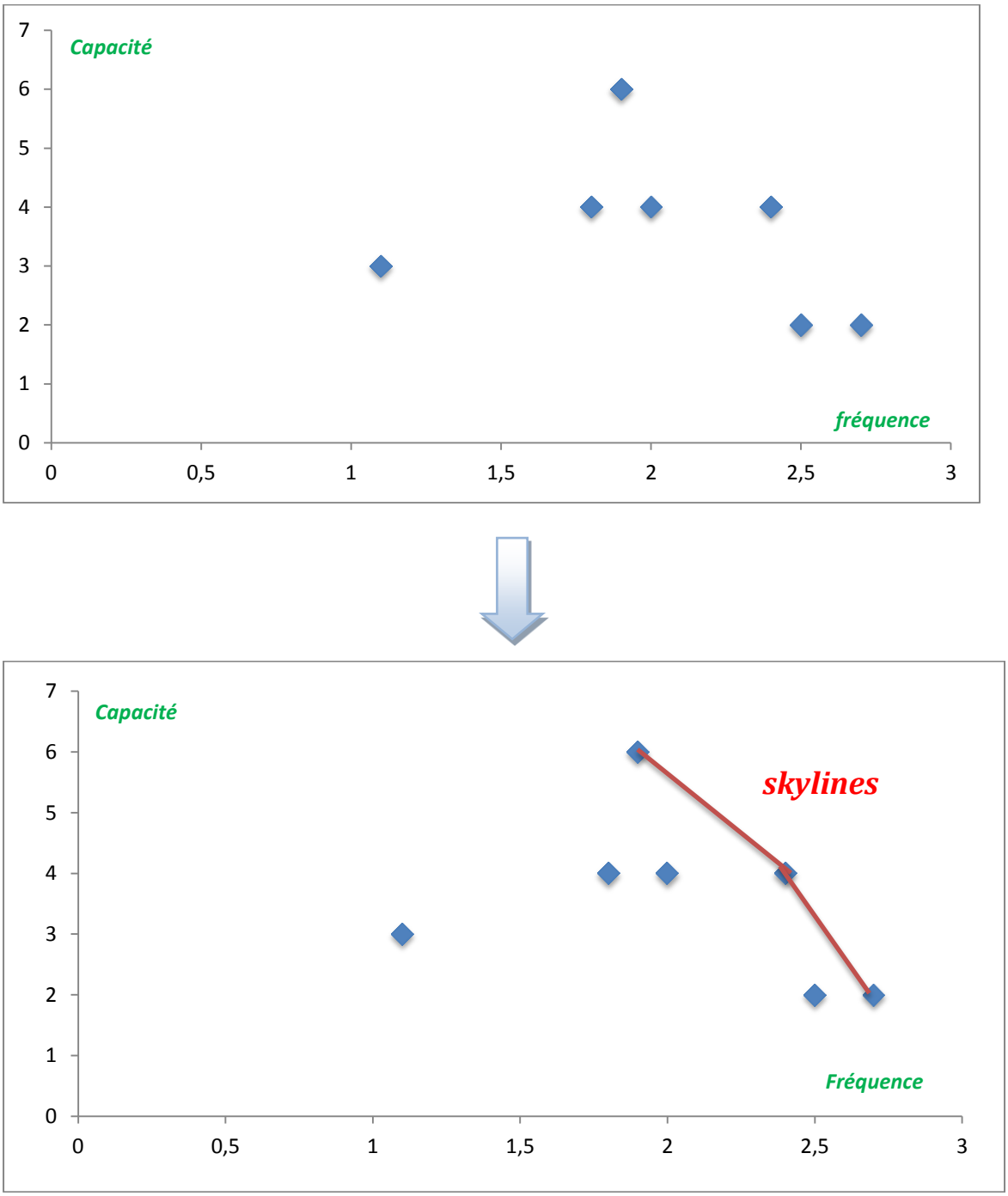



**Figure II.1: Illustration du concept de dominance et de Skyline sur les deux dimensions capacité et fréquence.**

## III. Propriétés des requêtes Skylines:

À présent, nous disposons d'une définition formelle du concept de requêtes Skylines. Il faut maintenant évaluer l'efficacité de ces requêtes en examinant leur complexité. Pour calculer l'ensemble des points Skylines, chaque point est comparé avec tous les autres points de l'ensemble de données (i.e. tests de dominance), ce qui représente $O(n^2)$ comparaisons (avec n le nombre de points dans l'ensemble de données). Dans le contexte de bases de données volumineuses, les calculs deviennent facilement intensifs (i.e. CPU intensifs) en raison du grand nombre de tests de dominance. Il est donc nécessaire d'utiliser des algorithmes spécifiquement conçus pour limiter le nombre de tests de dominance, et ainsi optimiser le calcul des Skylines pour permettre le passage à l'échelle en présence de gros volumes de données.

Les requêtes Skylines sont utilisées dans diverses applications, souvent dans des applications interactives. Il est important, dans ce type d'application, que l'algorithme de recherche de Skylines utilisé fournissent des résultats rapidement voire instantanément. C'est un défi difficile et les algorithmes de calcul de Skylines doivent satisfaire un ensemble de critères permettant l'évaluation de ces derniers en mettant en évidence leurs forces et leurs faiblesses. Ces critères d'évaluation ont été proposés par les auteurs de **[2, 3]**. Nous avons synthétisé ces critères en ne conservant que ceux qui nous semblent les plus pertinents dans le contexte de notre travail:

– **Efficacité :** Ce critère a trait à l'efficacité des algorithmes (i.e. temps de calculs, stockage mémoire ...etc.), mais aussi au passage à l'échelle de ces derniers (i.e. l'algorithme doit -- bien évoluer avec différentes distributions de données, différents types de données par exemple, numérique, catégoriques, et avec un nombre variable de dimensions, ...etc).

– **Progressivité:** Les points Skylines sont retournés le plus rapidement possible, au fur à mesure qu'ils sont découverts,

– **Préférences:** Le calcul des Skylines intègre les préférences utilisateurs.

– **Correction:** Tous les points retournés sont des Skylines.

– **Complétude:** Tous les points du Skylines sont retournés à la fin.

Tous les critères décrits ci-dessus constituent les défis majeurs que doit relever un bon algorithme de calcul de Skyline. Ces défis tant théoriques qu'expérimentaux, ont reçu une présentons quelques uns des algorithmes qui ont été proposés pour le calcul et l'extraction des Skylines.

Une fois présentés, ces algorithmes seront évalués et comparés selon les critères décrits précédemment.

## IV. Algorithmes de calcul des Skylines:

Les requêtes Skylines permettent l'extraction des points qui ne sont dominés par aucun autre point dans un espace multidimensionnel associé à des préférences. Divers travaux de recherche se sont intéressés au calcul des Skylines en présence de larges volumes de données. Ces travaux peuvent être subdivisés en deux catégories les algorithmes de calcul de Skylines dans un espace complet ou dans des sous-espaces. Dans la première catégorie, les algorithmes s'intéressent à l'optimisation du calcul des Skylines associés à toutes les dimensions de l'espace considéré, et ceux de la deuxième permettent à l'utilisateur de choisir les dimensions qui l'intéressent, tout en assurant un calcul optimal des Skylines. Dans cette section, nous présentons les principaux algorithmes de chacune de ces catégories.

### IV.1 Algorithmes de recherche dans un espace complet:

Nous distinguons deux familles d'algorithmes selon qu'ils utilisent ou non des techniques d'indexation. Les premiers n'utilisent pas de structure d e données particulière (i.e. pas de pré-traitements), et donc la recherche des Skylines se fait en analysant entièrement la base de données au moins une fois. Les seconds regroupent ceux qui exploitent des index afin d'améliorer l'efficacité de calcul et d'accélérer davantage les requêtes Skylines, en diminuant par exemple le nombre de tests de dominance.

#### IV.1.1 Les algorithmes sans index:

##### *IV.1.1.1 L'algorithme Block Nested Loop (BNL)*

Börzsönyi et al. **[4]** sont les premiers à avoir abordé la problématique du calcul des Skylines dans le contexte des bases de données. Ils ont introduit l'opérateur SKYLINE dans les systèmes de gestion des basses de données, comme étant une clause SQL étendue. Ils ont aussi proposé dans **[4]** le premier algorithme de recherche de Skyline Block Nested Loop (BNL)

Pour le calcul des Skylines, une approche naïve serait de comparer tous les points de l'ensemble de données deux à deux et de retourner comme Skyline tous les points qui ne sont dominés par aucun autre. L'algorithme BNL s'appuie sur cette approche en analysant entièrement l'ensemble des données et en gardant une liste de points candidats au Skylines en mémoire centrale. Si la mémoire centrale est saturée, l'algorithme gère un fichier temporaire stocké en mémoire secondaire dans lequel sont stockés tous les candidats non considérés faute de place. Ils seront traités lors d'une prochaine itération. Lors de la comparaison d'un point p avec les points stockés en mémoire centrale, trois cas de figure se présentent:

– Le point p est dominé par un point présent en mémoire centrale: p est alors directement écarté de l'ensemble Skyline (i.e. il est dominé) et ne sera plus pris en compte pour le reste des calculs.

– Le point p domine un ou plusieurs points présents en mémoire centrale: les points dominés sont directement écartés et ne seront plus pris en compte pour le reste des calculs (i.e. ils sont supprimés de la liste de points candidats au Skyline). Le point p quant à lui est inséré en mémoire centrale, puisqu'il y a au moins une place libre.

– Le point p est incomparable avec l'ensemble des points présents en mémoire centrale: c'est le cas le plus complexe à gérer, car p doit être inséré en mémoire centrale, mais il se peut que celle ci soit pleine vu que p n'a éliminé aucun autre point. Si la mémoire dispose de place, p est ajouté normalement en mémoire centrale. Sinon, p est mis de côté dans le fichier temporaire cité précédemment et sera examiné à nouveau au cours de la prochaine itération de l'algorithme.

Au vu de ce dernier cas de figure, nous pouvons constater que l'algorithme BNL peut nécessiter un grand nombre d'itérations avant que le Skyline final soit calculé.

Cet algorithme fonctionne bien dans le contexte de bases de données de petite ou moyenne taille, car le nombre de points candidats au Skyline est réduit. Le fichier temporaire est alors peu utilisé puisqu'il reste pratiquement toujours de la place en mémoire centrale. Dans le meilleur des cas, l'algorithme se termine après une seule itération avec une complexité O(n) (n représentant le nombre de points dans la base de données). Alors que dans le pire cas, sa complexité est quadratique $O(n^2)$. Cependant, l'algorithme BNL reste toujours plus optimal qu'une requête SQL utilisant l'opérateur SKYLINE, car il limite considérablement les entrées/sorties en chargeant en mémoire centrale un ensemble de points.

### *IV.1.1.2 L'algorithme Divide & Connquer*

A été proposé par les auteurs de **[4]**. Le principe général de cet algorithme consiste à travailler de manière récursive en divisant l'ensemble de données en plusieurs partitions de sorte à ce que chaque partition puisse être stockée en mémoire centrale. Le Skyline partiel des points de chaque partition est calculé en utilisant un algorithme de recherche de Skyline se basant sur un calcul en mémoire centrale (ex. l'algorithme BNL). Le Skyline final est ensuite obtenu en fusionnant tous les Skylines partiels. La complexité de cet algorithme est de l'ordre de O (n.(log n)$^{d-2}$ +n.log n) où n représente le cardinal de l'ensemble de données, et d le nombre de dimensions. Ce résultat est meilleur que la complexité en O(n$^2$) de l'algorithme BNL.

Cependant, l'algorithme D&C n'est efficace que pour les petits ensembles de donnée. Pour les bases de données volumineuses, le processus de partitionnement nécessite la lecture et l'écriture de l'ensemble de données au moins une fois. Ceci est extrêmement coûteux en E/S et détériore grandement les performances de cet algorithme. Aussi, cet algorithme n'est pas appropriée pour des traitements en ligne, car aucun point Skyline ne peut être retourné tant que la phase de partitionnement n'est pas terminée.

Comme souligné dans **[4, 5]**, la performance moyenne de D&C et BNL se détériore avec l'augmentation du nombre de dimensions du jeu de données considéré, en raison de l'augmentation du nombre de tests de dominance qui sollicitent de manière intensive le processeur. À partir de ce constat, d'autres variantes de l'algorithme BNL ont été proposées par [CGGL05], pour améliorer ses performances.

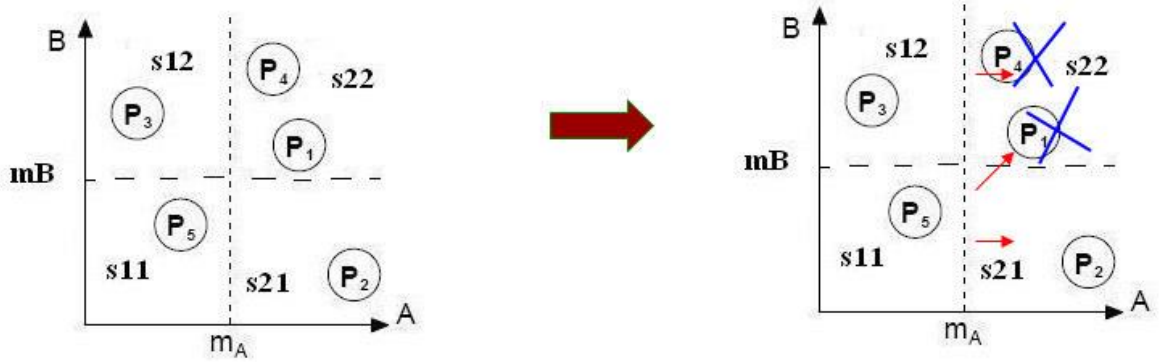



**Figure II. 2: illustration de l'algorithme D&C**

### *IV.1.1.3 L'algorithme Sort First Skyline (SFS):*

Proposé par les auteurs de **[6]**, cet algorithme est une version améliorée de l'algorithme BNL. 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 toute fonction monotone. Par conséquent, chaque point chargé en mémoire centrale peut immédiatement être retourné comme point Skylines. Le nombre de passes nécessaires à effectuer sur le disque pour lire les données est alors égal à la taille de l'ensemble Skylines final divisé par la taille de la mémoire centrale (en terme de nombre de points 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.

### *IV.1.1.4 L'algorithme Linear Elimination Sort for Skyline (LESS):*

L'algorithme LESS **[7]** est une version optimisée de l'algorithme **SFS**. Il utilise une mémoiretampon de petite taille, appelée elimination-filter window lors de l'étape du tri monotone de l'algorithme **SFS**. Cette mémoire tampon permet de sauvegarder un petit ensemble de points, qui seront utilisés pour un élagage précoce des autres points qu'ils dominent. Comme mentionné dans **[7]**, LESS surpasse constamment les performances de SFS. Cependant, les performances de **SFS** et LESS se détériorent grandement avec l'augmentation du nombre de dimensions. En effet, plus l'espace de dimensions est grand, plus les algorithmes **SFS** et LESS consacrent de temps dans la phase de filtre des Skylines. Cela est dû au fait qu'un plus grand nombre de points appartient au Skyline et que l'ensemble des données doit être balayé au moins une fois après leur tri.
Tous ces algorithmes basés sur le tri (**SFS** et LESS) pâtissent du grand nombre de pré-calculs requis durant la phase de filtre des Skylines, vu que chaque point lu doit être comparé avec les points Skylines chargés dans la mémoire tampon.

### IV.1.2 Les algorithmes avec index:

Les algorithmes mentionnés ci-dessus ne nécessitent pas que les données en entrée soient indexées. Par contre, une autre catégorie d'algorithmes **[5, 8, 2, 13, 10]** exploitent des structures d'index afin d'accélérer le calcul des Skylines. Un premier algorithme basé sur les structures d'index de type B-Tree ou R-Tree a été proposé dans **[4].** Ensuite, deux méthodes de traitement progressifs, Bitmap et Index, ont été proposées dans **[8].**

Différents types d'index ont été exploités pour l'optimisation du calcul des Skylines. Ces index peuvent être subdivisés en deux catégories : (i) les index classiques couramment utilisés dans les bases de données, tels que les B-Tree ou bien les index Bitmap (e.g. les algorithmes Bitmap et Index) ; (ii) les index spatiaux, introduits dans le contexte des bases de données spatiales, qui permettent d'appréhender le problème du calcul des Skylines sous un aspect géométrique en exploitant certaines propriétés pour l'optimisation du temps de calcul des algorithmes (e.g. les algorithmes NN et BBS). Dans ce qui suit, nous détaillons le fonctionnement de ces algorithmes et l'utilisation de ces différents index.

#### *IV.1.2.1 L'algorithme Bitmap*

L'algorithme Bitmap convertit chaque point p de l'ensemble de données en un vecteur de bits, représentant le nombre de points ayant une plus petite coordonnée que p dans chaque dimension. La longueur du vecteur de bits est déterminé 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 présente 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 | Coordinate | Bitmap representation |
|---|---|---|
| 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) | (1111100000,1111100000) |
| H | (4, 3) | (1111111000,1111111100) |
| I | (3, 2) | (1111111100,1111111110) |
| J | (9, 1) | (1100000000,1111111111) |
| K | (10, 4) | (1000000000,1111111000) |
| L | (6, 2) | (1111100000,1111111110) |
| M | (8, 3) | (1110000000,1111111100) |

**Table II.3: Exemple de l'algorithme de Bitmap**

C (4, 8)

$C_X$= 1110000110000

$C_Y$= 0011011111111

$C_X$ & $C_Y$=0010000110000

Le premier dans le résultat indiquent les points dominante C.

### *IV.1.2.2 L'algorithme Index*

L'algorithme Index organise l'ensemble des données en m listes (m étant le nombre de dimensions de l'espace), chaque liste est alors indexée sous forme d'un arbre B-Tree et est ordonnée de manière croissante. La i ème liste contient les points p qui vérifient la condition suivante : $p(di)= \min^{m}_{j=1} p(dj)$ (i.e. le point p appartient à la i ème liste si p(di) correspond à la plus petite valeur de p sur l'ensemble de dimensions associé). Les B-Tree sont utilisés pour calculer les Skylines locaux de chaque groupe de points, ces derniers seront fusionnés avec les autres Skylines pour former le Skyline final. Un groupe de points pour chaque liste, représente l'ensemble des points ayant la même valeur sur la dimension correspondante. Bien que cette technique permette de retourner rapidement et progressivement 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.

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

**Table II.4: Exemple de l'algorithme d'index**

### *IV.1.2.3 L'algorithme Nearest Neighbor (NN)*

L'algorithme NN a été la première méthode de calcul de Skylines exploitant les index spatiaux. Comme son nom l'indique, l'algorithme NN est basé sur des requêtes de plus proche voisinage, utilisant une distance donnée (e.g. distance de Manhattan). Ces requêtes sont implémentées à l'aide d'index de type R /*R_-Tree. L'algorithme utilise une constatation géométrique très simple: les points les plus intéressants sont ceux qui sont proches de l'origine du repère. À partir de cette observation, un algorithme de calcul de Skyline efficace (i.e. l'algorithme NN) a été proposé dans **[2]**. Cet algorithme, commence en cherchant le point le plus proche de l'origine du repère. Ce dernier représentant forcément un point Skyline puisqu'aucun autre point n'a pu le dominer. Ensuite, il segmente le reste des points en partitions (i.e. zones de l'espace) qui se chevauchent en appliquant la technique de recherche du plus proche voisin sur le dernier point extrait. Les prochains plus proches voisins sont ensuite itérativement trouvés dans chaque partition. Lors de la découverte d'un nouveau point, l'algorithme utilise l'observation citée précédemment; pour éliminer les régions de l'espace à ne plus explorer. Ainsi, il suffit de chercher uniquement dans les régions susceptibles de contenir d'autres points candidats. Chaque nouveau point Skyline p découvert est à l'origine de m appels récursifs de l'algorithme (m étant le nombre de dimensions de l'espace), une partition étant ajoutée pour chaque coordonnée de **p**. A chaque itération, l'arbre R/*R_-Tree sera parcouru plusieurs fois pour éliminer tous les doublons dans les zones de chevauchement de toutes les partitions (i.e. certaines zones de l'espace peuvent être parcourues plusieurs fois), ce qui engendre des calculs supplémentaires inutiles. Globalement, l'algorithme NN est un bon algorithme pour le calcul des Skylines à faible dimensions. Cependant, il est difficilement exploitable dès que le nombre de dimensions dépasse 4.

### *IV.1.2.4 L'algorithme Branch and Bound Skyline (BBS)*

L'algorithme BBS a été introduit par les auteurs de **[9, 10]**. Comme l'algorithme NN, BBS est aussi basé sur la recherche des plus proches voisins. Il 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 **[9].** L'algorithme BBS utilise un tas qui va contenir les entrées (feuilles / noeuds intermédiaires) de l'arbre en question, triée par ordre croissant selon leur pièces minimale (MinPc) à l'origine du repère .Au début, toutes les entrées au niveau du noeud racine sont insérées dans le tas en utilisant leur minPc comme index de tri. Ensuite, l'algorithme retire l'entrée ayant la plus petite valeur MinPc du tas et insère ses noeuds fils comme entrées dans ce dernier. Lorsque l'entrée courante est un point non dominé, il est directement ajouté au Skyline et tous les noeuds/ sous-arbres qu'ils dominent sont définitivement élagués du R-Tree. L'algorithme s'arrête quand le TAS est vide.

BBS est l'algorithme qui effectue le moins d'entrées sorties parmi tous les algorithmes utilisant les R-Trees (incluant NN). Il constitue la méthode de calcul la plus efficace que nous ayons étudiée pour la recherche des Skylines. Il est donc le plus appropriée dans le contexte de bases de données volumineuses et à dimensionnalité élevée.

## IV.2 Algorithmes de recherche dans des sous-espaces:

Comme nous l'avons vu dans la section précédente, de nombreux travaux récents se sont penchés sur le problème de l'extraction des points Skylines en proposant des méthodes de calcul efficaces. Cependant, lorsque le nombre de dimensions est trop important, les requêtes Skylines commencent à perdre leur pouvoir discriminant en renvoyant une grande partie des données. Cela dit, est-il vraiment pertinent de prendre en considération l'ensemble des dimensions à chaque nouvelle requête Skyline ? La réponse est non. En effet, il a été constaté, que suivant leurs centres d'intérêts, les utilisateurs pouvaient être amenés à poser des requêtes sur différents sous-ensembles de l'espace de dimensions et non sur la totalité. À partir de cette observation, divers travaux se sont alors intéressés à l'extraction de points Skylines dans des sous-espaces de dimensions. Dans un scénario général, les requêtes Skylines classiques peuvent être directement adaptées dans des sous espaces. Cependant, lorsqu'il s'agit de calculer les Skylines sur différents sous espaces, aucun des algorithmes existants ne sait exploiter à son avantage les liens qui peuvent exister entre les différents Skylines de ces différents sous-espaces.

De plus la connaissance de ces liens peut être très intéressante non seulement pour économiser les temps de calcul mais aussi pour un décideur qui souhaite comprendre les raisons pour lesquelles un point donné devient dominant ou à l'inverse dominé. Diverses études se sont alors focalisées sur ce problème **[11, 12, 14, 15, 16, 17, 18]**. Dans ce qui suit nous décrivons un ensemble d'algorithmes spécifiquement développés pour l'extraction de points Skyline dans des sous-espaces.

**V- Comparaison:**

Dans la section précédente, nous avons étudié les meilleurs représentants des deux grandes familles d'algorithmes (avec et sans index) de calculs de Skyline. Dans ce qui suit, nous présentons un tableau récapitulatif en prenant en compte les critères d'évaluation décrits précédemment. Les algorithmes de calcul de Skyline dans des sous espaces ne sont pas inclus dans cette comparaison, car ces méthodes se basent sur les algorithmes de calculs de Skyline dans un espace complet pour extraire leur Skyline. Nous constatons que tous les algorithmes décrits précédemment respectent parfaitement les deux critères Correction et Complétude, en retournant le Skyline exact. S'agissant du critère Efficacité, nous remarquons que les algorithmes utilisant les index spatiaux sont plus efficaces que les autres. Toutefois, ils deviennent peu efficaces voire inexploitables dès que le nombre de dimensions dépasse 6. Incontestablement, le comportement progressif (i.e. critère Progressivité) implique un pré-traitement des données en entrées, c'est à dire l'utilisation d'index (NN, BBS, Index et Bitmap) ou le tri des données (SFS et LESS). Ces pré-traitement peuvent être réutilisés par toutes les requêtes ultérieures de manière dynamique avec la structure correspondante. Concernant la gestion des mises à jour des données, les algorithmes NN et BBS utilisent des structures d'index dynamiques (R-Tree), pouvant être mises à jour de manière incrételmentale. Par contre, la maintenance des structures utilisées par les algorithmes SFS et LESS, peuvent être améliorées à 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 autres points. Concernant le dernier critère Préférences, à 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 préférences utilisateurs.

| Algorithme | Efficacité | progressivité | Préférences | Correction | Complétude |
|---|---|---|---|---|---|
| BNL | Non | Non | Non | Oui | Oui |
| D&C | Non | Non | Non | Oui | Oui |
| SFS | Non | Oui | Non | Oui | Oui |
| LESS | Non | Oui | Non | Oui | Oui |
| Bitmap | Non | Oui | Non | Oui | Oui |
| Index | Non | Oui | Non | Oui | Oui |
| NN | Oui | Oui | Oui | Oui | Oui |
| BBS | Oui | Oui | Non | Oui | Oui |

**Table II.3 – Comparaison des algorithmes de calcul de Skyline dans un espace complet**

## VI. Conclusion :

Nous avons présenté dans ce chapitre le concept de skyline ainsi qu'un survol sur les algorithmes qui permettre leur recherche, dans le chapitre suivant nous montrons la conception et la mise en oeuvre d'un algorithme particulier appelé **Divide & Conquer**.

# Chapitre2: la conception et l'implémentation de l'approche développée

**I Introduction:**

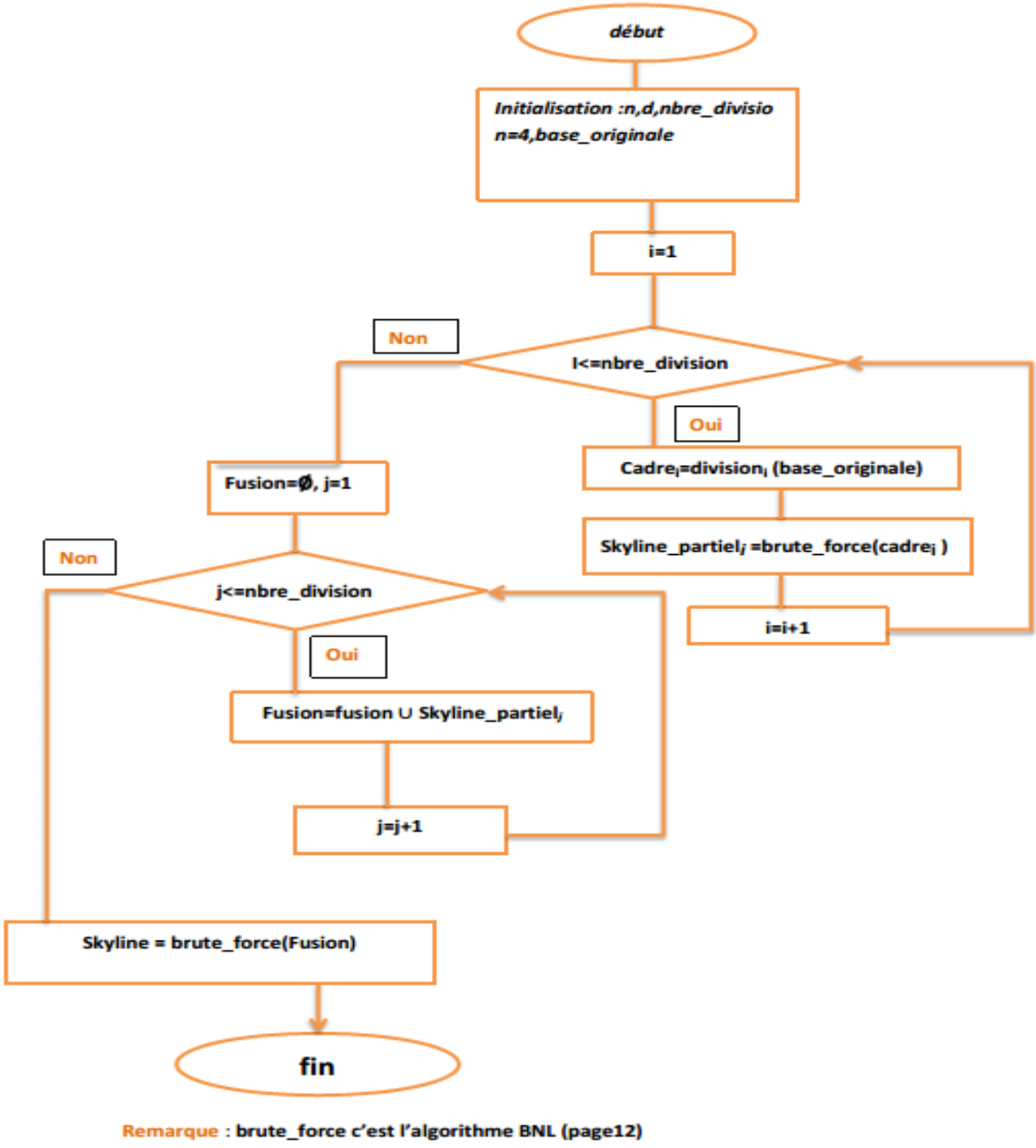
Les Requêtes Skyline ont attiré une attention considérable en raison de son importance dans de nombreuses applications telles que la prise multicritères de décision, l'extraction de données, et l'utilisateur des requêtes de préférence **[1]** ; ils renvoient un ensemble d'objets de données intéressantes de tous les autres objets.
Et pour ce dernier, nous considérons un algorithme performent « divide & conquer» qui se base sur la minimisation des objets.

**II Principe de l'algorithme Divide & Conquer:**

Dans les grandes bases de données, on analyse souvent une grande quantité d'objets retournés par les requêtes, pour réduire ce nombre nous sélectionnons juste les objets les plus importants et les plus significatifs, pour cela nous proposons un algorithme efficace pour le calcul des requêtes Skylines ce dernier est basé sur Divide & Conquer . La première étape de divide & conquer permet de diviser l'ensemble de données en plusieurs partitions de sorte à ce que chaque partition puisse être stockée en mémoire centrale.Le Skyline partiel des points de chaque partition est calculé en utilisant un algorithme de recherche de Skyline se basant sur un calcul en mémoire centrale (ex. l'algorithme BNL).Cette algorithme de recherche base sur l'utilisation d'une fonction domine tel que un objet P est dominé par un autre objet q s'il ya ($k \leq n$) dimensions dans lesquelles P est inférieur ou égal à q, Compte tenu d'une base de données X, un objet P est dit dans le Skyline de X s'il n'y a pas d'autre objet q en X de telle sorte que q vaut mieux que P dans toutes les dimensions. S'il existe par exemple une solution aqueuse, alors nous disons que P est dominé par q, ou q domine P. par conséquent on s'intéresse aux problèmes de base de données par minimisation.
Enfin, la dernière étape consiste à fusionner tous les Skylines partiels et refaire la meme algorithme de recherche sur ces Skylines pour obtenu les Skylines finaux.

**III Conception:**

**Organigramme de Divide & Conquer:**




Remarque : brute_force c'est l'algorithme BNL (page12)

**Figure III.1: L'organigramme de Divide & Conquer**

L’organigramme représente les deux fonctions principales pour la recherche des meilleurs points de Skyline :

- Il met ces derniers dans un tableau spécialement Skylines,
- En sortie (après exécution de cette opération) en aura un tableau qui contient tous les points qui sont incomparable entre eux.

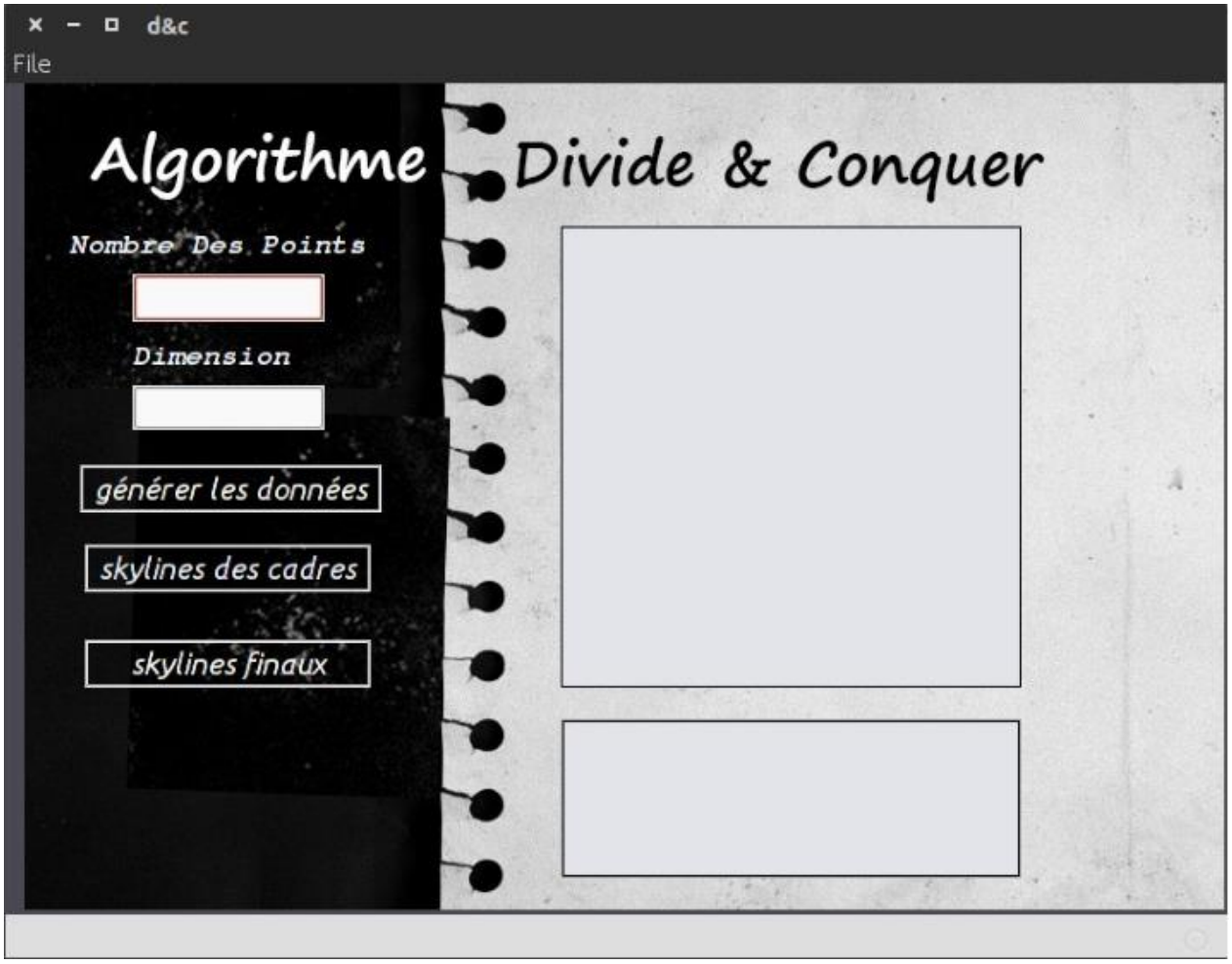



**Figure III.2: Création de la base de données**

## IV L’exécution:

1- On a entré le nombres des points (20) et la dimension (2).

2- On a exécuter la boutton générer les données:

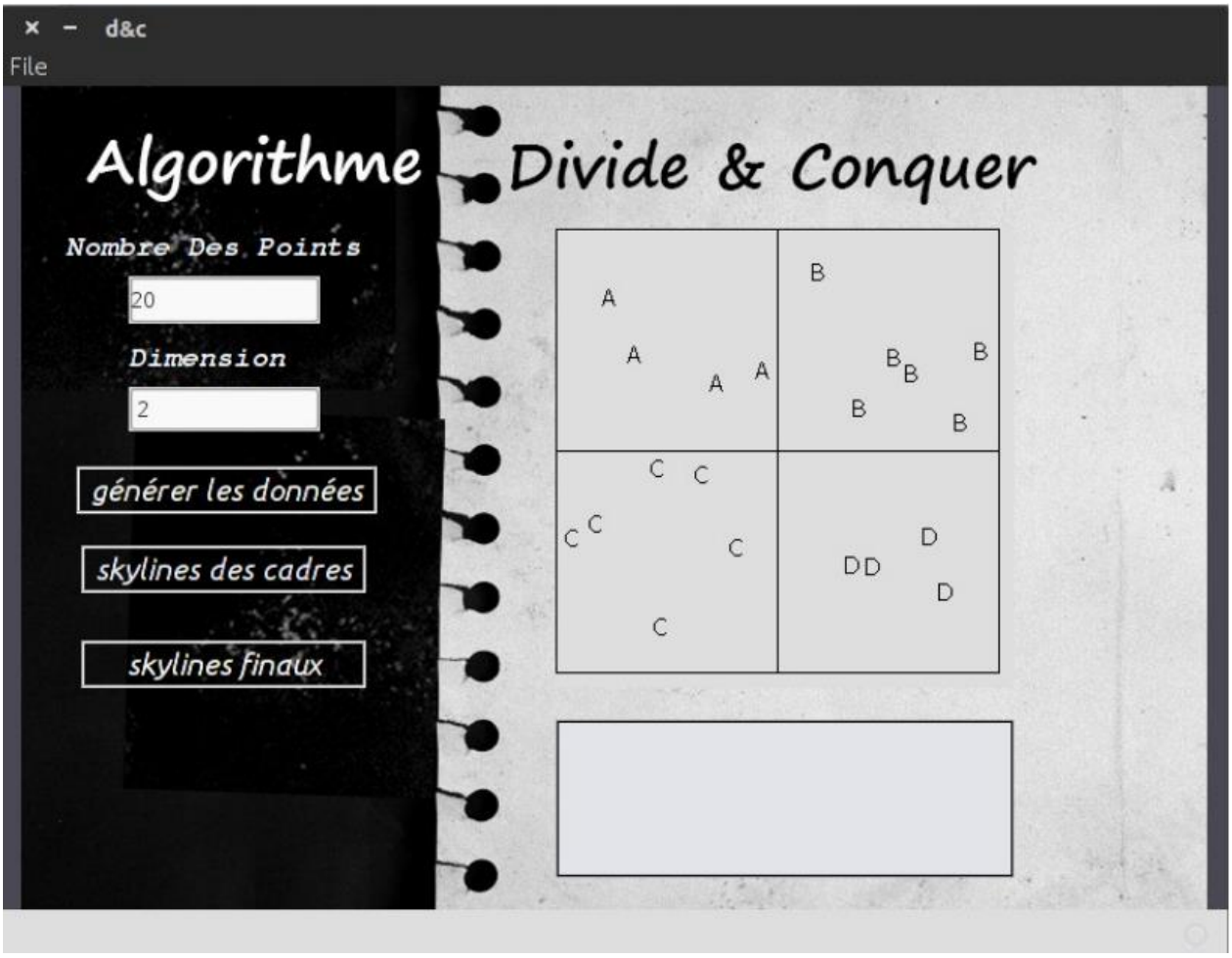



**Figure III.3: Le résultat de l’exécution de « générer les données »**

Nous montrons que la fenètre principale montre uniquement les 02 premiers axes , si la dimension depose 02 alors les axes de rang supèrieure ou égal à 03 ne sont pas visualisés, et par défaut le nombre de segment égal 4.

**3- On a exécuté la boutton skylines des cadres:**

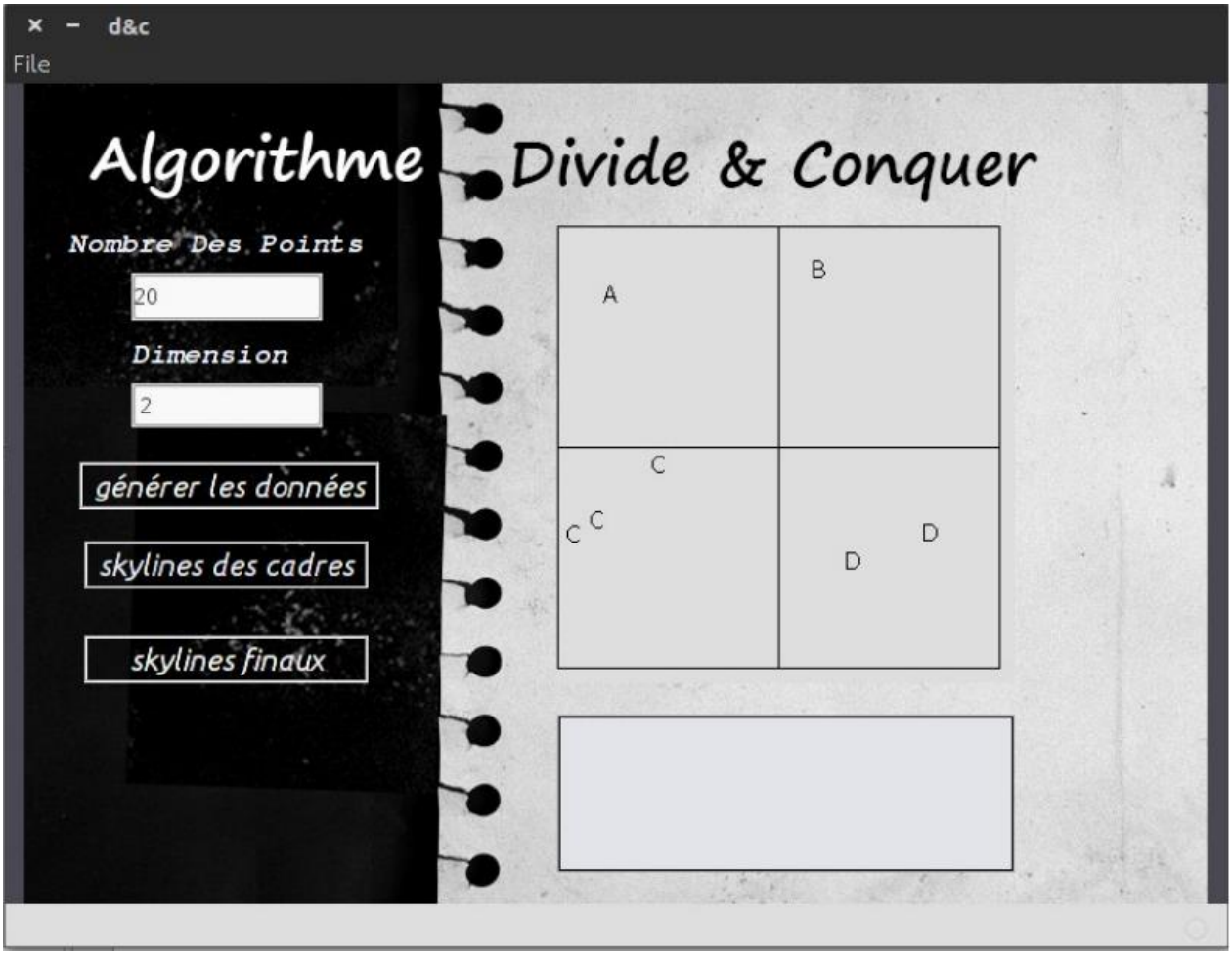



**Figure III.4 : Le résultat de l'exécution de « skylines des cadres »**

**4- On a exécuté la boutton skylines finaux:**

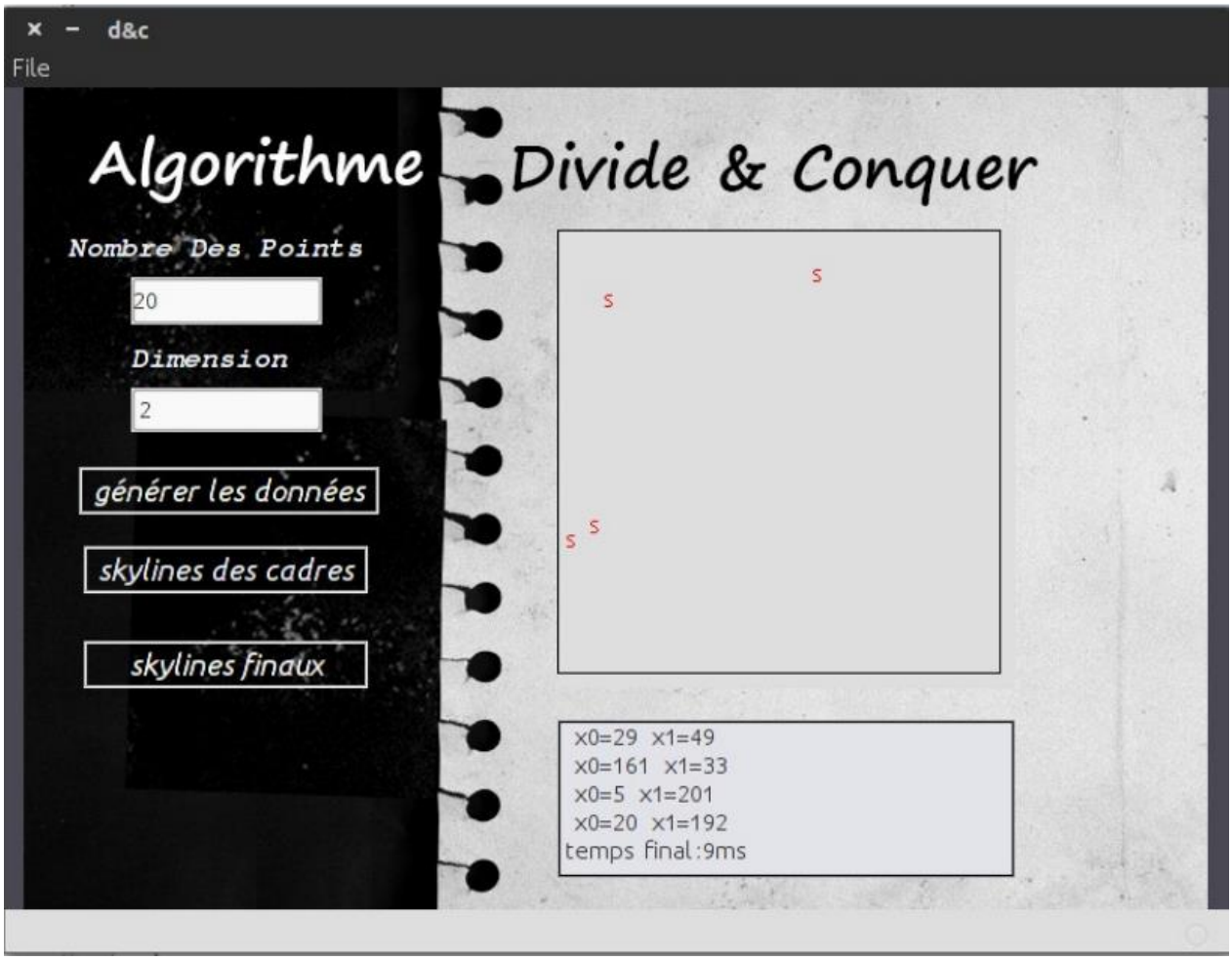



**Figure III.5: Le résultat de l'exécution de « skylines finaux »**

**5- L'ors de l' éxécution de Divide & Conquer:**


**Figure III.6: Le résultat de l'exécution de « Divide & Conquer »**

## V Expérimentation:

Nous avons mené une expérience pour évaluer la performance de notre application. Le but de cette application est de démontrer comment on peut obtenir des points meilleurs. Nous avons développé notre prototype sous NetBeans IDE de Sun Microsystems, la machine d'expérimentation possède les caractéristiques suivantes:

- La machine Sony Vaio version PCG - 91211M.
- Le système d'exploitation est ubuntu 14.04LTS (64bit).
- Processeur Intel® Core™ i3-2330M CPU @ 2.20GHz × 4.
- 3.8 GiB de RAM.

**(Tableau de comparaison)**

| Nombre des points | Dimension | Nombre de Skyline | Temps d'exécution |
|---|---|---|---|
| 20 | 2 | 4 | 9 |
| 40 | 2 | 6 | 14 |
| 80 | 5 | 40 | 46 |
| 100 | 7 | 73 | 118 |
| 300 | 10 | 246 | 526 |

**Table III.1: L'évaluation de l'application**

## VI.Conclusion:

Nous avons présenté Dans ce chapitre un algorithme d’optimisation multicritère qui permet de retourner les points Skylines de manière éfficace et diminue le nombre de comparaisons entre les points.

## Conclusion Général et Perspectives:

Dans ce travail nous avons montre les meilleurs représentants des deux grandes familles d'algorithmes (avec et sans index) de calculs de Skyline.
Tel que nous avons aussi développé une application qui se base sur l'algorithme « Divide & Conquer », ce dernier éxploite le principe de division de l'espace de recherche Pour réduire le temp d'éxtraction des skylines partiels, en plus le fait de definir 02 couches d'élagage permet un autre gain de temps.
Nous constatons que les critères de préférences et d'efficacité ne sont respectés que par peu d'algorithmes. Pourtant, ces deux critères sont indispensables au bon fonctionnement d'un algorithme de calcul de Skyline. Cependant, pour les bases de données volumineuses, l'algorithme **D&C** perd ses efficacités et devient inadaptés.
Comme perspectives a ce travail nous proposons deux approches importantes, le SKYCUBE qui représente le cube de tous les Skylines de tous les sous-espaces possibles et non vides, et SUBSKY qui permet un calcul efficace de requêtes Skylines dans des sous-espaces arbitraires.

## Références Bibliographiques:


**[1]** B.Tassadit, analyse multidimensionnelle interactive de résultat de simulation. Aide a la décision dans le domaine de l'argoécologie, Thèse /Université de Rennes 1, 28 novembre 2013.

**[2]** Kossmann D., Ramsak F., and Rost S. Shooting stars in the sky: An online For skyline queries. In Very Large Data Bases, pages 275–286, 2002.

**[3]** J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston,V. Raman,T. Roth, and P. J. Haas. Interactive data analysis: The control project. Computer, 32(8) Pages: 51–59, 1999.

**[4]** S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In Proc of The 17 th International Conference on Data Engineering, pages 421–430. Computer Society, 2001

**[5]** J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. IEEE Trans. on Knowl. and Data Eng., pages 717– 719, 2003.

**[6]** J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting: Theory and optimizations. In In Proc of Intelligent Information Systems, pages 595–604. Springer Berlin/Heidelberg, 2005.

**[7]** P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In Proceedings of the 31 st international conference on Very large data bases, pages 229–240. VLDB Endowment, 2005.

**[8]** K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In Proceedings of the 27th International Conference on Very Large Data Bases, Pages 301–310. Morgan Kaufmann Publishers Inc., 2001.

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

**[10]** D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. ACM Trans. Database Syst., pages 41–82, 2005.

**[11]** J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: a semantic Approach based on decisive subspaces. In Proc of the 31st international conference on Very large data bases, pages 253–264. VLDB Endowment, 2005.

**[12]** C. Raïssi, J. Pei, and T. Kister. Computing closed skycubes. Proc.VLDB Endow. Pages 838-847, 2010

**[13]** M. Morse, J. M. Patel, and H. V. Jagadish. Efficient skyline computation over Low-cardinality domains. In Proceedings of the 33rd international conference on Very large data bases, pages 267–278. VLDB Endowment, 2007.

**[14]** Z. Huang, J. Guo, S.L. Sun, and W. Wang. Efficient optimization of multiple subspace skyline queries. J. Comput. Sci. Technol., pages 103–111, 2008.

**[15]** Xia T. and Zhang D. Refreshing the sky : the compressed skycube with efficient support for frequent updates. In Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006, pages 491–502. ACM, 2006.

**[16]** C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang. Dada : a data cube for dominant relationship analysis. In Proceedings of the 2006 ACM SIGMOD international conference on Management of data, pages 659–670. ACM, 2006.

**[17]** W. Jin, A. K. H.Tung, M. Ester, and J. Han. On efficient processing of subspace skyline queries on high dimensional data. In Proc of the 19th International Conference on Scientific and Statistical Database Management, page 12. IEEE Computer Society, 2007.

**[18]** Y. Tao, X. Xiao, and J. Pei. Efficient skyline and top-k retrieval in subspaces. IEEE Trans. on Knowl. and Data Eng., pages 1072–1088, 2008.