Pearls of Database Literature
Notes: The Skyline Operator by Börzsönyi in 2001

Mario Preishuber, 1120643
Department of Computer Science — University of Salzburg
Summer Semester 2015

May 27, 2015

1 Questions: Robert

1.1 Was sind die Gründe warum im Paper eine Änderung des Datenbank Systems vorgeschlagen wird im Gegensatz zu der Methode den Skyline Operator auf einem unveränderten System auszuführen?

Integrating the Skyline operator into a database system offers several advantages.

- Encapsulation of the `skyline of` clause.
- Implementation of other operators need not be changed.
- Just simple modification of parser and query optimizer are required.

1.2 Warum bringt bei mehrdimensionalen Skylines das sortieren der Daten keinen Vorteil und warum tritt dieses Problem bei zweidimensionalen Skylines nicht auf?

For a two-dimensional Skyline all tuples are sorted topologically according to the two attributes (of `skyline of` clause). Hence, a tuple must only be compared with its predecessor, in other words the last previous tuple which is in the skyline. There is always only on tuple we have to look at.

For a multi-dimensional Skyline sorting won't work. The reason is there might be a tuple previous of the predecessor which dominates the current tuple. Figure 1.2 illustrates such a situation. The tuples $h1$ and $h2$ are not comparable, because $h1$ is just a 3-stars hotel and $h2$ is a 4-stars. So, non of these two is eliminated. If it is $h3$s turn and we just take a look at its predecessor (which is $h2$) $h3$ would not be eliminated. This is a problem because $h1$ dominates $h3$, so $h3$ has to be eliminated. Hence, the simple approach used for two-dimensional Skylines doesn't work for multi-dimensional Skylines.

```
<h1, 50 USD, 3 miles, ***>
<h2, 51 USD, 5 miles, ****>
<h3, 52 USD, 4 miles, ***>
<h4, 53 USD, 2 miles, ***>
```

Figure 1: 3d Skyline example.

1.3 Was sind die zwei Änderungen am Fenster die im Paper bei dem Block Algorithmus vorschlagen werden?

Maintaining the Window as a Self-organizing List If a tuple t which is in the window dominates another tuple, t is moved to the beginning of the window. The next t_n tuple is first checked against t which increases the probability that t_n maybe eliminated earlier. This approach requires less comparisons which gains performance.

Replacing Tuples in the Window This approach tries to keep the *most dominant* set of tuples in the window. Think of following situation: The window size is 2 and the window contains the tuples `<h1, 50 USD, 1.0 miles>` and `<h2, 59 USD, 0.9 miles>`. Assume the next tuple look like this `<h3, 60 USD, 0.1 miles>`. So, both $h1$ and $h2$ are incomparable with $h3$. However, it is obvious that if $h1$ and $h3$ would be in the window they eliminate more tuples than the combination $h1$, $h2$. There are many replacement algorithms, i.e., we could extend our *self-organizing* list and implement a LRU replacement policy.

2 Questions: Alexander

2.1 The paper presents an algorithm to find interesting points in a large data set. Relying on an algorithm to return interesting results requires trust. How do the authors justify this trust?

Definition 1 *A point is interesting if it is not dominated by any other point.*

Definition 2 *A point dominates another point if it is as good or better in all dimensions and better in at least one dimension.* Hence, we can't lose any

relevant tuples, because a tuple t_1 is only dominating another tuple t_2 if t_1 is at least in one dimension better and in all others equal to t_2. This follows by the definition of dominating. In other words, we always get all relevant points of interest (independent of our criteria).

```
SELECT ...FROM ...WHERE ...
GROUP BY ...HAVING ...
SKYLINE OF [DISTINCT] d_1 [MIN | MAX | DIFF], ...d_n [MIN | MAX | DIFF]
ORDER BY ...
```

Figure 2: SQL Extensions

2.2 Which implementations of the Skyline operator are proposed and what distinguishes them?

Figure 2.2 shows all SQL extensions.

- d_1, ..., d_n denote the dimensions of the Skyline.
- `MIN, MAX, DIFF` denotes whether the value of this dimension should be minimized, maximized or just different.
- `DISTINCT` denotes how the system should handle duplicates (optional).
- The `SKYLINE OF` clause is always executed after the `SELECT ...FROM ...WHERE ...GROUP BY ...HAVING ...` and before `ORDER BY ...`

2.3 How does the properties (i.e. correlation) of data influence the results?

If there are correlations on the data all block-nested-loop (BNL) based algorithms perform much better than the divide-and-conquer (D&C) based. They claim that the performance of BNL variants are good if the size of the Skyline is small. Furthermore, if there are no correlations the BNL based algorithms perform well for small dimensions but they don't scale, i.e., for 5 dimensions the performance good for 10 dimensions D&C perform significantly better. To summarize correlations between data influence the performance. It depends on the used algorithm how massive the effects on the execution are.

3 Questions: Philipp

3.1 In der Arbeit wird angeführt, dass der SKYLINE OF - Operator nach scan, join und group-by Operatoren ausgeführt werden muss, und vor einem Order-by. Des Weiteren wird besprochen, dass es Ausnahmen zu diesen Regeln gibt, welche wären das und was bedeuten sie? Könnte eine Kombination von den Order-by Attributen und den Attributen des Skyline Operators den Algorithmus zum ermitteln der Skyline verbessern bzw. konkret beschleunigen? Z.B.

```
select * from Emp e, Dept d
where e.dno = d.dno
skyline of e.age MIN, e.salary MAX
order by e.age asc, e.salary desc
```

There two interesting approaches:

Pushing the Skyline Operator *Through* a Join There is a situation where it's possible to execute they skyline operator before a join. Therefore, the join has to be *non-reductive* (the definition can be found in another paper). Applying the skyline operator before a join is nice because the skyline operator reduces the size of the result. Hence, joining gets cheaper. Also the execution of the skyline operator can get cheaper, because an index may be used.

Pushing the Skyline Operator *Into* a Join The idea is to compute the skyline before joining on one of the tables. After that execute the join and apply the skyline operator again at the end. As the approach above this reduces the input for the join operation. To ensure that the skyline is correct we have to apply the skyline operation on the result of the join.

3.2 Bei der Erklärung des Divide and Conquer Algorithmus und selbiger Erweiterungen *m-way Partitioning* wird die *Early Skyline* erläutert. Was ist die Early Skyline und das m-way Partitioning Verfahren und wie verbessert sie den Basis D&C Algorithmus?

The approaches *m-way partitioning* and *early skyline* are a solution for the situation that the main memory it to small for the total input. In that case the basic algorithm performs terribly bad.

m-way partitioning The idea is very simple: The input is partitioned as long as it fits into the main memory. This requires to compute bounds (which

depend on the size of the main memory) instead of the median.

early skyline In this case we read as many tuples as possible into main memory. Now we compute the skyline, and save it to an temporal file. We repeat this procedure until there are no more tuples to read. In the next step we compute the skyline over our *early skylines*. This can be done, e.g., by using *m-way partitioning*.

3.3 Welche Aussagen wurden über die Multi-Dimensionalen Syklines aufgrund der Ergebnisse getroffen? Warum wird eine Early Skyline weniger nützlich mit der Anzahl an zusätzlichen Dimensionen.

A kind of a overview messages could be said that block-nested-loop (BNL) based algorithms are the clear winner for correlated databases. For non-correlated databases it depends on the dimensions. For small dimensions block-nested-loop variants are still good, but they don't scale with a increasing number of dimensions. They claim to get in every case a good performance both approaches should be implemented in a system. Further, they show that BNL based algorithms have the lowest I/O costs in all cases. However, as expected outperforms the *early skyline* version of the divide-and-conquer algorithm the *m-way* version.

Notizen zur Diskussion des Papers: The Skyline Operatore, von Börzsönyi, Kossmann und Stocker

Note Taker: Patrick Rappensberger

27.05.2015

1 Was sind die Gründe warum im Paper eine Änderung des Datenbank Systems vorgeschlagen wird im Gegensatz zu der Methode den Skyline Operator auf einem unveränderten System auszuführen?

Im Prinzip geht es um die Performance von Nested Queries. In diesem Paper werden vor allem jene Nested Queries diskutiert die nicht aufgelöst werden können und bei jeder Iteration neu berechnet werden müssen. Mithilfe des Skyline Operators sollen Nested Query's eliminiert werden uns somit die Performance des Systems verbessert werden.

2 Warum bringt bei mehrdimensionalen Skylines das sortieren der Daten keinen Vorteil und warum tritt dieses Problem bei zweidimensionalen Skylines nicht auf ?

Bei zweidimensionalen Skylines werden die Daten nach den Attributen sortiert. Primär wird nach dem ersten Attribut sortiert und somit müssen nur direkte Nachfolger/Vorgänger von einem Tuple verglichen werden um festzustellen, ob ein Tuple ein anderes dominiert. Bei der Sortierung wird schon garantiert das, das erste Tuple mit dem ersten Attribut alle anderen in dieser Dimension dominiert. Somit muss nur mehr auf das andere Attribut überprüft werden. Bei mehrdimensionalen Skylines funktioniert dieses Prinzip des direkten Vergleichs nicht mehr. Somit sind mehre Iterationen über alle Tuple notwendig und somit ineffizient.

3 Was sind die zwei Änderungen am Fenster die im Paper bei dem Block Algorithmus vorschlagen werden?

Die Block Algorithmen funktionieren im Prinzip folgendermaßen:
Es wird ein Fenster (Window) im Hauptspeicher definiert. In diesem Window sind Tuple die untereinander incomparable sind. Alle Tuple werden gelesen und mit denen im

Window verglichen. Es können drei Fälle für das gelesene Tuple p auftreten:

- p wird von einem Tuple im Window dominiert: Somit wird p eliminiert und nicht mehr weiter beachtet.
- p dominiert eines, oder mehrere, Tuple im Window: Die dominierten Tuple werden aus dem Window entfernt und p wird hinzugefügt.
- p ist incomparable mit allen Window-Tuplen: Falls noch Platz im Window ist wird p hinzugefügt, sonst wird p in eine temporäre Datei geschrieben und in den nächsten Iterationen verarbeitet.

Am Ende jeder Iteration können jene Tuples aus dem Window genommen werden, die mit allen Tuples in der temporären Datei verglichen wurden. Diese Outputtuples sind dann Teil der Skyline. Mithilfe von Timestamps wird sichergestellt das der Algorithmus terminiert und das jedes Tuple im Window mit jedem Tuple verglichen wurde.
Im Paper wurden nun folgende Änderungen vorgeschlagen:
Maintaining the Window as a Self-organizing List: Bei dieser Methode wird das Window als self-organizing list implementiert. Das bedeutet falls ein Tuple w im Fenster ist und ein Tuple eliminiert, wird w an den Beginn der Liste geschoben. Somit ist w das erste Tuple mit dem das nächst gelesene Tuple verglichen wird. Falls ein "Tuple-Killer" gefunden wurde ist dieser immer am Anfang des Windows/der Liste und somit sind weniger Vergleiche notwendig.
Replacing Tuples in the Window: Bei dieser Implementationsart wird versucht nur Tuple im Window zu halten, die eine hohe Selektivität besitzen. Somit sollen möglichst viele Tuple eliminiert werden und die Anzahl der nötigen Iterationen gering gehalten werden. Diese vorgehensweise hat allerdings zur folge das CPU-Zeit verwendet werden muss um festzustellen ob ein Tuple im Window erstetzt werden soll obwohl es nicht dominiert wird.
Zum Beispiel befinden sich folgende Tuple im Window:(h_1,\$50,1.0 mile), (h_2,\$59,0.9 mile) und diese werden mit folgenden Tuple verglichen: (h_3,\$60,0.1 mile). Es ist klar ersichtlich das h_3 eine höhere Selektivität besitzt (geringe Mehrkosten, aber viel geringere Distanz) als h_2 und somit sollte h_2 im Fenster durch h_3 ersetzt werden, obwohl h_3 nicht h_2 dominiert.

4 The paper presents an algorithm to find interesting points in a large data set. Relying on an algorithm to return interesting results requires trust. How do the authors justify this trust?

Durch die Definition von **dominieren** wird sichergestellt das keine Ergebnistuple verloren gehen können. **Definition von Dominieren**:
Ein Punkt dominiert einen anderen, wenn er mindestens gleich gut in allen Dimensionen ist und mindestens in einer Dimension besser.

Daraus und aus der Definition des Skyline Operators folgt das nur (alle) Punkte in der Skyline sind, die nicht dominiert werden. Somit ist sichergestellt das keine Punkte entfallen die in einer Dimension bester sind.

5 Which implementations of the Skyline operator are proposed and what distinguishes them?

Im Paper werden mehrere Skyline Operator Algorithmen/Implementationen vorgestellt und diese wurden in folgende Kategorien eingeteilt:

- 2D Skyline Operator
- Block-Nested-Loop Algorithm
  - Maintaining the Window as a Self-organizing list
  - Replacing Tuples in the Window
- Divide and Conquer Algorithm
  - M-way Partitioning
  - Early Skyline
- Sonstige (Verwendung von Indizes)
  - Ordered Indices
  - Pushing the Skyline Operator into/trough a Join

Eine 2D Skyline trivial zu berechnen. Die Daten werden nach den Attributen sortiert und jedes Tuple muss nur mit seinem Vorgänger verglichen werden. (Genauer beschrieben in Frage 2)
Bei den Block-Nested-Loop Algorithmen wird im Prinzip jedes Tuple mit jedem anderen Tuple verglichen. Da dies sehr I/O lastig ist und vor allem ineffizient ist werden bessere Verfahren präsentiert. Diese sind in Frage 3 beschrieben.
Bei den Divide and Conquer Algorithmen wird die Menge an Input Daten geteilt und jeweils getrennt verarbeitet und später wieder zusammengeführt. Das Verfahren und die Subverfahren sind in Frage 8 ausführlich beantwortet.
Falls Indizes verwendet werden, bieten sich R-Bäume an. Die Idee bei den Indizes Operatoren besteht darin, geignete Tuple zu finden um ganze Teilbäume entfernen zu können.

6 How does the properties (i.e. correlation) of data influence the results?

Folgende drei Typen von Verteilungen wurden im Paper behandelt:

- independent: Die Werte der Attribute wurde zufällig verteilt.
- correlated: Falls ein Punkt in einer Dimension gut ist, ist der Punkt auch in den anderen gut.
- anti-correlated: Falls ein Punkt in einer Dimension gut ist, ist der Punkt in den anderen schlecht.

Der Divide & Conquer Algorithmus ist bei unabhängigen Daten gut. Die Block basierten bei vielen dominierenden Tuplen (also bei Korrelierten Daten). DIe Anti-Correlated führen bei allen Verfahren zu vielen Incompareables.

7 (Betrifft 3. und 4.) In der Arbeit wird angeführt, dass der SKYLINE OF - Operator nach scan, join und group-by Operatoren ausgeführt werden muss, und vor einem order by. Des Weiteren wird besprochen, dass es Ausnahmen zu diesen Regeln gibt, welche wären das und was bedeuten sie? Könnte eine Kombination von den Order-By Attributen und den Attributen des Skyline Operators den Algorithmus zum ermitteln der Skyline verbessern bzw. konkret beschleunigen?

Folgende zwei Regeln werden im Paper besprochen:**Pushing the Skyline Operator Trough a Join** und **Pushing the Skyline Operator Into a Join**

- Pushing the Skyline Operator Trough a Join: Diese Regel kann angewandt werden wenn der join (oder group-by) non-reductive ist. In diesem Fall kann der Skyline Operator vor dem Join ausgeführt werden. Das senkt auch die Kosten des Joins, weil durch den Skyline Operator weniger Tuple gejoined werden müssen. Eventuell kann auch schon bei der Skyline Operation ein Index verwendet werden. Das bringt weitere Zeitersparnis.
- Pushing the Skyline Operator Into a Join: Im Prinzip stellt diese Variante einen Early Skyline Operator dar. Im Paper wird allerdings nur ein Beispiel dargestellt bei dem diese Variante funktioniert und effektiv ist. Leider wird im Paper nicht genauer auf diese Operationsart eingegangen.

8 (Betrifft 3.4.2) Bei der Erklärung des Divide and Conquer Algorithmus und selbiger Erweiterungen "m-way Partitioning" wird die "Early Skyline" erläutert. Was ist die Early Skyline und das m-way Partitioning Verfahren und wie verbessert sie den Basis D& C Algorithmus?

Der Divide and Conquer Algorithmus funktioniert besteht aus drei Schritten:

- Den Median von einer Dimension berechnen und den Input in zwei Partitionen einteilen.
- Die Skyline von beiden Partitionen berechnen. Das geschieht durch das wiederholte anwenden des Algorithmus und passiert so lange bis die Skyline trivial berechnet werden kann, weil nur mehr sehr wenige Tuple sich in jeder Partition befinden.
- Die globale Skyline berechnen indem alle Partitionen wieder zusammen geführt werden. Das passiert indem jene Tuple entfernt werden die von Tuple in einer anderen Partition dominiert werden. Durch intelligente Partitionierung müssen nicht alle Partitionen untereinander auf dominierende Tuple überprüft werden.

Folgende Erweiterungen sind im Paper beschrieben:

- M-Way Partitioning: Falls der gesamte Input nicht im Hauptspeicher Platz hat, kann m-way Partitioning angewendet werden. Bei diesem Verfahren bestimmt in der ersten Phase des Divide & Conquer Algorithmus nicht der Median wo geteilt wird, sondern die Größe des Hauptspeichers. Falls eine Partition nicht Platz hat wird sie so lange geteilt, bis sie in den Hauptspeicher passt. Im Merge Schritt wird ebenfalls so lange geteilt bis die benötigten Partitionen in den Hauptspeicher passen.
- Early Skyline: Dieses Verfahren kommt ebenfalls zum Einsatz wenn der Hauptspeicher limitiert ist. Bei diesem Verfahren werden so viele Tuple geladen, wie in den Hauptspeicher passen. Danach wird auf die geladenen Tuple das Divide & Conquer Verfahren angewandt um durch Dominierung Tuple zu eliminieren. Das bezeichnet man als Early Skyline. Danach werden die noch nicht verarbeiteten Blöcke mithilfe von m-way Partitioning verarbeitet. Dieses Verfahren benötigt mehr CPU-Zeit, allerdings wird der I/O minimiert. Dadurch eignet sich das Verfahren besonders wenn die Selektivität groß ist.

9 (Betrifft 5.3) Welche Aussagen wurden über die Multi-Dimensionalen Syklines aufgrund der Ergebnisse getroffen? Warum wird eine Early Skyline weniger nützlich mit der Anzahl an zusätzlichen Dimensionen.

Der Early-Skyline Operator ist mit vielen Dimensionen weniger effektiv, weil die Selektivität der einzelnen Partitionen sinkt und mehr Partitionen erstellt werden müssen.

Die Resultate zusammengefasst:
Block-Nested-Loop Algorithmen sind gut falls die Größe der Skyline klein ist und sehr sensitiv ist. Die Divide & Conquer Algorithmen sind weniger sensitiv im Bezug auf die Dimensionen und Korrelationen. Allgemein sind Divide & Conquer Algorithmen besser, wenn mehr Hauptspeicher vorhanden ist und Block-Nested-Loop Verfahren werden mit steigender Größe der Datenbank effektiver.