Aggregate Skyline Join Queries: Skylines with Aggregate Operations over Multiple Relations

Arnab Bhattacharya, B. Palvali Teja
arnabb@iitk.ac.in, tpalvali@amazon.com

Dept. of Computer Science and Engineering, Indian Institute of Technology, Kanpur
Amazon Development Centre, Hyderabad

COMAD
8th December, 2010
A practical problem

- Flying from city A to city B where there is no direct flight
- Join flights from city A to those to city B (using one intermediate city)
- Prefer flights with better ratings and amenities
- More importantly, prefer combination of flights with lower total cost and lower total duration
- Translates nicely to skyline paradigm
Skylines address the problem of multi-criteria decision making where there is no clear preference function.

In the above example, ratings, amenities, total cost and total duration are all important.

Obviously, a flight pair having lower ratings, lower amenities, higher cost and higher duration will never be preferred.

However, for all other flight pairs, it is not clear what the user wants.

Skyline shows an overall big picture for more thorough consideration.
Skylines address the problem of multi-criteria decision making where there is no clear preference function. In the above example, ratings, amenities, total cost and total duration are all important. Obviously, a flight pair having lower ratings, lower amenities, higher cost and higher duration will never be preferred. However, for all other flight pairs, it is not clear what the user wants. Skyline shows an overall big picture for more thorough consideration. Skyline is incorporated in PostgreSQL systems with a SQL syntax:

```sql
SELECT f1.fno, f2.fno, f1.dst, f2.src,
      f1.arr, f2.dep, f1.rtg, f2.rtg, f1.amn, f2.amn,
      cost AS f1.cost + f2.cost, duration AS f1.duration + f2.duration
FROM FlightsA AS f1, FlightsB AS f2
WHERE f1.dst = f2.src AND f1.arr < f2.dep AND
      SKYLINE of cost MIN, duration MIN,
      f1.rtg MAX, f2.rtg MAX, f1.amn MAX, f2.amn MAX
```

Arnab Bhattacharya, CSE, IITK
## Example

### Flights from city A (FlightsA)

<table>
<thead>
<tr>
<th>fno</th>
<th>dep</th>
<th>Join (H)</th>
<th>Aggregate (G)</th>
<th>Local (L)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>arr</td>
<td>dst</td>
<td>duration</td>
</tr>
<tr>
<td>11</td>
<td>06:30</td>
<td>08:40</td>
<td>C</td>
<td>2h 10m</td>
</tr>
<tr>
<td>12</td>
<td>07:00</td>
<td>09:00</td>
<td>E</td>
<td>2h 00m</td>
</tr>
<tr>
<td>14</td>
<td>08:05</td>
<td>10:00</td>
<td>E</td>
<td>1h 55m</td>
</tr>
<tr>
<td>15</td>
<td>09:50</td>
<td>10:40</td>
<td>C</td>
<td>1h 40m</td>
</tr>
<tr>
<td>13</td>
<td>12:00</td>
<td>13:50</td>
<td>C</td>
<td>1h 50m</td>
</tr>
<tr>
<td>16</td>
<td>16:00</td>
<td>17:30</td>
<td>D</td>
<td>1h 30m</td>
</tr>
<tr>
<td>17</td>
<td>17:00</td>
<td>20:20</td>
<td>C</td>
<td>3h 20m</td>
</tr>
</tbody>
</table>

### Flights to city B (FlightsB)

<table>
<thead>
<tr>
<th>fno</th>
<th>src</th>
<th>dep</th>
<th>Join (H)</th>
<th>Aggregate (G)</th>
<th>Local (L)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>arr</td>
<td>dst</td>
<td>duration</td>
</tr>
<tr>
<td>21</td>
<td>C</td>
<td>09:50</td>
<td>12:00</td>
<td>2h 10m</td>
<td>162</td>
</tr>
<tr>
<td>26</td>
<td>C</td>
<td>16:00</td>
<td>18:49</td>
<td>2h 49m</td>
<td>160</td>
</tr>
<tr>
<td>23</td>
<td>C</td>
<td>16:00</td>
<td>18:45</td>
<td>2h 45m</td>
<td>160</td>
</tr>
<tr>
<td>25</td>
<td>D</td>
<td>16:00</td>
<td>17:49</td>
<td>1h 49m</td>
<td>220</td>
</tr>
<tr>
<td>22</td>
<td>D</td>
<td>17:00</td>
<td>19:00</td>
<td>2h 00m</td>
<td>166</td>
</tr>
<tr>
<td>27</td>
<td>E</td>
<td>20:00</td>
<td>21:46</td>
<td>1h 46m</td>
<td>200</td>
</tr>
<tr>
<td>24</td>
<td>E</td>
<td>20:00</td>
<td>21:30</td>
<td>1h 30m</td>
<td>160</td>
</tr>
</tbody>
</table>

### Part of the joined relation (FlightsA \⋊ ⋉ FlightsB)

<table>
<thead>
<tr>
<th>f1.fno</th>
<th>f2.fno</th>
<th>f1.dst</th>
<th>f2.src</th>
<th>f1.arr</th>
<th>f2.dep</th>
<th>f1.amn</th>
<th>f2.amn</th>
<th>f1.rtg</th>
<th>f2.rtg</th>
<th>cost</th>
<th>duration</th>
<th>Skyline</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>21</td>
<td>C</td>
<td>C</td>
<td>08:40</td>
<td>09:50</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>324</td>
<td>4h 20m</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>23</td>
<td>C</td>
<td>C</td>
<td>08:40</td>
<td>16:00</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>322</td>
<td>4h 55m</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>23</td>
<td>C</td>
<td>C</td>
<td>13:50</td>
<td>16:00</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>333</td>
<td>4h 35m</td>
<td>No</td>
</tr>
<tr>
<td>15</td>
<td>23</td>
<td>C</td>
<td>C</td>
<td>10:40</td>
<td>16:00</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>430</td>
<td>4h 25m</td>
<td>No</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
<td>E</td>
<td>E</td>
<td>09:00</td>
<td>20:00</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>326</td>
<td>3h 30m</td>
<td>Yes</td>
</tr>
<tr>
<td>14</td>
<td>24</td>
<td>E</td>
<td>E</td>
<td>10:00</td>
<td>20:00</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>300</td>
<td>3h 25m</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Aggregated Skyline Join Queries

- **f1.fno**
- **f2.fno**
- **f1.dst**
- **f2.src**
- **f1.arr**
- **f2.dep**
- **f1.amn**
- **f2.amn**
- **f1.rtg**
- **f2.rtg**
- **cost**
- **duration**
- **Skyline**
Efficient algorithms exist for:

- Skyline computation on a single relation
- Skyline computation on a joined relation where the preferences are on attributes of the base relations
Efficient algorithms exist for:

- Skyline computation on a single relation
- Skyline computation on a joined relation where the preferences are on attributes of the base relations

We propose skyline computation on a joined relation where preferences are both on:

- Individual attributes that are local to a base relation
- Attributes whose values are aggregates of attributes from the two relations
  - Total cost, i.e., cost of flight 1 + cost of flight 2
  - Total duration, i.e., duration of flight 1 + duration of flight 2

We coin these queries “aggregate skyline join queries” or ASJQ
Efficient algorithms exist for:

- Skyline computation on a single relation
- Skyline computation on a joined relation where the preferences are on attributes of the base relations

We propose skyline computation on a joined relation where preferences are both on:

- Individual attributes that are local to a base relation
- Attributes whose values are aggregates of attributes from the two relations
  - Total cost, i.e., cost of flight 1 + cost of flight 2
  - Total duration, i.e., duration of flight 1 + duration of flight 2

We coin these queries “aggregate skyline join queries” or ASJQ

Useful in many applications

- Buying a digital camera and a compatible memory card
- Buying a team of good batsmen and bowlers
**Skyline Tuple**

**Definition (Dominance)**

A tuple \( r \) in a relation \( R \) *dominates* another tuple \( s \in R \), denoted by \( r \succ s \), if there exists at least one attribute where \( r \) is strictly preferred over \( s \) and in all other attributes, \( r \) is at least as preferred as \( s \).

- **Example:** preference functions are *minimum*
  - \( A = \{4, 5, 7\}, \ B = \{2, 5, 6\}, \ C = \{3, 6, 7\} \)
  - \( B \succ A; \ B \succ C; \ A \not\succ C; \ C \not\succ A \)

- A *skyline tuple* is one that is *not* dominated by any other tuple in the relation
  - For above example, it is only \( B \)
**Local attributes**

**Definition (Local attributes)**

The attributes of a relation on which preferences are applied for the purposes of skyline computation, but no aggregate operation with an attribute from the other relation is performed, are denoted as *local attributes*.

- Example: amenities, rating
Definition (Aggregate attributes)

The attributes of a relation, on which an aggregate operation is performed with another attribute from the other relation, and then preferences are applied on the aggregated value for skyline computation, are denoted as aggregate attributes.

- Example: cost, duration
**Definition (Join attributes)**

The attributes of a relation, on which no skyline preferences are specified, but are used to specify the join conditions between the two relations, are denoted as *join attributes*.

- Example: source, destination, departure, arrival
**Dominance**

- **Full dominance**: A tuple $r$ fully dominates $s$ if $r$ dominates $s$ in both the local and aggregate attributes.

- **Local dominance**: A tuple $r$ locally dominates $s$ if $r$ dominates $s$ in only the local attributes.

- Full dominance implies local dominance but not vice versa.

- If a tuple does not dominate another tuple locally, it does not dominate it fully either.
Dominance with join attributes

- Dominance relationships help infer certain properties in the final joined set.
- For that, it is necessary that whenever a tuple \( t' = u \bowtie v' \) exists in the final relation, the tuple \( t = u \bowtie v \), where \( v' \succ v \), also exists.
- However, the join attributes of \( v' \) and \( v \) may be such that only \( v' \) satisfies the join condition with \( u \), but \( v \) does not.
- Hence, inference about \( t' \) on the assumption that \( t \) exists is wrong.
- Example
  - Flight 15 is dominated by flight 16.
  - However, flight 15 can join with flight 23 which flight 16 cannot.
- Therefore, preferences over join attributes need to be considered while considering dominance.
Preferences over join attributes

• Suppose join condition for two join attributes $a \in A$ and $b \in B$ is $A.a \odot B.b$
• $\odot$ may be any of $\neq, <, \leq, >, \geq$
• For tuple $u' \in A$ to be dominated by $u \in A$, whenever $u'$ joins with $v \in B$, $u$ must be able to join with $v$ as well
• If $\odot$ is $\neq$, then $u.a = u'.a$, both being equal to $v.b$
• If $\odot$ is $<$, then $u.a < u'.a$ (sufficient)
• Thus, join attribute is also considered a skyline attribute
• Definitions of full and local dominance are modified to include preferences over join attributes as well

<table>
<thead>
<tr>
<th>Join condition</th>
<th>$u \in A \succ u' \in A$ if</th>
<th>$v \in B \succ v' \in B$ if</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A.a = B.b$</td>
<td>$u.a = u'.a$</td>
<td>$v.b = v'.b$</td>
</tr>
<tr>
<td>$A.a &lt; B.b, A.a \leq B.b$</td>
<td>$u.a \leq u'.a$</td>
<td>$v.b \geq v'.b$</td>
</tr>
<tr>
<td>$A.a &lt; B.b, A.a \geq B.b$</td>
<td>$u.a \geq u'.a$</td>
<td>$v.b \leq v'.b$</td>
</tr>
</tbody>
</table>
Naïve Algorithm

- Compute join
- Perform aggregates
- Compute skylines over all preferences
Naïve Algorithm

- Compute join
- Perform aggregates
- Compute skylines over all preferences
- Computationally expensive
- Impractical
Performing skylines before join: Full skylines

- Some skyline computation can be done before joining
- Denote *full* skyline sets by $A_0$ and $B_0$
- Non-skyline sets are $A'_0 = A - A_0$ and $B'_0 = B - B_0$
Performing skylines before join: Full skylines

- Some skyline computation can be done before joining
- Denote *full* skyline sets by \( A_0 \) and \( B_0 \)
- Non-skyline sets are \( A'_0 = A - A_0 \) and \( B'_0 = B - B_0 \)
- Theorem: Tuples formed by joining \( A'_0 \) or \( B'_0 \) cannot be part of the final skyline set
- Proof
  - Assume a tuple \( t' = u \in A_0 \bowtie v' \in B'_0 \)
  - Consider another tuple \( t = u \in A_0 \bowtie v \in B_0 \).
  - Since \( v \succeq v' \), \( t \succeq t' \)
Performing skylines before join: Full skylines

- Some skyline computation can be done before joining
- Denote *full* skyline sets by $A_0$ and $B_0$
- Non-skyline sets are $A'_0 = A - A_0$ and $B'_0 = B - B_0$
- Theorem: Tuples formed by joining $A'_0$ or $B'_0$ cannot be part of the final skyline set
- Proof
  - Assume a tuple $t' = u \in A_0 \bowtie v' \in B'_0$
  - Consider another tuple $t = u \in A_0 \bowtie v \in B_0$.
  - Since $v \succ v'$, $t \succ t'$
- Effect: Prunes all tuples in $A'_0 \bowtie B_0$, $A_0 \bowtie B'_0$ and $A'_0 \bowtie B'_0$
Performing Skylines before Join: Local Skylines

- Denote local skyline sets in $A_0$ and $B_0$ by $A_1$ and $B_1$ respectively.
- Non-skyline sets are $A'_1 = A_0 - A_1$ and $B'_1 = B_0 - B_1$. 
Performing skylines before join: Local skylines

- Denote *local* skyline sets in $A_0$ and $B_0$ by $A_1$ and $B_1$ respectively
- Non-skyline sets are $A'_1 = A_0 - A_1$ and $B'_1 = B_0 - B_1$
- Theorem: Tuples formed by joining $A_1$ or $B_1$ are surely part of the final skyline set
- Proof
  - Assume a tuple $t = u \in A_1 \Join v' \in B'_1$
  - Consider any other tuple $t' = u' \in A_0 \Join v' \in B'_1$.
  - Since $u$ is a local skyline, $\not\exists u', u' \not\succ u$
  - Therefore, $\not\exists t', t' \succ t$
Performing skylines before join: Local skylines

- Denote local skyline sets in $A_0$ and $B_0$ by $A_1$ and $B_1$ respectively.
- Non-skyline sets are $A'_1 = A_0 - A_1$ and $B'_1 = B_0 - B_1$.
- Theorem: Tuples formed by joining $A_1$ or $B_1$ are surely part of the final skyline set.
- Proof
  - Assume a tuple $t = u \in A_1 \bowtie v' \in B'_1$.
  - Consider any other tuple $t' = u' \in A_0 \bowtie v' \in B'_1$.
  - Since $u$ is a local skyline, $\not\exists u', u' \not\succ u$.
  - Therefore, $\not\exists t', t' \succ t$.
- Effect: Outputs all tuples in $A'_1 \bowtie B'_1$, $A_1 \bowtie B'_1$ and $A_1 \bowtie B_1$.
- Only $A'_1 \bowtie B'_1$ needs to be examined.
**Naïve Algorithm**
Performing Skylines before Join

**Multiple Skyline Computations Algorithm**

**Dominator-based Approach**
Iterative Algorithm

---

### Example

<table>
<thead>
<tr>
<th>fno</th>
<th>dep</th>
<th>arr</th>
<th>dst</th>
<th>duration</th>
<th>cost</th>
<th>amn</th>
<th>rtg</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>06:30</td>
<td>08:40</td>
<td>C</td>
<td>2h 10m</td>
<td>162</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>07:00</td>
<td>09:00</td>
<td>E</td>
<td>2h 00m</td>
<td>166</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>08:05</td>
<td>10:00</td>
<td>E</td>
<td>1h 55m</td>
<td>140</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>09:50</td>
<td>10:40</td>
<td>C</td>
<td>1h 40m</td>
<td>270</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>12:00</td>
<td>13:50</td>
<td>C</td>
<td>1h 50m</td>
<td>173</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>16:00</td>
<td>17:30</td>
<td>D</td>
<td>1h 30m</td>
<td>230</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>17:00</td>
<td>20:20</td>
<td>C</td>
<td>3h 20m</td>
<td>183</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**Flights from city A (FlightsA)**

<table>
<thead>
<tr>
<th>fno</th>
<th>src</th>
<th>dep</th>
<th>arr</th>
<th>duration</th>
<th>cost</th>
<th>amn</th>
<th>rtg</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>C</td>
<td>09:50</td>
<td>12:00</td>
<td>2h 10m</td>
<td>162</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>26</td>
<td>C</td>
<td>16:00</td>
<td>18:49</td>
<td>2h 49m</td>
<td>160</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>23</td>
<td>C</td>
<td>16:00</td>
<td>18:45</td>
<td>2h 45m</td>
<td>160</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>25</td>
<td>D</td>
<td>16:00</td>
<td>17:49</td>
<td>1h 49m</td>
<td>220</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>22</td>
<td>D</td>
<td>17:00</td>
<td>19:00</td>
<td>2h 00m</td>
<td>166</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>27</td>
<td>E</td>
<td>20:00</td>
<td>21:46</td>
<td>1h 46m</td>
<td>200</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>E</td>
<td>20:00</td>
<td>21:30</td>
<td>1h 30m</td>
<td>160</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**Flights to city B (FlightsB)**

<table>
<thead>
<tr>
<th>Set</th>
<th>Flight numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_0$</td>
<td>$A_1$</td>
</tr>
<tr>
<td></td>
<td>11, 12</td>
</tr>
<tr>
<td>$A_1'$</td>
<td>$A_2$</td>
</tr>
<tr>
<td></td>
<td>13, 14</td>
</tr>
<tr>
<td>$A_2'$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15, 16</td>
</tr>
<tr>
<td>$A_0'$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>Flight numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_0$</td>
<td>$B_1$</td>
</tr>
<tr>
<td></td>
<td>21, 22</td>
</tr>
<tr>
<td>$B_1'$</td>
<td>$B_2$</td>
</tr>
<tr>
<td></td>
<td>23</td>
</tr>
<tr>
<td>$B_2'$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>24, 25</td>
</tr>
<tr>
<td>$B_0'$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>26, 27</td>
</tr>
</tbody>
</table>

**Aggregate Skyline Join Queries**

---

 Arnab Bhattacharya, CSE, IITK
Multiple skyline computations (MSC) algorithm

- Utilizes Theorem 1 to prune all tuples in $A'_0 \Join B_0$, $A_0 \Join B'_0$ and $A'_0 \Join B'_0$
- Utilizes Theorem 2 to output all tuples in $A'_1 \Join B_1$, $A_1 \Join B'_1$ and $A_1 \Join B_1$
- Examines $A'_1 \Join B'_1$ fully
  - Tests every tuple by checking whether any other tuple in $A_0 \Join B_0$ dominates it
**Dominator-based approach**

- A tuple $t' = u' \in A'_1 \bowtie v' \in B'_1$ can be dominated only by certain tuples in $A_0 \bowtie B_0$
- Suppose the *local* dominators of $u'$ and $v'$ are denoted by $ld(u')$ and $ld(v')$ respectively
- Lemma: $t'$ can be dominated only by $t$ of the form $t = u \in ld(u') \bowtie v \in ld(v')$
- Proof
  - Consider a tuple $u \notin ld(u')$ and consider any tuple $t = u \bowtie v$
  - Local attributes of $u'$ are *not* dominated by $u$
  - Therefore, local attributes of $t'$ are also not dominated by $t$
**Dominator-based approach**

- A tuple $t' = u' \in A'_1 \bowtie v' \in B'_1$ can be dominated only by certain tuples in $A_0 \bowtie B_0$
- Suppose the *local* dominators of $u'$ and $v'$ are denoted by $ld(u')$ and $ld(v')$ respectively
- Lemma: $t'$ can be dominated only by $t$ of the form $t = u \in ld(u') \bowtie v \in ld(v')$
- **Proof**
  - Consider a tuple $u \notin ld(u')$ and consider any tuple $t = u \bowtie v$
  - Local attributes of $u'$ are *not* dominated by $u$
  - Therefore, local attributes of $t'$ are also not dominated by $t$
- **Effect**: A tuple $t \in A'_1 \bowtie B'_1$ need not be checked against all tuples in $A_0 \bowtie B_0$, but only those in $ld(u') \bowtie ld(v')$
- **Maintaining local dominator sets** $ld(.)$ may be costly
Iterative Algorithm

- Cost of comparing all tuples in $ld(A'_1)$ and $ld(B'_1)$ is high
- Divide $A'_1$ and $B'_1$ further into *local* skyline sets $A_2$ and $B_2$ respectively
- Non-skyline sets are $A'_2 = A'_1 - A_2$ and $B'_2 = B'_1 - B_2$
- This division of $A_0$ is carried on iteratively into $A_1, A_2, \ldots, A_k, A'_k$
- Similar division of $B_0$ into $B_1, B_2, \ldots, B_k, B'_k$
TARGET SETS

- Dominators of a certain set can exist only in certain other sets
- For example, a tuple in $A_2 \Join B_2$ needs to be compared with tuples in $A_1 \Join B_1$ only
- No unnecessary comparison with $(A_1 \Join B'_1) \cup (A'_1 \Join B_1) \cup (A'_1 \Join B'_1)$

<table>
<thead>
<tr>
<th>Set</th>
<th>Target Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2 \Join B_2$</td>
<td>$A_1 \Join B_1$</td>
</tr>
<tr>
<td>$A_2 \Join B'_2$</td>
<td>$A_1 \Join B_1, A_1 \Join B'_1$</td>
</tr>
<tr>
<td>$A'_2 \Join B_2$</td>
<td>$A_1 \Join B_1, A'_1 \Join B_1$</td>
</tr>
<tr>
<td>$A'_2 \Join B'_2$</td>
<td>$A_1 \Join B_1, A_1 \Join B'_1, A'_1 \Join B_1$</td>
</tr>
</tbody>
</table>
When there is only one aggregate attribute, the case is quite simpler

- Lemma: All tuples in $A_0 \bowtie B_0$ are part of the final answer set
When there is only one aggregate attribute, the case is quite simpler.

Lemma: All tuples in $A_0 \Join B_0$ are part of the final answer set.

Proof:
- Consider a tuple $t' = u' \in A_1 \Join v' \in B_1$
- Claim: $\not\exists t, t \succ t'$
- Suppose such a $t = u \Join v$ exists.
- Therefore, $u \succ_{ld} u'$ and $v \succ_{ld} v'$
- However, since $u' \in A_0$ and $v' \in B_0$, $u \not\succ_{fd} u'$ and $v \not\succ_{fd} v'$
- Therefore, it must be that $u' \succ_g u$ and $v' \succ_g v$
- This implies that $t \not\succ t'$
**Single aggregate attribute**

- When there is only one aggregate attribute, the case is quite simpler
- Lemma: All tuples in $A_0 \bowtie B_0$ are part of the final answer set
- Proof
  - Consider a tuple $t' = u' \in A'_1 \bowtie v' \in B'_1$
  - Claim: $\not\exists t, t \succ t'$
  - Suppose such a $t = u \bowtie v$ exists
  - Therefore, $u \succ_{id} u'$ and $v \succ_{id} v'$
  - However, since $u' \in A_0$ and $v' \in B_0$, $u \not\succ_{fd} u'$ and $v \not\succ_{fd} v'$
  - Therefore, it must be that $u' \succ_g u$ and $v' \succ_g v$
  - This implies that $t \not\succ t'$
- Effect: Finding local skylines is enough
Performance of naïve algorithm

- Naïve algorithm takes much more time
- Performance is independent of cardinality of final answer set
- Overall, iterative algorithm is the best
# Default experimental parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of local attributes</td>
<td>$L$</td>
<td>2</td>
</tr>
<tr>
<td>Number of aggregate attributes</td>
<td>$G$</td>
<td>2</td>
</tr>
<tr>
<td>Cardinality of datasets</td>
<td>$N$</td>
<td>40000</td>
</tr>
<tr>
<td>Number of categories</td>
<td>$C$</td>
<td>10</td>
</tr>
<tr>
<td>Distribution of datasets</td>
<td>$D$</td>
<td>Correlated</td>
</tr>
</tbody>
</table>

Arnab Bhattacharya, CSE, IITK

Aggregate Skyline Join Queries
Effect of number of local attributes

- Running time increases almost exponentially with number of local attributes
- Iterative shows best scalability
Effect of number of aggregate attributes

- Running time increases almost exponentially with number of aggregate attributes
- Absolute times are lower
Effect of dataset cardinality

- Scalability is better than quadratic
Effect of dataset distribution

- Cardinality of final answer set is much higher in anti-correlated datasets
- Iterative shows the best comparative advantage in this case
Effect of categories of join attribute measures the possible values of the join attribute (equi-join)

- When number of join categories increases
  - Full skyline sets $A_0$ and $B_0$ become larger as there is less probability of a tuple matching another tuple in the join attribute, and therefore, dominating it.
Effect of categories of join attribute

- For two relations having $N$ tuples with $C$ categories, the cardinality of the joined set is $C \times (N/C)^2 = N^2/C$
- At higher number of join categories
  - The cardinality of the joined set is low leading to a lower cardinality
- When number of join categories is low
  - The number of tuples in each category is high
  - However, there is a higher chance of a tuple being dominated thereby leading to a lower cardinality
**Conclusions**

- Proposed a novel query – **Aggregate Skyline Join Query**
- Extended the general skyline operator to multiple relations involving joins using aggregate operations over attributes from different relations
- Extensions to distributed and parallel environments
Conclusions

- Proposed a novel query – **Aggregate Skyline Join Query**
- Extended the general skyline operator to multiple relations involving joins using aggregate operations over attributes from different relations
- Extensions to distributed and parallel environments

THANK YOU!
CONCLUSIONS

- Proposed a novel query – **Aggregate Skyline Join Query**
- Extended the general skyline operator to multiple relations involving joins using aggregate operations over attributes from different relations
- Extensions to distributed and parallel environments

THANK YOU!

Questions?