Skylines: Databases' Answer to Multiple Preferences

Arnab Bhattacharya
arnabb@cse.iitk.ac.in

Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Faculty Seminar Series and SigData

7th September, 2012

Motivation

- Suppose you are flying from Kolkata to Agartala
- Any booking site will show you hundreds of flights
- Which one to choose?

Motivation

- Suppose you are flying from Kolkata to Agartala
- Any booking site will show you hundreds of flights
- Which one to choose?
  - Cost not always the *only* criterion
  - Time of day, ratings, amenities of flights, etc. may also be important
- In short, multiple preferences for selecting a flight
  - Flight A has less cost but less amenities than flight B
  - Not clear which one to choose

Motivation

- Suppose you are flying from Kolkata to Agartala
- Any booking site will show you hundreds of flights
- Which one to choose?
  - Cost not always the *only* criterion
  - Time of day, ratings, amenities of flights, etc. may also be important
- In short, multiple preferences for selecting a flight
  - Flight A has less cost but less amenities than flight B
  - Not clear which one to choose
- Leave the choice to user by letting her examine both

SKYLINE QUERIES

- However, if preferences are strictly better, then the choice is clear
  - Flight C with more cost and less amenities than flight A
  - Flight C will *never* be preferred over flight A

Skyline Queries

- However, if preferences are strictly better, then the choice is clear
  - Flight C with more cost and less amenities than flight A
  - Flight C will *never* be preferred over flight A
- Skylines are those objects that are not dominated by some other object for all preferences
- Also known as Pareto optimal curve or maximal vector problem

Formal Definition

- Each object in database D has k' attributes
- Skyline query over $k \leq k'$ attributes

Formal Definition

- Each object in database D has k' attributes
- Skyline query over $k \leq k'$ attributes
- For each skyline attribute, a preference function is defined
  - Can be any comparison operator, denoted by $\succ$
  - If it is at least equal, then it is denoted by $\succeq$

Formal Definition

- Each object in database D has k' attributes
- Skyline query over $k \leq k'$ attributes
- For each skyline attribute, a preference function is defined
  - Can be any comparison operator, denoted by $\succ$
  - If it is at least equal, then it is denoted by $\succeq$
- An object O_p dominates another object O_q if

$$O_p \succ O_q \iff \forall i,\ O_{pi} \succeq O_{qi} \text{ and } \exists j,\ O_{pj} \succ O_{qj}$$

Formal Definition

- Each object in database D has k' attributes
- Skyline query over $k \leq k'$ attributes
- For each skyline attribute, a preference function is defined
  - Can be any comparison operator, denoted by $\succ$
  - If it is at least equal, then it is denoted by $\succeq$
- An object O_p dominates another object O_q if

$$O_p \succ O_q \iff \forall i,\ O_{pi} \succeq O_{qi} \text{ and } \exists j,\ O_{pj} \succ O_{qj}$$

- The skyline query [BKS01] returns the set of objects $S \subseteq D$ such that for all objects $O_p \in S$ and all objects $O_q \notin S$,

$$O_p \in S \iff \nexists O_q \in D,\ O_q \succ O_p$$
$$O_q \notin S \iff \exists O_p \in S,\ O_p \succ O_q$$

Skyline Syntax in SQL

- PostgreSQL has adopted skyline keyword in SQL [Ede09]

```
SELECT fno, dep, arr, cost, rtg, amn
FROM flights
WHERE src = ``Kolkata'' and dest = ``Agartala'' and
SKYLINE of cost min, rtg max, amn max
```

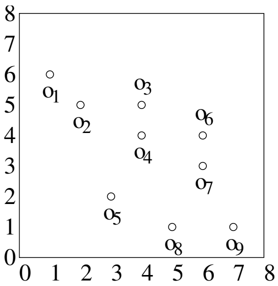
Preference Functions

- If object O_p dominates object O_q, then for all monotone preference functions, O_p is better than O_q
- Moreover, for every skyline object O_p, there exists a monotone preference function f such that O_p optimizes f
- This is why this is much more complex and interesting than a single optimization function

Preference Functions

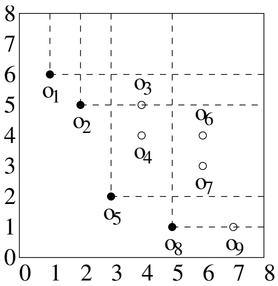
- If object O_p dominates object O_q, then for all monotone preference functions, O_p is better than O_q
- Moreover, for every skyline object O_p, there exists a monotone preference function f such that O_p optimizes f
- This is why this is much more complex and interesting than a single optimization function
- There is no ranking or ordering among skyline objects

Example



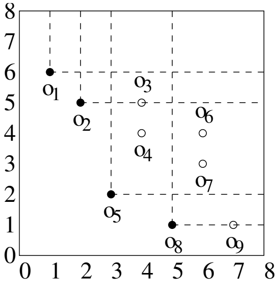
- Assume preference functions are less ($<$)

Example



- Assume preference functions are less ($<$)
- - - lines show the region dominated by the corresponding object

Example



- Assume preference functions are less ($<$)
- - - lines show the region dominated by the corresponding object
- Skyline objects are O_1, O_2, O_5, O_8

Naïve Algorithm

- Compare each object with every other object
- An object is a skyline only if no object dominates it

Naïve Algorithm

- Compare each object with every other object
- An object is a skyline only if no object dominates it
- $O(n^2)$ for n objects

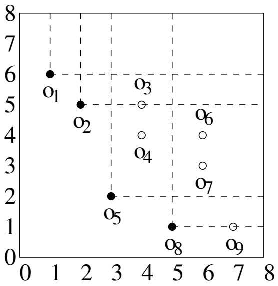
Block-nested-loop (BNL) algorithm

- First skyline algorithm [BKS01]
- Maintain a window of objects not yet dominated
- Check every new object against the window
  1. If it is dominated, prune it
  2. If it dominates some other object(s) in the window, prune those object(s)
  3. If neither, add to the window

Block-nested-loop (BNL) algorithm

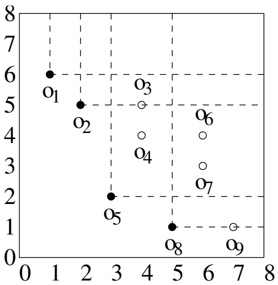
- First skyline algorithm [BKS01]
- Maintain a window of objects not yet dominated
- Check every new object against the window
  1. If it is dominated, prune it
  2. If it dominates some other object(s) in the window, prune those object(s)
  3. If neither, add to the window
- Finally, window contains *only* the skyline objects

Example



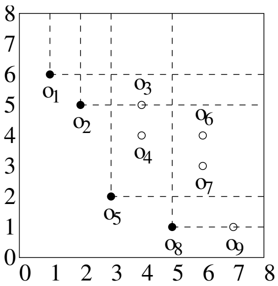
Window: Φ

Example



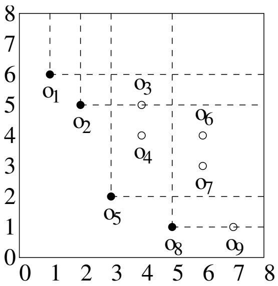
Window: O_1

Example



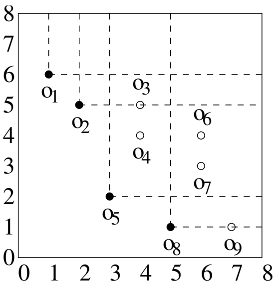
Window: O_1, O_2

Example



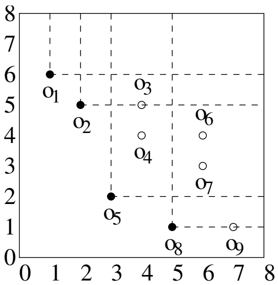
Window: O_1, O_2
Pruned: O_3

Example



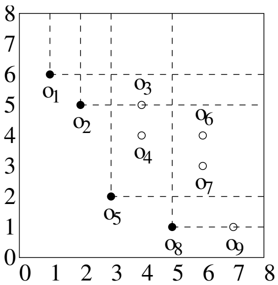
Window: O_1, O_2, O_4
Pruned: O_3

Example



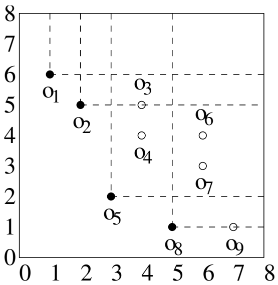
Window: O_1, O_2, O_5
Pruned: O_3, O_4

Example



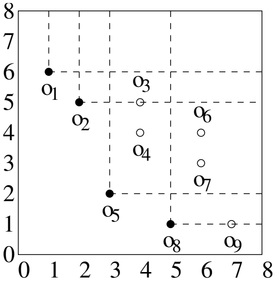
Window: O_1, O_2, O_5
Pruned: O_3, O_4, O_6

Example



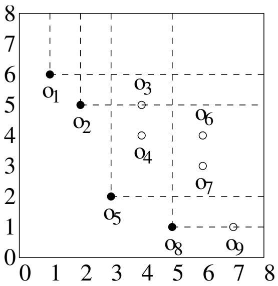
Window: O_1, O_2, O_5
Pruned: O_3, O_4, O_6, O_7

Example



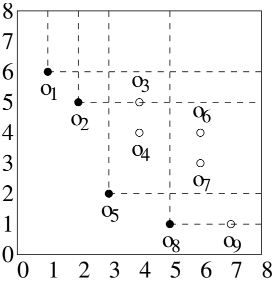
Window: O_1, O_2, O_5, O_8
Pruned: O_3, O_4, O_6, O_7

Example



Window: O_1, O_2, O_5, O_8
Pruned: O_3, O_4, O_6, O_7, O_9

Example



Window: O_1, O_2, O_5, O_8
Pruned: O_3, O_4, O_6, O_7, O_9
Skylines: O_1, O_2, O_5, O_8

Correctness of BNL Algorithm

- Consider two objects *A* and *B* that do not dominate each other
  - Both are added to window

Correctness of BNL Algorithm

- Consider two objects A and B that do not dominate each other
  - Both are added to window
- Otherwise, suppose $A \succ B$
- If A arrives before B
  - When B arrives, it is checked against A and pruned
- If B arrives before A
  - If B is in window, A prunes it and is added to window
  - If B is not in window, A is simply added to window

Correctness of BNL Algorithm

- Consider two objects A and B that do not dominate each other
  - Both are added to window
- Otherwise, suppose $A \succ B$
- If A arrives before B
  - When B arrives, it is checked against A and pruned
- If B arrives before A
  - If B is in window, A prunes it and is added to window
  - If B is not in window, A is simply added to window
- Every non-skyline object is dominated by at least one skyline object
  - Uses transitivity relationship of object dominance

Sort-filter-skyline (SFS) Algorithm

- Sort objects by some *monotone* scoring function [CGGL03]
- Use the ordered list as input for BNL algorithm

Sort-filter-skyline (SFS) Algorithm

- Sort objects by some *monotone* scoring function [CGGL03]
- Use the ordered list as input for BNL algorithm
- Example: entropy function

$$E(O_i) = \sum_{j=1}^{d} \ln(O_{ij} + 1)$$

Sort-filter-skyline (SFS) Algorithm

- Sort objects by some *monotone* scoring function [CGGL03]
- Use the ordered list as input for BNL algorithm
- Example: entropy function

$$E(O_i) = \sum_{j=1}^{d} \ln(O_{ij} + 1)$$

- Assume that all preferences are less ($<$)
- If $A \succ B$,

Sort-filter-skyline (SFS) Algorithm

- Sort objects by some *monotone* scoring function [CGGL03]
- Use the ordered list as input for BNL algorithm
- Example: entropy function

$$E(O_i) = \sum_{j=1}^{d} \ln(O_{ij} + 1)$$

- Assume that all preferences are less ($<$)
- If $A \succ B$, then necessarily $E(A) < E(B)$
- If $E(A) < E(B)$,

Sort-filter-skyline (SFS) Algorithm

- Sort objects by some *monotone* scoring function [CGGL03]
- Use the ordered list as input for BNL algorithm
- Example: entropy function

$$E(O_i) = \sum_{j=1}^{d} \ln(O_{ij} + 1)$$

- Assume that all preferences are less ($<$)
- If $A \succ B$, then necessarily $E(A) < E(B)$
- If $E(A) < E(B)$, nothing can be concluded

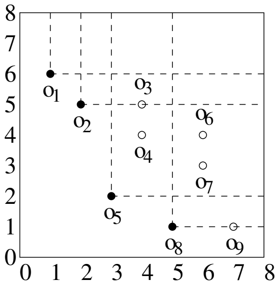
Sort-filter-skyline (SFS) Algorithm

- Sort objects by some *monotone* scoring function [CGGL03]
- Use the ordered list as input for BNL algorithm
- Example: entropy function

$$E(O_i) = \sum_{j=1}^{d} \ln(O_{ij} + 1)$$

- Assume that all preferences are less ($<$)
- If $A \succ B$, then necessarily $E(A) < E(B)$
- If $E(A) < E(B)$, nothing can be concluded
  - $A \succ B$ or A and B do not dominate each other
  - $B \not\succ A$ as then $E(B) < E(A)$

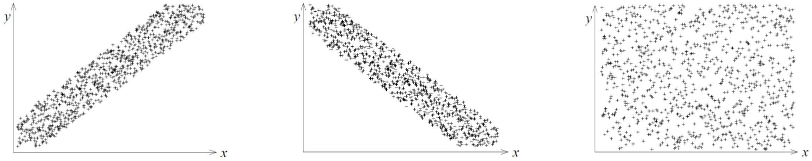
Example



Object	Value	Entropy
O_1	(1,6)	2.6390
O_2	(2,5)	2.8903
O_3	(4,5)	3.4011
O_4	(4,4)	3.2188
O_5	(3,2)	2.4848
O_6	(6,4)	3.5553
O_7	(6,3)	3.3321
O_8	(5,1)	2.4848
O_9	(7,1)	2.7725

- Sorting produces a much better scanning order
- Here, it is O_5, O_8, O_1, O_9, O_2, O_4, O_7, O_3, O_6

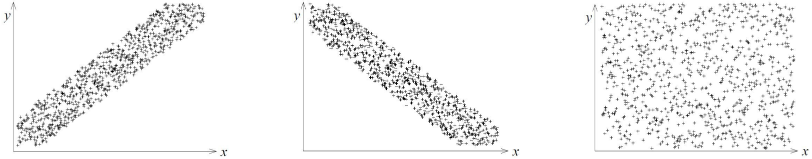
Number of Skyline Objects



Correlated Anti-correlated Independent

- Assume N objects having d skyline preference atributes
- For *correlated* data, skyline cardinality is

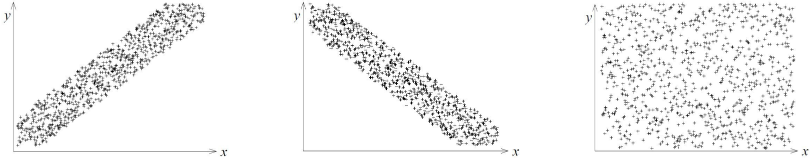
Number of Skyline Objects



Correlated | Anti-correlated | Independent

- Assume N objects having d skyline preference atributes
- For *correlated* data, skyline cardinality is *low*
  - For perfectly correlated data, number of skyline objects is

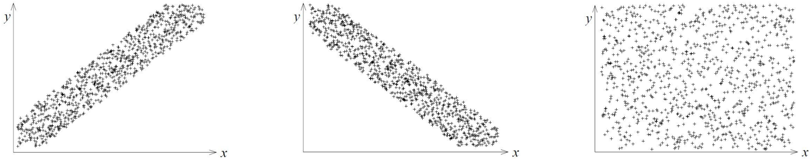
Number of Skyline Objects



Correlated Anti-correlated Independent

- Assume *N* objects having *d* skyline preference atributes
- For *correlated* data, skyline cardinality is *low*
  - For perfectly correlated data, number of skyline objects is 1
- For *anti-correlated* data, skyline cardinality is

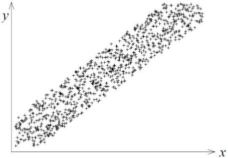
Number of Skyline Objects



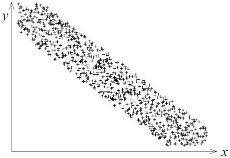
Correlated | Anti-correlated | Independent

- Assume *N* objects having *d* skyline preference atributes
- For *correlated* data, skyline cardinality is *low*
  - For perfectly correlated data, number of skyline objects is 1
- For *anti-correlated* data, skyline cardinality is *high*
  - For perfectly anti-correlated data, number of skyline objects is

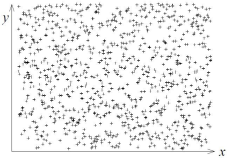
Number of Skyline Objects



Correlated



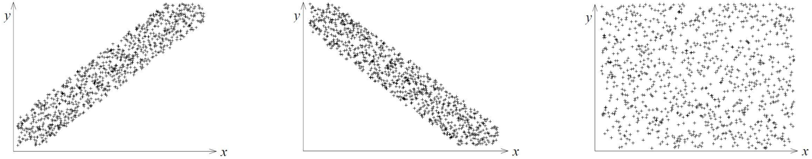
Anti-correlated



Independent

- Assume N objects having d skyline preference atributes
- For *correlated* data, skyline cardinality is *low*
  - For perfectly correlated data, number of skyline objects is 1
- For *anti-correlated* data, skyline cardinality is *high*
  - For perfectly anti-correlated data, number of skyline objects is N
- For *independent* data, skyline cardinality is

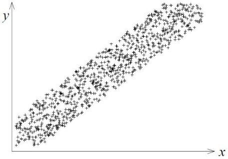
Number of Skyline Objects



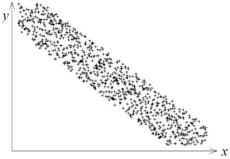
Correlated | Anti-correlated | Independent

- Assume *N* objects having *d* skyline preference atributes
- For *correlated* data, skyline cardinality is *low*
  - For perfectly correlated data, number of skyline objects is 1
- For *anti-correlated* data, skyline cardinality is *high*
  - For perfectly anti-correlated data, number of skyline objects is *N*
- For *independent* data, skyline cardinality is *medium*
  - Estimated skyline cardinality is

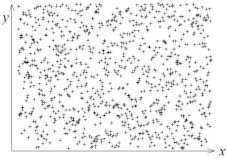
Number of Skyline Objects



Correlated



Anti-correlated



Independent

- Assume N objects having d skyline preference atributes
- For *correlated* data, skyline cardinality is *low*
  - For perfectly correlated data, number of skyline objects is 1
- For *anti-correlated* data, skyline cardinality is *high*
  - For perfectly anti-correlated data, number of skyline objects is N
- For *independent* data, skyline cardinality is *medium*
  - Estimated skyline cardinality is $\ln^{d-1} N/(d-1)!$ [God04]

Skylines in High Dimensions

- For large high-dimensional datasets, number of skyline objects quickly becomes unmanageable

N	d				
	1	2	3	5	10
10^3	1	7	24	95	99
10^4	1	9	42	300	1314
10^5	1	12	66	732	9793
10^6	1	14	95	1518	50529
10^7	1	16	130	2812	202330

- In high dimensions, it is rare that an object will dominate another in all skyline attributes

Skylines in High Dimensions

- For large high-dimensional datasets, number of skyline objects quickly becomes unmanageable

N	d				
	1	2	3	5	10
10^3	1	7	24	95	99
10^4	1	9	42	300	1314
10^5	1	12	66	732	9793
10^6	1	14	95	1518	50529
10^7	1	16	130	2812	202330

- In high dimensions, it is rare that an object will dominate another in all skyline attributes
- It does not make sense to even show 300 flights for a database of 10,000 flights and 5 preference functions
- How to handle it?

k-Dominant Skylines

- Relax dominance to k-dominance [CJT+06]
- An object O_p k-dominates another object O_q if it is preferred in k of the d skyline attributes, i.e.,

$$O_p \succ_k O_q \iff \exists I, |I| = k, \quad \forall i \in I, O_{pi} \succeq O_{qi}, \quad \exists j \in I, O_{pj} \succ O_{qj}$$

k-Dominant Skylines

- Relax dominance to k-dominance [CJT+06]
- An object O_p k-dominates another object O_q if it is preferred in k of the d skyline attributes, i.e.,

$$O_p \succ_k O_q \iff \exists I, |I| = k, \quad \forall i \in I, O_{pi} \succeq O_{qi}, \quad \exists j \in I, O_{pj} \succ O_{qj}$$

- A k-dominant skyline set is always a subset of the full skyline set

k-Dominant Skylines

- Relax dominance to k-dominance [CJT+06]
- An object O_p k-dominates another object O_q if it is preferred in k of the d skyline attributes, i.e.,

$$O_p \succ_k O_q \iff \exists I, |I| = k, \quad \forall i \in I, O_{pi} \succeq O_{qi}, \quad \exists j \in I, O_{pj} \succ O_{qj}$$

- A k-dominant skyline set is always a subset of the full skyline set
- If $k \leq d/2$, then it may be that $A \succ_k B$ and $B \succ_k A$

k-Dominant Skylines

- Relax dominance to k-dominance [CJT+06]
- An object O_p k-dominates another object O_q if it is preferred in k of the d skyline attributes, i.e.,

$$O_p \succ_k O_q \iff \exists I, |I| = k, \quad \forall i \in I, O_{pi} \succeq O_{qi}, \quad \exists j \in I, O_{pj} \succ O_{qj}$$

- A k-dominant skyline set is always a subset of the full skyline set
- If $k \leq d/2$, then it may be that $A \succ_k B$ and $B \succ_k A$
- Even otherwise, it may be that $A \succ_k B$, $B \succ_k C$ and $C \succ_k A$

k-Dominant Skylines

- Relax dominance to k-dominance [CJT+06]
- An object O_p k-dominates another object O_q if it is preferred in k of the d skyline attributes, i.e.,

$$O_p \succ_k O_q \iff \exists I, |I| = k, \quad \forall i \in I, O_{pi} \succeq O_{qi}, \quad \exists j \in I, O_{pj} \succ O_{qj}$$

- A k-dominant skyline set is always a subset of the full skyline set
- If $k \leq d/2$, then it may be that $A \succ_k B$ and $B \succ_k A$
- Even otherwise, it may be that $A \succ_k B$, $B \succ_k C$ and $C \succ_k A$
- A particular k-dominant skyline set may even be empty

A Two-Pass Algorithm for k-Dominant Skylines

- Maintain window W of k-dominant objects
- For every object $p \in D$, check against all objects $q \in W$
- If $p \succ_k q$, remove q from W
- If $q \succ_k p$, prune p
- Else, add p to W

A Two-Pass Algorithm for k-Dominant Skylines

- Maintain window W of k-dominant objects
- For every object $p \in D$, check against all objects $q \in W$
- If $p \succ_k q$, remove q from W
- If $q \succ_k p$, prune p
- Else, add p to W
- This first pass is not enough

A Two-Pass Algorithm for k-Dominant Skylines

- Maintain window W of k-dominant objects
- For every object $p \in D$, check against all objects $q \in W$
- If $p \succ_k q$, remove q from W
- If $q \succ_k p$, prune p
- Else, add p to W
- This first pass is not enough
- It may happen that $\exists r \in D \succ_k p$ but $\exists q \in W \succ_k r$ and, therefore, $r \notin W$ and r was not compared against p
- r must come earlier than p

A Two-Pass Algorithm for k-Dominant Skylines

- Maintain window W of k-dominant objects
- For every object $p \in D$, check against all objects $q \in W$
- If $p \succ_k q$, remove q from W
- If $q \succ_k p$, prune p
- Else, add p to W
- This first pass is not enough
- It may happen that $\exists r \in D \succ_k p$ but $\exists q \in W \succ_k r$ and, therefore, $r \notin W$ and r was not compared against p
- r must come earlier than p
- So, in second pass, check all $p \in W$ against all $r \notin W$ that came *earlier* in first pass

Skyline Joins

- More often, there are no suitable direct flights
- So, to go to Agartala from Kolkata, one may need to go via Gouhati or Delhi or Dhaka

Skyline Joins

- More often, there are no suitable direct flights
- So, to go to Agartala from Kolkata, one may need to go via Gouhati or Delhi or Dhaka
- Skyline join [SWLT08] to handle this
- Relation R_1 contains all flights departing from Kolkata
- Relation R_2 contains all flights arriving at Agartala
- Relation $R = R_1 \bowtie R_2$ on $R_1.dest = R_2.src$

Skyline Joins

- More often, there are no suitable direct flights
- So, to go to Agartala from Kolkata, one may need to go via Gouhati or Delhi or Dhaka
- Skyline join [SWLT08] to handle this
- Relation R_1 contains all flights departing from Kolkata
- Relation R_2 contains all flights arriving at Agartala
- Relation $R = R_1 \bowtie R_2$ on $R_1.dest = R_2.src$
- Naïve way is to join and then perform skyline query

Skyline Joins

- More often, there are no suitable direct flights
- So, to go to Agartala from Kolkata, one may need to go via Gouhati or Delhi or Dhaka
- Skyline join [SWLT08] to handle this
- Relation R_1 contains all flights departing from Kolkata
- Relation R_2 contains all flights arriving at Agartala
- Relation $R = R_1 \bowtie R_2$ on $R_1.dest = R_2.src$
- Naïve way is to join and then perform skyline query
- Skyline can be computed before join
  - If $u_1 \succ v_1 \in R_1$ and $u_2 \succ v_2 \in R_2$, then $u_1 \bowtie u_2 \succ v_1 \bowtie v_2 \in R$, and therefore, v_1 and v_2 need not be joined
  - Non-skyline sets need not be joined
  - Join of a skyline tuple is surely a skyline in R

Skylines over Aggregate Attributes

- Skyline preferences for joined relations are more often on attributes that have been aggregated from the base relations
- For example, a user would prefer less total cost of the two flights and not the individual costs
- However, total cost is only available *after* the join
- Aggregate skyline join queries [BT10]

Aggregate Skyline Join Queries (ASJQ)

- j join attributes; m_1, m_2 local attributes; n aggregate attributes
  - Join attributes: R_1.dest $= R_2$.src, R_1.time of arrival $< R_2$.time of departure
  - Local attributes: rating, amenities
  - Aggregate attributes: R.duration = R_1.duration + R_2.duration, R.cost = R_1.cost + R_2.cost

Aggregate Skyline Join Queries (ASJQ)

- j join attributes; m_1, m_2 local attributes; n aggregate attributes
  - Join attributes: R_1.dest $=$ R_2.src, R_1.time of arrival $<$ R_2.time of departure
  - Local attributes: rating, amenities
  - Aggregate attributes: R.duration = R_1.duration + R_2.duration, R.cost = R_1.cost + R_2.cost
- $A = \{h_{a_1}, \dots, h_{a_j}, \quad l_{a_1}, \dots, l_{a_{m_1}}, \quad g_{a_1}, \dots, g_{a_n}\}$
- $B = \{h_{b_1}, \dots, h_{b_j}, \quad l_{b_1}, \dots, l_{b_{m_2}}, \quad g_{b_1}, \dots, g_{b_n}\}$
- $J = \{h_{a_1} \bowtie h_{b_1}, \dots, h_{a_j} \bowtie h_{b_j}, \quad l_{a_1}, \dots, l_{a_{m_1}}, l_{b_1}, \dots, l_{b_{m_2}}, \quad g_{a_1} \oplus g_{b_1}, \dots, g_{a_n} \oplus g_{b_n}\}$
- Skyline is over preferences for $m_1 + m_2 + n$ attributes

Aggregate Skyline Join Queries (ASJQ)

- j join attributes; m_1, m_2 local attributes; n aggregate attributes
  - Join attributes: R_1.dest $=$ R_2.src, R_1.time of arrival $<$ R_2.time of departure
  - Local attributes: rating, amenities
  - Aggregate attributes: R.duration = R_1.duration + R_2.duration, R.cost = R_1.cost + R_2.cost
- $A = \{h_{a_1}, \dots, h_{a_j}, \quad l_{a_1}, \dots, l_{a_{m_1}}, \quad g_{a_1}, \dots, g_{a_n}\}$
- $B = \{h_{b_1}, \dots, h_{b_j}, \quad l_{b_1}, \dots, l_{b_{m_2}}, \quad g_{b_1}, \dots, g_{b_n}\}$
- $J = \{h_{a_1} \bowtie h_{b_1}, \dots, h_{a_j} \bowtie h_{b_j}, \quad l_{a_1}, \dots, l_{a_{m_1}}, l_{b_1}, \dots, l_{b_{m_2}}, \quad g_{a_1} \oplus g_{b_1}, \dots, g_{a_n} \oplus g_{b_n}\}$
- Skyline is over preferences for $m_1 + m_2 + n$ attributes
- Naïve is too costly

Example

fno	dep	Join (H)		Aggregate (G)		Local (L)	
		arr	dst	duration	cost	amn	rtg
11	06:30	08:40	C	2h 10m	162	5	4
12	07:00	09:00	E	2h 00m	166	4	5
14	08:05	10:00	E	1h 55m	140	3	4
15	09:50	10:40	C	1h 40m	270	3	2
13	12:00	13:50	C	1h 50m	173	4	3
16	16:00	17:30	D	1h 30m	230	3	3
17	17:00	20:20	C	3h 20m	183	4	3

(a) Flights from city A (FlightsA)

fno	Join (H)		arr	Aggregate (G)		Local (L)	
	src	dep		duration	cost	amn	rtg
21	C	09:50	12:00	2h 10m	162	5	4
26	C	16:00	18:49	2h 49m	160	2	3
23	C	16:00	18:45	2h 45m	160	4	4
25	D	16:00	17:49	1h 49m	220	3	4
22	D	17:00	19:00	2h 00m	166	4	5
27	E	20:00	21:46	1h 46m	200	3	3
24	E	20:00	21:30	1h 30m	160	4	3

(b) Flights to city B (FlightsB)

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

(c) Part of the joined relation (FlightsA $\bowtie$ FlightsB)

SQL Syntax for ASJQ

- It is easy to incorporate ASJQ into existing SQL syntax

```
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,
  dur as f1.dur + f2.dur
FROM FlightsA as f1, flightsB as f2
WHERE f1.dst = f2.src and f1.arr < f2.dep and
SKYLINE of cost min, dur min,
  f1.rtg max, f2.rtg max, f1.amn max, f2.amn max
```

Full and Local Skylines

- u fully dominates v, i.e., $u \succ_f v$ iff u dominates v in *both* local and aggregate attributes
- u locally dominates v, i.e., $u \succ_l v$ iff u dominates v in *only* local attributes

Full and Local Skylines

- u fully dominates v, i.e., $u \succ_f v$ iff u dominates v in *both* local and aggregate attributes
- u locally dominates v, i.e., $u \succ_l v$ iff u dominates v in *only* local attributes
- Full dominance implies local dominance but not vice versa

Full and Local Skylines

- u fully dominates v, i.e., $u \succ_f v$ iff u dominates v in *both* local and aggregate attributes
- u locally dominates v, i.e., $u \succ_l v$ iff u dominates v in *only* local attributes
- Full dominance implies local dominance but not vice versa
- An object that is in the local skyline set is also in the full skyline set but not vice versa

Break-up using Full and Local Skylines

- Each relation R is broken up in the following way
- First, it is broken based on full skylines, i.e.,

$$R = R_f \cup R_{f'}$$

where R_f is the full skyline set of R and $R_{f'}$ is the rest

Break-up using Full and Local Skylines

- Each relation R is broken up in the following way
- First, it is broken based on full skylines, i.e.,

$$R = R_f \cup R_{f'}$$

where R_f is the full skyline set of R and $R_{f'}$ is the rest

- Then, R_f is further broken based on local skylines, i.e.,

$$R_f = R_{fl} \cup R_{fl'}$$

where R_{fl} is the local skyline set of R_f and $R_{fl'}$ is the rest of R_f

Break-up using Full and Local Skylines

- Each relation R is broken up in the following way
- First, it is broken based on full skylines, i.e.,

$$R = R_f \cup R_{f'}$$

  where R_f is the full skyline set of R and $R_{f'}$ is the rest
- Then, R_f is further broken based on local skylines, i.e.,

$$R_f = R_{fl} \cup R_{fl'}$$

  where R_{fl} is the local skyline set of R_f and $R_{fl'}$ is the rest of R_f
- Therefore,

$$R = R_f \cup R_{f'} = (R_{fl} \cup R_{fl'}) \cup R_{f'}$$

Two Pruning Theorems

- Any object from $A_{f'} \bowtie B_{f'}$, $A_f \bowtie B_{f'}$ and $A_{f'} \bowtie B_f$ cannot exist in the final skyline answer set

Two Pruning Theorems

- Any object from $A_{f'} \bowtie B_{f'}$, $A_f \bowtie B_{f'}$ and $A_{f'} \bowtie B_f$ cannot exist in the final skyline answer set
  - Consider $t' = u \bowtie v'$ where $u \in A_f$ and $v' \in B_{f'}$
  - $\exists v \in B_f \succ v'$
  - Surely, $t = u \bowtie v \succ t' = u \bowtie v'$

Two Pruning Theorems

- Any object from $A_{f'} \bowtie B_{f'}$, $A_f \bowtie B_{f'}$ and $A_{f'} \bowtie B_f$ cannot exist in the final skyline answer set
  - Consider $t' = u \bowtie v'$ where $u \in A_f$ and $v' \in B_{f'}$
  - $\exists v \in B_f \succ v'$
  - Surely, $t = u \bowtie v \succ t' = u \bowtie v'$
- Any object from $A_{fl} \bowtie B_{fl}$, $A_{fl} \bowtie B_{fl'}$ and $A_{fl'} \bowtie B_{fl}$ must exist in the final skyline answer set

Two Pruning Theorems

- Any object from $A_{f'} \bowtie B_{f'}$, $A_f \bowtie B_{f'}$ and $A_{f'} \bowtie B_f$ cannot exist in the final skyline answer set
  - Consider $t' = u \bowtie v'$ where $u \in A_f$ and $v' \in B_{f'}$
  - $\exists v \in B_f \succ v'$
  - Surely, $t = u \bowtie v \succ t' = u \bowtie v'$
- Any object from $A_{fl} \bowtie B_{fl}$, $A_{fl} \bowtie B_{fl'}$ and $A_{fl'} \bowtie B_{fl}$ must exist in the final skyline answer set
  - Consider $t' = u \bowtie v'$ where $u \in A_{fl}$ and $v' \in B_{fl'}$
  - $\nexists u' \in A \succ_l u \implies \nexists u' \in A \succ_f u$
  - $\exists v \in B_{fl} \succ_l v'$ *only* in local as $v \in B_f$
  - So, $\nexists v \in B \succ_f v'$
  - Hence, the aggregate attributes of t' can never be dominated (monotone property)
  - Thus, t' is a skyline

State After using the Two Theorems

Set	Skyline	Target
$A_{fl} \bowtie B_{fl}$	yes	-
$A_{fl} \bowtie B_{fl'}$	yes	-
$A_{fl} \bowtie B_{f'}$	no	-
$A_{fl'} \bowtie B_{fl}$	yes	-
$A_{fl'} \bowtie B_{fl'}$	-	$A_f \bowtie B_f$
$A_{fl'} \bowtie B_{f'}$	no	-
$A_{f'} \bowtie B_{fl}$	no	-
$A_{f'} \bowtie B_{fl'}$	no	-
$A_{f'} \bowtie B_{f'}$	no	-

- Only the set $A_{fl'} \bowtie B_{fl'}$ needs to be checked
- It need to be checked only against $A_f \bowtie B_f$ and not the entire $A \bowtie B$

Some Interesting Skyline Issues

- SkyCube
  - Skylines over all possible subsets of attributes

Some Interesting Skyline Issues

- SkyCube
  - Skylines over all possible subsets of attributes
- Update-intensive databases
  - How to update skylines and skycube

Some Interesting Skyline Issues

- SkyCube
  - Skylines over all possible subsets of attributes
- Update-intensive databases
  - How to update skylines and skycube
- Query-aware skylines
  - Origin changes

Some Interesting Skyline Issues

- SkyCube
  - Skylines over all possible subsets of attributes
- Update-intensive databases
  - How to update skylines and skycube
- Query-aware skylines
  - Origin changes
- Caching
  - More in context of skycube

Some Interesting Skyline Issues

- SkyCube
  - Skylines over all possible subsets of attributes
- Update-intensive databases
  - How to update skylines and skycube
- Query-aware skylines
  - Origin changes
- Caching
  - More in context of skycube
- Uncertain databases
  - An object is a skyline with only some non-zero probability
  - This is because the dominator may not exist

Conclusions

- Skylines provide a nice way of handling multiple preferences
- Algorithms need to be practical
- Some nice theoretical issues as well
- Still many open questions

Conclusions

- Skylines provide a nice way of handling multiple preferences
- Algorithms need to be practical
- Some nice theoretical issues as well
- Still many open questions

THANK YOU

Conclusions

- Skylines provide a nice way of handling multiple preferences
- Algorithms need to be practical
- Some nice theoretical issues as well
- Still many open questions

THANK YOU

QUESTIONS?
ANSWERS!

References I

S. Börzsönyi, D. Kossmann, and K. Stocker.
The skyline operator.
In *ICDE*, pages 421–430, 2001.

A. Bhattacharya and B. P. Teja.
Aggregate skyline join queries: Skylines with aggregate operations over multiple relations.
In *Int. Conf. Management of Data (COMAD)*, pages 15–26, 2010.

J. Chomicki, P. Godfrey, J. Gryz, and D. Liang.
Skyline with presorting.
In *ICDE*, pages 717–719, 2003.

References II

C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang.
Finding k-dominant skylines in high dimensional space.
In *SIGMOD*, pages 503–514, 2006.

H. Eder.
On extending PostgreSQL with the skyline operator.
Master's thesis, Vienna University of Technology, 2009.

P. Godfrey.
Skyline cardinality for relational processing.
In *FoIKS*, pages 78–97, 2004.

D. Sun, S. Wu, J. Li, and A. K. H. Tung.
Skyline-join in distributed databases.
In *ICDE Workshop*, pages 176–181, 2008.