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

- 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

- 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

```
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
```

Example

		Join (H)		Aggregate (G)		Local (L)	
fno	dep	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

Flights from city A (FlightsA)

	Join (H)			Aggregate (G)		Local (L)	
fno	src	dep	arr	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

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

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

Aggregate skyline join queries

- 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

Aggregate skyline join queries

- 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

Aggregate skyline join queries

- 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

Aggregate attributes

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

Join attributes

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 attrbutes

- Dominance relationships help infer certain properties in 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 $=, <, \leq, >, \geq$
- 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 $=$, 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

Join condition	$u \in A \succ u' \in A$ if	$v \in B \succ v' \in B$ if
$A.a = B.b$	$u.a = u'.a$	$v.b = v'.b$
$A.a < B.b$, $A.a \leq B.b$	$u.a \leq u'.a$	$v.b \geq v'.b$
$A.a < B.b$, $A.a \geq B.b$	$u.a \geq u'.a$	$v.b \leq v'.b$

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 $v \succ v'$, $t \succ t'$

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 $t = u \in A_1 \bowtie v' \in B'_1$
  - Consider any other tuple $t' = u' \in A_0 \bowtie v' \in B'_1$.
  - Since u is a local skyline, $\nexists u', u' \succ u$
  - Therefore, $\nexists t', t' \succ t$

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, $\nexists u', u' \not\succ u$
  - Therefore, $\nexists t', t' \succ t$
- 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

Example

		Join (H)		Aggregate (G)		Local (L)	
fno	dep	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

Flights from city A (FlightsA)

	Join (H)			Aggregate (G)		Local (L)	
fno	src	dep	arr	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

Flights to city B (FlightsB)

Set			Flight numbers
A_0	A_1		11, 12
	A'_1	A_2	13, 14
		A'_2	15, 16
A'_0			17

Set			Flight numbers
B_0	B_1		21, 22
	B'_1	B_2	23
		B'_2	24, 25
B'_0			26, 27

Multiple skyline computations (MSC) algorithm

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

Dominator-based approach

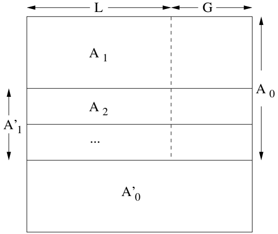
- A tuple $t' = u' \in A'_1 \bowtie v' \in B'_1$ can be dominated only by certain tuples in $A_0 \bowtie B_0$
- Suppose the *local* dominators of u' and v' are denoted by $ld(u')$ and $ld(v')$ respectively
- Lemma: t' can be dominated only by t of the form $t = u \in ld(u') \bowtie v \in ld(v')$
- Proof
  - 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 $A_2 \bowtie B_2$ needs to be compared with tuples in $A_1 \bowtie B_1$ only
- No unnecessary comparison with $(A_1 \bowtie B_1') \cup (A_1' \bowtie B_1) \cup (A_1' \bowtie B_1')$

Set	Target Sets
$A_2 \bowtie B_2$	$A_1 \bowtie B_1$
$A_2 \bowtie B_2'$	$A_1 \bowtie B_1$, $A_1 \bowtie B_1'$
$A_2' \bowtie B_2$	$A_1 \bowtie B_1$, $A_1' \bowtie B_1$
$A_2' \bowtie B_2'$	$A_1 \bowtie B_1$, $A_1 \bowtie B_1'$, $A_1' \bowtie B_1$

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

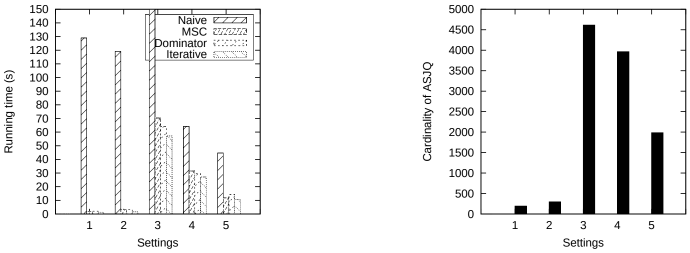
Single aggregate attribute

- 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: $\nexists t, t \succ t'$
  - Suppose such a $t = u \bowtie v$ exists
  - Therefore, $u \succ_{ld} u'$ and $v \succ_{ld} v'$
  - However, since $u' \in A_0$ and $v' \in B_0$, $u \nsucc_{fd} u'$ and $v \nsucc_{fd} v'$
  - Therefore, it must be that $u' \succ_g u$ and $v' \succ_g v$
  - This implies that $t \nsucc t'$

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: $\nexists t, t \succ t'$
  - Suppose such a $t = u \bowtie v$ 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 $t \not\succ t'$
- Effect: Finding local skylines is enough

Performance of naïve algorithm

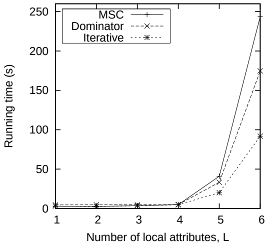


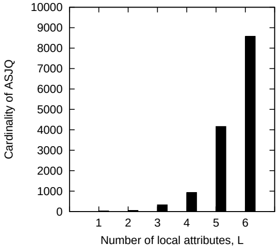
- Naïve algorithm takes much more time
- Performance is independent of cardinality of final answer set
- Overall, iterative algorithm is the best

Default experimental parameters

Parameter	Symbol	Default value
Number of local attributes	L	2
Number of aggregate attributes	G	2
Cardinality of datasets	N	40000
Number of categories	C	10
Distribution of datasets	D	Correlated

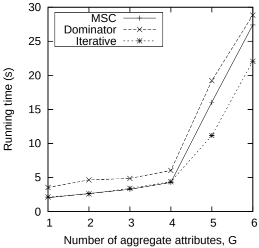
Effect of number of local attributes

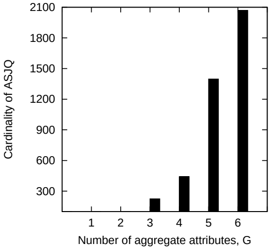




- Running time increases almost exponentially with number of local attributes
- Iterative shows best scalability

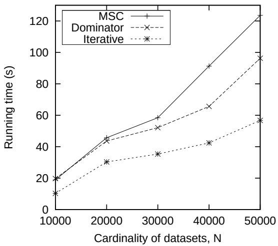
Effect of number of aggregate attributes

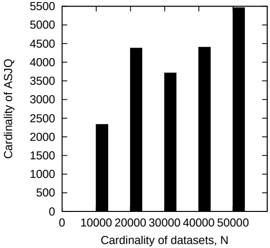




- Running time increases almost exponentially with number of aggregate attributes
- Absolute times are lower

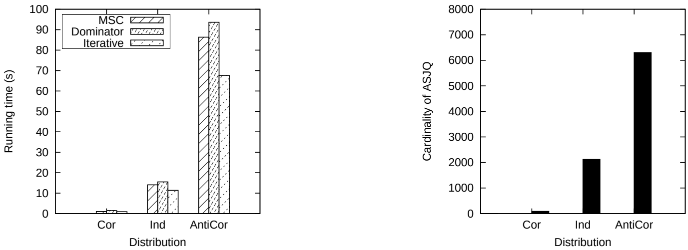
Effect of dataset cardinality





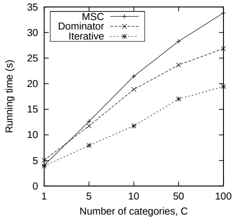
- Scalability is better than quadratic

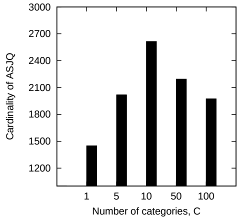
Effect of dataset distribution



- Cardinality of final answer set is much higher in anti-correlated datasets
- Iterative shows the best comparative advantage in this case

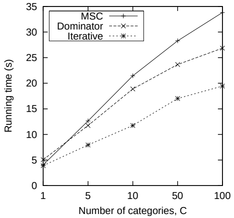
Effect of categories of join attribute

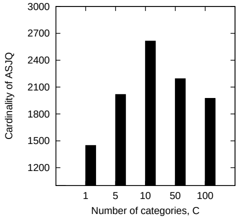




- Number of categories of join attribute measures the possible values of the join attribute (equi-join)
- When number of join categories increases
  - Full skyline sets A_0 and B_0 become larger as there is less probability of a tuple matching another tuple in the join attribute, and therefore, dominating 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?