CompSci 590.6

Understanding Data:

Theory and Applications

Lecture 2

Data Cube Basics

Instructor: Sudeepa Roy
Email: *sudeepa@cs.duke.edu*

Today's Papers

1.

Gray-Chaudhuri-Bosworth-Layman-Reichart-Venkatrao-Pellow-Pirahesh

Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals

ICDE 1996/Data Mining and Knowledge Discovery 1997

- Thinking process at that time

2.

Agarwal-Agrawal-Deshpande-Gupta-Naughton-Ramakrishnan-Sarawagi

On the Computation of Multidimensional Aggregates

VLDB 1996

- Technical

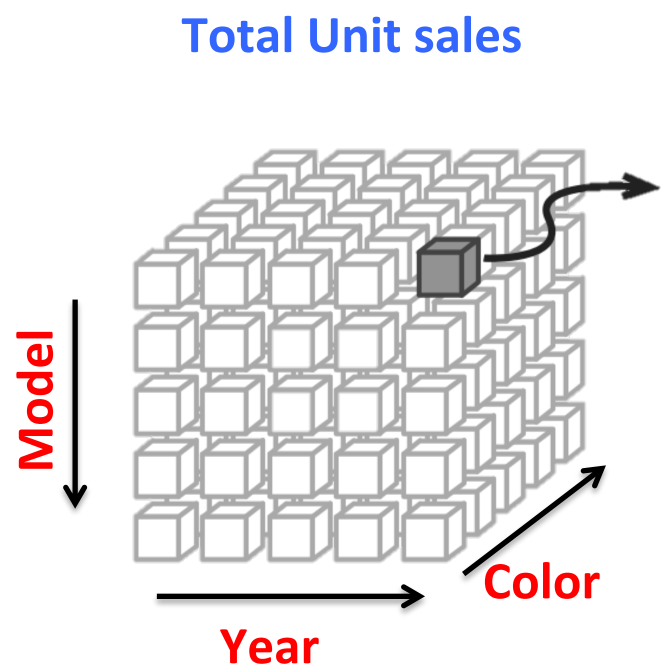
(more than 2630 and 750 citations resp. on Google Scholar)

Naïve Approach

- Data analysts are interested in exploring trends and anomalies
- Possibly by visualization (Excel) - 2D or 3D plots
- "Dimensionality Reduction" by summarizing data and computing aggregates

- Find total unit sales for each

1. Model
2. Model, broken into years
3. Year, broken into colors
4. Year
5. Model, broken into colors
6. ….



Naïve Approach

Run a number of queries

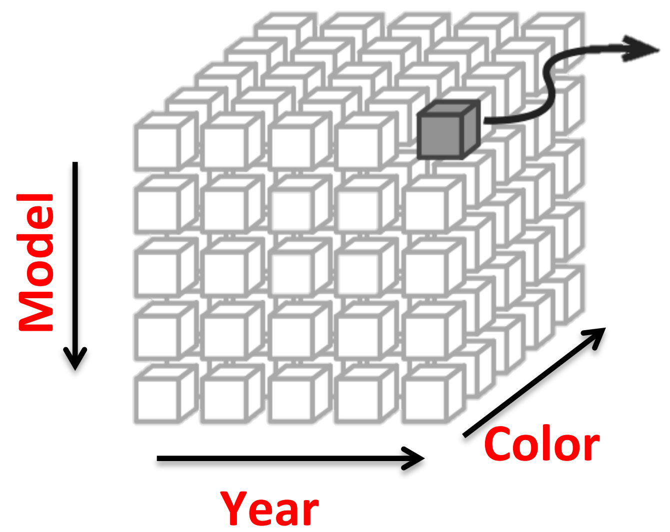
```
SELECT sum(units)
FROM Sales

SELECT Color, sum(units)
FROM Sales
GROUP BY Color

SELECT Year, sum(units)
FROM Sales
GROUP BY Year

SELECT Model, Year, sum(units)
FROM Sales
GROUP BY Model, Year
….
```

Total Unit sales



- Data cube generalizes Histogram, Roll-Ups, Cross-Tabs
- More complex to do these with GROUP-BY

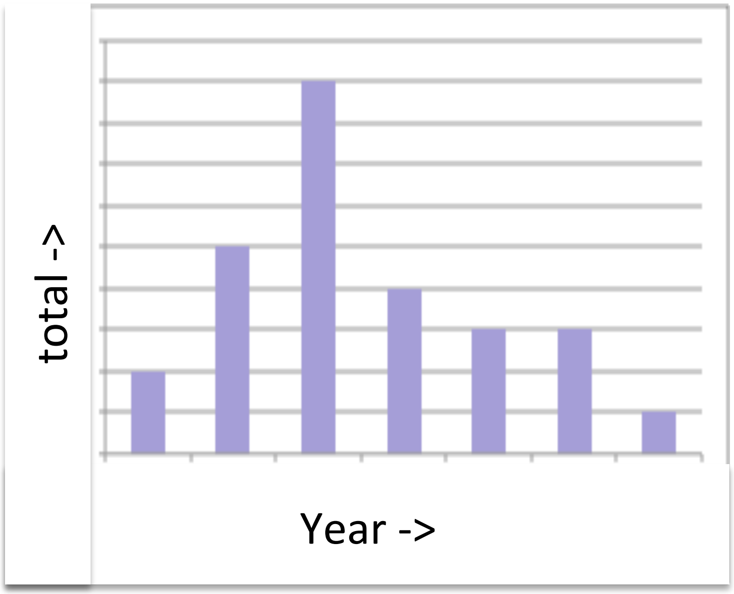
- How many sub-queries?
- How many sub-queries for 8 attributes?

Sales (Model, Year, Color, Units)

Histograms

A tabulated frequency of computed values

```
SELECT Year, COUNT(Units) as total
FROM Sales
GROUP BY Year
ORDER BY Year
```



May require a nested SELECT to compute

Sales (Model, Year, Color, Units)

Roll-Ups

- Analysis reports start at a coarse level, go to finer levels
- Order of attribute matters
- Not relational data (empty cells no keys)

Roll-ups

Drill-downs

GROUP BY

Model	Year	Color	Model, Year, Color	Model, Year	Model
Chevy	1994	Black	50		
Chevy	1994	White	40		
				90	
Chevy	1995	Black	115		
Chevy	1995	White	85		
				200	
					290

Sales (Model, Year, Color, Units)

Roll-Ups

- Another representation (Chris Date'96)
- Relational, but
  - long attribute names
  - hard to express in SQL and repetition

GROUP BY

Model	Year	Color	Model, Year, Color	Model, Year	Model
Chevy	1994	Black	50	90	290
Chevy	1994	White	40	90	290
Chevy	1995	Black	85	200	290
Chevy	1995	Black	115	200	290

Sales (Model, Year, Color, Units)

'ALL' Construct

Easier to visualize roll-up if allow ALL to fill in the super-aggregates

```
SELECT Model, Year, Color, SUM(Units)
     FROM Sales
     WHERE Model = 'Chevy'
         GROUP BY Model, Year, Color
UNION
SELECT Model, Year, 'ALL', SUM(Units)
     FROM Sales
     WHERE Model = 'Chevy'
     GROUP BY Model, Year
UNION
…
UNION
SELECT 'ALL', 'ALL', 'ALL', SUM(Units)
     FROM Sales
     WHERE Model = 'Chevy';
```

Model	**Year**	**Color**	**Units**
Chevy	1994	Black	50
Chevy	1994	White	40
Chevy	1994	'ALL'	90
Chevy	1995	Black	85
Chevy	1995	White	115
Chevy	1995	'ALL'	200
Chevy	'ALL'	'ALL'	290

Sales (Model, Year, Color, Units)

Traditional Roll-Up

Model	Year	Color	Model, Year, Color	Model, Year	Model
Chevy	1994	Black	50		
Chevy	1994	White	40		
				90	
Chevy	1995	Black	115		
Chevy	1995	White	85		
				200	
					290

‘ALL’ Roll-Up

Model	Year	Color	Units
Chevy	1994	Black	50
Chevy	1994	White	40
Chevy	1994	‘ALL’	90
Chevy	1995	Black	85
Chevy	1995	White	115
Chevy	1995	‘ALL’	200
Chevy	‘ALL’	‘ALL’	290

- Roll-ups are asymmetric

Cross Tabulation

If we made the roll-up symmetric, we would get a cross-tabulation

Generalizes to higher dimensions

```
SELECT Model, 'ALL', Color, SUM(Units)
     FROM Sales
     WHERE Model = 'Chevy'
     GROUP BY Model, Color
```

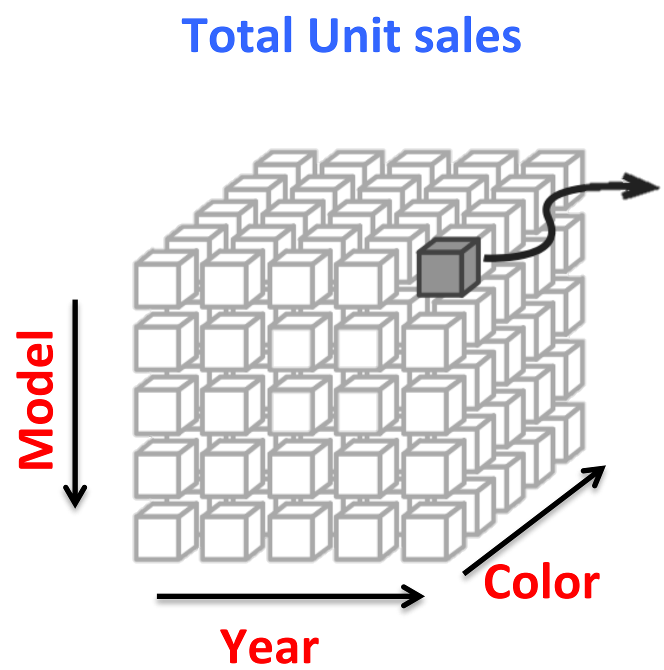
Chevy	1994	1995	Total (ALL)
Black	50	85	135
White	40	115	155
Total (ALL)	90	200	290

Is the problem solved with Cross-Tab and GROUP-BYs with 'ALL'?

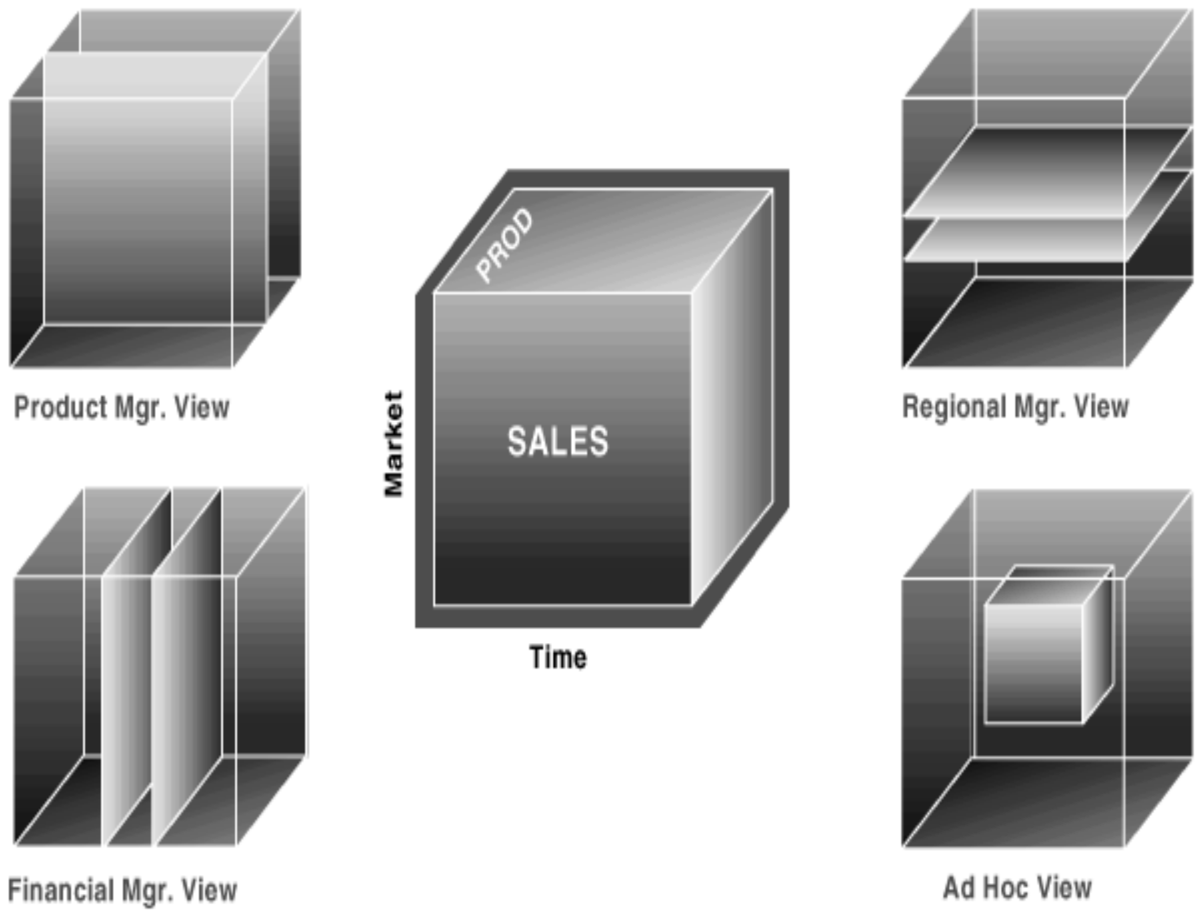
- Requires a lot of GROUP BYs (64 for 6-dimension)
- Too complex to optimize (64 scans, 64 sort/hash, slow)

Data Cube: Intuition

```
SELECT 'ALL', 'ALL', 'ALL', sum(units)
FROM Sales
UNION
SELECT 'ALL', 'ALL', Color, sum(units)
FROM Sales
GROUP BY Color
UNION
SELECT 'ALL', Year, 'ALL', sum(units)
FROM Sales
GROUP BY Year
UNION
SELECT Model, Year, 'ALL', sum(units)
FROM Sales
GROUP BY Model, Year
UNION
….
```



Data Cube



Ack: from slides by Laurel Orr and Jeremy Hyrkas, UW

Data Cube

- Computes the aggregate on all possible combinations of group by columns.
- If there are N attributes, there are 2^N-1 super-aggregates.
- If the cardinality of the N attributes are C_1,..., C_N, then there are a total of $(C_1+1)...(C_N+1)$ values in the cube.
- ROLL-UP is similar but just looks at N aggregates

Sales (Model, Year, Color, Units)

Data Cube Syntax

- SQL Server

```
SELECT Model, Year, Color, sum(units)
FROM Sales
GROUP BY Model, Year, Color
WITH CUBE
```

Types of Aggregates

- Distributive: input can be partitioned into disjoint sets and aggregated separately
  - COUNT, SUM, MIN
- Algebraic: can be composed of distributive aggregates
  - AVG
- Holistic: aggregate must be computed over the entire input set
  - MEDIAN

Types of Aggregates

Efficient computation of the CUBE operator depends on the type of aggregate

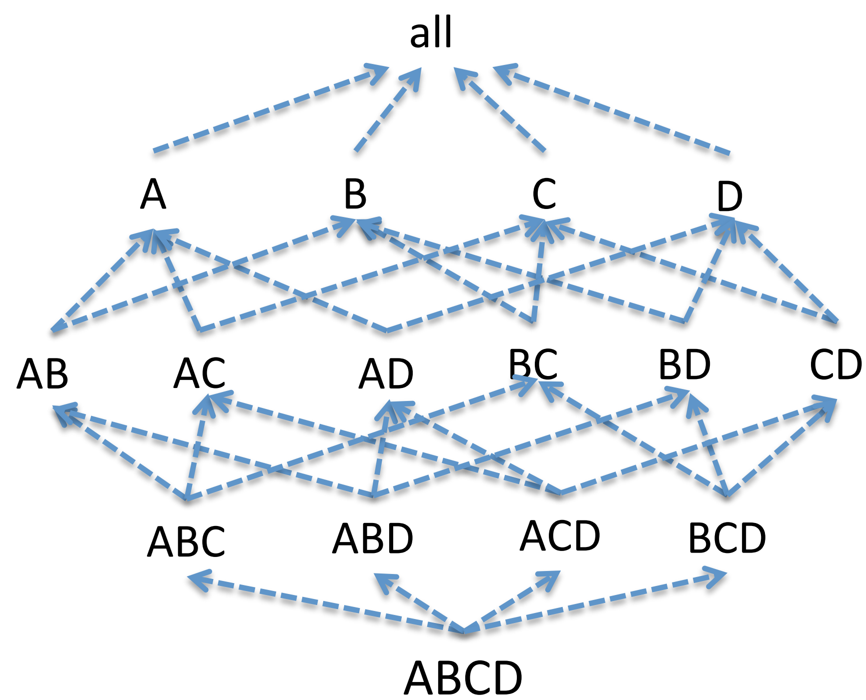
Distributive and Algebraic aggregates motivate optimizations

Agarwal et al paper

- Compute GROUP-BYs from previously computed GROUP-BYs
- Which direction?
- Next, some generic optimizations

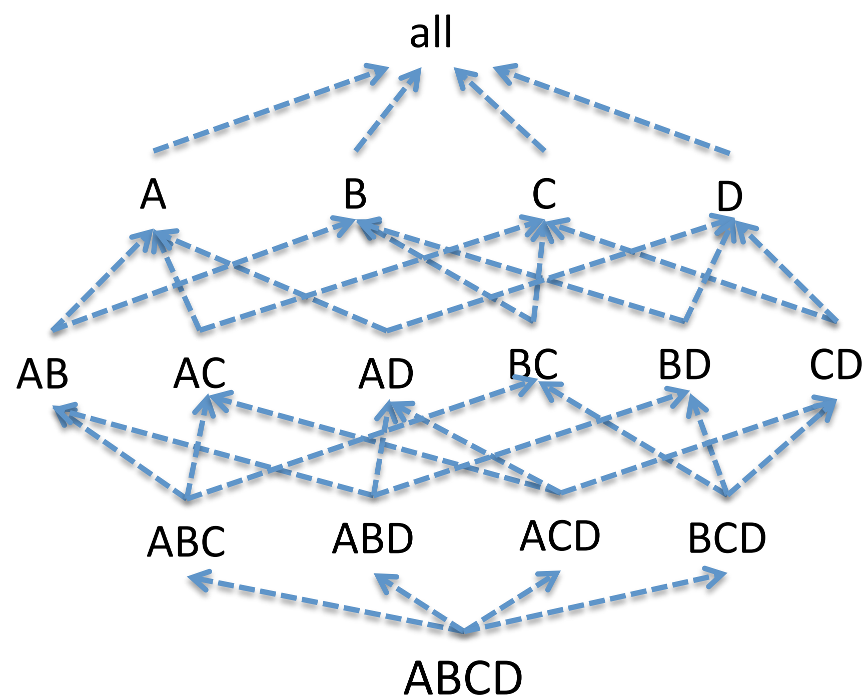
Optimization 1: Smallest Parent

- Compute GROUP-BY from the smallest (size) previously computed GROUP-BY as a parent
  - AB can be computed from ABC, ABD, or ABCD
  - ABC or ABD better than ABCD
  - Even ABC or ABD may have different sizes



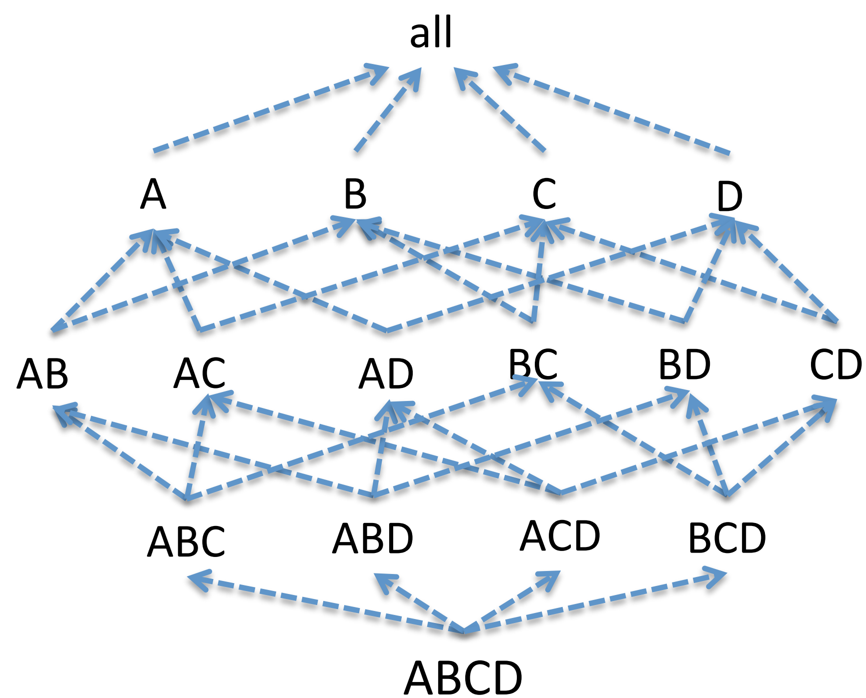
Optimization 2: Cache Results

- Cache result of one GROUP-BY in memory to reduce disk I/O
  - Compute AB from ABC while ABC is still in memory



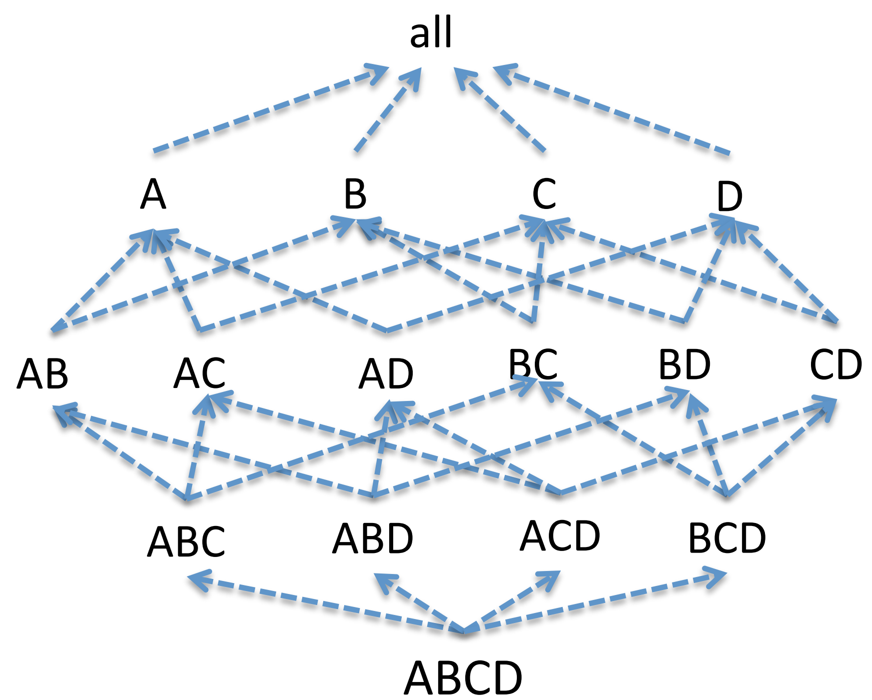
Optimization 3: Amortize Disk Scans

- Amortize disk reads for multiple GROUP-Bys
  - Suppose the result for ABCD is stored on disk
  - Compute all of ABC, ABD, ACD, BCD simultaneously in one scan of ABCD



Optimization 4, 5 (later)

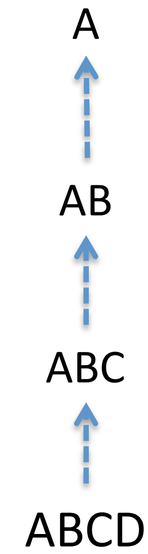
- 4. Share-sort
  - for sort-based algorithms
- 5. Shared-partition
  - for hash-based algorithms



PipeSort Algorithm

PipeSort: Basic Idea

- **Share-sort optimization:**
  - Data sorted in one order
  - Compute all GROUP-BYs prefixed in that order
  - Example:
    - GROUP-BY over attributes ABCD
    - Sort raw data by ABCD
    - Compute ABCD -> ABC -> AB -> A in pipelined fashion
  - No additional sort needed
  - BUT, may have a conflict with "smallest-parent" optimization
    - ABD -> AB could be a better choice

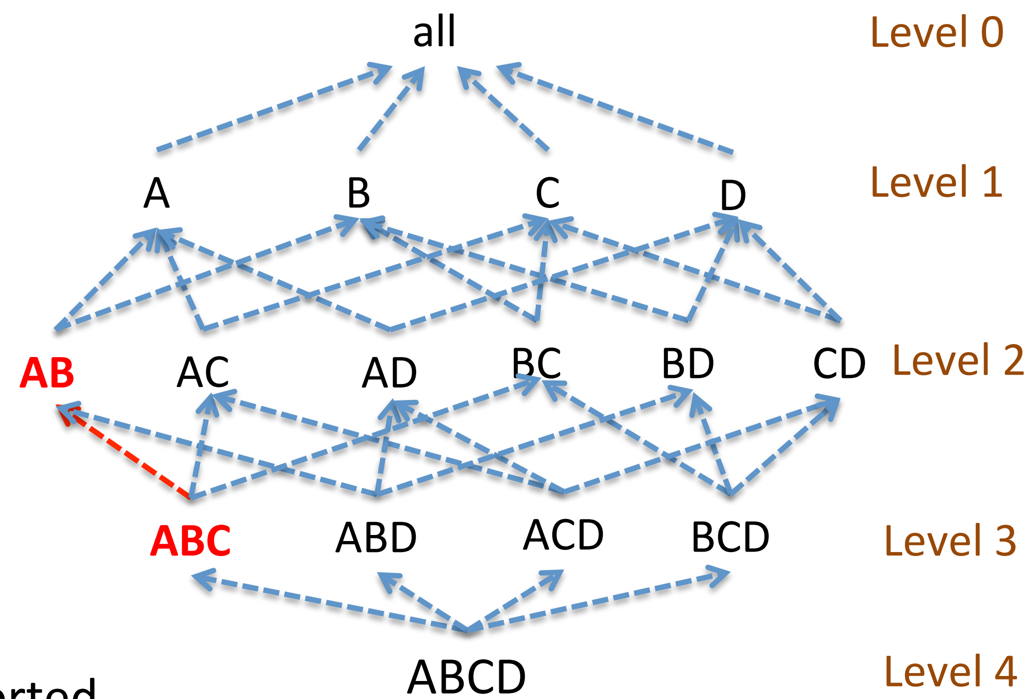


- **Pipe-sort algorithm:**
  - Combines two optimizations: "shared-sorts" and "smallest-parent"
  - Also includes "cache-results" and "amortized-scans"
    - Compute one tuple of ABCD, propagate upward in the pipeline by a single scan

Harinarayan et al. ’96
Lecture 3

Search Lattice

- Directed edge => one attribute less and possible computation
- Level k contains k attributes
  - all = 0 attribute
- Two possible costs for each edge e_{ij} = i ---> j
- $A(e_{ij})$: i is sorted for j
- $S(e_{ij})$: i is NOT sorted for j



Sorted

A	B	C	sum
a1	b1	c1	5
a1	b1	c2	10
a1	b2	c3	8
a2	b2	c1	2
a2	b2	c3	11

Not Sorted

A	B	C	sum
a2	b2	c3	11
a1	b1	c2	10
a2	b2	c1	2
a1	b1	c1	5
a1	b2	c3	8

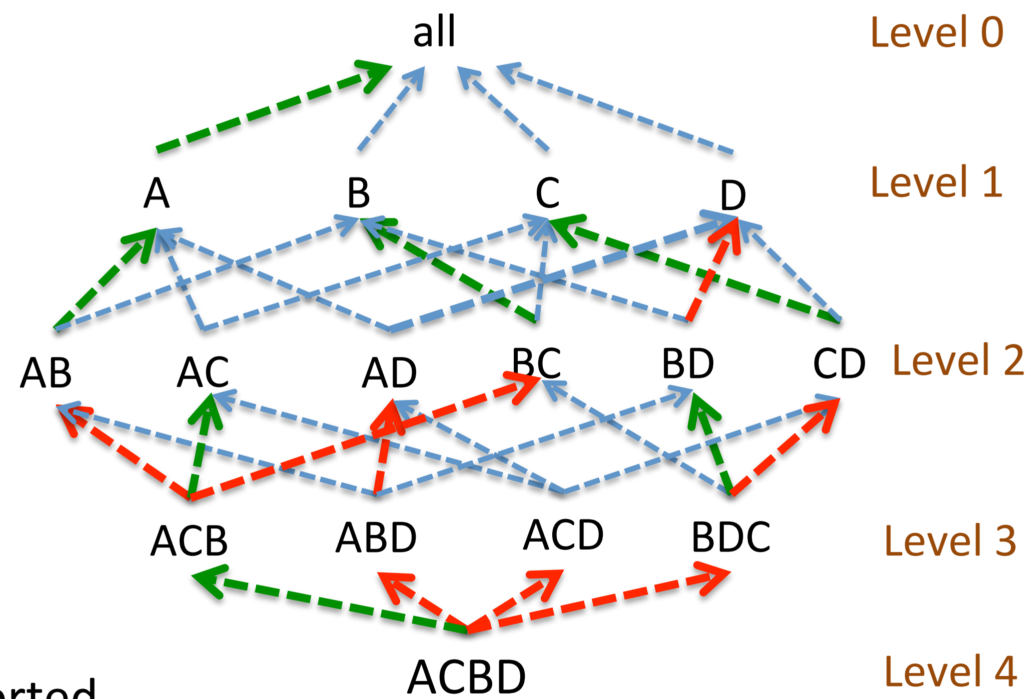
A	B	sum
a1	b1	15
a1	b2	8
a2	b2	13

PipeSort Output

Sorted (A)

Not-Sorted (S)

- A subgraph O
- each node has a single parent
- each node has a sorted order of attributes
- if parent's sorted order is a prefix, cost = $A(e_{ij})$, else $S(e_{ij})$
- Mark by A or S
- At most one A-marked out-edge
- Goal: Find O with min total cost
- Q. Should we always have a green out-edge?



Sorted

A	B	C	sum
a1	b1	c1	5
a1	b1	c2	10
a1	b2	c3	8
a2	b2	c1	2
a2	b2	c3	11

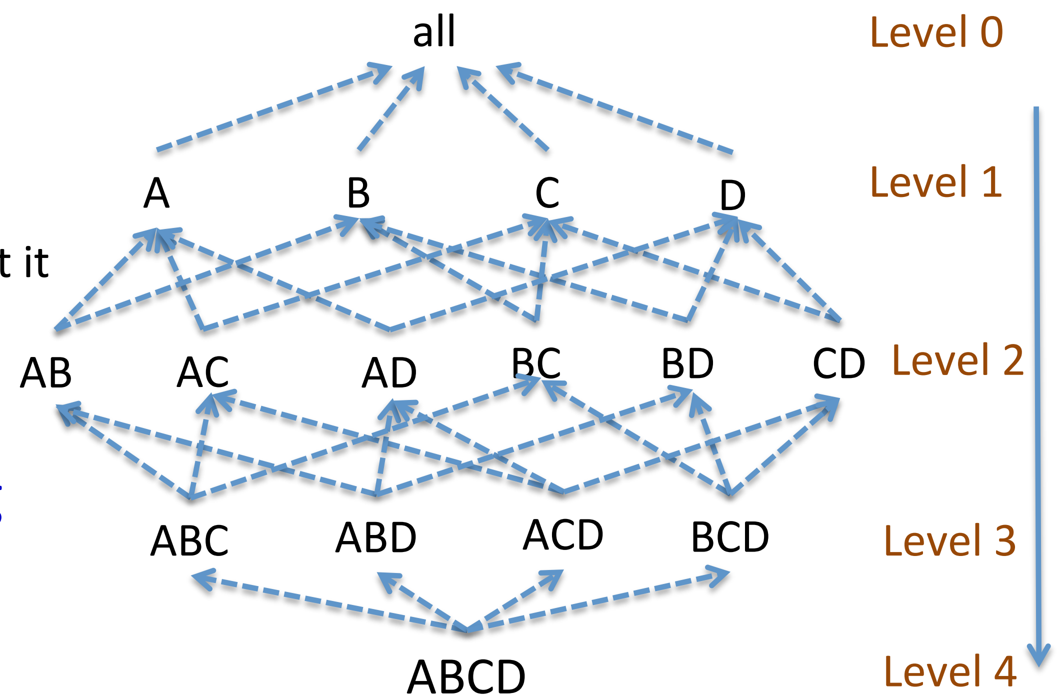
Not Sorted

A	B	C	sum
a2	b2	c3	11
a1	b1	c2	10
a2	b2	c1	2
a1	b1	c1	5
a1	b2	c3	8

A	B	sum
a1	b1	15
a1	b2	8
a2	b2	13

Outline: PipeSort Algorithm (1)

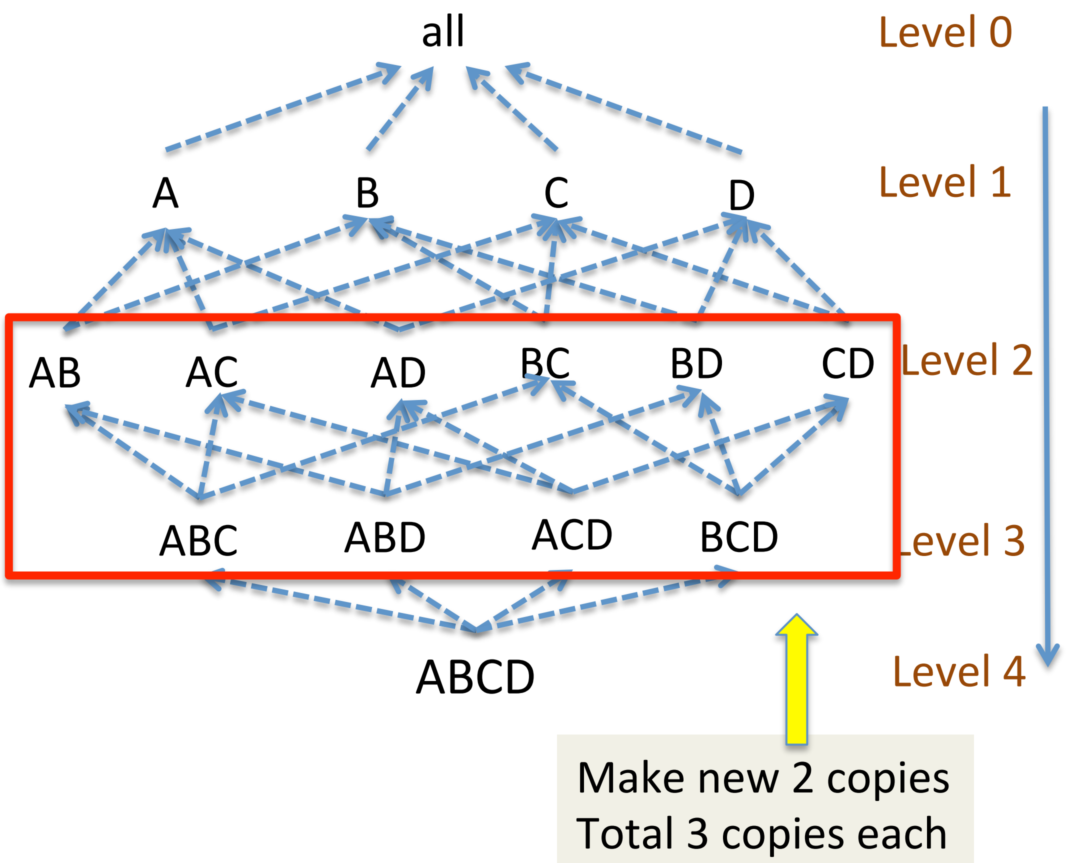
- Go from level 0 to N-1
  - here N = 4
- For each level k
  - find the best way to construct it from level k+1



- Weighted Bipartite Matching
  - G(V1, V2, E)
  - Weight on edges
  - each vertex in V1 should be connected to at most one vertex in V2
  - Find a matching of max total weight
  - Here min total weight
  - w -> max_weight – w
  - Requires |V2| >= |V1|

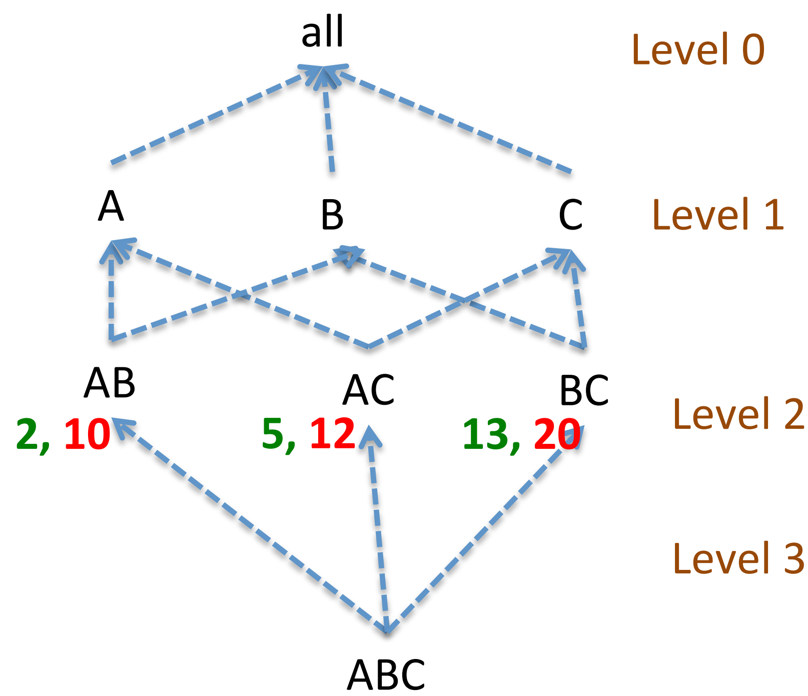
Outline: PipeSort Algorithm (2)

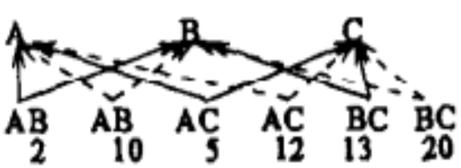
- Reduction to a weighted bipartite matching between level k and k+1
- Make k new copies of each node in level k+1
  - k+1 copies for each in total
  - replicate edges
- Original copy = cost $A(e_{ij})$ = sorted
  - sorted order of i fixed
- New copies = cost $S(e_{ij})$ = not sorted
  - need to sort i



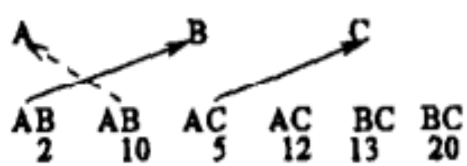
Outline: PipeSort Algorithm (3)

- Illustration with a smaller example
- Level k = 1 from level k+1 = 2
  - one new copy (dotted edges)
  - one existing copy (solid edge)
- Assumption for simplicity
  - same cost for all outgoing edges
  - $A(e_{ij}) = A(e_{ij'})$
  - $S(e_{ij}) = S(e_{ij'})$





(a) Transformed search lattice



(b) Minimum cost matching

- Optimal on total cost
- Not on #sorts
  - can be suboptimal (size)

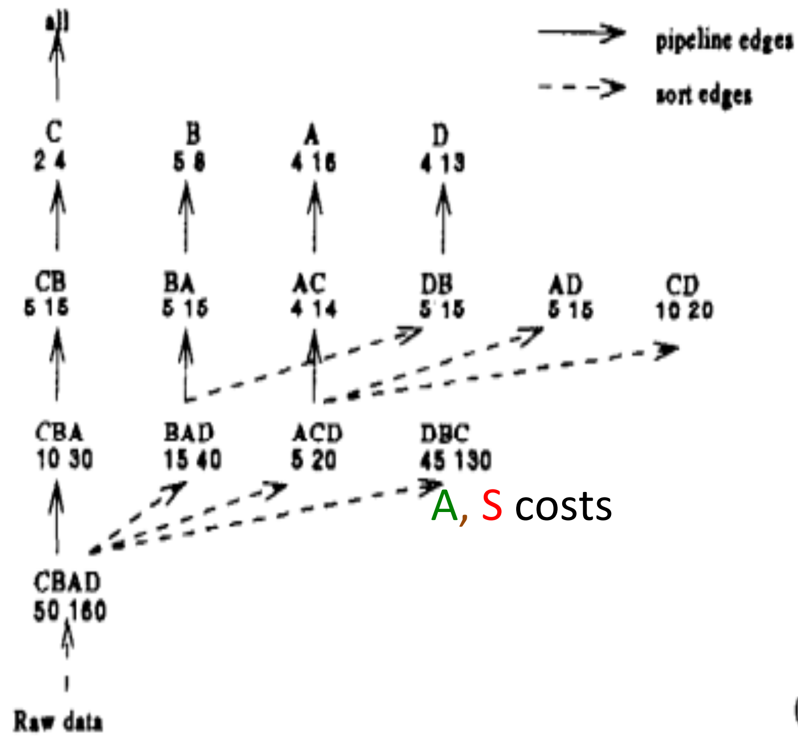
Outline: PipeSort Algorithm (4)

After computing the plan, execute all pipelines

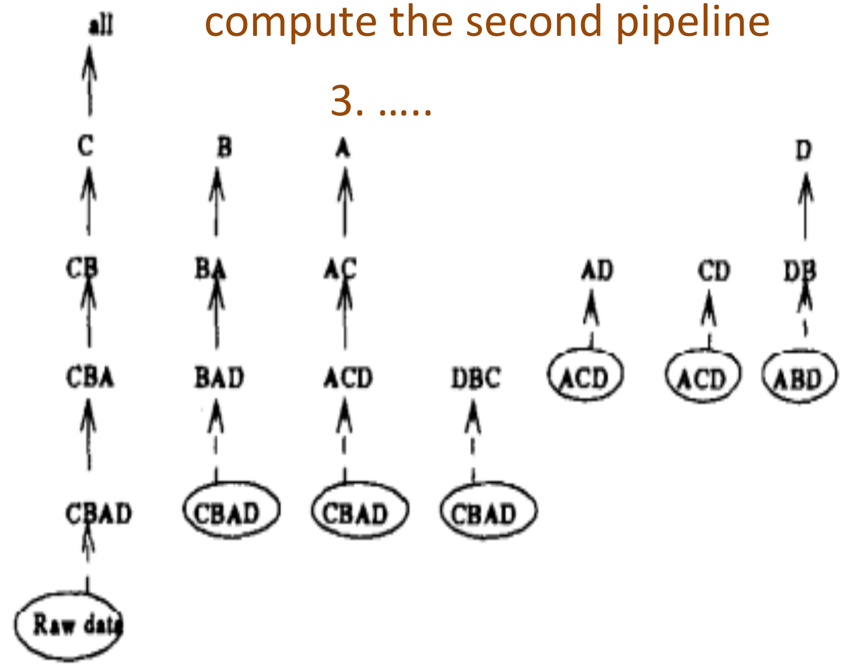
1. First pipeline is executed by one scan of the data

2. Sort CBAD -> BADC,
compute the second pipeline

3. …..



(a) The minimum cost sort plan

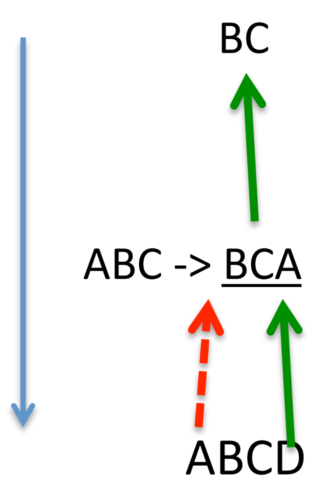


(b) The pipelines that are executed

Outline: PipeSort Algorithm (5)

Observations:

- Finds the best plan for computing level k from level k+1
  - Assuming the cost of sorting "BAD" does not depend on how the GROUP-BY on "BAD" has been computed
- Generating plan k+1 -> k does not prevent generating plan k+2 -> k+1 from finding the best choice
- Not provably globally optimal
  - e.g. can the optimal plan compute AB from ABCD?
  - something to explore!



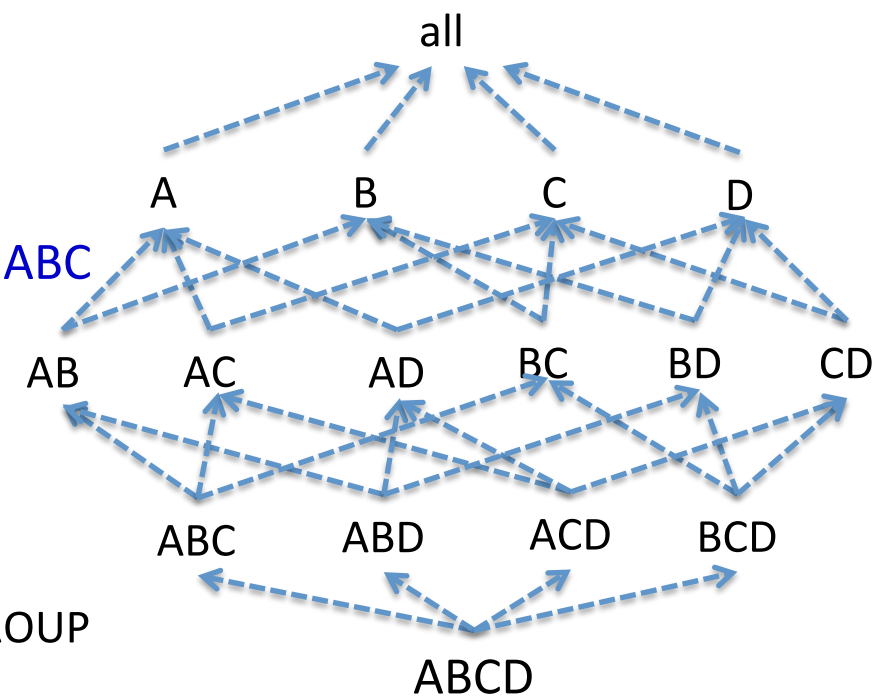
If the green edge is chosen, the sorted order of ABCD will be BCAD

PipeHash Algorithm

PipeHash: Basic Idea (1)

N = 4

- Use hash tables to compute smaller GROUP-BYs
- If the hash tables for AB and AC fit in memory, compute both in one scan of ABC
- With no memory restrictions



for k = N…0:

For each k+1-attribute GROUP BY g

Compute in one scan of g all k-attribute GROUP BY where g is smallest parent

Save g to disk and destroy the hash table of g

A	B	C	sum
a1	b1	c1	5
a1	b1	c2	10
a2	b2	c3	8
a2	b2	c1	2
a2	b2	c3	11

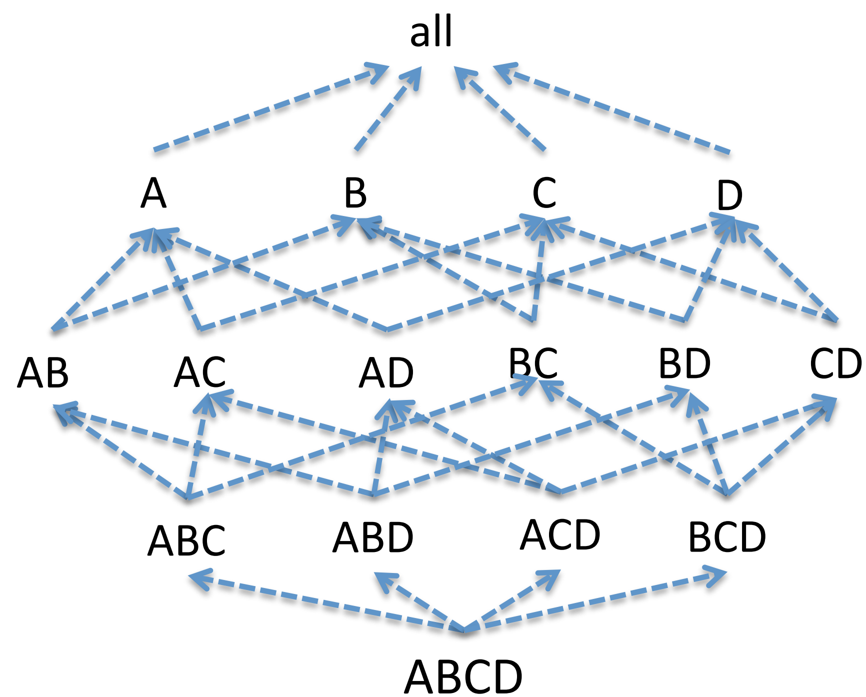
A	C	sum
a1	c1	5
a1	c2	10
a2	c3	19
a2	c1	2

A	B	sum
a1	b1	15
a1	b2	8
a2	b2	13

PipeHash: Basic Idea (2)

N = 4

- But, data might be large, Hash Tables may not fit in memory
- Solution: optimization "shared-partition"
  - partition data on one or more attributes
  - Suppose the data is partitioned on attribute A
  - All GROUP-Bys containing A (AB, AC, AD, ABC…) can be computed independently on each partition
  - Cost of partitioning is shared by multiple GROUP-BYs



A	B		sum
a1	b1	→	15
a1	b2	→	8
a2	b2	→	13

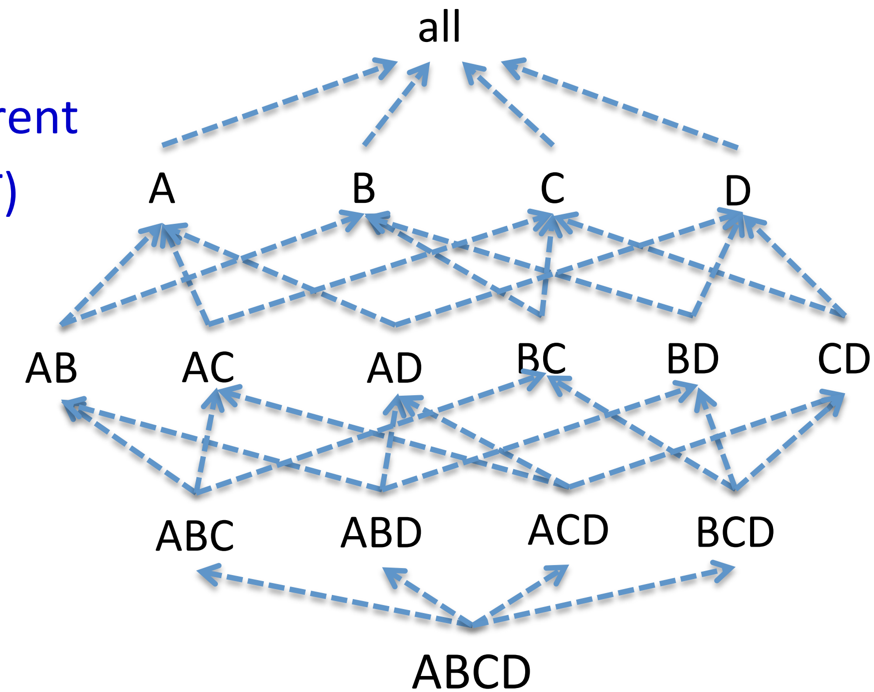
A	C		sum
a1	c1	→	5
a1	c2	→	10
a2	c3	→	19
a2	c1	→	2

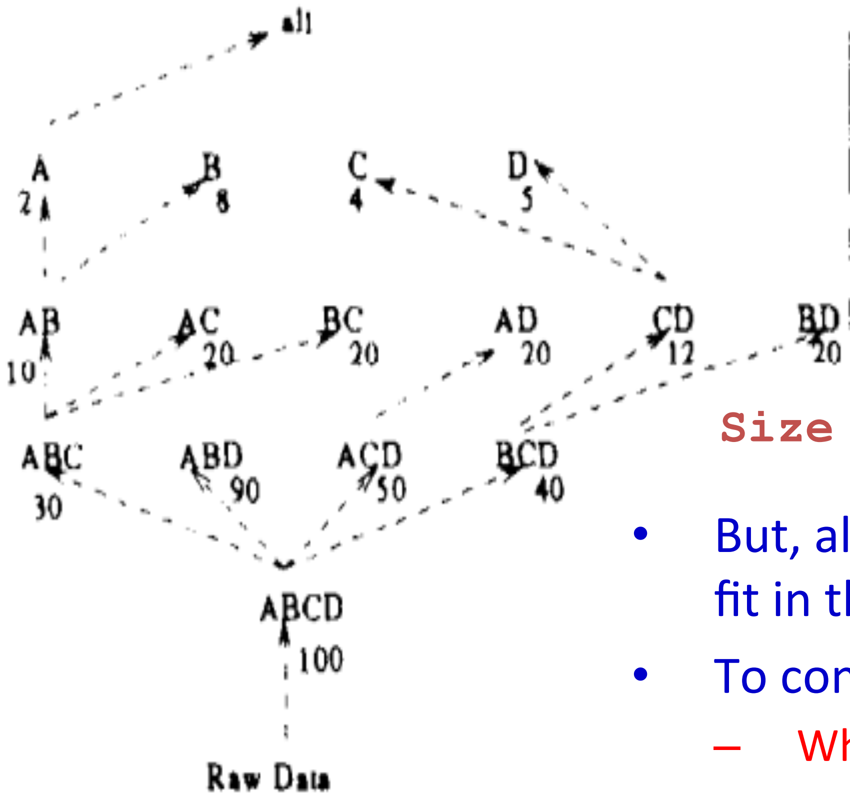
A	B	C	sum
a1	b1	c1	5
a1	b1	c2	10
a2	b2	c3	8
a2	b2	c1	2
a2	b2	c3	11

PipeHash: Basic Idea (3)

N = 4

- Input: search lattice
- For each group-by, select smallest parent
- Result: Minimum Spanning Tree (MST)





(a) Minimum spanning tree

Size of GROUP-BY

- But, all Hash Tables (HT) in the MST may not fit in the memory together
- To consider:
  - Which GROUP-BYs to compute together?
  - When to allocate-release memory for HT?
  - What attributes to partition on?

Outline: PipeHash Algorithm (1)

- Once again, a combinatorial optimization problem
- This problem is conjectured to be NP-complete in the paper
  - something to explore!
- Use heuristics

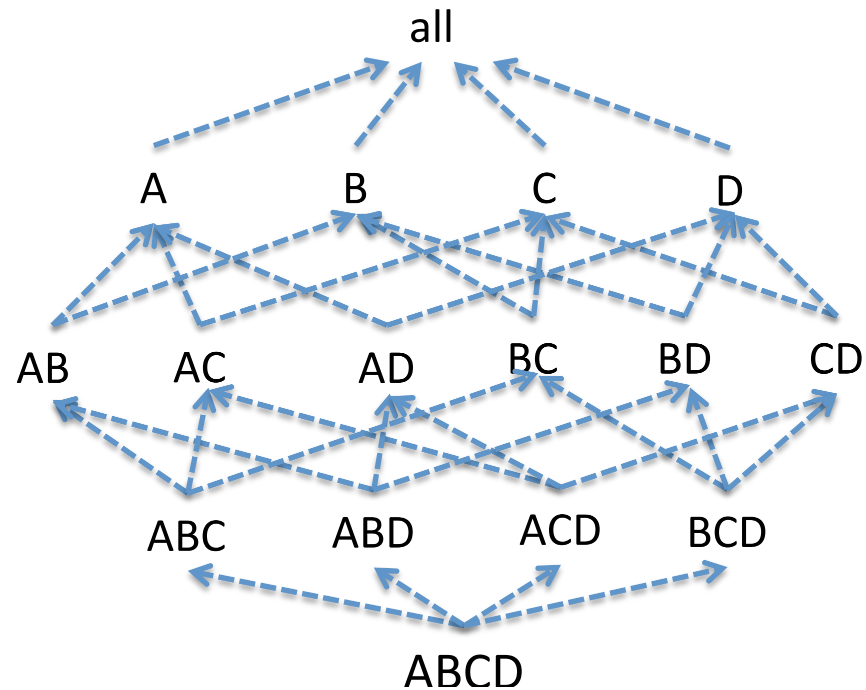
Trade-offs

1. Choose as large sub-tree of MST as possible ("cache-results", "amortized scan")
2. The sub-tree must include the partitioning attribute(s)

Heuristic

Choose a partitioning attribute that allows selection of the largest subtree of MST

Outline: PipeHash Algorithm (2)

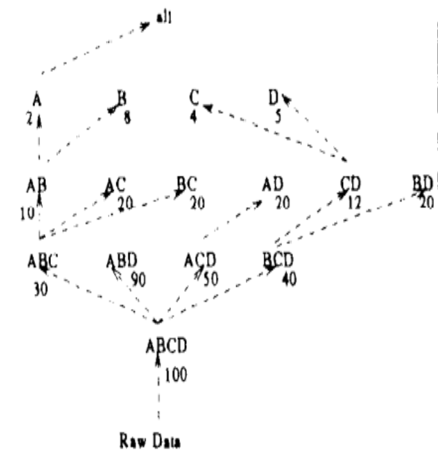


Algorithm

- Input: search lattice
- worklist = {MST}
- while worklist not empty
  - select one tree T from the worklist
  - T' = select-subtree(T)
  - Compute-subtree(T')

Next, through examples

- Select-subtree(T)
  - May add more subtrees to worklist
- Compute-subtree(T')



(a) Minimum spanning tree

Outline: PipeHash Algorithm (3)

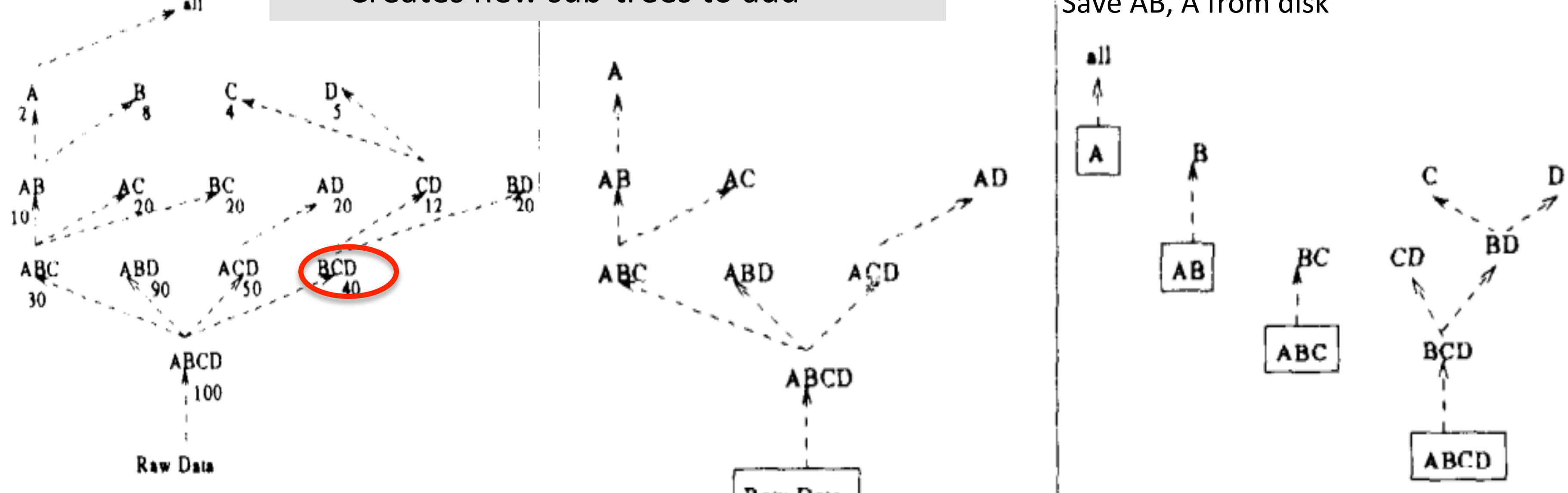
- T' = Select-Subtree(T) = T_A
- Compute-Subtree(T')

Partition T_A
For each partition,

- Compute GROUP-BY ABCD
- Scan ABCD to compute ABC, ABD, ACD
- Save ABCD, ABD to disk
- Compute AD from ACD
- Save ACD, AD to disk
- Compute AB, AC from ABC
- Save ABC, AC to disk
- Compute A from AB
- Save AB, A from disk

- s= {A} is such that
  - T_s per partition in P_s fits in memory
    P_s= #partitions
  - T' = T_s is the largest
- Creates new sub-trees to add

Hash-Table in memory until all children are created



(a) Minimum spanning tree

(b) First subtree: partitioned on A

(c) Remaining subtrees

Experiments

5 Experimental evaluation

In this section, we present the performance of our cube algorithms on several real-life datasets and analyze the behavior of these algorithms on tunable synthetic datasets. These experiments were performed on a RS/6000 250 workstation running AIX 3.2.5. The workstation had a total physical memory of 256 MB. We used a buffer of size 32 MB. The datasets were stored as flat files on a local 2GB SCSI 3.5" drive with sequential throughput of about 1.5 MB/second.

NH:NaiveHash PH:PipeHash NS:NaiveSort PS:PipeSort

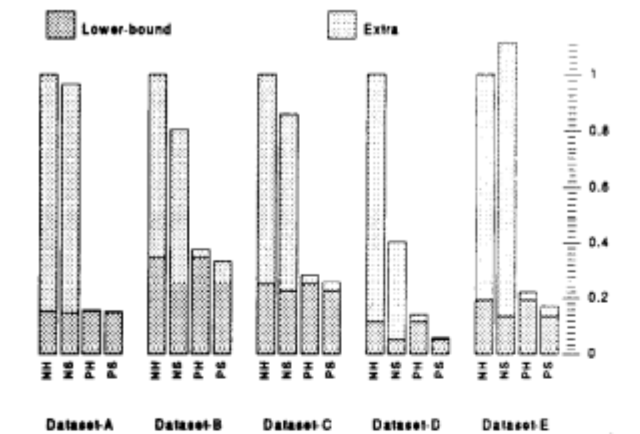


Figure 5: Performance of the cube computation algorithms on the five real life datasets. The y-axis denotes the total time normalized by the time taken by the NaiveHash algorithm for each dataset.

- Here sort-based better than hash-based (new hash-table for each GROUP-BY)
- Another experiment on synthetic data (see paper)
- For less sparse data, hash-based better than sort-based

Summary

- Similar Overlap algorithm by Deshpande et al. (see paper)

- All algorithms try to pick the best plan to compute aggregates with fewer scans and maximal memory usage
- Finding optimal decisions for each algorithm may be NP-complete
- Algorithms use heuristics that work well in practice

- Next class: other efficient implementations and index for cube