Data Mining:

Concepts and Techniques

(3rd ed.)

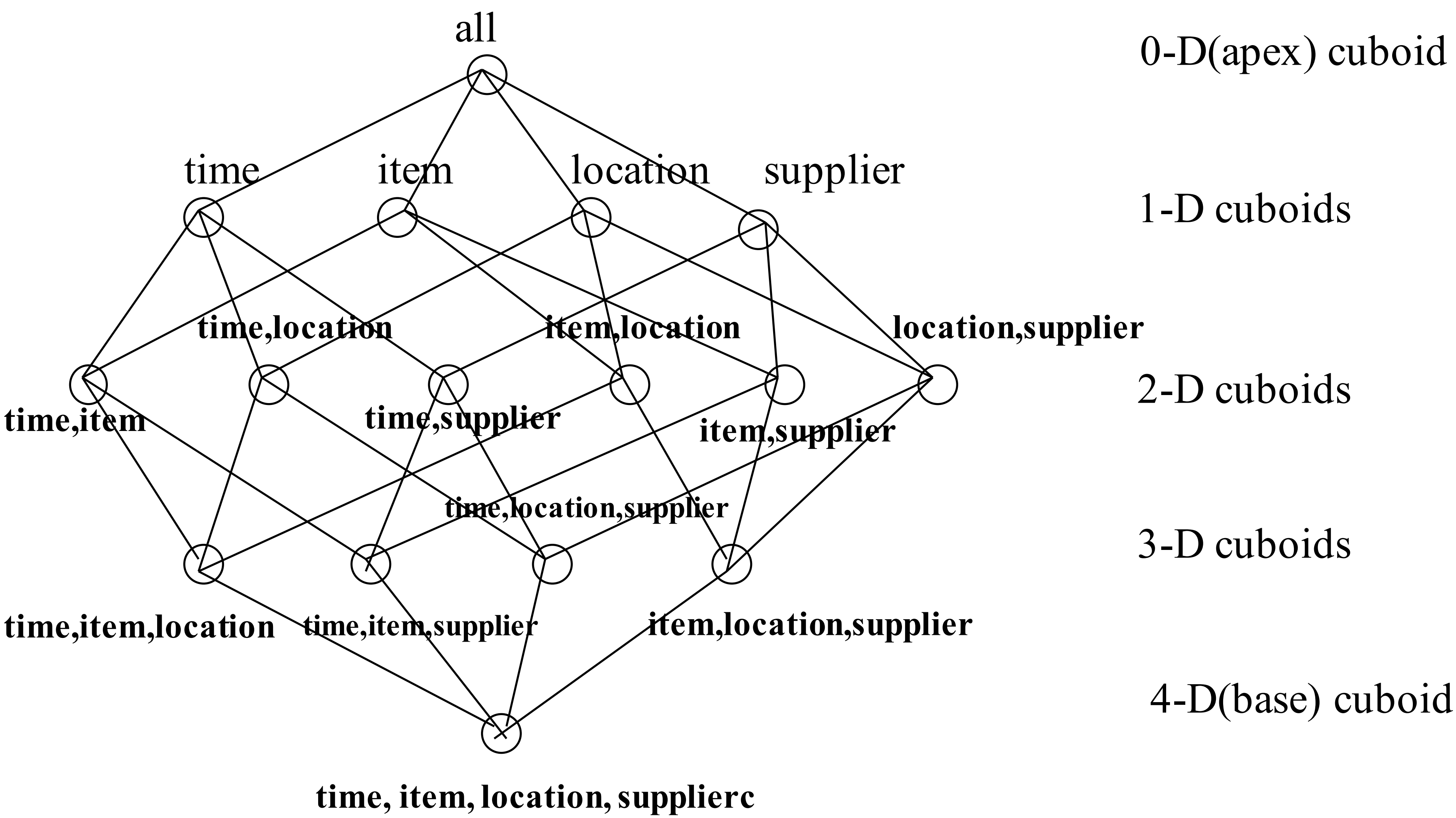
— Chapter 5 —

Slides Courtesy of Textbook

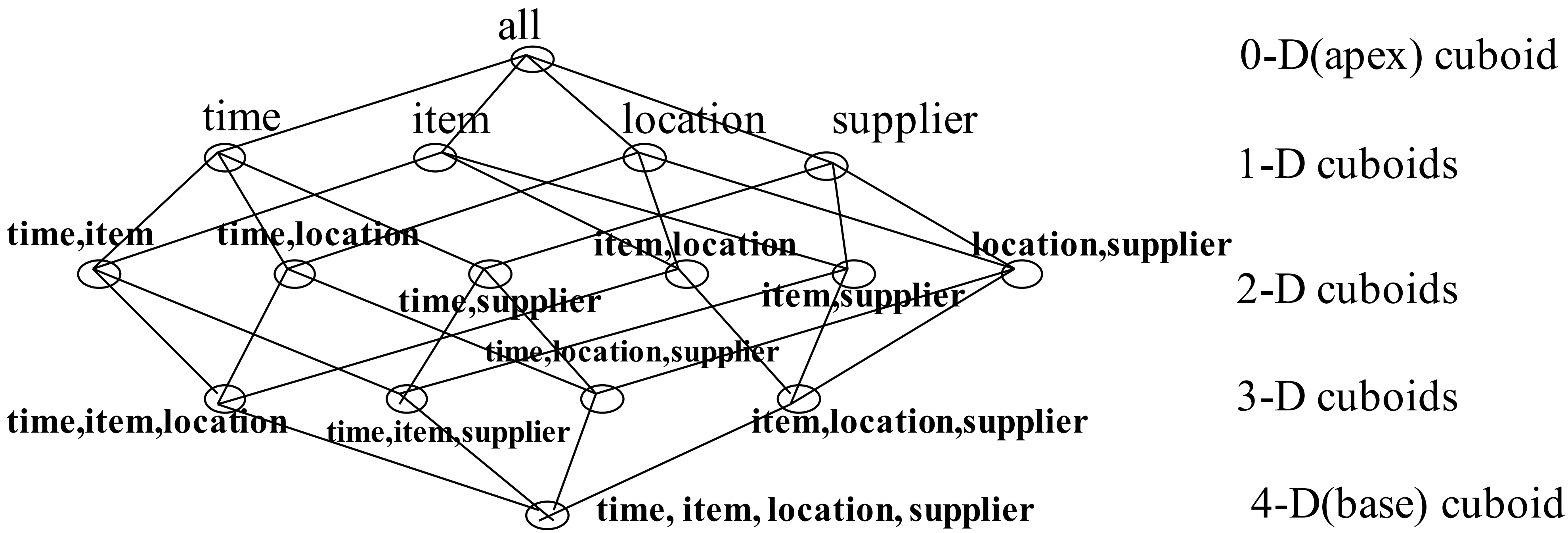
Chapter 5: Data Cube Technology

- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
- Processing Advanced Queries by Exploring Data Cube Technology
- Summary

Data Cube: A Lattice of Cuboids



Data Cube: A Lattice of Cuboids



- Base vs. aggregate cells; ancestor vs. descendant cells; parent vs. child cells
  1. (9/15, milk, Urbana, Dairy_land)
  2. (9/15, milk, Urbana, *)
  3. (*, milk, Urbana, *)
  4. (*, milk, Urbana, *)
  5. (*, milk, Chicago, *)
  6. (*, milk, *, *)

Cube Materialization: Full Cube vs. Iceberg Cube

- Motivation: To ensure fast OLAP, we pre-compute the full cube or partial cube.
- Pre-compute Full Cube: Pre-compute every cell in the cube.
  - Data analysts have really fast access to data in each cuboid.
  - Huge cost in memory & time. (e.g. Explosive growth with the # of dimension.)

- Pre-compute Partial Cube (Iceberg Cube): Pre-compute cells satisfying some conditions (e.g. count > min support,…)
  - Many cells are of no interest to data analysts.
  - Save cost in memory & time.

Iceberg Cube

- Computing *only* the cuboid cells whose measure satisfies the iceberg conditions

- Only a small portion of cells may be "above the water'' in a sparse cube

iceberg condition

compute cube sales iceberg as
select month, city, customer group, count(*)
from sales Information
cube by month, city, customer group
having count(*) >= min support

Iceberg Cube

- Avoid explosive growth: A cube with 100 dimensions
  - 2 base cells with count = 1 (Others' count = 0).
    - $A = (a_1, a_2, \ldots., a_{100}), B = (b_1, b_2, \ldots, b_{100})$
  - How many aggregate cells if "having count >= 1"?
  - What about "having count >= 2"?

- Is iceberg cube good enough?
  - 2 non-zero base cells:
    - $\{(a_1, a_2, a_3 \ldots, a_{100}){:}10, (a_1, a_2, b_3, \ldots, b_{100}){:}10\}$
  - How many cells will the iceberg cube have if having count(*) >= 10? Hint: A huge but tricky number!

Closed Cube & Cube Shell

- Close cube:
  - Closed cell c: if there exists no cell d, s.t. d is a descendant of c, and d has the same measure value as c.
  - Closed cube: a cube consisting of only closed cells
  - Given 2 base cells: $\{(a_1, a_2, a_3 \ldots, a_{100}):10, (a_1, a_2, b_3, \ldots, b_{100}):10\}$, what is the closed cube of the base cuboid? Hint: only 3 cells
- Cube Shell
  - Pre-compute only the cuboids involving a small # of dimensions, e.g., 3
  - More dimension combinations will need to be computed on the fly

For $(A_1, A_2, \ldots A_{10})$, how many combinations to compute?

Chapter 5: Data Cube Technology

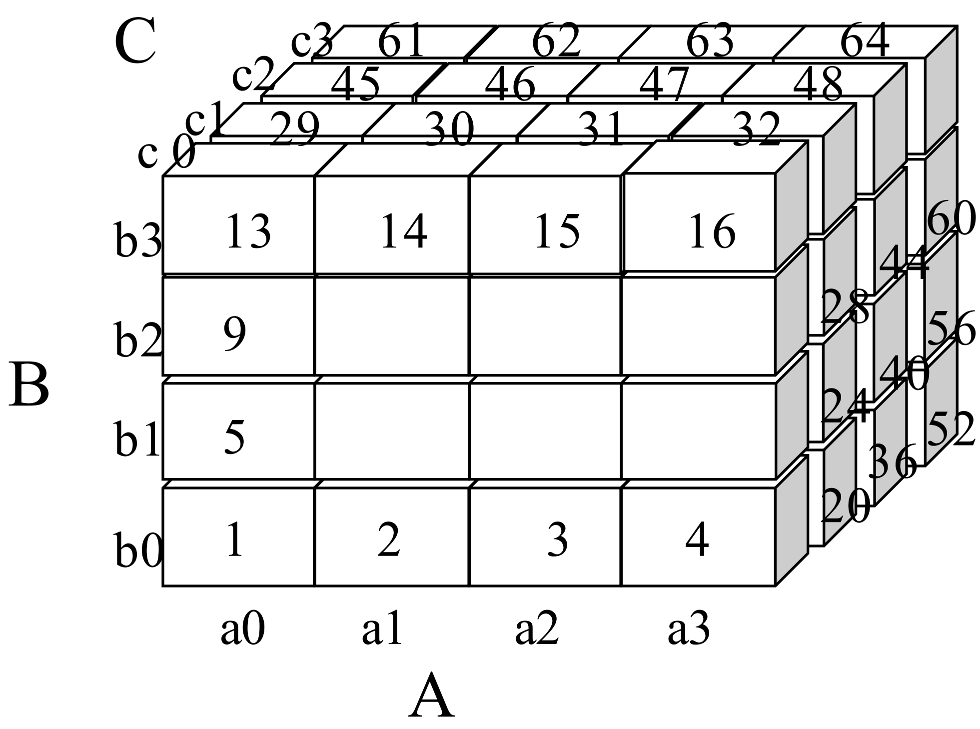
- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
  - Multi-Way Array Aggregation
  - BUC
- Processing Advanced Queries by Exploring Data Cube Technology
- Summary

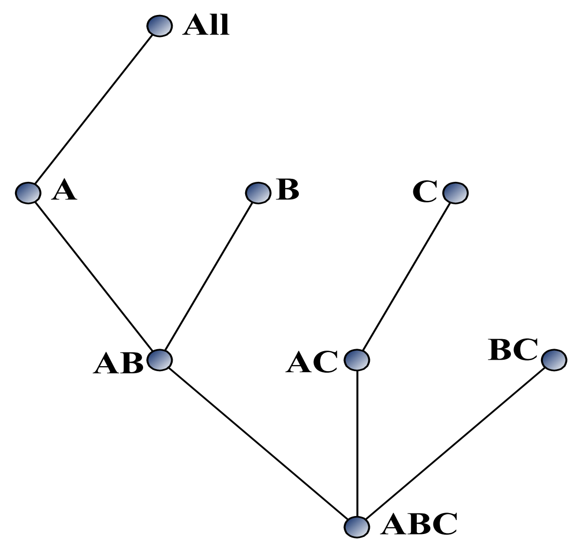
Multi-Way Array Aggregation

- Introduction:
  - Array-based "bottom-up" algorithm
  - Using multi-dimensional chunks
- High Efficiency for Full Cube Computation:
  - Simultaneous aggregation on multiple dimensions
  - Intermediate aggregate values are re-used for computing ancestor cuboids
- No Iceberg Optimization:
  - Cannot do *Apriori* pruning

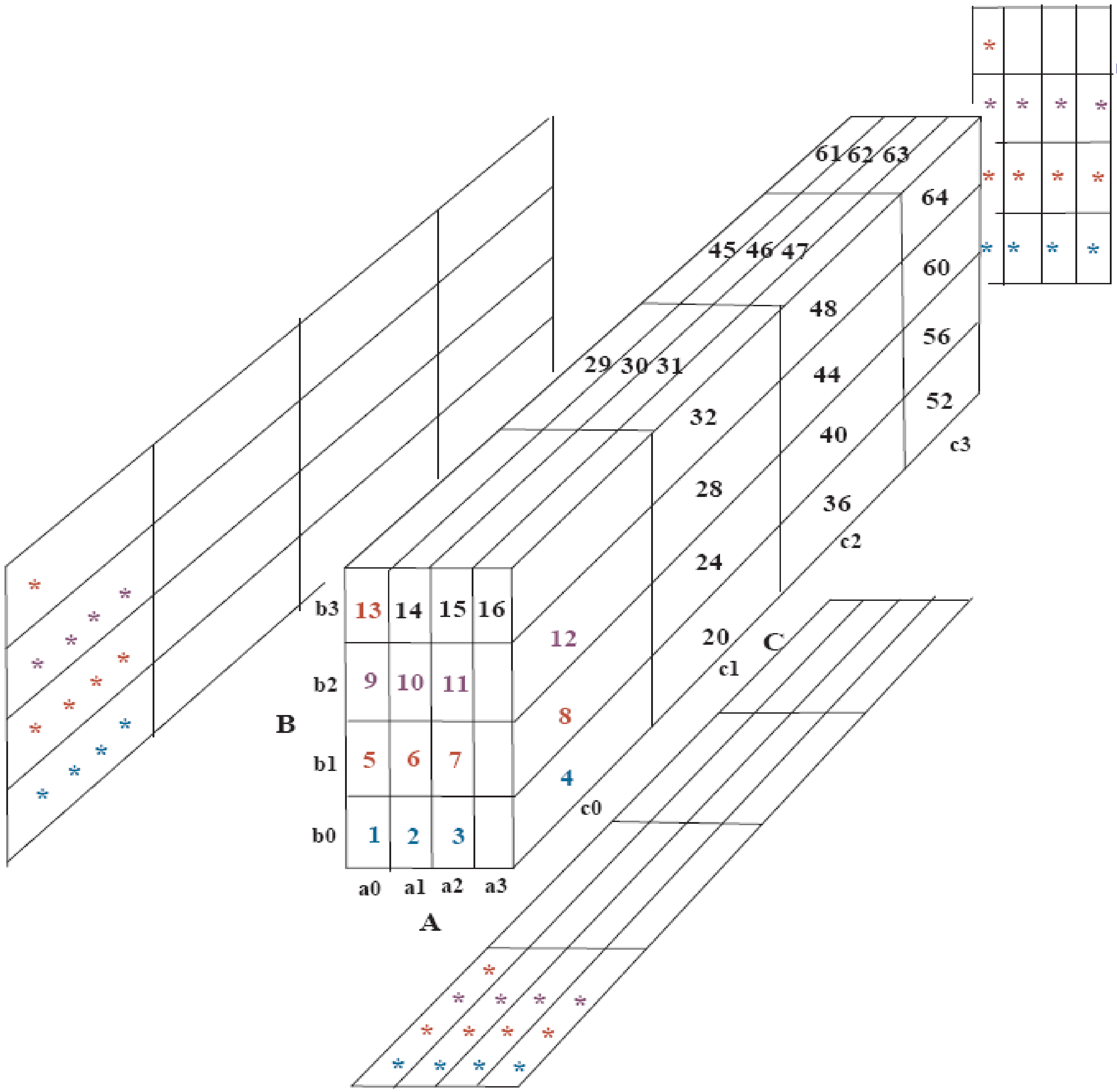
Multi-way Array Aggregation for Cube Computation

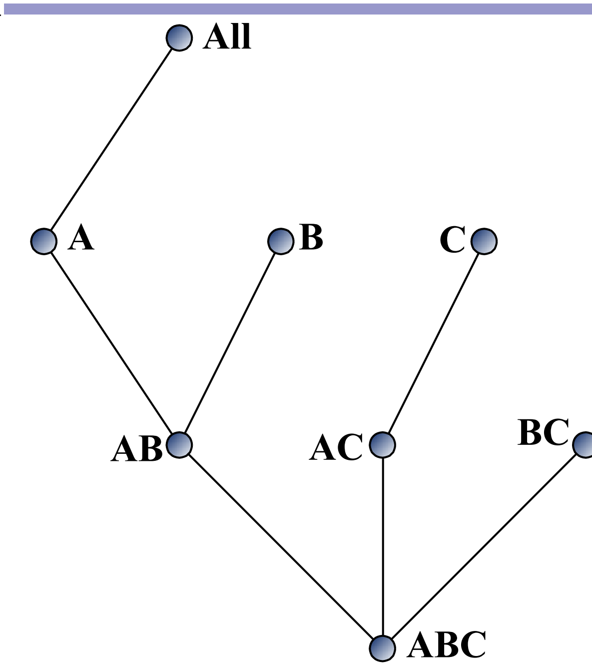
- Partition arrays into chunks (a small sub-cube which fits in memory).
- Compute aggregates in "multi-way" by visiting cube cells in the **order** which minimizes the # of times to visit each cell, and reduces memory access and storage cost.





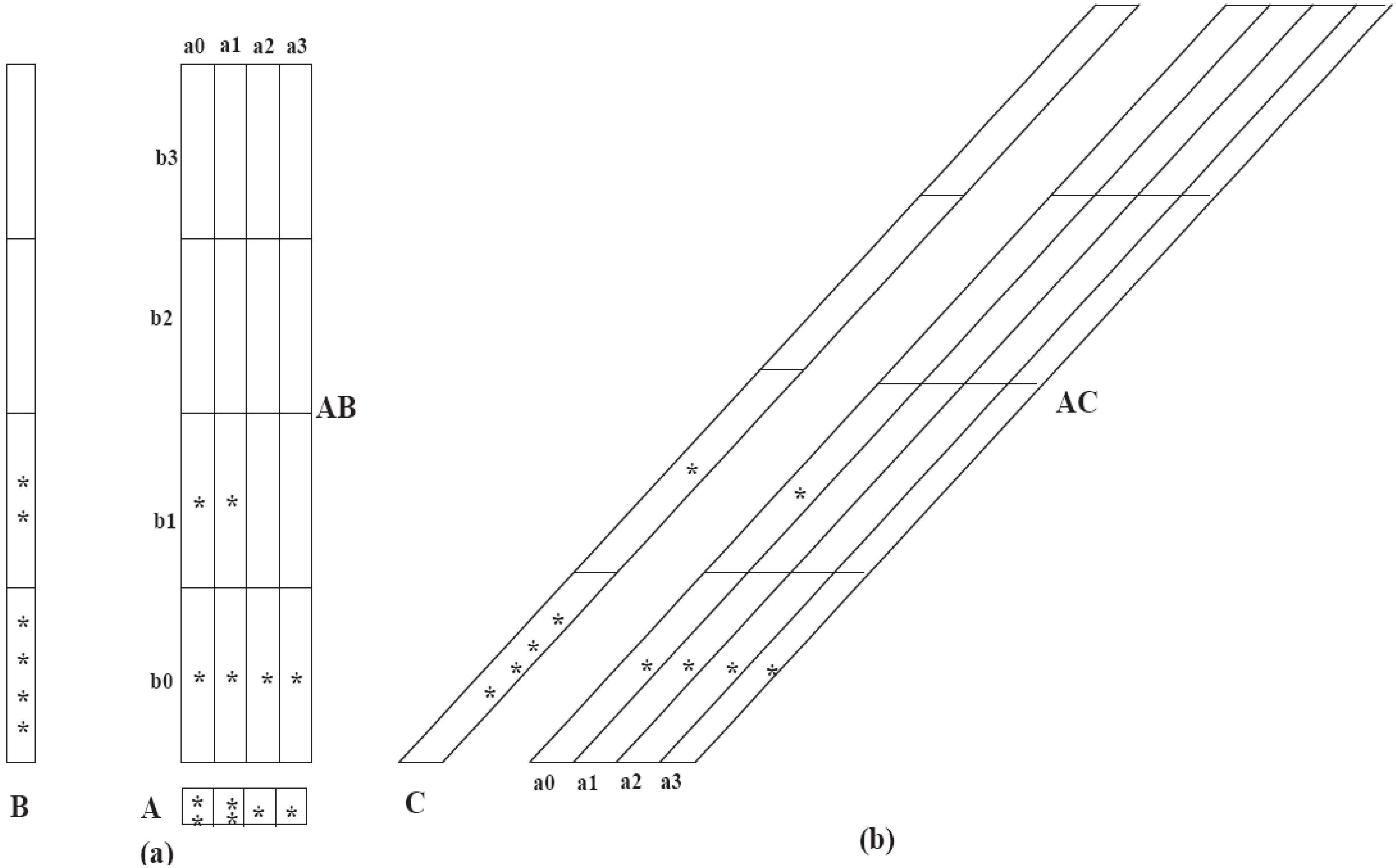
Multi-way Array Aggregation for Cube Computation (3-D to 2-D)





- The best order is the one that minimizes the memory requirement and reduced I/Os

Multi-way Array Aggregation for Cube Computation (2-D to 1-D)



Multi-Way Array Aggregation for Cube Computation (Method Summary)

- Method: The planes should be sorted and computed according to their size in ascending order
  - Idea: Keep the smallest plane in the main memory, fetch and compute only one chunk at a time for the largest plane
- Limitation of the method: Computing well only for a small number of dimensions
  - If there are a large number of dimensions, "top-down" computation and iceberg cube computation methods can be explored

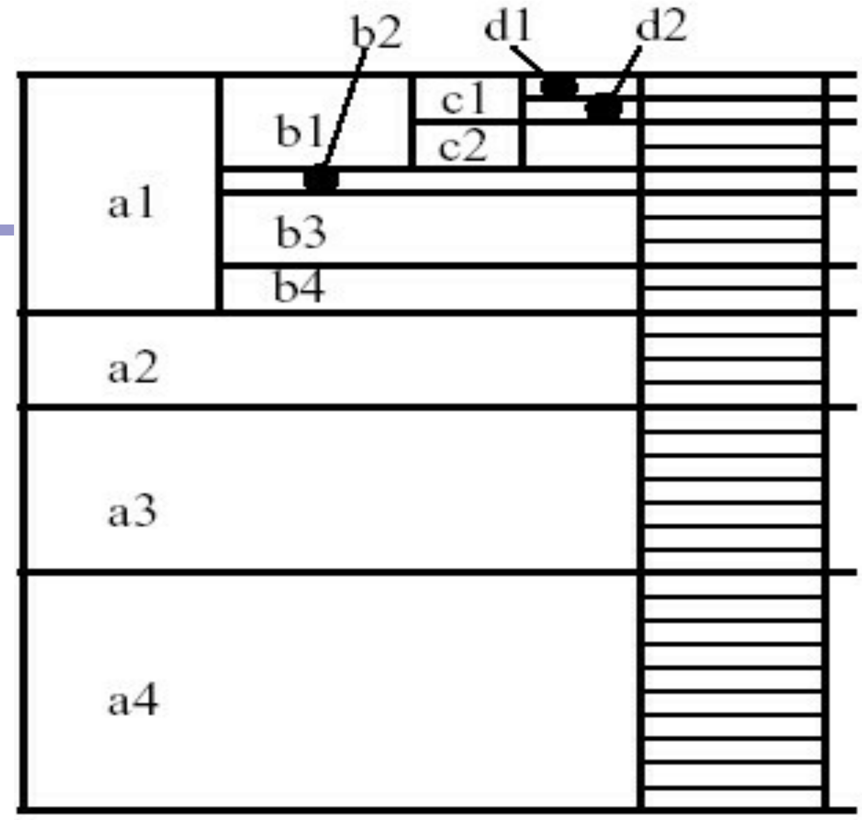
Bottom-Up Computation (BUC)

Top Down

- Introduction:
  - BUC (Beyer & Ramakrishnan, SIGMOD'99)
  - Bottom-Up cube computation

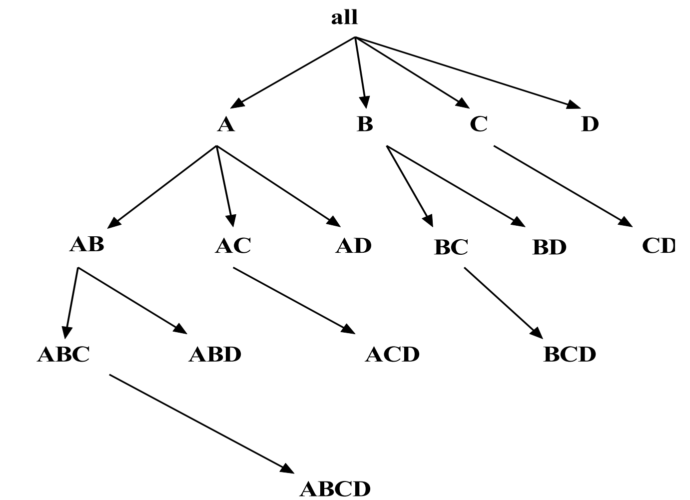
    (Note: **Top-Down** in our view!)
- Partition and Optimization: Divides dimensions into partitions and facilitates iceberg pruning
  - If a partition does not satisfy *min_sup*, its descendants can be pruned
  - If *min_sup* = 1 $\Rightarrow$ compute full CUBE!
- No simultaneous aggregation: Difference to Multi-Way Array Aggregation

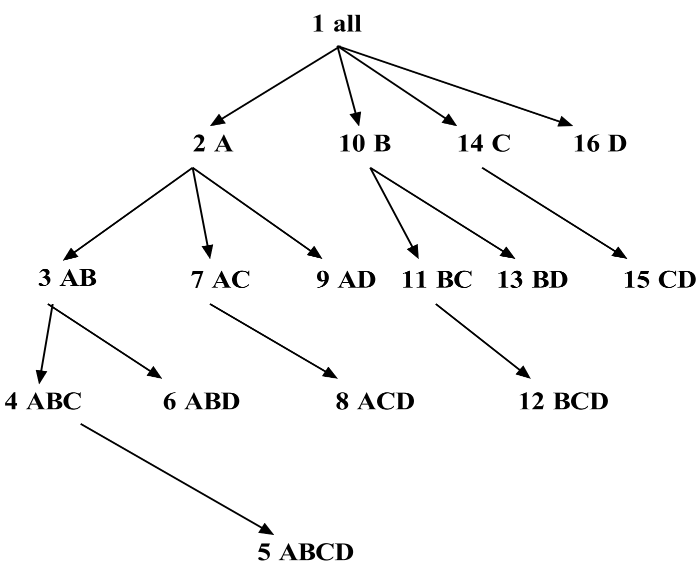
BUC: Partitioning



- Usually, entire data set can't fit in main memory
- Sort *distinct* values
  - partition into blocks that fit
- Continue processing
- Optimizations
  - Partitioning
    - External Sorting, Hashing, Counting Sort
  - Ordering dimensions to encourage pruning
    - Cardinality, Skew, Correlation
  - Collapsing duplicates
    - Can't do holistic aggregates anymore!

BUC Algorithm






```
Procedure BottomUpCube(input, dim)
Inputs:
   input: The relation to aggregate.
   dim: The starting dimension for this iteration.
Globals:
   constant numDims: The total number of dimensions.
   constant cardinality[numDims]: The cardinality of
         each dimension.
   constant minsup: The minimum number of tuples in a
         partition for it to be output.
   outputRec: The current output record.
   dataCount[numDims]: Stores the size of each partition.
         dataCount[i] is a list of integers of size
         cardinality[i].
Outputs:
   One record that is the aggregation of input.
   Recursively, outputs CUBE(dim, ..., numDims) on
         input (with minimum support).
Method:
   Aggregate(input); // Places result in outputRec
   if input.count() == 1 then // Optimization
         WriteAncestors(input[0], dim); return;
   write outputRec;
   for d = dim ; d < numDims ; d++ do
      let C = cardinality[d];
      Partition(input, d, C, dataCount[d]);
      let k = 0;
      for i = 0 ; i < C ; i++ do // For each partition
         let c = dataCount[d][i]
         if c >= minsup then // The BUC stops here
            outputRec.dim[d] = input[k].dim[d];
            BottomUpCube(input[k ... k+c], d+1);
         end if
         k += c;
      end for
      outputRec.dim[d] = ALL;
   end for
```

BUC Running 1

output: (*, *, *, *)
BUC(input[0..30], d=1)

Dimension Order: A, B, C, D

Return output: (*, *, *, *): 31

<u>d=1 → Partition by dim A</u>

output: (a1, *, *, *)
BUC(input[0..10], d=2)
output: (a2, *, *, *)
BUC(input[10..14], d=2)
output: (a3, *, *, *)
BUC(input[14..21], d=2)
output: (a4, *, *, *)
BUC(input[21..30], d=2)

<u>d=2 → Partition by dim B</u>

output: (*, b1, *, *)
BUC(input[0..7], d=3) // // Any more expansion from here?
output: (*, b2, *, *)
BUC(input[7..8], d=3)
output: (*, b3, *, *, *)
BUC(input[8..23], d=3)
output: (*, b4, *, *)
BUC(input[23..30], d=3)

<u>d=3 → Partition by dim C</u>

output: (*, *, c1, *)
BUC(input[0..13], d=4)
output: (*, c2, *, *)
BUC(input[13..30], d=4)

<u>d=4 → Partition by dim D</u>

output: (*, *, *, d1)
BUC(input[0..11], d=5) // Any more expansion from here?
output: (*, *, *, d2)
BUC(input[11..30], d=5)

BUC Running 2

output: (a1, *, *, *)
BUC(input[0..10], d=2)

Return output: (a1, *, *, *): 10

<u>d=2 → Partition by dim B</u>
output: (a1, b1, *, *)
BUC(input[0..4], d=3)
output: (a1, b2, *, *)
BUC(input[4..5], d=3)
output: (a1, b3, *, *, *)
BUC(input[5..8], d=3)
output: (a1, b4, *, *)
BUC(input[8..10], d=3)

<u>d=3 → Partition by dim C</u>
output: (a1, *, c1, *)
BUC(input[0..4], d=4) // Any more expansion from here?
output: (a1, *, c2, *, *)
BUC(input[4..10], d=4)

<u>d=4 → Partition by dim D</u>
output: (a1, *, *, d1)
BUC(input[0..9], d=5) // Any more expansion from here?
output: (a1, *, *, d2)
BUC(input[0:10], d=5)

BUC: Trace Tree of Expansion

How if different dimension order?

Chapter 5: Data Cube Technology

- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
- Processing Advanced Queries by Exploring Data Cube Technology
  - Sampling Cube: X. Li, J. Han, Z. Yin, J.-G. Lee, Y. Sun, "Sampling Cube: A Framework for Statistical OLAP over Sampling Data", SIGMOD'08
- Summary

Statistical Surveys and OLAP

- Statistical Survey: A popular tool to collect information about a **population** based on a **sample**
  - Ex.: TV ratings, US Census, election polls
- A common tool in politics, health, market research, science, and many more
- An efficient way of collecting information (Data collection is expensive)
- Many **statistical tools** available, to determine validity
  - Confidence intervals
  - Hypothesis tests
- OLAP (multidimensional analysis) on survey data
  - highly desirable but can it be done well?

Surveys: Sample vs. Whole Population

Data is only a sample of **population**

Age\Education	High-school	College	Graduate
18			
19			
20			
…			

Problems for Drilling in Sampling Cube

- OLAP on Survey (i.e., Sampling) Data
- Semantics of query is unchanged, but input data is changed

Age/Education	High-school	College	Graduate
18			
19			
20			
…			

Data is only a **sample** of population but samples could be small when drilling to certain multidimensional space

Challenges for OLAP on Sampling Data

Q: What is the average income of 19-year-old high-school students?

A: Returns not only query result but also confidence interval

- Computing confidence intervals in OLAP context
- No data?
  - Not exactly. No data in subspaces in cube
  - Sparse data
  - Causes include sampling bias and query selection bias
- Curse of dimensionality
  - Survey data can be high dimensional
  - Over 600 dimensions in real world example
  - Impossible to fully materialize

Confidence Interval

- *Confidence interval at : $\bar{x}$ $\bar{x} \pm t_c \hat{\sigma}_{\bar{x}}$*
  - *x is a sample of data set; i $\bar{x}$ he mean of sample*
  - *t_c is the critical t-value, calculated by a look-up*
  - $\hat{\sigma}_{\bar{x}} = \frac{s}{\sqrt{l}}$ the estimated standard error of the mean
- *Example**:** \$50,000 ± \$3,000 with 95% confidence*
  - Treat points in cube cell as samples
  - Compute confidence interval as traditional sample set
- Return answer in the form of confidence interval
  - Indicates **quality** of query answer
  - User selects desired confidence interval

Efficient Computing Confidence Interval Measures

- Efficient computation in all cells in data cube
  - Both mean and confidence interval are **algebraic**
  - Why confidence interval measure is algebraic?

$$\bar{x} \pm t_c \hat{\sigma}_{\bar{x}}$$

$\bar{x}$ is algebraic

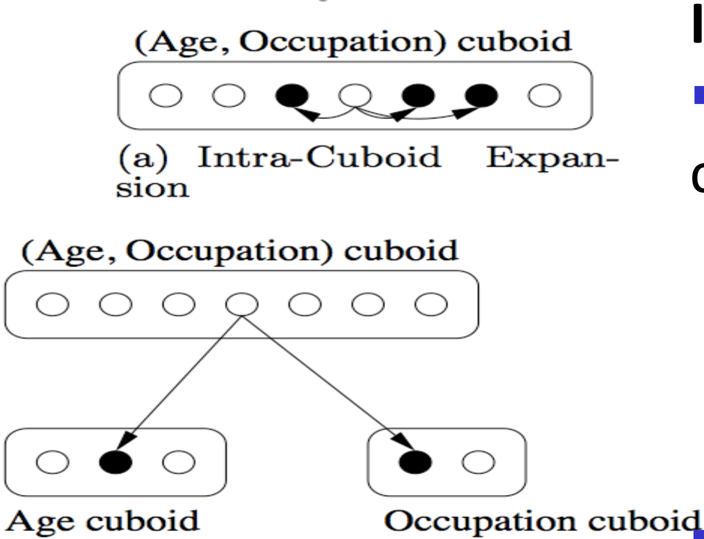
$\hat{\sigma}_{\bar{x}} = \frac{s}{\sqrt{l}}$ where both *s* and *l (count)* are algebraic

- Thus one can calculate cells efficiently at more general cuboids without having to start at the base cuboid each time

Boosting Confidence by Query Expansion

- From the example: The queried cell "19-year-old college students" contains only 2 samples
- Confidence interval is large (i.e., low confidence). why?
  - Small sample size
  - High standard deviation with samples
- Small sample sizes can occur at relatively low dimensional selections
  - Collect more data?— expensive!
  - Use data in other cells? Maybe, but have to be careful

uery Expansion: Intra-Cuboid Expansion



(a) Intra-Cuboid Expansion
(b) Inter-Cuboid Expansion

Intra-Cuboid Expansion

- Combine other cells' data into own to "boost" confidence
  - If share semantic and cube similarity
  - Use only if necessary
  - Bigger sample size will decrease confidence interval
- Cell segment similarity
  - Some dimensions are clear: **Age**
  - Some are fuzzy: **Occupation**
  - May need domain knowledge
- Cell value similarity
  - How to determine if two cells' samples come from the same population?
  - Two-sample t-test (confidence-based)

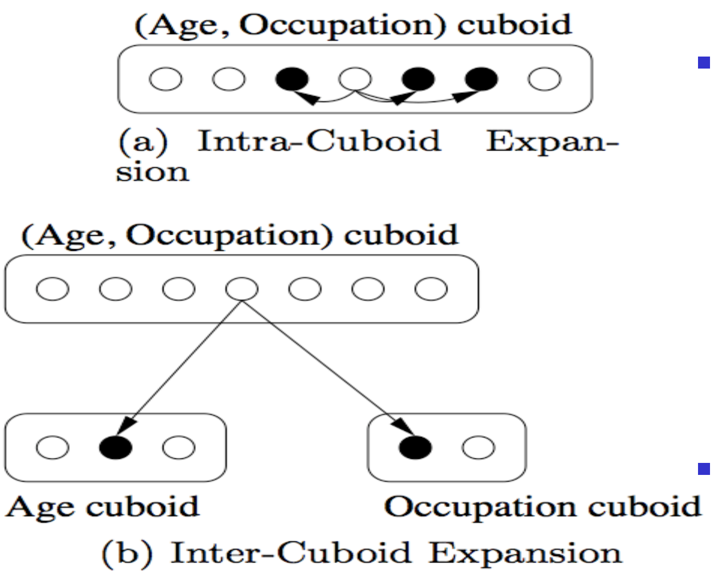
Intra-Cuboid Expansion

What is the average income of 19-year-old college students?

Age/Education	**High-school**	**College**	**Graduate**
18			
19			
20			
…			

Expand query to include **18** and **20** year olds? Vs. expand query to include **high-school** and **graduate** students?

Query Expansion: Inter-Cuboid Expansion



- If a query dimension is
  - Not correlated with cube value
  - But is causing small sample size by drilling down too much
- Remove dimension (i.e., generalize to *) and move to a more general cuboid
- Can use two-sample t-test to determine similarity between two cells across cuboids
- Can also use a different method to be shown later

Chapter 5: Data Cube Technology

- Data Cube Computation: Preliminary Concepts
- Data Cube Computation Methods
- Processing Advanced Queries by Exploring Data Cube Technology
- Summary

Data Cube Technology: Summary

- Data Cube Computation: Preliminary Concepts

- Data Cube Computation Methods
  - Multi-Way Array Aggregation
  - BUC

- Processing Advanced Queries by Exploring Data Cube Technology
  - Sampling Cubes