Efficient Implementation of Data Cubes Via Materialized Views

Jeffrey D. Ullman

Department of Computer Science
Stanford University
Stanford CA 94305
`ullman@cs.stanford.edu`
`http://db.stanford.edu/~ullman`

Abstract

Data cubes are specialized database management systems designed to support multidimensional data for such purposes as decision support and data mining. For a given mix of queries, we can optimize the implementation of a data cube by materializing some projections of the cube. A greedy approach turns out to be very effective; it is both polynomial-time as a function of the number of possible views to materialize and guaranteed to come close to the optimum choice of views. The work reported here is a summary of results appearing in the following two papers:

V. Harinarayan, A. Rajaraman, and J. D. Ullman, "Implementing data cubes efficiently." To appear in 1996 SIGMOD. An extended version is available by anonymous ftp from `db.stanford.edu` as `pub/harinarayan/1995/cube.ps`.

H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman, "Index selection for OLAP." Available by anonymous ftp from `db.stanford.edu` as `pub/hgupta/1996/CubeIndex.ps`.

Data Cubes

- Special-purpose DBMS for storing multidimensional data and handling queries that aggregate over some dimensions.

Example 1: Consider information about sales at a chain store. Dimensions might include day of sale, item sold, store at which sold, color of item, etc. Figure 1 suggests a 4-dimensional cube.

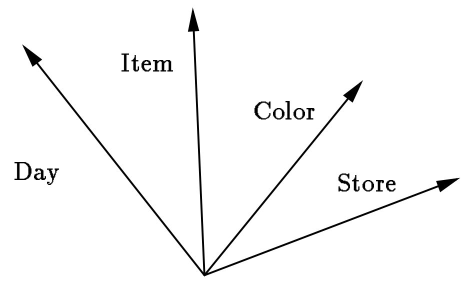


Fig. 1. A 4-dimensional data cube.

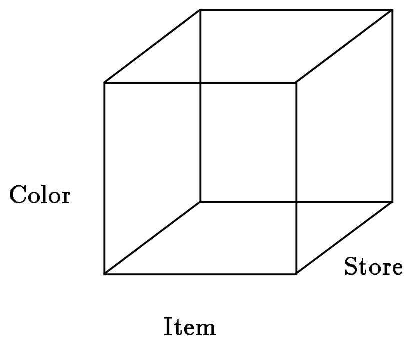
- One *critical* attribute represents the quantity to be analyzed, e.g. dollar amount of sale.

Views

The natural choice of views to materialize is a subset of the views of the form

```
SELECT <grouped attributes>,
    SUM(<critical attribute>)
FROM <the data cube relation>
GROUP BY <some attributes>
```

- That is, a view is a projection of the cube onto some of its dimensions. Example:

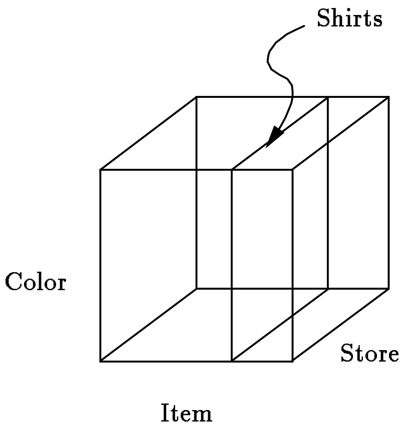


Queries

Typically group by some dimensions, select particular values for some other dimensions, and aggregate (sum the critial attribute) over the other dimensions.

```
SELECT <grouped attributes>,
    SUM(<critical attribute>)
FROM <the data cube relation>
WHERE <some attributes equated
    to particular values>
GROUP BY <some other attributes>
```

Example 2: The "shirts" plane of the Item-Color-Store cube represents the query "list the sales of shirts by color and store."



Relationship Between Queries and Views

Each query has a natural view from which it is most easily answered. However, it can be answered from views that group by more attributes; those views are larger and require additional cost. Most extreme: the raw data is a view (the *top view*) from which any query can be answered at great cost.

A (Slightly) More General Model

The *pure-or* model of view definitions:

- A collection of views (not necessarily projections of a data cube).
  - ✧ Each view can be constructed from any (perhaps none) of a set of "larger" views.
  - ✧ One view is the *top view*. It cannot be constructed from any view, and all views can be constructed from it.
- A collection of queries.
  - ✧ Each query has a weight, representing the likelihood of its being asked.
- For each query-view pair, there is a cost of answering the query from that view (may be ∞ if the view is unsuitable).
  - ✧ If query Q can be answered from view V, and V can be constructed from view W, then Q can be answered from W, and the cost is no greater than the cost of constructing V from W and then answering Q from V.
  - ✧ Each query can be answered from the top view at some large, fixed cost.

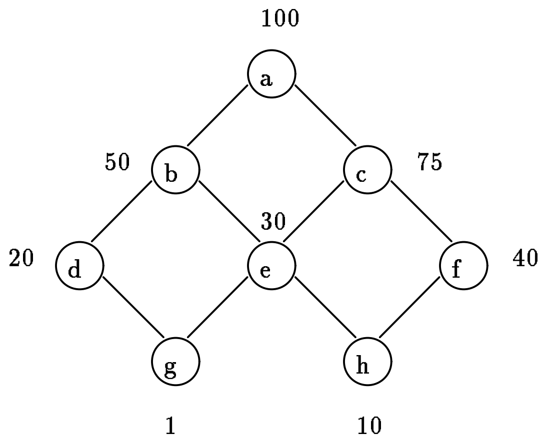


Fig. 2. Lattice of views to materialize.

Relationship to Data Cube

- Views form a hypercube.
- A view can be constructed from any view "above," i.e., a view that groups on a superset of attributes.
- A query has a natural "best" view, which groups by the same set of attributes.
- But a query can be answered from any view above its best view, at a cost equal to the size of that view (or some fraction if the appropriate indexes are available).

The Greedy Algorithm

- Assume the top view is materialed.
- Select additional views to materialize, one at a time, until some total cost of selected views is reached.
- At each step, select that unmaterialized view with the greatest *benefit*, i.e., the view that most reduces the average cost of answering a query, per unit space.

Example 3: In Fig. 2 is a lattice of views and their query costs.

- We shall assume that the queries each ask to see one of these views.
- a is the top view, already assumed materialized.
- *Simplifying assumptions*:
  - ✧ All views have unit cost of materialization.
  - ✧ All queries (= views) are equally likely.

- Want to select the 3 views that most improve the average query cost.

	First Choice	Second Choice	Third Choice
b	$50 \times 5 = 250$		
c	$25 \times 5 = 125$	$25 \times 2 = 50$	$25 \times 1 = 25$
d	$80 \times 2 = 160$	$30 \times 2 = 60$	$30 \times 2 = 60$
e	$70 \times 3 = 210$	$20 \times 3 = 60$	$20 + 20 + 10 = 50$
f	$60 \times 2 = 120$	$60 + 10 = 70$	
g	$99 \times 1 = 99$	$49 \times 1 = 49$	$49 \times 1 = 49$
h	$90 \times 1 = 90$	$40 \times 1 = 40$	$30 \times 1 = 30$

Analysis of Greedy Algorithm

Theorem (Harinarayan, Rajaraman, and Ullman): The benefit of the greedy algorithm can never be less than $(e-1)/e = 0.63$ times the benefit of the optimum choice of materialized views.

- *Oddity*: Frequently, after looking at the selection made by the greedy algorithm, we can deduce a much tighter bound. In particular, if either all chosen views contribute the same benefit, or the last view chosen contributes negligible benefit, then the greedy solution is optimal.

- A similar proof of the 0.63 bound appears in a different context by Cornujols, Fisher, and Nemhauser, "Location of bank accounts to optimize float," *Management Science* **23**, pp. 789–810, 1977.

- There is no tighter bound possible for the greedy algorithm, in general. Figure 3 is a counterexample.

- A recent result of C. Chekuri shows that *no* polynomial time whatsoever can have a worst-case bound better than the 0.63 that the greedy algorithm achieves.

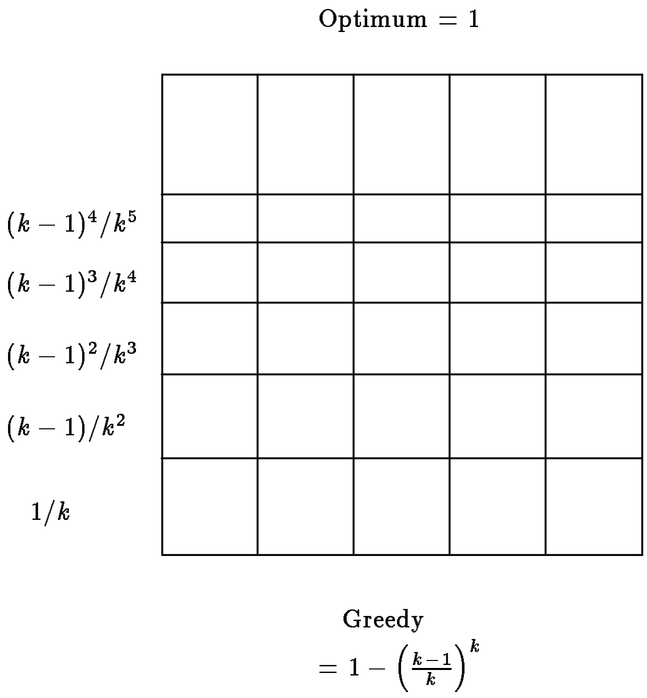


Fig. 3. Why greedy cannot do better than 63%.

More Complex Warehouses

The reason greedy is so successful is that the benefit function is *monotone*; that is, materializing a view never increases the benefit of some other view.

- If the costs of queries and views are nonmonotone, then greedy can be arbitrarily bad.

Views and Indexes

When queries involve specification of particular values for some attributes, e.g., "give the total sales of red items at the Des Moines store by month," indexes on materialized views can help. But there is no benefit to choosing an index without first choosing a view upon which it is an index.

- Thus, if we treat views and indexes equally as things to materialize, there is nonmonotonicity, and greedy can be arbitrarily bad.

A Greedy Algorithm for Views and Indexes

For any view, its *tail* of indexes is chosen by greedily adding one index at a time, until the benefit per unit space of the view and the chosen indexes can no longer be increased.

The full algorithm is to repeatedly choose either

1. An index for a previously selected view, or
2. A view plus its tail of indexes that has the maximum benefit per unit space

until all available space is consumed.

Theorem (H. Gupta, V. Harinarayan, A. Rajaraman, J. D. Ullman): The above algorithm runs in time polynomial in the number of views and indexes and never performs worse than 47% of the optimal solution.

- The actual constant is $1 - 1/e^{0.63}$.

Acknowledgements

This work was supported by NSF grant IRI–92–23405, ARO grant DAAH04–95–1–0192, and USAF contract F33615–93–1–1339.