Attribute-Value Reordering for Efficient Hybrid OLAP

Owen Kaser
Dept. of Computer Science and Applied Statistics
University of New Brunswick, Saint John, NB Canada

Daniel Lemire
National Research Council of Canada
Fredericton, NB Canada
Overview

✔ Coding dimensional values as integers
✔ Meet the problem (visually)
✔ Background (multidimensional storage)
✔ Packing data into dense chunks
✔ Experimental results
Background

Cube $C$ is a partial function from dimensions to a measure value.

e.g.,

$C : \text{Item} \times \text{Place} \times \text{Time} \rightarrow \text{Sales Amount}$

$C_{\text{Iced Tea, Auckland, January}} = 20000.0.$

$C_{\text{Car Wax, Toronto, February}} = \text{—}.$
Usefulness of Integer Indices in Cube $C$

Conceptually, $C_{\text{Iced Tea, Auckland, January}} = 20000.0$.

Suggestion: “replace strings by integers” often made.

For storage, system [or database designer] likely to code

*for Months*: January = 1, February = 2, …

*for Items*: Car Wax = 1, Cocoa Mix = 2, Iced Tea = 3, …

e.g., with row numbers in dimension tables (star schema)
Freedom in Choosing Codes

For **Item**, these codes are arbitrary. Any other assignment of \( \{1, \ldots, n\} \) to **Items** is a permutation of the initial one.

But for **Month**, there is a natural ordering.

And for **Place**, there may be a hierarchy (**City**, **State**, **Country**).

**Code assignments for Month and Place should be restricted.**

But to study the full impact, we don’t.
To display a 2-d cube $C$, plot pixel at $(x,y)$ when $C_{x,y} \neq 0$.

- rearranging (permuting) rows and columns can cluster/uncluster data
- left: nicely clustered; middle: columns permuted; right: rows too
Normalization

Let $C$ be a $d$-dimensional cube, size $n_1 \times n_2 \times \ldots \times n_d$

“Normalization” $\pi = (\gamma_1, \gamma_2, \ldots, \gamma_d)$, with each $\gamma_i$ a permutation for dimension $i$. i.e., $\gamma_i$ is a permutation of $1, 2, \ldots, n_i$.

Define “normalized cube” $\pi(C)$ by

$$\pi(C)[i_1, i_2, \ldots, i_d] = C[\gamma_1(i_1), \gamma_2(i_2), \ldots, \gamma_d(i_d)].$$

Note: $\gamma_i$: “came from”; thus $\gamma_i^{-1}$: “went to”

To retrieve $C[i_1, \ldots, i_d]$, use $\pi(C)[\gamma_1^{-1}(i_1), \ldots, \gamma_d^{-1}(i_d)].$
Sparse vs Dense Storage

$\#C$ — number of nonzero elements of $C$.

Density $\rho = \frac{\#C}{n_1 \times n_2 \times \ldots \times n_d}$; $\rho \ll 1$: sparse cube. Otherwise, dense.

**Sparse coding:**

▷ goal: storage space depends on $\#C$, not $n_1 \times \ldots \times n_d$.

▷ many approaches developed (decades-old work)
A Storage-Cost Model

Idea for sparse case: to record that $A[x_1, x_2, \ldots, x_d] = \nu$ we record a $d + 1$-tuple $(x_1, x_2, \ldots, x_d, \nu)$. The $x_i$’s are typically small.

Our model: To store a $d$-dimensional cube $C$ of size $n_1 \times n_2 \times \ldots \times n_d$ costs

1. $n_1 \times n_2 \times \ldots \times n_d$, if done densely,

2. $(d/2 + 1) \cdot \#C$, if done sparsely.
Chunked/Blocked Storage (Sarawagi’94)

Partition $d$-dim cube into $d$-dim subcubes, blocks.

For simplicity, assume block size $m_1 \times m_2 \times \ldots \times m_d$.

Choose “store sparsely” or “store densely” on a chunk-by-chunk basis.
Normalization Affects Storage Costs

Worst case: all blocks sparse, with \(0 < \rho < \frac{1}{d/2+1}\).

Best case: each block has \(\rho = 1\) or \(\rho = 0\).

Lemma 1: there are cubes where normalization can turn worst cases into best cases. Example above isn’t quite one!
Optimal Normalization Problem

**Given:** $d$-dimensional cube $C$, chunk sizes in each dimension $(m_1, m_2, \ldots, m_d)$

**Output:** normalization $\varpi$ that minimizes storage cost $H(\varpi(C))$

“Code assignment affects chunked storage efficiency”, observed by Deshpande et al., SIGMOD’98.

Sensible heuristic: let dimension’s hierarchy guide you.

Issue apparently never addressed in depth after this (?)
Complexity

Consider the “decision problem” version that adds storage bound $K$. Asks “Is there a normalization $\pi$ with $H(\pi(C)) \leq K$?”

**Theorem 1.** The decision problem for Optimal Normalization is NP-complete, even for $d = 2$ and $m_1 = 1$ and $m_2 = 3$.

Proved by reduction from Exact-3-Cover.
Volume-2 Blocks

There is an efficient algorithm when $\prod_{i=1}^{d} m_i = 2$.

**Theorem 2.** For blocks of size $1 \times \ldots \times 1 \times 2 \times 1 \ldots \times 1$, the best normalization can be computed in $O(n_k \times (n_1 \times n_2 \times \ldots \times n_d) + n_k^3)$ time.

Algorithm relies on a cubic-time weighted-matching algorithm.

Probably can be improved, so time depends on $\#C$, not $\prod_{i=1}^{d} n_i$. 
Volume-2 Algorithm

Here, optimal orderings for vertical dimension include $A, B, C, D$ and $C, D, B, A$. 
Heuristics

Tested many heuristics. Two more noteworthy:

- Iterated Matching (IM). Applies the volume-2 algorithm to each dimension in turn, getting blocks of size $2 \times 2 \times 2 \ldots \times 2$. Not optimal.

- Frequency Sort (FS). \( \gamma_i \) orders dimension \( i \) values by descending frequency.
Frequency Sort (Results)
Independence and Frequency Sort

Frequency Sort (FS) is quickly computed. In our tests, it worked well. Traced to “much independence between dimensions”.

Result: we can quantify the dependence between the dimensions, get factor δ, where \(0 \leq \delta \leq 1\).

Small \(\delta \Rightarrow\) FS solution is nearly optimal.

Calculating \(\delta\) is easy. (In the paper, we used “IS”, where \(IS = 1 - \delta\).)
Relating $\delta$ to Frequency Sort Quality

FS is actually an approximation algorithm.

**Theorem 3.** *FS has an absolute error bound* $\delta(d/2 + 1)\#C$.

**Corollary.** *FS has relative error bound* $\delta(d/2 + 1)$.

E.g., for a 4-d cube with $\delta = .1$, FS solution is at most $.1 \times (4/2 + 1) = 30\%$ worse than optimal.
Experimental Results

Synthetic data does not seem appropriate for this work.
Got some large data sets from UCI’s KDD repository and elsewhere:

▷ Weather: 18-d, 1.1M facts, $\rho = 1.5 \times 10^{-30}$

▷ Forest: 11-d, 600k facts, $\rho = 2.4 \times 10^{-16}$

▷ Census: projected down to 18-d, 700k facts, also very sparse.

Seem too sparse by themselves.
**Test Data**

To get test data, randomly chose 50 cubes each of

- Weather datacube (5-d subsets)
- Forest datacube (3-d subsets)
- Census datacube (6-d subsets)

Most had $0.0001 \leq \rho \leq 0.2$ Also required that, if stored densely, had to fit in 100MB.
## Experimental Results

Compression relative to sparse storage (ROLAP):

<table>
<thead>
<tr>
<th>data sets</th>
<th>HOLAP chunked storage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>default normalization</td>
</tr>
<tr>
<td>Census</td>
<td>31%</td>
</tr>
<tr>
<td>Forest</td>
<td>31%</td>
</tr>
<tr>
<td>Weather</td>
<td>19%</td>
</tr>
</tbody>
</table>

FS did poorly on many Forest cubes.

Is an additional 10% compression helpful? Disastrous to ignore?

Hopefully, ↑ Yes  ↑ No
δ versus FS quality

FrequencySort’s solutions theoretically improve when δ ↓. Do we see this experimentally?

Yes. Problem: don’t know optimal. Substitute: try IM!
Conclusions/Summary

✔ Good normalization leads to useful space savings.

✔ Going for optimal normalization is too ambitious.

✔ FS is provably good when δ is low; experiments show bound seems pessimistic.

✔ Should help in a chunk-based OLAP engine being developed.
Questions??
Extra Slides
IS Preliminaries

Underlying probabilistic model: nonzero cube cells uniformly likely to be chosen.

For each dimension \( j \), get probability distribution \( \phi^j \)

\[
\phi^j_{\nu} = \frac{\text{nonzero cells with index } \nu \text{ in dimension } j}{\#C}
\]

If all \( \{\phi^j \mid j \in \{1, \ldots, d\}\} \) jointly independent:

\[
\Pr[C[i_1, i_2, \ldots, i_d] \neq 0] = \prod_{j=1}^{d} \phi^j_{i_j}
\]

and (claim) clearly FS gives an optimal algorithm.
\[ IS = \sum_{C[i_1, i_2, \ldots, i_d] \neq 0} \left( \prod_{j=1}^{d} \phi_{i_j}^j \right) \]

Roughly, \((1 - IS)\#C\) is the expected number of nonzero cells that, if we assume independence, we would mispredict as zero. At worst, such cells will have to be stored sparsely, at cost \((d/2 + 1)\) each.
Relating IS to Frequency Sort Quality

**Theorem 4.** Given cube $C$, let $\varnothing$ be an optimal normalization and $fs$ be a Frequency Sort normalization, then

$$H(fs(C)) - H(\varnothing(C)) \leq \left(\frac{d}{2} + 1\right) (1 - IS) \#C$$

where $H(\cdot)$ gives the storage cost of an cube.

Not even considering block dimensions; further improvements?
Exact 3-Cover (X3C)

[See Garey and Johnson, 1979]

**Given**: Set $S$ and a set $\mathcal{T}$ of three-element subsets of $S$.

**Question**: Is there a $\mathcal{T}' \subseteq \mathcal{T}$ such that each $s \in S$ occurs in exactly one member of $\mathcal{T}'$?

X3C is known to be NP-complete.
Transforming X3C to Optimal Normalization

Given an instance of X3C, make a \(|T| \times |S|\) cube. For \(s \in S\) and \(T \in \mathcal{T}\), the cube has an allocated cell corresponding to \((T, s) \leftrightarrow s \in T\).

Cube has \(3|\mathcal{T}|\) cells to be stored.

Can be stored for \(\leq 9|\mathcal{T}| - |S| \leftrightarrow\) the answer to the instance of X3C is “yes”.

Thus Optimal Normalization is NP-hard.
Example Transformation

**X3C:** $X = \{1, 2, 3, 4, 5, 6\}, \mathcal{T} = \{\{1, 2, 3\}, \{1, 2, 4\}, \{3, 5, 6\}\}$

**Optimal Normalization:** Blocks $3 \times 1$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>${1,2,3}$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>${1,2,4}$</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>${3,5,6}$</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

and set storage bound to 21.

Storage model: elements in full blocks cost 2 each, elements in non-full blocks cost 3 each.

Answer here is “yes”: swap columns 3 and 4.

6 elements in full blocks, 3 in nonfull blocks; $(6 \times 2 + 3 \times 3 = 21)$. 
IM is not optimal

\[
\begin{bmatrix}
1 & - & 1 & 1 \\
1 & - & - & - \\
1 & - & - & - \\
\end{bmatrix}
\]

This is optimal for $1 \times 2$ and $2 \times 1$ (storage cost 6)

but has cost 8 for $2 \times 2$ blocks, whereas

\[
\begin{bmatrix}
1 & 1 & - & 1 \\
1 & - & - & - \\
\end{bmatrix}
\] has cost 6 for $2 \times 2$ blocks.
Test Data

In OLAP, various aggregated views might be materialized. Group by of some subset of dimensions: is one cube in the overall datacube [Gray et al ’96]

To get test cases, randomly choose cubes from the datacube. (i.e., randomly select some subset of dimensions to get a test case).
Compression Relative to What?

What default normalization do we compare against?

Data sets were obtained “relationally” : lists of records, we scan sequentially.

Default normalization: code 0 for attribute value used in first record, code 1 goes to the next-seen attribute, etc.

“First seen, first numbered”.

Unused alternatives: sorted-as-strings, random, . . .
Index of Extra Slides

- more IS details
- BBT
- NP Completeness of 1x3
- Iterated Matching is Suboptimal
- Why cuboids from datacube
- Default normalization