Pruning Attribute Values From Data Cubes with Diamond Dicing

Hazel Webb and Owen Kaser
University of New Brunswick
hazel.webb@unb.ca, o.kaser@computer.org

Daniel Lemire
Université du Québec à Montréal
lemire@acm.org

ABSTRACT

Data stored in a data warehouse are inherently multidimensional, unlike most data-pruning techniques (such as iceberg and top-k queries). However, analysts need to issue multidimensional queries. For example, an analyst may need to select not just the most profitable stores or—separately—the most profitable products, but simultaneous sets of stores and products fulfilling some profitability constraints. To fill this need, we propose a new operator, the *diamond dice*. Because of the interaction between dimensions, the computation of diamonds is challenging. We present the first diamond-dicing experiments on large data sets. Our external memory algorithm avoids potentially expensive random accesses. Experiments show that we can compute diamond cubes over fact tables containing 100 million facts and 500,000 distinct attribute values in less than an hour using a single-core PC.

Topics: Access methods and data structures; data mining, OLAP and knowledge discovery; data warehousing; database query languages; databases and information retrieval

Keywords: Diamond cube, data warehouses, OLAP, algorithms, information retrieval, multidimensional queries

1. INTRODUCTION

In signal and image processing, software subsamples data [26] for visualization, compression, or analysis purposes: commonly, images are cropped to focus the attention on a segment. Researchers have proposed similar subsampling techniques in databases [3, 13], including iceberg queries [12, 25, 30] and top-k queries [20, 21]. Formally, subsampling is the selection of a subset of the data, often with desirable properties such as representativity, conciseness, or homogeneity. Of the subsampling techniques applicable to OLAP, only the dice operator focuses on reducing the number of attribute values without aggregation whilst retaining the original number of dimensions.

Such reduced representations are sometimes of critical importance to get good online performance in Business Intelligence (BI) applications [2,12]. Even when performance is not an issue, browsing and visualizing the data often benefit from reduced views [4].

IDEAS08 2008, September 10-12, Coimbra [Portugal]
Editor: Bipin C. DESAI

Table 1: Sales (in million dollars) with a 4,10 sum-diamond shaded: stores need to have sales above $10 million whereas product lines need sales above $4 million

	London	Montreal	Miami	Paris	Berlin
TV	3.4	0.9	0.1	0.9	2.0
Camcorder	0.1	1.4	3.1	2.3	2.1
Phone	0.2	8.4	2.1	4.5	0.1
Camera	0.4	2.7	6.3	4.6	3.5
Game console	3.2	0.3	0.3	2.1	1.5
DVD player	0.2	0.5	0.5	2.2	2.3

Often, business analysts are interested in distinguishing elements that are most crucial to their business, such as the k products jointly responsible for 50% of all sales, from the *long tail* [1]—the lesser elements. The computation of icebergs, top-k elements, or heavy-hitters has received much attention [7, 9, 8]. We wish to generalize this type of query so that interactions between dimensions are allowed. For example, a business analyst might want to compute a small set of stores and business hours jointly responsible for over 80% of the sales. In this new setting, the heads and tails of the distributions must be described using a multidimensional language; computationally, the queries become significantly more difficult. Hence, analysts often process dimensions one at a time: perhaps they would focus first on the most profitable business hours, and then aggregate sales per store, or perhaps they would find the must profitable stores and aggregate sales per hour. We propose a general model, of which the unidimensional analysis is a special case, that has acceptable computational costs and a theoretical foundation. In the two-dimensional case, our proposal is a generalization of ITERATIVE PRUNING [17], a graph-trawling approach used to analyze social networks. It also generalizes iceberg queries [12, 25, 30].

To illustrate our proposal in the BI context, consider the following example. Table 1 represents the sales of different items in different locations. Typical iceberg queries might be requests for stores having sales of at least 10 million dollars or product lines with sales of at least 4 million dollars. However, what if the analyst wants to apply both thresholds simultaneously? He might contemplate closing both some stores and some product lines. In our example, applying the constraint on stores would close Chicago, whereas applying the constraint on product lines would not terminate any product line. However, once the shop in Chicago is closed, we see that the product line TV must be terminated which causes the closure of the Berlin store and the termination of two new product lines (Game console and DVD player). This multidimensional pruning query selects a subset of attribute values from each dimension that are simultaneously important. There are many other similar applications:

- In Web traffic analysis, we could seek predominant traffic sources linking to pages benefiting substantially from these sources.
- We could provide a corporation with a list of services bringing in substantial revenues from large consumers of these same services.
- We could list all movies highly popular among individuals watching many of these movies.
- We could find products with sales above 10 k units during months where 100 k units of these products were sold, to customers who bought at least 5 k such units during the selected months.
- We could expand the ITERATIVE PRUNING algorithm so that it includes geography or text content in addition to the inbound/outbound link information currently considered.

The operation is a *diamond dice* [28] and produces a *diamond*, as formally defined in Section 3.

Other approaches that seek important attribute values, e.g. the Skyline operator [6,22], Dominant Relationship Analysis [19], and Top-k dominating queries [32], require dimension attribute values to be ordered, e.g. distance between a hotel and a conference venue, so that data points can be ordered. Our approach requires no such ordering.

2. NOTATION

Notation used in this paper is tabulated below.

C	a data cube
σ	aggregator COUNT or SUM
$C_{\text{dim},j}$	σ(slice j of dimension dim in cube C)
$\lvert C\rvert = \sum_j C_{1,j}$	the number of allocated cells in cube C
A, B	cubes
D_i	i^{th} dimension of a data cube
n_i	number of attribute values in dimension D_i
k	number of carats
k_i	number of carats of order 1 for D_i
d	number of dimensions
p	maximum number of attribute values per dim
p_i	maximum number of attribute values for D_i
$\kappa(C)$	maximum carats in C
COUNT-$\kappa(C)$	maximum carats in C, σ is COUNT

3. PROPERTIES OF DIAMOND CUBES

Given a database relation, a dimension D is the set of values associated with a single attribute. A cube C is the set of dimensions together with a map from some tuples in $D_1 \times \cdots \times D_d$ to real-valued measure values. Without losing generality, we shall assume that $n_1 \leq n_2 \leq \ldots \leq n_d$, where n_i is the number of distinct attribute values in dimension i.

A *slice* of order δ is the set of cells we obtain when we fix a single attribute value in each of δ different dimensions. For example, a slice of order 0 is the entire cube, a slice of order 1 is the more traditional definition of a slice and so on. For a d-dimensional cube, a slice of order d is a single cell. An aggregator is a function, σ, from arrays of values to the real numbers.

DEFINITION 1. *Let σ be an aggregator such as* SUM *or* COUNT, *and let k be some real-valued number. A cube has k* carats *over dimensions $i_1, \ldots, i_\delta$, if for every slice x of order δ along dimensions $i_1, \ldots, i_\delta$, we have $\sigma(x) \geq k$.*

We can recover iceberg cubes by seeking cubes having carats of order d where $\sigma(x)$ returns the measure corresponding to cell x. Requiring k SUM-carats results in the iceberg that contains cells whose measures are at least k.

We say that an aggregator σ is monotonically increasing if $S' \subset S$ implies $\sigma(S') \leq \sigma(S)$. Similarly, σ is monotonically decreasing if $S' \subset S$ implies $\sigma(S') \geq \sigma(S)$. Monotonically increasing operators include COUNT, MAX and SUM (over non-negative measures). Monotonically decreasing operators include MIN and SUM (over non-positive measures).

DEFINITION 2. *Let A and B be two cubes with the same dimensions and measures. Their union is denoted $A \cup B$. It is the set of attributes together with their measures, on each dimension, that appear in A, or B or both. The union of A and B is B if and only if A is contained in B: A is a subcube of B.*

PROPOSITION 1. *If the aggregator σ is monotonically increasing, then the union of any two cubes having k carats over dimensions $i_1, \ldots, i_\delta$ has k carats over dimensions $i_1, \ldots, i_\delta$ as well.*

Hence, as long as σ is monotonically increasing , there is a cube of maximal volume having k carats over dimensions $i_1, \ldots, i_\delta$, and we call such a cube the *diamond*. When σ is not monotonically increasing, there may not be a unique diamond. Indeed, consider the even-numbered rows and columns of the following matrix, then consider the odd-numbered rows and columns. Both are cubes of maximal volume with 2 carats (of order 1) under the SUM operator:

$$\begin{array}{rrrr} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{array}$$

Because we wish diamonds to be unique, we will require σ to be monotonically increasing. The next proposition shows that diamonds are themselves nested.

PROPOSITION 2. *The diamond having k' carats over dimensions $i_1, \ldots, i_\delta$ is contained in the diamond having k carats over dimensions $i_1, \ldots, i_\delta$ whenever $k' \geq k$.*

For simplicity, we only consider carats of order 1 for the rest of the paper. For example, a cube has 5 COUNT carats if for every *traditional* slice, x, we have COUNT$(x) \geq 5$.

We write that a cube has $k_1, k_2, \ldots, k_d$-carats if it has k_i carats over dimension D_i; when $k_1 = k_2 = \ldots = k_d = k$ we simply write that it has k carats.

One consequence of Proposition 2 is that the diamonds having various number of carats form a lattice (see Fig. 1) under the relation "is included in." This lattice creates optimization opportunities: if we are given the $2,1$-carat diamond X and the $1,2$-carat diamond Y, then we know that the $2,2$-carat diamond must lie in both X and Y. Likewise, if we have the $2,2$-carat diamond, then we know that its attribute values must be included in the diamond above it in the lattice (such as the $2,1$-carat diamond).

Given the size of a sum-based diamond cube (in cells), there is no upper bound on its number of carats. However, it cannot have more carats than the sum of its measures. Conversely, if a cube has dimension sizes $n_1, n_2, \ldots, n_d$ and k carats, then its sum is at least $k \max(n_1, n_2, \ldots, n_d)$.

Given the dimensions of a COUNT-based diamond cube, $n_1 \leq n_2 \leq \ldots \leq n_{d-1} \leq n_d$, an upper bound for the number of carats k of a subcube is $\prod_{i=1}^{d-1} n_i$. An upper bound on the number of carats k_i for dimension i is $\prod_{j=1, j\neq i}^{d} n_i$. An alternate (and trivial)

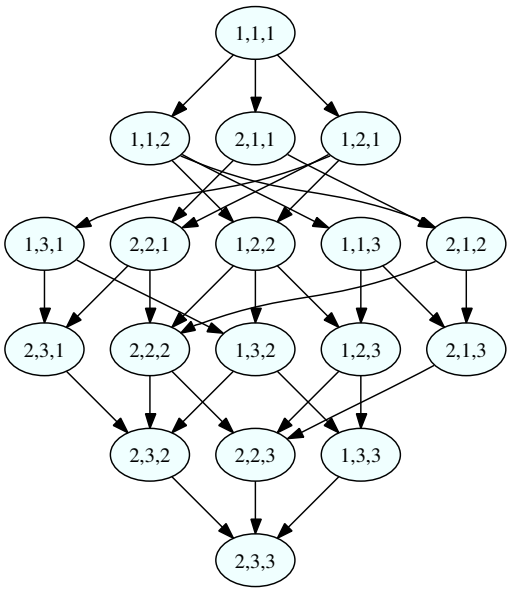

Figure 1: Part of the COUNT-based diamond-cube lattice of a $2 \times 2 \times 2$ cube

upper bound on the number of carats in any dimension is $|C|$, the number of allocated cells in the cube. For sparse cubes, this bound may be more useful.

Intuitively, a cube with many carats needs to have a large number of allocated cells: accordingly, the next proposition provides a lower bound on the size of the cube given the number of carats.

PROPOSITION 3. *For $d > 1$ and $\sigma =$ COUNT, the size S, or number of allocated cells, of a d-dimensional cube of k carats satisfies $S \geq k \max_{i \in \{1,2,\ldots,d\}} n_i \geq k^{d/(d-1)}$; more generally, a $k_1, k_2, \ldots, k_d$-carat cube has size S satisfying*

$$S \geq \max_{i \in \{1,2,\ldots,d\}} k_i n_i \geq (\prod_{i=1,\ldots,d} k_i)^{1/(d-1)}.$$

We calculate the *volume* of a cube C as $\prod_{i=1}^{i=d} n_i$ and its *density* is the ratio of allocated cells, $|C|$, to the volume $(|C|/\prod_{i=1}^{i=d} n_i)$. Given σ, its *carat-number*, $\kappa(C)$, is the largest number of carats for which the cube has a non-empty diamond. Intuitively, a small cube with many allocated cells should have a large $\kappa(C)$, and the following proposition makes this precise.

PROPOSITION 4. *For COUNT-based carats, we have $\kappa(C) \geq |C|/\sum_i (n_i - 1) - 3$.*

Many OLAP aggregators are distributive, algebraic and linear. An aggregator σ is *distributive* [15] if there is a function F such that for all $0 \leq k < n - 1$,

$$\sigma(a_0, \ldots, a_k, a_{k+1}, \ldots, a_{n-1}) = F(\sigma(a_0, \ldots, a_k), \sigma(a_{k+1}, \ldots, a_{n-1})).$$

An aggregator σ is *algebraic* if there is an intermediate tuple-valued **distributive** range-query function G from which σ can be computed. An algebraic example is AVERAGE: given the tuple (COUNT, SUM) one can compute AVERAGE by a ratio. In other words, if σ is an algebraic function then there must exist G and F such that

$$G(a_0, \ldots, a_k, a_{k+1}, \ldots, a_{n-1}) = F(G(a_0, \ldots, a_k), G(a_{k+1}, \ldots, a_{n-1})).$$

An algebraic aggregator σ is *linear* [18] if the corresponding intermediate query G satisfies

$$G(a_0 + \alpha d_0, \ldots, a_{n-1} + \alpha d_{n-1}) = G(a_0, \ldots, a_{n-1}) + \alpha G(d_0, \ldots, d_{n-1}).$$

for all arrays a, d, and constants α. SUM and COUNT are linear functions; MAX is not linear.

4. RELATED PROBLEMS

In this section, we discuss four problems, three of which are NP-hard, and show that the diamond—while perhaps not providing an exact solution—is a good starting point. The first two problems, Trawling the Web for Cyber-communities and Largest Perfect Subcube, assume use of the aggregator COUNT whilst for the remaining problems we assume SUM.

4.1 Trawling the Web for Cyber-communities

In 1999, Kumar et al. [17] introduced the ITERATIVE PRUNING algorithm for discovering emerging communities on the Web. They model the Web as a directed graph and seek large dense bipartite subgraphs or cores, and therefore their problem is a 2-D version of our problem. Although their paper has been widely cited [27, 31], to our knowledge, we are the first to propose a multidimensional extension to their problem suitable for use in more than two dimensions and to provide a formal analysis.

4.2 Largest Perfect Cube

A *perfect* cube contains no empty cells, and thus it is a diamond. Finding the largest perfect diamond is NP-hard. A motivation for this problem is found in Formal Concept Analysis [14], for example.

PROPOSITION 5. *Finding a perfect subcube with largest volume is NP-hard, even in 2-D.*

Finding a diamond might be part of a sensible heuristic to solve this problem, as the next lemma suggests.

LEMMA 1. *For COUNT-based carats, a perfect subcube of size $n_1 \times n_2 \times \ldots \times n_d$ is contained in the $\prod_{i=1}^{d} n_i / \max_i n_i$-carat diamond and in the $k_1, k_2, \ldots, k_d$-carat diamond where $k_i = \prod_{j=1}^{d} n_j / n_i$.*

This helps in two ways: if there is a nontrivial diamond of the specified size, we can search for the perfect subcube within it; however, if there is only an empty diamond of the specified size, there is no perfect subcube.

4.3 Heaviest Cube with Limited Dimensions

In the OLAP context, given a cube, a user may ask to "find a subcube with 10 attribute values per dimension." Meanwhile, he may want the resulting subcube to have maximal average—he is, perhaps, looking for the 10 attributes from each dimension that, in combination, give the greatest profit. Note that this problem does not restrict the number of attribute values (p) to be the same for each dimension.

We call this problem the HEAVIEST CUBE WITH LIMITED DIMENSIONS (HCLD), which we formalize as: pick $\min(n_i, p_i)$ attribute values for dimension D_i, for all i's, so that the resulting

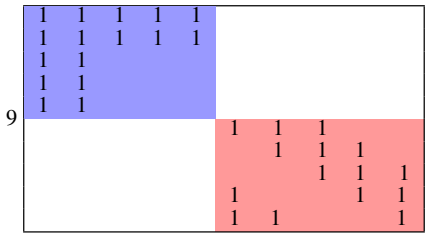

Figure 2: Example showing that a diamond (bottom-right quadrant) may not have optimal density.

subcube has maximal average. We have that the HCLD must intersect with diamonds.

THEOREM 1. *Using the* SUM *operator, a cube without any* $k_1, k_2, \ldots, k_d$*-carat subcube has sum less than* $\sum_{i=1}^{d}(n_i+1)k_i + \max(k_1, k_2, \ldots, k_d)$ *where the cube has size* $n_1 \times n_2 \times \ldots \times n_d$.

COROLLARY 1. *Any solution to the HCLD problem having average greater than*

$$\frac{\sum_{i=1}^{d}(n_i+1)k_i + \max(k_1, k_2, \ldots, k_d)}{\prod_{i=1}^{d} n_i}$$

must intersect with the $k_1, k_2, \ldots, k_d$*-sum-carat diamond.*

The solution to a special case of this problem, where all measures have the value 1, is the DENSEST CUBE WITH LIMITED DIMENSIONS. We might hope that when the dimensions of the diamond coincide with the required dimensions of the densest cube, we would have a solution to the DCLD problem. Alas, this is not true. Consider the 2-D cube in Fig. 2. The bottom-right quadrant forms the largest 3-carat subcube. In the bottom-right quadrant, there are 15 allocated cells whereas in the upper-left quadrant there are 16 allocated cells. This proves the next result.

LEMMA 2. *Even if a* COUNT*-based diamond has exactly* $\min(n_i, p_i)$ *attribute values for dimension* D_i*, for all* i*'s, it may still not be a solution to the DCLD problem.*

We are interested in large data sets; the next theorem shows that solving DCLD and HCLD is difficult.

THEOREM 2. *The DCLD and HCLD problems are NP-hard.*

5. ALGORITHM

We have developed and implemented an algorithm for computing diamonds. Its overall approach is illustrated by Example 1. That approach is to repeatedly identify an attribute value that cannot be in the diamond, and then (possibly not immediately) remove the attribute value and its slice. The identification of "bad" attribute values is done conservatively, in that they are known already to have a sum less than required (σ is sum), or insufficient allocated cells (σ is count). When the algorithm terminates, we are left with only attribute values that meet the condition in every slice: a diamond.

EXAMPLE 1. *Suppose we seek a 4,10-carat diamond in Table 1 using Algorithm 1. On a first pass, we can delete the attribute values "Chicago" and "TV" because their respective slices have sums below 10 and 4. On a second pass, value "Berlin," "Game console" and "DVD" can be removed because the sums of their slices were reduced by the removal of the values "Chicago" and "TV." The algorithm then terminates.*

Algorithms based on this approach will always terminate, though they might sometimes return an empty cube. The correctness of our algorithm is guaranteed by the following result.

THEOREM 3. *Algorithm 1 is correct, that is, it always returns the* $k_1, k_2, \ldots, k_d$*-carat diamond.*

For simplicity of exposition, in the rest of the paper, we assume that the number of carats is the same for all dimensions.

input: file `inFile` containing $d-$dimensional cube C, integer $k > 0$
output: the diamond data cube
`// preprocessing scan computes` σ `values for each slice`
foreach *dimension* i **do**
  Create hash table $\texttt{ht}_i$
  foreach *attribute value* v *in dimension* i **do**
    if σ(*slice for value* v *of dimension* i *in* C) $\geq k$ **then**
      $\texttt{ht}_i(v) = \sigma$(slice for value v of dimension i in C)
    end
  end
end
`so there appears to be a hash table that has some specific values from a dimension`
`stable` $\leftarrow$ **false**
while $\neg$`stable` **do**
  Create new output file `outFile` `// iterate main loop`
  `stable` $\leftarrow$ **true**
  foreach *row* r *of* `inFile` **do**
    $(v_1, v_2, \ldots, v_d) \leftarrow r$
    if $v_i \in \text{dom}\,\texttt{ht}_i$*, for all* $1 \leq i \leq d$ **then**
      write r to `outFile`
    else
      for $j \in \{1, \ldots, i-1, i+1, \ldots, d\}$ **do**
        if $\texttt{v}_j \in \text{dom}\,\texttt{ht}_j$ **then**
          $\texttt{ht}_j(v_j) = \texttt{ht}_j(v_j) - \sigma(\{r\})$
          if $\texttt{ht}_j(v_j) < k$ **then**
            remove v_j from $\text{dom}\,\texttt{ht}_j$
          end
        end
      end
      `stable` $\leftarrow$ **false**
    end
  end
  if $\neg$`stable` **then**
    `inFile` $\leftarrow$ `outFile` `// prepare for another iteration`
  end
end
return `outFile`

Algorithm 1: Diamond dicing for relationally stored cubes. Each iteration, less data is processed.

Our algorithm employs a preprocessing step that iterates over the input file creating d hash tables that map attributes to their σ-values. When σ = COUNT, the σ-values for each dimension form a histogram, which might be precomputed in a DBMS.

These values can be updated quickly as long as σ is linear: aggregators like SUM and COUNT are good candidates. If the cardinality of any of the dimensions is such that hash tables cannot be stored in main memory, then a file-based set of hash tables could be constructed. However, given a d-dimensional cube, there are only $\sum_{i=1}^{d} n_i$ slices and so the memory usage is $O(\sum_{i=1}^{d} n_i)$: for our tests, main memory hash tables suffice.

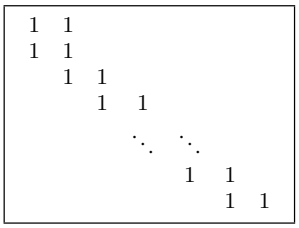

Figure 3: An $n \times n$ cube with $2n$ allocated cells (each indicated by a 1) and a 2-carat diamond in the upper left: it is a difficult case for an iterative algorithm.

Algorithm 1 reads and writes the files sequentially from and to disk and does not require potentially expensive random access, making it a candidate for a data parallel implementation in the future.

Let I be the number of iterations through the input file till convergence; ie no more deletions are done. Value I is data dependent and (by Fig. 3) is $\Theta(\sum_i n_i)$ in the worst case.In practice, we do not expect I to be nearly so large, and working with our largest "real world" data sets we never found I to exceed 100.

Algorithm 1 runs in time $O(Id|C|)$; each attribute value is deleted at most once. In many cases, the input file decreases substantially in the first few iterations and those cubes will be processed faster than this bound suggests. The more carats we seek, the faster the file will decrease initially.

The speed of convergence of Algorithm 1 and indeed the size of an eventual diamond may depend on the data-distribution skew. Cell allocation in data cubes is very skewed and frequently follows Zipfian distributions [23]. Suppose the number of allocated cells $C_{\texttt{dim},i}$ in a given slice i follows a Zipfian distribution: $P(C_{\texttt{dim},i} = j) \propto j^{-s}$ for $s > 1$. The parameter s is indicative of the skew. We then have that $P(C_{\texttt{dim},i} < k_i) = \sum_{j=1}^{k_i-1} j^{-s} / \sum_{j=1}^{\infty} j^{-s} = P_{k_i,s}$. The expected number of slices marked for deletion after one pass of over all dimensions using $\sigma = \text{COUNT}$, prior to any slice deletion, is thus $\sum_{i=1}^{d} n_i P_{k_i,s}$. This quantity grows fast to $\sum_{i=1}^{d} n_i$ (all slices marked for deletion) as s grows (see Fig. 4). For SUM-based diamonds, we not only have the skew of the cell allocation, but also the skew of the measures to accelerate convergence. In other words, we expect Algorithm 1 to converge quickly over real data sets, but more slowly over synthetic cubes generated using uniform distributions.

5.1 Finding the Largest Number of Carats

The determination of $\kappa(C)$, the largest value of k for which C has a non-trivial diamond, is a special case of the computation of the diamond-cube lattice (see Proposition 2). Identifying $\kappa(C)$ may help guide analysis. Two approaches have been identified:

1. Assume $\sigma = \text{COUNT}$. Set the parameter k to 1 + the lower bound (provided by Proposition 4 or Theorem 1) and check whether there is a diamond with k carats. Repeat, incrementing k, until an empty cube results. At each step, Proposition 2 says we can start from the cube from the previous iteration, rather than from C. When σ is SUM, there are two additional complications. First, the value of k can grow large if measure values are large. Furthermore, if some measures are not integers, the result need not be an integer (hence we would compute $\lfloor \kappa(C) \rfloor$ by applying this method, and not $\kappa(C)$).

2. Assume $\sigma = \text{COUNT}$. Observe that $\kappa(C)$ is in a finite interval. We have a lower bound from Proposition 4 or Theorem 1 and an upper bound $\prod_{i=1}^{d-1} n_i$ or $|C|$. (If this upper bound is unreasonably large, we could start with the lower bound and repeatedly double it.) Execute the diamond-dicing algorithm and set k to a value determined by a binary search over its valid range. Every time we test a new midpoint k, we can begin the computation from the largest nonempty k-diamond that has been seen so far. (See Proposition 2 and note that Algorithm 1 does not destroy its input.) If σ is SUM and measures are not integer values, it might be difficult to know when the binary search has converged exactly.

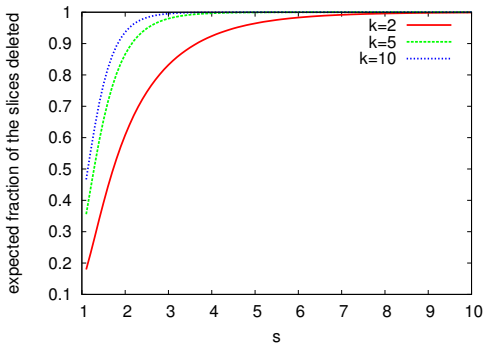

Figure 4: Expected fraction of slices marked for deletion after one pass under a Zipfian distribution for various values of the skew parameter s.

We believe the second approach is better. Let us compare one iteration of the first approach (which begins with a k-carat diamond and seeks a $k+1$-carat diamond) and a comparable iteration of the second approach (which begins with a k-carat diamond and seeks a $(k + k_{\text{upper}})/2$-carat diamond). Both will end up making at least one scan, and probably several more, through the k-carat diamond. Now, we experimentally observe that k values that slightly exceed $\kappa(C)$ tend to lead to several times more scans through the cube than with other values of k. Our first approach will make only one such unsuccessful attempt, whereas the binary search would typically make several unsuccessful attempts while narrowing in on $\kappa(C)$. Nevertheless, we believe the fewer attempts will far outweigh this effect. We recommend binary search, given that it will find $\kappa(C)$ in $O(\log \kappa(C))$ iterations.

If one is willing to accept an approximate answer for $\kappa(C)$ when aggregating with SUM, a similar approach can be used.

5.2 Diamond-Based Heuristic for DCLD

In Section 4.3, we noted that a diamond with the appropriate shape will not necessarily solve the DCLD problem. Nevertheless, when we examined many small random cubes, the solutions typically coincided. Therefore, we suggest diamond dicing as a heuristic for DCLD.

A heuristic for DCLD can start with a diamond and then refine its shape. Our heuristic first finds a diamond that is only somewhat too large, then removes slices until the desired shape is obtained. See Algorithm 2.

6. EXPERIMENTS

We wish to show that diamonds can be computed efficiently. We also want to review experimentally some of the properties of diamonds including their density (count-based diamonds) and the range of values the carats may take in practice. Finally, we want to

input: d-dimensional cube C, integers $p_1, p_2, \ldots p_d$
output: Cube with size $p_1 \times p_2 \times \ldots \times p_d$
// Use binary search to find k
Find max k where the k-carat diamond Δ has shape $p'_1 \times p'_2 \times \ldots \times p'_d$, where $\forall i.p'_i \geq p_i$
for $i \leftarrow 1$ *to* d **do**
  Sort slices of dimension i of Δ by their σ values
  Retain only the top p_i slices and discard the remainder from Δ
end
return Δ

Algorithm 2: DCLD heuristic that starts from a diamond.

provide some evidence that diamond dicing can serve as the basis for a DCLD heuristic.

6.1 Data Sets

We experimented with diamond dicing on several different data sets, some of whose properties are laid out in Tables 5 and 6.

Cubes TW1, TW2 and TW3 were extracted from TWEED [11], which contains over 11,000 records of events related to internal terrorism in 16 countries in Western Europe between 1950 and 2004. Of the 52 dimensions in the TWEED data, 37 were measures since they decomposed the number of people killed/injured into all the affected groups. Cardinalities of the dimensions ranged from 3 to 284. Cube TW1 retained dimensions Country, Year, Action and Target with cardinalities of $16 \times 53 \times 11 \times 11$. For cubes TW2 and TW3 all dimensions not deemed measures were retained. A detailed description of how each cube was extracted and its cardinalities can be found in our technical report [29].

We also processed the Netflix data set [24], which has dimensions: MovieID $\times$ UserID $\times$ Date $\times$ Rating ($17766 \times 480189 \times 2182 \times 5$). Each row in the fact table has a distinct pair of values (MovieID, UserID). We extracted two 3-D cubes NF1 and NF2 both with about 10^8 allocated cells using dimensions MovieID, UserID and Date. For NF2 we use Rating as the measure and the SUM aggregator, whereas NF1 uses the COUNT aggregator. The Netflix data set is the largest openly available movie-rating database (≈ 2 GiB).

Our third real data set, Census-Income, comes from the UCI KDD Archive [16]. The cardinalities of the dimensions ranged from 2 to 91 and there were 199,523 records. We extracted 27 dimensions from the original 41 and used two measures, income from stocks and hourly wage. Full details of the dimensions retained and rolled-up can be found in our technical report [29].

We also generated synthetic data. As has already been stated, cell allocation in data cubes is skewed. We modelled this by generating values in each dimension that followed a power distribution. The values in dimension i were generated as $\lfloor n_i u^{1/a} \rfloor$ where $u \in [0, 1]$ is a uniform distribution. For $a = 1$, this function generates uniformly distributed values. The dimensions are statistically independent. We picked the first 250,000 distinct facts. Since cubes S2A and S3A were generated with close to 250,000 distinct facts we decided to keep them all.

The cardinalities for all synthetic cubes are laid out in Table 7. All experiments on our synthetic data were done using the measure COUNT.

All experiments were carried out on a Linux-based (Ubuntu 7.04) dual-processor machine with Intel Xeon (single core) 2.8 GHz processors with 2 GiB RAM. The disk, a Seagate Cheetah ST373453LC (SCSI 320, 15 kRPM, 68 GiB), was formatted to the ext3 filesystem. Our implementation was done with Sun's SDK 1.6.0 and to handle the large hash tables generated when processing Netflix, we set the maximum heap size for the JVM to 2 GiB.

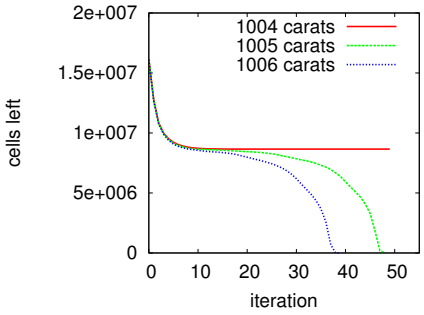

Figure 8: Cells remaining after each iteration of Algorithm 1 on NF1, computing a 1004-, 1005- and 1006-carat diamonds.

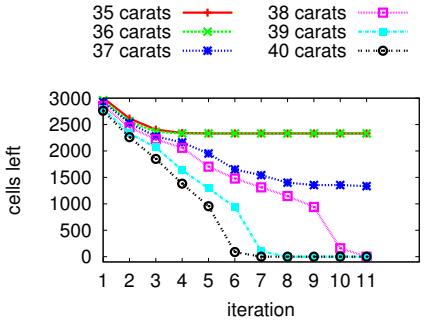

Figure 9: Cells remaining after each iteration, TW2

6.2 Iterations to Convergence

Algorithm 1 required 19 iterations and an average of 35 minutes to compute the 1004-carat κ-diamond for NF1. However it took 50 iterations and an average of 60 minutes to determine that there was no 1005-carat diamond. The preprocessing time for NF1 was 22 minutes. For a comparison, sorting lexicographically the Netflix comma-separated data file took 29 minutes using the Unix utility sort. Times were averaged over 10 runs. Fig. 8 shows the number of cells present in the diamond after each iteration for 1004–1006 carats. The curve for 1006 reaches zero first, followed by that for 1005. Since $\kappa(\text{NF1}) = 1004$, that curve stabilizes at a nonzero value. We see a similar result for TW2 in Fig. 9 where κ is 37. It takes longer to reach a critical point when k only slightly exceeds κ.

As stated in Section 5, the number of iterations required until convergence for all our real and synthetic cubes was far fewer than the upper bound, e.g. cube S2B: 2,195 (upper bound) and 12 (actual). We had expected to see the uniformly distributed data taking longer to converge than the skewed data. This was not the case. It may be that a clearer difference would be apparent on larger synthetic data sets. This will be investigated in future experiments.

6.3 Largest Carats

According to Proposition 4, COUNT-κ(NF1) $\geq$ 197. Experimentally, we determined that it was 1004. By the definition of the carat, it means we can extract a subset of the Netflix data set where each user entered at least 1004 ratings on movies rated at least 1004 times by these same users during days where there were

Figure 5: Real data sets used in experiments

	TWEED			Netflix		Census income	
cube	TW1	TW2	TW3	NF1	NF2	C1	C2
dimensions	4	15	15	3	3	28	28
$\|C\|$	1957	4963	4963	100,478,158	100,478,158	196054	196054
$\sum_{i=1}^{d} n_i$	88	674	674	500,137	500,137	533	533
measure	count	count	killed	count	rating	stocks	wage
iters to converge	6	10	3	19	40	6	4
κ	38	37	85	1,004	3,483	99,999	9,999

Figure 6: Synthetic data sets used in experiments

cube	S1A	S1B	S1C	S2A	S2B	S2C	S3A	S3B	S3C
dimensions	4	4	4	8	8	8	16	16	16
skew factor	0.02	0.2	1.0	0.02	0.2	1.0	0.02	0.2	1.0
$\|C\|$	250k	250k	250k	251k	250k	250k	262k	250k	250k
$\sum_{i=1}^{d} n_i$	11,106	11,098	11,110	22,003	22,195	22,220	38,354	44,379	44,440
iters to converge	12	9	2	6	12	12	8	21	6
κ	135	121	30	133	32	18	119	8	15

Figure 7: Dimensional cardinalities for our synthetic data cubes

Cube	Dimensional cardinalities
S1A	$6 \times 100 \times 1000 \times 10000$
S1B	$2 \times 100 \times 1000 \times 9996$
S1C	$10 \times 100 \times 1000 \times 10000$
S2A	$10 \times 100 \times 1000 \times 9881 \times 10 \times 100 \times 1000 \times 9902$
S2B	$10 \times 100 \times 1000 \times 9987 \times 10 \times 100 \times 1000 \times 9988$
S2C	$10 \times 100 \times 1000 \times 10000 \times 10 \times 100 \times 1000 \times 10000$
S3A	$10 \times 100 \times 1000 \times 8465 \times 10 \times 100 \times 1000 \times 8480$ $\times 10 \times 100 \times 1000 \times 8502 \times 10 \times 100 \times 1000 \times 8467$
S3B	$10 \times 100 \times 1000 \times 9982 \times 10 \times 100 \times 1000 \times 9987$ $\times 10 \times 100 \times 1000 \times 9988 \times 10 \times 100 \times 1000 \times 9982$
S3C	$10 \times 100 \times 1000 \times 10000 \times 10 \times 100 \times 1000 \times 10000$ $\times 10 \times 100 \times 1000 \times 10000 \times 10 \times 100 \times 1000 \times 10000$

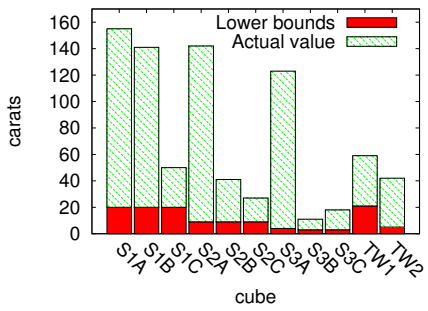

Figure 10: Comparison between estimated κ, based on the lower bounds from Proposition 4, and number of (COUNT-based) carats found.

at least 1004 ratings by these same users on these same movies. The 1004-carat diamond had dimensions $3082 \times 6833 \times 1351$ and 8,654,370 cells, for a density of about 3×10^{-4} or two orders of magnitude denser than the original cube. The presence of such a large diamond was surprising to us. We believe nothing similar has been observed about the Netflix data set before [5].

Comparing the two methods in Section 5.1, we see that sequential search would try 809 values of k before identifying κ. However, binary search would try 14 values of k (although 3 are between 1005 and 1010, where perhaps double or triple the normal number of iterations are required). To test the time difference for the two methods, we used cube TW1. We executed a binary search, repeatedly doubling our lower bound to obtain the upper limit until we established the range where κ must exist. Whenever we exceeded κ, a copy of the original data was used for the next step. Even with this copying step and the unnecessary recomputation from the original data, the time for binary search averaged only 2.75 seconds. Whereas a sequential search, that started with the lower bound and increased k by one, averaged 9.854 seconds over ten runs.

Fig. 10 shows our lower bounds on κ, given the dimensions and numbers of allocated cells in each cube, compared with their actual κ values. The plot indicates that our lower bounds are further away from actual values as the skew of the cube increases for the synthetic cubes. Also, we are further away from κ for TW2, a cube with 15 dimensions, than for TW1.

For uniformly-distributed cubes S1C, S2C and S3C there was no real difference in density between the cube and its diamond. However, all other diamonds experienced an increase of between 5 and 9 orders of magnitude.

Diamonds found in C1, C2, NF2 and TW3 captured .25%, .09%, 66.8% and 0.6% of the overall sum for each cube respectively. The very small fraction captured by the diamond for TW3 can be explained by the fact that κ(TW3) is based on a diamond that has only one cell, a bombing in Bologna in 1980 that killed 85 people. Similarly, the diamond for C2 also comprised a single cell.

6.4 Effectiveness of DCLD Heuristic

To test the effectiveness of our diamond-based DCLD heuristic (Subsection 5.2), we used cube TW1 and set the parameter p to 5. We were able to establish quickly that the 38-carat diamond was the closest to satisfying this constraint. It had density of 0.169 and cardinalities of $15 \times 7 \times 5 \times 8$ for the attribute values; year, country, action and target. The solution we generated to this DCLD ($p = 5$) problem had exactly 5 attribute values per dimension and density of 0.286.

Since the DCLD problem is NP-complete, determining the quality of the heuristic poses difficulties. We are not aware of any known approximation algorithms and it seems difficult to formulate a suitably fast exact solution by, for instance, branch and bound. Therefore, we also implemented a second computationally expensive heuristic, in hope of finding a high-quality solution with which to compare our diamond-based heuristic. This heuristic is based on local search from an intuitively reasonable starting state. (A greedy steepest-descent approach is used; states are $\langle A_1, A_2, \ldots, A_d \rangle$, where $|A_i| = p_i$, and the local neighbourhood of such a state is $\langle A'_1, A'_2, \ldots, A'_d \rangle$, where $A_i = A'_i$ except for one value of i, where $|A_i \cap A'_i| = p_i - 1$. The starting state consists of the most frequent p_i values from each dimension i. Our implemention actually requires the i^{th} local move be chosen along dimension $i \bmod d$, although if no such move brings improvement, no move is made.)

The density reported by this second heuristic was 0.283, a similar outcome, but at the expense of more work. Our diamond-based heuristic, starting with the 38-carat diamond, required a total of 15 deletes. Whereas our expensive comparision heuristic, starting with its $5 \times 5 \times 5 \times 5$ subcube, required 1420 inserts/deletes.

Our diamond heuristic might indeed be a useful starting point for a solution to the DCLD problem.

7. CONCLUSION AND FUTURE WORK

We introduced the diamond dice, a new OLAP operator that dices on all dimensions simultaneously. This new operation represents a multidimensional generalization of the iceberg query and can be used by analysts to discover sets of attribute values jointly satisfying multidimensional constraints.

We have shown that the problem is tractable. We were able to process the 2 GiB Netflix data with 500,000 distinct attribute values and 100 million cells in about 35 minutes, excluding preprocessing. As expected from the theory, real-world data sets have a fast convergence using Algorithm 1: the first few iterations quickly prune most of the false candidates. We have identified potential strategies to improve the performance further. First, we might selectively materialize elements of the diamond-cube lattice (see Proposition 2). The computation of selected components of the diamond-cube lattice also opens up several optimization opportunities. Second, we believe we can use ideas from the implementation of ITERATIVE PRUNING proposed by Kumar et al. [17]. Third, Algorithm 1 is suitable for parallelization [10]. Also, our current implementation uses only Java's standard libraries and treats all attribute values as strings. We believe optimizations can be made by the preprocessing step that will greatly reduce overall running time.

We presented theoretical and empirical evidence that a non-trivial, single, dense chunk can be discovered using the diamond dice and that it provides a sensible heuristic for solving the DENSEST CUBE WITH LIMITED DIMENSIONS. The diamonds are typically much denser than the original cube. Over moderate cubes, we saw an increase of the density by one order of magnitude, whereas for a large cube (Netflix) we saw an increase by two orders of magnitude and more dramatic increases for the synthetic cubes. Even though Lemma 2 states that diamonds do not necessarily have optimal density given their shape, informal experiments suggest that they do with high probability. This may indicate that we can bound the sub-optimality, at least in the average case; further study is needed.

We have shown that sum-based diamonds are no harder to compute than count-based diamonds and we plan to continue working towards an efficient solution for the HEAVIEST CUBE WITH LIMITED DIMENSIONS (HCLD).

8. REFERENCES

[1] C. Anderson. *The long tail*. Hyperion, 2006.

[2] K. Aouiche, D. Lemire, and R. Godin. Collaborative OLAP with tag clouds: Web 2.0 OLAP formalism and experimental evaluation. In *WEBIST'08*, 2008.

[3] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD'03*, pages 539–550, 2003.

[4] R. Ben Messaoud, O. Boussaid, and S. Loudcher Rabaséda. Efficient multidimensional data representations based on multiple correspondence analysis. In *KDD'06*, pages 662–667, 2006.

[5] J. Bennett and S. Lanning. The Netflix prize. In *KDD Cup and Workshop 2007*, 2007.

[6] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE '01*, pages 421–430. IEEE Computer Society, 2001.

[7] M. J. Carey and D. Kossmann. On saying "enough already!" in SQL. In *SIGMOD'97*, pages 219–230, 1997.

[8] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Diamond in the rough: finding hierarchical heavy hitters in multi-dimensional data. In *SIGMOD '04*, pages 155–166, New York, NY, USA, 2004. ACM Press.

[9] G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1):249–278, 2005.

[10] F. B. Dehne, T. B. Eavis, and A. B. Rau-Chaplin. The cgmCUBE project: Optimizing parallel data cube generation for ROLAP. *Distributed and Parallel Databases*, 19(1):29–62, 2006.

[11] J. O. Engene. Five decades of terrorism in Europe: The TWEED dataset. *Journal of Peace Research*, 44(1):109–121, 2007.

[12] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In *VLDB'98*, pages 299–310, 1998.

[13] V. Ganti, M. L. Lee, and R. Ramakrishnan. ICICLES: Self-tuning samples for approximate query answering. In *VLDB'00*, pages 176–187, 2000.

[14] R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on Galois (concept) lattices. *Computational Intelligence*, 11:246–267, 1995.

[15] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *ICDE '96*, pages 152–159, 1996.

[16] S. Hettich and S. D. Bay. The UCI KDD archive. `http://kdd.ics.uci.edu` (checked 2008-04-28), 2000.

[17] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. In *WWW '99*, pages 1481–1493, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[18] D. Lemire and O. Kaser. Hierarchical bin buffering: Online local moments for dynamic external memory arrays. *ACM Trans. Algorithms*, 4(1):1–31, 2008.

[19] C. Li, B. C. Ooi, A. K. H. Tung, and S. Wang. DADA: a data cube for dominant relationship analysis. In *SIGMOD'06*, pages 659–670, 2006.

[20] Z. X. Loh, T. W. Ling, C. H. Ang, and S. Y. Lee. Adaptive method for range top-k queries in OLAP data cubes. In *DEXA'02*, pages 648–657, 2002.

[21] Z. X. Loh, T. W. Ling, C. H. Ang, and S. Y. Lee. Analysis of pre-computed partition top method for range top-k queries in OLAP data cubes. In *CIKM'02*, pages 60–67, 2002.

[22] M. D. Morse, J. M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *VLDB*, pages 267–278, 2007.

[23] T. P. E. Nadeau and T. J. E. Teorey. A Pareto model for OLAP view size estimation. *Information Systems Frontiers*, 5(2):137–147, 2003.

[24] Netflix, Inc. Nexflix prize. `http://www.netflixprize.com` (checked 2008-04-28), 2007.

[25] J. Pei, M. Cho, and D. Cheung. Cross table cubing: Mining iceberg cubes from data warehouses. In *SDM'05*, 2005.

[26] D. N. Politis, J. P. Romano, and M. Wolf. *Subsampling*. Springer, 1999.

[27] P. K. Reddy and M. Kitsuregawa. An approach to relate the web communities through bipartite graphs. In *WISE'01*, pages 302–310, 2001.

[28] H. Webb. Properties and applications of diamond cubes. In *ICSOFT 2007 – Doctoral Consortium*, 2007.

[29] H. Webb, O. Kaser, and D. Lemire. Pruning attribute values from data cubes with diamond dicing. Technical Report TR-08-011, Computer Science and Applied Statistics, University of New Brunswick Saint John, 2008. available from `http://http://arxiv.org/abs/0805.0747`.

[30] D. Xin, J. Han, X. Li, and B. W. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In *VLDB*, pages 476–487, 2003.

[31] K. Yang. Information retrieval on the web. *Annual Review of Information Science and Technology*, 39:33–81, 2005.

[32] M. L. Yiu and N. Mamoulis. Efficient processing of top-k dominating queries on multi-dimensional data. In *VLDB'07*, pages 483–494, 2007.