# Summary of "Pruning Attribute Values from Data Cubes with Diamond Dicing"

Eduardo Gutarra

2011-03-29

Webb et al. propose a new operator, the diamond dice, which allows analysts to issue multi-dimensional queries with simultaneous constraints on the data. The diamond dice operation is a multi-dimensional generalization of the Iterative pruning graph-trawling algorithm which is used to analyze social networks. The diamond dice operation produces a diamond. The diamond is a sample of the data that meets the constraints on all dimensions simultaneously. They call the constraints on the dimensions the carats of the cube.

Researchers have proposed other similar sub sampling techniques, among which are iceberg queries and top-$k$ queries. These techniques can eliminate attribute values from dimensions by aggregating through different groupings; thus, these techniques can also decrease the number of dimensions. The diamond dice, however, eliminates attribute values from dimensions without aggregating and maintains the original number of dimensions. In fact, the dice operation is the only OLAP operation that reduces the number of attribute values without reducing the number of dimensions.

Webb et al. consider that a cube has $k$-carats over a number of dimensions when for every slice $x$ in every dimension the cube has an aggregation of values greater than or equal to $k$. Moreover, a diamond of $k$-carats is the largest cube with $k$-carats possible over a given number of dimensions. The authors introduce certain properties that can be found in diamond cubes, among which are the following:

A first property is that when the aggregator function is monotonically increasing or decreasing, the union of 2 cubes with $k$-carats will produce a larger cube of $k$-carats. A monotonically increasing aggregator function implies that if set $S'$ is contained in the set S, then the aggregation of the elements in $S \geq S'$. Similarly, the monotonically decreasing function has that if set $S'$ is contained in the set $S$, then the aggregation of the elements in $S \leq$S′. When the aggregator function is not monotonically increasing or decreasing, finding the diamond of a cube is NP-Hard, because when uniting cubes of $k$-carats we are not guaranteed that we will obtain a larger $k$-carat cube; thus when uniting cubes with non-monotonic functions, the number of carats in the new cube can change.

A second property is that a diamond cube having $k$' carats over a given set of dimensions is contained in a diamond having $k$ carats over those same dimensions, when $k' \geq k$. For example, the diamond that contains one carat contains all diamonds of higher carat values

such as 2, 3, and 4. Because the higher carat diamonds are contained in lower carat diamonds, they can form a lattice that organizes what diamonds may be derived from a given diamond. This lattice allows for possible optimization techniques for deriving diamonds from other diamonds.

The authors implemented an algorithm that computes diamonds from a given d-dimensional cube. The algorithm builds a hash table that is used to keep track of the values of the aggregator function over a given slice of the cube. This hash table maps the aggregation of values for a given slice to the dimension's slice value. A hash table is kept for each dimension. When certain slice values are not included in the hash table, it makes dependent slices that were previously valid, invalid, as they are unable to meet the threshold. The algorithm checks for these changes by repeatedly eliminating invalid slices, until no new slices are eliminated on each of the different dimensions.

The authors also address some problems that can be approached by finding diamonds in cubes. One such problem is finding the largest value of $k$ for which a non-empty $k$-carat diamond exists. They present two approaches. The first approach is to iteratively traverse the lattice deriving higher karat diamonds from previous lower karat diamonds until an empty diamond is found; thus the last value of $k$ for which a non-empty diamond is found is the largest $k$. The second approach consists in using binary search to guess the largest $k$ value given a higher and lower bound. Unlike the first approach, the second approach can encounter an empty diamond more than once. This approach progressively uses binary search to approach the largest $k$ value, presumably in fewer iterations than the first approach.

Webb et al. also consider using the diamond dice operation as a heuristic for finding perfect cubes. Perfect cubes are cubes that have all cells allocated for a given $k$-value. The diamond dice operation may also be used to estimate what the Densest Cube with Limited Dimensions (DCLD) and the Heaviest Cube with Limited Dimensions (HCLD) might be for a given cube. The DCLD and HCLD cubes are sub-cubes of a given cube that have the highest aggregation of values possible. The difference between them is that for the DCLD problem the aggregator function used is COUNT, and for the HCLD problem SUM is used.

Webb et al. performed experiments to illustrate that diamonds can be computed efficiently and to review experimentally some of the properties of diamonds, including the range of values the carats may take as well as some of their properties. They experimented with several different datasets. In their experiments on the Netflix dataset, they attempted to find the largest number of carats for which a diamond exists. They found it to be 1004, and the diamond produced was two orders of magnitude denser than the original cube. They found, however, that it took considerably more time to find that there was no 1005-carat or 1006-carat diamond. They also found that it took more iterations to find the empty 1005-carat diamond, than it did to find the 1006-carat empty diamond. This effect is due to the faster elimination of slices as the number of carats sought increases. A similar result is seen in a second experiment where they also illustrate the number of iterations needed to compute diamonds with the following number of carats: 35, 36, 37, 38, 39 and 40. They found that the number of iterations until convergence for the different diamonds varied. Computing diamonds with fewer than 37 carats took fewer iterations for convergence than for a 37-carat

diamond. Yet still, some diamonds with more than 37 carats also took fewer iterations for convergence than the 37-carat diamond.

Finally, they conclude with a DCLD heuristic to calculate the densest cube, starting with a 38-carat diamond. They compare this solution against a second heuristic based on a greedy algorithm. This algorithm consists in a local search from an intuitively reasonable starting state. The density reported by the second heuristic was of 0.283 as opposed to 0.286 reported by the DCLD heuristic. Moreover, the DCLD heuristic required a total of 15 deletes, as opposed to 1420 insert/deletes required in their more greedy-algorithm-based heuristic. Thus, they determine that the diamond heuristic might be a useful starting point for solving the DCLD problem.