Optimal Path Finding Algorithm For Massive Queries In Lattice of Cuboids

Prashant R and Eashwaran R Maharaja Surajmal Institute of Technology, GGSIPU, New delhi-1100058, India B.Tech (2017-21)	Suman Mann Maharaja Surajmal Institute of Technology, GGSIPU, New delhi-1100058, India HOD(IT dept),MSIT

Abstract

Multidimensional analysis requires the computation of many aggregate functions over a large volume of collected data. To provide the various viewpoints for the analysts, these data are organized as a multi-dimensional data model called data cubes. Each cell in a data cube represents a unique set of values for the different dimensions and contains the metrics of interest. The different abstraction and concretization associated with a dimension may be represented as a lattice. The focus is to move up and drill down within the lattice using an algorithm with optimal space and computation. In the lattice of cuboids, there exist multiple paths for summarization from a lower to an upper level of the cuboid. The alternate paths involve different amounts of storage space and different volume of computations. The objective of this paper is to design an algorithm that can answer queries of path finding in O(n) time with a preprocessing of O(nlog(n)).

1. Introduction

A data warehouse (DW) is a repository of integrated information available for querying and analysis. The information in the data warehouse is stored in the form of the multidimensional model. The multidimensional model viewdata in the form of the data cube. Data cube computes the aggregates along all possible combinations of dimensions. It is defined by dimensions and facts. Dimensions are the entities with respect to which an organization wants to keep records [4]. Facts are the numerical measures/ quantities by which we want to analyze the relationship between dimensions. In general terms, we consider data cube as 3-D geometric structures, but in data warehousing, it is n-dimensional. The data cube is a metaphor for multidimensional data storage. Each cell of datacube shows a specific view in which users are interested. Given a set of dimensions, we can generate a cuboid for each of the possible subsets of the given dimensions which result in a lattice of the cuboid. Figure 1 shows a lattice of cuboid for the dimensions of time, product, market and supplier. For the number of dimensions, we may find 2n cuboid and the main challenge is to understand how the cuboids are related to each other[3]. As shown in figure1 different paths are available for a particular cuboid. These paths consider the different amount of storage space and time computation. In this paper, an algorithm is proposed for computing the cuboid that minimizes the storage space and computation. The algorithm is proposed based on lowest first for selection of the optimal path. The proposed algorithm gives the optimal solution in terms of space and time computation.

2. Definition and Properties of Larrice Theory

Some important definitions related to lattice are given below:
1. Lattice: A partial order set (POS) (A, ≤) is called a lattice if, x, y ε A, there exist sup(x, y) and inf (x, y) [27]. For supermum we use the symbol and for infimum we use the symbol .In the lattice theory, these operations are called binary operations. 2. POS: Partially ordered set is a set with a binary relation ≤ that, for any x, y, and z, satisfies the following conditions: (1) x ≤ x (reflexivity); (2) if x ≤ y and y ≤ x, then x = y (anti_symmetry); (3) if x ≤ y and y ≤ z, then x ≤ z (transitivity).
If, for a POS (A, ≤), x, y ε A: x ≤ y or y ≤ x, then this set is called a linearly ordered set, or chain.
3. Supremum or Least Upper Bound: Let (A, R) be a POS. An element l ε A is called supremum of a and b in A if and only if i. aRl and bRl i.e. l is the upper bound of a and b.
ii. If an element l' ε A such that aRl' and bRl'then lRl'
That is if l' is another upper bound of a and b then l' is also the upper bound of l. Thus l is the least upper bound of a and b. 4. Greatest Lower Bound(GLB) or Infimum Let (A, R) be a POS. An element l ε A is called infimum of a and b in A if and only if i. lRa and lRb i.e. l is the lower bound of a and b. ii. If there exist one element l' ε A such that l' is also a lower bound of a and b then l'Rl that is if l'is also a lower bound of a, b then l' is the lower bound of l also i.e. l is the greatest of all lower bounds of a and b. That is if l' is another upper bound of a and b then l' is also the upper bound of l. Thus l is the least upper bound of a and b. 5. Roll Up & Drill Down: The roll-up operation performs aggregation on a data cube, may be climbing up a concept hierarchy for a dimension or by dimension reduction. Drilldown navigates from less detailed data to more detailed data. It is the reverse of roll-up operation. Drill down is realized by stepping down a concept hierarchy or adding a new dimension.

2.1 Some Characteristics of Lattices

If A is any lattice, then for any a, b, c ε L the following properties hold:

- $\overbrace{(a \cup a = a) \wedge (a \cap a = a)}^{Idempotent\ Law}$
- $\overbrace{(a \cup (b \cup c) = (a \cup b) \cup c) \wedge (a \cap (b \cap c) = (a \cap b)) \cap c}^{Associative\ Law}$
- $\overbrace{(a \cup b = b \cup a) \wedge (a \cap b = b \cap a)}^{Commutative\ Law}$
- $\overbrace{(a \cap (a \cup b) = a) \wedge (a \cup (a \cap b) = a)}^{Absorption\ Law}$

3. Lowest First Approach

The idea is to head towards the locally optimal solution by visiting cuboid that have all the attributes of current cuboid plus one more which has lowest value among the valid and unvisited attributes where being valid means they are present in the target cuboid and continuing until we reach the target. In Further sections we will prove using principle of mathematical induction how the local optima leads us to the global optima. The algorithm has a preprocessing complexity of O(nlog(n)) and query time complexity of O(n) where n is the number of total attributes.

Consider a lattice of cuboids having a total of 6 attributes Q, P, M, S, U given to us that 0<=Q <=P<=S <=U <=M (values belong to set of natural number) and we have to find optimal path from <Q> to <Q, P , S ,U , M>, Now a path is optimal if the intermediate cuboids generated while traversal occupy minimum space and take minimum processing time so we write a generic expression for path value and try to optimize it, which is for the current case $A_1*A_2*A_3 + A_1*A_2 + A_1$ and now we need to find a bijection between the variables in our generic expression and the sorted difference set [P,S,U,M] which gives us the optimal value for the expression. To minimize

the whole expression what we do is minimize each term in the expression while making sure it stays valid, So starting from the last term we have

· A_1 is minimum when its value is P. So, A_1 = P (as P has lowest value in our difference set followed by S, U and M)

· A_1*A_2 is minimum when its value is P*S and we already have A1 =P so A2 = S.

· A1 *A2* A3 is minimum when its value is P*S*U and we already have A1 =P, A2=S so A3=U and our optimal assignment order is A1 =P, A2=S, A3=U and the path becomes <Q> => <Q,P> => <Q,P,S> => <Q,P,S,U> => <Q,P,S,U,M> which gives us lowest possible space and time complexity for any possible path from source cuboid to target cuboid.

4. Proof using principle of Mathematical Induction

We propose that for any natural number n if the values of attributes are in order $A_1<=A_2<=A_3<=A_4$ ... <= A_n then the correct bijection which gives minimum value for the expression $B_1*B_2*B_3*B_4*...*B_{n-1}*B_n + ... + B_1*B_2*B_3 + B_1*B_2 + B_1$ is when $B_1=A_1$, $B_2=A_2$ $B_3=A_3$, ... $B_{n-1}=A_{n-1}$, $B_n=A_n$,.

Base Case (n=1)
for n=1 we have only one attribute, ie A_1 and the expression becomes A_1 which has no other possible variation hence it is the minimum possible, base case is proved hence.

Inductive Step.

Fix k ≥ 1, and suppose that the following holds(assumption), that is for n=k the minimum value of expression $B_1*B_2*B_3*B_4*...*B_{k-1}*B_k + B_1*B_2*B_3*B_4*...*B_{k-1} + ... + B_1*B_2*B_3 + B_1*B_2 + B_1$ is when $B_1=A_1$, $B_2=A_2$, $B_3=A_3$,, ..., $B_{k-1}=A_{k-1}$, $B_k=A_k$.

Now It remains to show that for n=k+1 our proposal holds, So according to what we proposed minimum value of expression $B_1*B_2*B_3*B_4*...*B_k*B_{k+1} + B_1*B_2*B_3*B_4*...*B_{k-1}*B_k + B_1*B_2*B_3*B_4*...*B_{k-1} + ... + B_1*B_2*B_3 + B_1*B_2 + B_1$ is when $B_1=A_1$, $B_2=A_2$, $B_3=A_3$,, ..., $B_{k-1}=A_{k-1}$, $B_k=A_k$,$B_{k+1}=A_{k+1}$

Now moving on to proving if this is true, we know that from our inductive assumption for n=k that expression containing terms till n=k are minimum possible and they are common in expression for n=k+1, So now we just need to show the $k+1^{th}$ term is minimum it can be, then the whole n=k+1 expression will become optimal, and that can done So, we have n non negative integer variables in order $0<=A_1<=A_2<=A_3<=A_4$ ... <= A_n and if for some k+1<=n we have to select k+1 variables such that their products is minimum then it is obviously to select the least k+1 variables as they would result in least value when multiplied, So the $k+1^{th}$ term in our expression comes out to be $B_1*B_2*B_3*B_4*...*B_k*B_{k+1}$ with $B_{k+1}=A_{k+1}$ which proves that it holds for n=k+1 also and hence using principle of mathematical induction we have proved our Hypothesis.

5. Algorithm

I) Preprocessing

1) User input the values and names of all attributes.

2) Check if they are valid.

3) Create an array named sorted_attr by sorting the attributes given by the user based on the values of those attributes in ascending order.

II) Query

1) User input for the source and target cuboids.

2) Check for their validity using boundary checks.

3) Create an array named difference_set containing attributes that arent present in source cuboid but are present in target cuboid(these are the attributes that would create possible paths from source to target)

4) Create Hash map named hash_diff_set of the elements of difference_set to reduce random access complexity down to O(1)

5) Iterate over the sorted_attr containing all the attributes in sorted order by values, and for each element check if hash_diff_set[sorted_attr[i]] is present, if its present then push that element to a new array named sorted_diff_set so that after iterating over sorted_attr we get sorted_diff_set containing elements of difference_set in O(n) time complexity.

6) Now the path from source to target is starting from source and then to cuboid containing attributes of source plus first element of sorted_diff_set, from there to cuboid containing first 2 elements of sorted_diff_set and so on till we finally reach target cuboid.

6. Implementation and example

Let us consider a lattice of cuboids having 4 attributes <P,O,I,U>, having values 20, 5, 10, 30 respectively and we have two path finding queries A)from <P> to <P,O,I,U>, B) from <P> to <P,O,U>

Steps:

A) Query #1 <P> to <P,O,I,U>

1) Sorting all the attributes we get [O,I,P,U]

2) Our difference_set contains attributes present in target but not in source ie, [O,I,U]

3) Sorting it using hash maps and previously sorted attribute array we get sorted_diff_set to be [O,I,U].

4) Now starting from <P>(source) we first visit <P,O>(first element of sorted_diff_set) then from <P,O> we visit <P,O,I> and then from <P,O,I> we visit <P,O,I,U> which is our target. And this optimal path matches with the result of brute-force algorithm having O(2^n) worst-case time complexity.

B) Query #1 <P> to <P,O,U>

1) Sorting all the attributes we get [O,I,P,U]

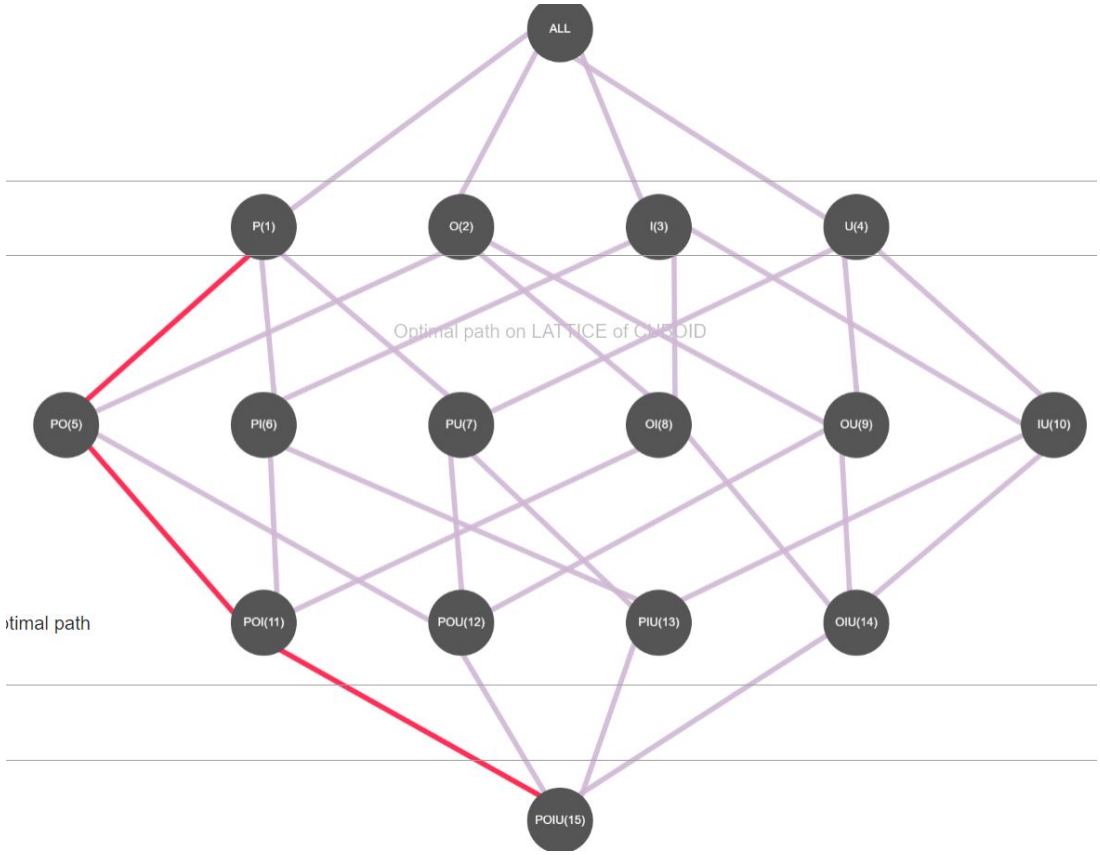
2) Our difference_set contains attributes present in target but not in source ie, [O,U]

3) Sorting it using hash maps and previously sorted attribute array we get sorted_diff_set to be [O,U].

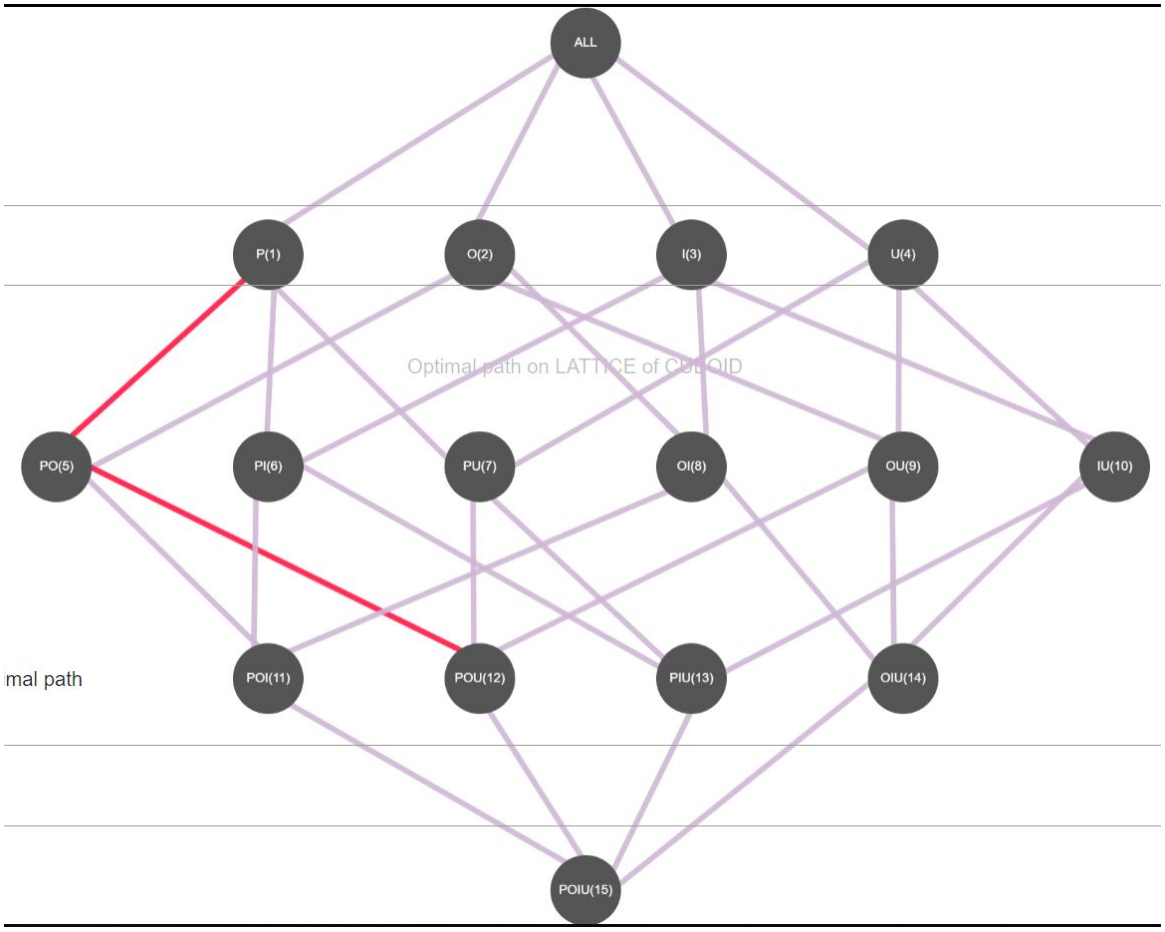
4) Now starting from <P>(source) we first visit <P,O>(first element of sorted_diff_set) then from <P,O> we visit <P,O,U> which is our target. And this optimal path also matches with the result of brute-force algorithm having O(2^n) worst-case time complexity.

Below are the optimal paths highlighted for both Queries A and B, generated by a web implementation of the above algorithm available at https://latticeofcuboid.ml/ . The time complexity was O(nlog(n)) for the first preprocessing thereafter it is O(n) per Query which is 2 in our case.

Query A)



Query B)



7. Conclusion

In this paper we have proposed an algorithm which has a preprocessing time complexity of O(nlogn) and query time complexity of O(n) So it can handle massive queries to find optimal path in lattice of cuboids.