HKU CSIS Tech Report TR-99-04

Requirement-Based Data Cube Schema Design

David W. Cheung[†] Bo Zhou[††] Ben Kao[†]
Hongjun Lu[‡] Tak Wah Lam[†] Hing Fung Ting [†]

[†] Department of Computer Science and Information Systems, The University of Hong Kong, Hong Kong.
[††] Department of Computer Science and Engineering, Zhejiang University, Hangzhou, China.
[‡] Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong.
email: {dcheung,bzhou,kao,twlam,hfting}@csis.hku.hk, luhj@cs.ust.hk

Abstract

On-line analytical processing (OLAP) requires efficient processing of complex decision support queries over very large databases. It is well accepted that pre-computed data cubes can help reduce the response time of such queries dramatically. A very important design issue of an efficient OLAP system is therefore the choice of the right data cubes to materialize. We call this problem the *data cube schema design problem*. In this paper we show that the problem of finding an optimal data cube schema for an OLAP system with limited memory is NP-hard. As a more computationally efficient alternative, we propose a greedy approximation algorithm cMP and its variants. Algorithm cMP consists of two phases. In the first phase, an initial schema consisting of all the cubes required to efficiently answer the user queries is formed. In the second phase, cubes in the initial schema are selectively merged to satisfy the memory constraint. We show that cMP is very effective in prunning the search space for an optimal schema. This leads to a highly efficient algorithm. We report the efficiency and the effectiveness of cMP via an empirical study using the TPC-D benchmark. Our results show that the data cube schemas generated by cMP enable very efficient OLAP query processing.

Keywords: Data cubes, Data cube schema design, OLAP, DSS.

1 Introduction

With wide acceptance of the data warehousing technology, corporations are building their decision support systems (DSS) on large data warehouses. Many of these DSS's have on-line characteristics and are termed On-line Analytical Processing (OLAP) systems. Different from the conventional database applications, a DSS usually needs to analyze accumulative information, e.g., the total sales in a particular region within a given period of time. Very often, the system needs to scan almost the entire database to compute query answers, resulting in a very poor response time. Conventional database techniques are simply not fast enough for today's corporate decision process.

The *data cube* technology has been becoming a core component of many OLAP systems. Data cubes are pre-computed multi-dimensional views of the data in a data warehouse [6]. The advantage of a data cube system is that once the data cubes are built, answers to decision support queries can be retrieved from the cubes in real-time.

An OLAP system can be modeled by a **three-level architecture** that consists of: (1) a query client; (2) a data cube engine; and (3) a data warehouse server. Figure 1 shows such an architecture.

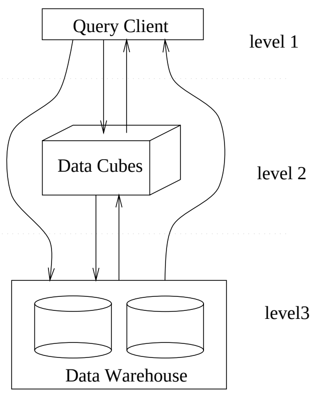

Figure 1: Three-level architecture of an OLAP system

The bottom level of an OLAP system is a data warehouse built on top of a DBMS. Data in the warehouse comes from source operational databases. (In the simplest case, the data warehouse could be the DBMS itself.) The warehouse needs to support fast aggregations, for example, by means of different indexing techniques such as bit-map indices and join indices [11, 12].

The middle level of an OLAP system is a set of data cubes, generated from the data warehouse. These cubes are called *base data cubes*. Each base cube is defined by a set of attributes taken from the warehouse schema. It contains the aggregates over the selected set of attributes. Other aggregates can be computed from the base cubes. For example, if a base cube is defined on the attributes `<part, customer, date>`, then the aggregates grouped by `part` and `year` can be derived. The set of base data cubes together define a **data cube schema** for the OLAP system.

The top level of an OLAP system is a query client. The client, besides supporting DSS queries, allows users to browse through the data it caches from the data cubes. Therefore, a query could be a very complicated DSS query or a simple slicing and dicing request. A query submitted to the query client, after being checked against the data cube schema, will be directed to the cube level if it can be answered by the data cubes there; otherwise, the query is passed on to the warehouse where the result is computed. Since data cubes store pre-computed results, servicing queries with the cubes is much faster than with the warehouse.

Various developments and research studies have been made on the design of the three levels in an OLAP system. Many commercial products are also now available. Some example query clients include Seagate Info Worksheet [13] and Microsoft PivotTable Services [10]. Currently, these query client products mainly provide browsing and report generation services on cached data. In general, unless the answer is already cached, complex DSS queries submitted to the client module will have to be compiled into accesses to the data cubes or to the warehouse. For the warehouse and the data cube levels, there are products like Microsoft SQL OLAP Services [10], Hyperion Essbase OLAP Server, and IBM DB2 OLAP Server [8]. At the warehouse level, many vendors have been enhancing their DBMS products to improve the performance on data aggregation [3, 7]. As for the data cube level, most of the research studies focus on two issues: (1) how to compute aggregates from a base cube efficiently, and (2) what data structures should be used to represent the cubes. For the first issue, studies have been done comparing sorting-based techniques with hashing-based techniques [1]; For the second issue, there is the debate on choosing between a relational OLAP (ROLAP) representation or a multi-dimensional OLAP (MOLAP) representation [1, 16].

As we have mentioned, the OLAP system would be able to support real-time responses if the cube level can intercept (or answer) all the queries. Unfortunately, materializing all possible cubes so that all possible queries can be answered by the cubes is clearly impractical due to the high storage and maintenance costs. Instead, one should carefully choose the right combination of cubes so that query response time is optimized subject to the constraints of the system's capacity (such as storage space). We call the set of materialized base cubes the *data cube schema* of the OLAP system. We also call the problem of selecting a data cube schema the *data cube schema design problem.*

The key to the design of a query-efficient OLAP system thus lies on the design of a good data cube schema. In particular, two very important questions one needs to address are: *on what basis shall we design such a schema?* And *where should the schema be derived from?* We claim that the data cube schema should not be based solely on the database schema in the warehouse. Instead, *a practical approach to the cube design problem should be based on the users' query requirements.* For example, in the TPC-D benchmark [15], the *requirement* is to answer the 17 DSS queries that are specified in the benchmark efficiently. This is because these queries presumably are driven from the applications that use the data warehouse most often. Given the user query requirements (i.e., a set of frequently asked queries) and a set of system capacity constraints (e.g., storage limitation), our goal is to derive a data cube schema that optimizes query response time without violating the system capacity constraints.

As we will see in Section 2.5, we prove that the optimization problem is NP-hard. Finding the optimal data cube schema is thus computationally very expensive. As an alternative, we propose an efficient greedy approximation algorithm cMP for the requirement-based data cube schema design problem. Our algorithm consists of two phases:

(1) Define an initial schema

The first phase is to derive an initial set of data cubes from the application requirements. In this study, we assume that the requirements are captured by a set of *frequently-asked queries* or FAQs. The initial set of data cubes are selected such that all the FAQs can be answered directly and efficiently. In the TPC-D example, we can define a cube to answer each one of the 17 DSS queries. For example, the 7th query of the TPC-D benchmark involve three attributes: `supp_nation`, `cust_nation`, and `shipdate_yr`. The query can be answered efficiently by materialing the base cube that consists of those three attributes. We call the schema that consists of the set of cubes derived from the FAQs the *initial schema.*

(2) Schema Optimization

The second phase is to modify the initial schema so that query response time is optimized subject to the system's capacity constraints. the data cubes derived in an initial schema may have lots of redundancy caused by overlapping attributes. The total size of the cubes may exceed the storage or memory limitation of the system. Too many cubes would always induce a large maintenance cost when the data in the underlying warehouse changes. Therefore, it may be more cost-effective to merge some of cubes. Cube merging may result in fewer but perhaps larger cubes. In terms of query response time, query processed using the merged cubes will in general be slower than using the original cubes. Hence, there is a trade-off between query performance and cube maintenance. *Schema optimization is to determine a set of data cubes that replace some of the cubes in the initial schema such that the query performance on the resulted cubes is optimal under the constraint that the total size of the resulted cubes is within an acceptable system limit.*[1] The set of data cubes obtained from the optimization process is called the *optimal schema* for the OLAP system.

Note that the cubes in the initial schema produce the best query performance. However, since these cubes are derived from the FAQs, they are numerous, causing a high maintenance cost. In the TPC-D example, we would have to manage 17 data cubes. We believe that reducing the number and hence the total size of the cubes in the schema will have a significant impact on the applicability and the cost of an OLAP system.

The rest of the paper is organized as follows. In Section 2 we present a formal definition of the schema optimization problem. Section 3 introduces the greedy algorithm cMP for schema optimization. A performance study of cMP is presented in Section 4. We use data from the TPC-D benchmark in the study. Finally, we conclude our paper in Section 5.

2 Schema Optimization

In our framework, data cube schema design is a two-phase process. It involves the design of an initial data cube schema followed by an optimization exercise. Again, one possible way to capture the requirements in a DSS is to define a set of frequent queries. The initial schema can then be derived from these queries. In this section, we demonstrate this design process by an example of deriving a data cube from a TPC-D query. Also, we define formally the schema optimization problem.

2.1 Deriving a data cube from a DSS query

A data cube which provides an answer efficiently to a query can be derived by analyzing the syntax and semantic of the query. As an example, the 7th query of the TPC-D benchmark is shown in Figure 2.

The query finds, for two given nations, `nation1` and `nation2`, the gross discounted revenues derived from `lineitems` in which parts were shipped from a supplier in either nation to a customer in the other nation during 1995 and 1996. To answer this query, one

[1]It is reasonable to correlate the maintenance cost with the total size of the cubes in the schema.

```
select supp_nation, cust_nation, year, sum(volume) as revenue
from (
    select n1.n_name as supp_nation,
            n2.n_name as cust_nation,
            extract(year from l_shipdate) as year,
            l_extendedprice × (1-l_discount) as volume
    from supplier, lineitem, order, customer, nation n1, nation n2
    where s_suppkey = l_suppkey
        and o_orderkey = l_orderkey
        and c_custkey = o_custkey
        and s_nationkey = n1.nationkey
        and c_nationkey = n2.nationkey
        and ( (n1.n_name='[nation1]' and n2.n_name='[nation2]')
            or (n1.n_name='[nation2]' and n2.n_name='[nation1]'))
        and l_shipdate between date '1995-1-1' and '1996-12-31'
    ) as shipping
group by supp_nation, cust_nation, year
order by supp_nation, cust_nation, year;
```

Figure 2: The 7th query of TPC-D

needs to join 6 tables and has to go through almost the entire `lineitem` table. The long response time makes it impossible to process the query on-line at the data warehouse level. To answer the query at the data cube level, one can build the 3-dimensional data cube: [`supp_nation`,`cust_nation`,`shipdate_yr`].[2] Using the data cube, the query can be answered easily: only 4 data points need to be retrieved from the cube, i.e., the 4 combinations of two nations and two `shipdate_yr`'s.

2.2 Search space of an optimal schema

In this paper we assume that the requirements of the OLAP system is captured in a set of frequent queries. We use Q to denote the initial schema of the data cubes derived from the queries. The second phase of our cube design process is to refine the set Q so that the maintenance cost (such as storage) of the cube set is within the capacity of the system. We can consider the refinement as an optimization problem with the set of all possible cube sets as the search space.

To simplify the problem, we assume that the database in the data warehouse is represented by a star schema [2]. Attributes in the queries come from the fields of the dimensions and fact tables. Usually, the number of dimensions and fact tables is not large, e.g., there are only 2 fact tables and 6 dimension tables in the TPC-D database. However, there may be many attributes in one table. For example, the table `part` includes the attributes `p_partkey, p_brand, p_type, p_size, p_container`, etc. The dimension is in fact a multi-hierarchical dimension

[2]We use $[d_1, d_2, \cdots, d_n]$ to represent a data cube, where the d_i's are the dimensions of the cube.

as shown in Figure 3. In addition, in the star schema, some attributes are stored directly in

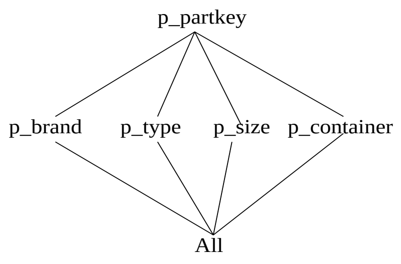

Figure 3: The multi-hierarchical structure of the `part` dimension in TPC-D

the fact table. For example, the attributes `l_shipdate, l_commitdate, l_receiptdate` in the fact table `lineitem` are such attributes. As a result, the number of attributes (dimensions) needs to be considered in a data cube design is much more than the number of dimension tables. In TPC-D, 33 attributes need to be considered.

In [9], the notion of a *composite lattice* is used to integrate multi-hierarchical dimensions with the lattice of aggregates in a data cube. Assume that $A = \{a_1, a_2, \cdots, a_n\}$ is the set of all attributes on which query can be posted. Any subset of A can be used as the dimension attributes to construct a data cube. The composite lattice $L = (\mathcal{P}(A), \prec)$ is the lattice of data cubes constructed from all the subsets of A. ($\mathcal{P}(A)$ is the power set of A.) The cube associated with the set A is the root of the lattice L. For two different cubes c_1, $c_2 \in L$, the *derived from* relationship, $c_1 \prec c_2$, holds if c_1 can be derived from c_2 by aggregation. For example the cube c_1 = [`part, year`] can be derived from c_2 = [`part, customer, date`]. The lattice L is the search space of the optimization problem. As has been mentioned, n is large in general. For example, in the TPC-D benchmark, $n = 33$. Thus, the search space L of the optimization problem is enormous.

2.3 Schema optimization

Given an initial data cube schema Q, a search space L, and a maintenance cost bound *LIM*, the schema optimization problem is defined in Table 1.

Objective:	Find $C \subset L$ such that $Cost(Q, C)$ is minimal
Constraint:	$\forall q \in Q,\ \exists c \in C,\ such\ that\ q \prec c$ and $MC(C) \leq LIM$

Table 1: Schema Optimization Problem

The objective is to find a cube set C such that the cost of answering the frequent queries, $Cost(Q, C)$ is the smallest. The constraint states that any frequent query q can be answered by some cube c in C and that the total maintenance cost $MC(C)$ of the cube set is smaller than the system limit *LIM*. We will discuss various measures of *Cost* and *MC* shortly.

For simplicity, we assume that the frequent queries are equally probable. Since each cube in the initial schema Q is derived from one distinct frequent query, we use the same symbol q to denote both a cube in the initial schema and its associated frequent query. Since we do not want to make any assumption on the implementation of the cubes and the structure of the queries, a good measure of $Cost(Q, C)$ is the linear cost model suggested in [9]. In that model, if $q \prec c$, then the cost of computing the answer for a query q using a cube c is linearly proportional to the number of data points in c. We use $S(c)$ to denote the number of data points in c. For each query $q \in Q$, we use $F_C(q)$ to denote the smallest cube in C that answers q. Formally,

$$F_C(q) \textit{ is a cube in } C \textit{ such that } q \prec F_C(q) \textit{ and } \forall x \in C, \textit{ if } q \prec x, \textit{ then } S(F_C(q)) \leq S(x). \quad (2.1)$$

We now define $Cost(Q, C)$ by:

$$Cost(Q, C) = \sum_{q \in Q} (S(F_C(q))). \quad (2.2)$$

Maintaining a data cube requires disk storage and CPU computation. Without assuming any implementation method, two measures can be used to estimate the maintenance cost $MC(C)$ of a cube set.

- $MC_1(C) = |C|$, i.e., the number of cubes in C. With this cost function, the bound *LIM* is expressed as the maximum number of cubes the OLAP system can materialized.
- $MC_2(C) = \sum_{c \in C} S(c)$, i.e., the total number of data points in the cubes. This is an estimate of the total disk storage required. The bound *LIM* is the maximum storage needed to maintain the cubes.

2.4 Related works

To the best of our knowledge, this paper is the first to explore the data cube schema design problem. Several papers have been published on data cube implementation. Cube selection algorithms have been proposed in [9, 14]. These cube selection algorithms assume that there is one root base cube c_0 which encompasses all the attributes in the queries. They also assume that some queries are associated with this root base cube c_0; therefore $c_0 \in Q$. Very different from the schema optimization problem, their selections always include c_0 in the answer, i.e., $c_0 \in C$. However, in a general DSS such as TPC-D, we do not anticipate many frequent queries that involve all the attributes; hence, our cube schema design problem is more general.

Cube selection algorithms start from a base cube and determine what cubes deducible from it should be implemented so that queries on the aggregates in the base cube can be answered efficiently. Tackling a very different problem, cube schema design tries to merge the cubes in an initial schema bottom-up to generate a set of cubes which provide an optimal query performance, while system capacity constraints are satisfied. The search space of the design problem is in general much larger because of the large numbers of attributes present in the initial schema. In short, cube selection algorithms are for cube implementation but not for cube schema design.

An interesting question is whether it is possible to modify the selection algorithm [9] to solve the schema design problem. One solution is to apply the selection algorithm on the *maximal cubes* of Q, i.e., those that cannot be deduced from any other cube in Q. In general, there are more than one maximal cubes in Q. In order to adopt the selection algorithm, we can include all the maximal cubes in the answer set C as the initial members, and then expand C by applying the selection algorithms on them. The expansion stops when the total size exceeds the storage bound *LIM*.

However, the above solution has a few undesirable problems. First, if the maximal cubes alone have already exceeded the maintenance bound, then the cube selection algorithm fails. Second, if some of the maximal cubes are highly correlated, e.g., with many overlapping attributes, then merging some of them could be beneficial and is sometimes even necessary. Selection algorithms, however, never merge cubes. For example, given a lattice $L = (\{A, B, C, \cdots\}, \prec)$, suppose both cubes $ABCD$ and $BCDE$ are maximal, and $S(ABCD) = S(BCDE) = S(ABCDE)$, then using a selection algorithm, both $ABCD$ and $BCDE$ are selected. However, replacing them by the cube $ABCDE$ decreases the maintenance cost without increasing the query cost. Hence, the selection algorithm is not always applicable to the cube schema design probelm.

2.5 Complexity of the optimization problem

The schema optimization problem is computationally difficult. Here, we summarize its complexity in the following theorem.

Theorem 1 *(1) Given an initial schema Q, a search space L, and a bound LIM, the problem of finding a subset $C \subset L$, such that C does not contain the root of L, every $q \in Q$ can be derived from some $c \in C$ and $|C| \leq$ LIM is NP-complete.*

(2) Given a performance ratio r, $r > 1$, to find an algorithm A for the Schema Optimization Problem defined in Table 1 whose performance is bounded by r times the optimal performance is NP-hard.

$\boxed{\textbf{Version 3}}$

Proof. We first prove (1). Obviously, the problem is in *NP*. To show that the problem is *NP-complete*, we reduce the well-known *NP-complete* problem *3SAT* to it.

Consider any instance of *3SAT*, which is a boolean formula ϕ in conjunctive normal form. Suppose ϕ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $c_1, c_2, \ldots, c_m$. (Each clauses has exactly three distinct literals, where a literal is either a variable or its negation.) We construct the following searching space L, which comprises the cubes r, b, and two sets of cubes V and W. The cubes r and b are the *universal upper bound* (i.e. the root) and the *universal lower bound*, respectively. For each variable x_i, there are cubes v_i and v_i' in V, and a cube w_i in W. For each clause c_j, there is a corresponding cube z_j in W. Note that V has $2n$ cubes and W has $n + m$ cubes. The partial ordering for L is defined as follows. For every $v \in V$, $v \prec r$, and for every $w \in W$, $b \prec w$. Also, for each of the three literals ℓ in some clause c_j, if ℓ is a variable x_i, then $z_j \prec v_i$; if ℓ is its negation, then $z_j \prec v_i'$. For example, if $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$, the corresponding search space L is given in Figure 4. Q comprises all cubes in W and *LIM* is

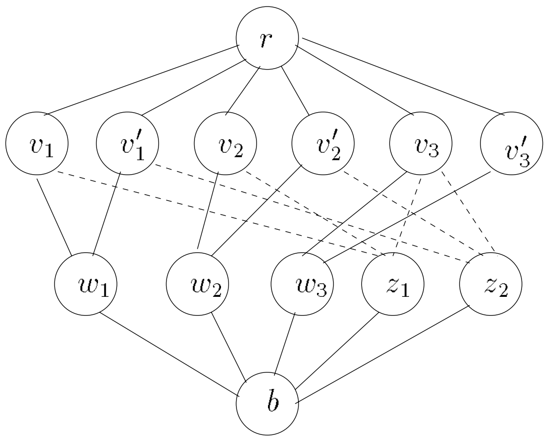

Figure 4: The search space L

n. It can be verified that ϕ is satisfiable if and only if there exists a subset C of L, $r \notin C$, such that every $q \in Q$ can be derived from some $c \in C$ and $|C| \leq LIM$. It follows that the problem formulated in (1) is *NP-complete*.

To prove (2), it can be shown that if we have an efficient algorithm A for the Schema Optimization Problem whose performance is bounded by some ratio r, then we can use A to design an efficient algorithm for solving the *set-covering* problem [5], which is *NP-complete*. It follows that finding such an approximation algorithm is *NP-hard*. Details will be given in the full paper. □

Theorem 1 tells us that the optimization problem is a very difficult one. In theory, it is impossible to find even an efficient approximation algorithm that can give a performance guarantee. In the next section, we will discuss a greedy approximation algorithm and discuss the heuristics it uses to prune the vast search space looking for a "good" solution. The efficiency and the effectiveness of the algorithm are studied in Section 4.

3 The Algorithm cMP

We have developed a greedy algorithm called cMP (**c**ube **M**erging and **P**runing). The outline of cMP is shown in Figure 5.

During each iteration of the loop (lines 2 to 6) of algorithm cMP, we select two cube sets D and A. The cubes in D are removed from C, and the cubes in A are added into C. The cube sets D and A are selected such that the cubes in Q can still be answered by the new C. In order to reduce the maintenance cost, the number (or size) of the cubes added is smaller than that of the removed ones. The new cube set C however might have a larger query cost. The algorithm terminates when the maintenance cost of the cube set no longer exceeds the limit *LIM*.

The selection of the cube sets D and A is governed by the evaluation function α. The

```
   /* input: $L, Q, LIM$; output: $C$ */
1) $C = Q$;
2) while $MC(C) > LIM$ do {
3)    SelectCubes($D \subseteq C, A \subseteq L - C$) such that $\alpha(C, D, A)$ is maximum;
4)    /* $\alpha$ is an evaluation function */
5)    $C = C \cup A - D$;
6) }
7) return $C$;
```

Figure 5: The algorithm cMP

evaluation function is defined such that the reduction in the maintenance cost is large while the increment in the query cost is small. In our algorithm, we use the following α function:

$$\alpha(C, D, A) = \frac{SavingInMC(C)}{IncreaseInQueryCost} = \frac{\sum_{t \in D} S(t) - \sum_{t \in A} S(t)}{\sum_{t \in D} n(t) \times [S(F_{C^+}(t)) - S(t)]} \tag{3.3}$$

where $C^+ = C \cup A - D$ is the new C. The numerator of formula 3.3 is the saving in maintenance cost. (We have used MC_2 in this formula. The results in the rest of the paper, unless stated explicitly, are also valid for MC_1.) The denominator is the increment in the query cost. $F_{C^+}(t)$ is the smallest ancestor of t in C^+. Since $t \in D$ is removed, the queries which were answered by t before C is changed to C^+ now have to be answered by $F_{C^+}(t)$. The number of such queries in Q is denoted by $n(t)$. Hence, the increment in the query cost is $n(t) \times [S(F_{C^+}(t)) - S(t)]$.

3.1 Properties of the evaluation function α

In cMP, the search space for D and A is enormous. In this subsection, we show some properties of the evaluation function which can be used to prune the search space effectively.

Theorem 2 *Suppose L and α are defined as above, and C is the set of cubes when cMP enters an iteration. Suppose that D_s and A_s are selected based on C which maximizes the value of α over all $D \subseteq C$ and $A \subseteq L - C$. (If there are more than one combinations that give the maximum value, D_s is the combination with the fewest cubes.) Then D_s and A_s must have the following properties.*

1. *If $|D_s| = 1$, then $A_s = \emptyset$.*
2. *If $D_s = \{b_1, b_2, \cdots, b_k\}$, $k > 1$, then the following is true:*
   (a) *$A_s = \{a\}$, and $a = F_L(\{b_1, b_2, \cdots, b_k\})$, which is the smallest common ancestor of $b_1, b_2, \cdots, b_k$ in L;*
   (b) *$\forall b_i$, $i = 1, 2, \cdots, k$, $S(F_C(b_i)) > S(a)$.*

Proof: See Appendix A. □

According to the above theorem, A is determined by D in case α attains its maximum value. Also, A contains only the smallest common ancestor of the removed cubes — this significantly reduces the search space. Furthermore, item 2.b of Theorem 2 makes the evaluation of many combinations of D unnecessary.

Corollary 1 *If $D_s = \{b_1, b_2, \cdots, b_k\}, k > 1$, then $\forall i, j \leq k, i \neq j$, $b_i \prec b_j$ is* **not** *true.*

Proof:

Proof by contradiction.

Assume there exists $i, j \leq k$, and $b_i \prec b_j$. From the definition of $F_C(b_i)$, we have $S(F_C(b_i)) \leq S(b_j)$. If $a = F_L(\{b_1, b_2, \cdots, b_k\})$ is the smallest common ancestor, we have $b_j \prec a$, and $S(b_j) \leq S(a)$. Therefore, $S(F_C(b_i)) \leq S(a)$. A contradiction with (2.b) of Theorem 2. $\square$

Corollary 1 tells us that we do not need to consider a D which contains a cube that can be derived from another cube in D. For such D, the corollary implies that the α value is not the maximum. We develop the procedure **SelectCubes** which uses this result to prune candidates in the search space of cMP:

1. Build a directed acyclic graph (DAG) of all the cubes in C in which the edges are the derived from relationship $\prec$ among the cubes in C.
2. Partition the graph into disjoint *paths*. We partition the DAG by traversing the graph from a *maximal node* which has no ancestor towards a bottom node which has no descendant. The visited nodes (and their associated edges) are removed. We repeat the same procedure on the remaining nodes until all nodes are removed. Figure 6 shows an example. The first path starts from *node* 1 to *node* 2 and then to *node* 4. Removing this path reduces the graph to one that contains nodes 3, 5 and 6, from which two more paths are constructed.
3. The nodes on the same path have a derived-from relationship. According to Corollary 1, no two nodes from the same path should be picked together for D. Hence, we pick at most one node from each path. In practice, the number of paths should not be large. This pruning significantly reduces the number of possible candidates of D and hence A from all possible combinations.

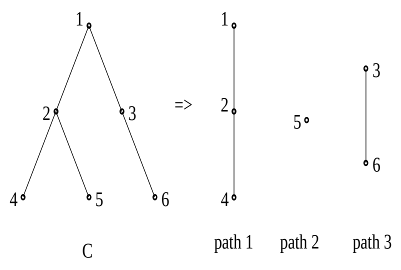

Figure 6: An example of partitioning a DAG into disjoint paths

Following Theorem 2 and the path-based processing, we derive another pruning technique for further reduction of the search space.

Corollary 2 *Assume C is partitioned into p paths: $P_1, P_2, \cdots, P_p$, ($p > 1$), where $\forall i = 1, 2, \cdots, p$, $P_i = < b_{i,1}, b_{i,2}, \cdots, b_{i,n_i} >$, and $b_{i,s} \prec b_{i,s+1}$, $s = 1, \cdots, n_i - 1$.*

Let $a_{k_1,k_2,\cdots,k_m} = F_L(\{b_{1,k_1}, b_{2,k_2}, \cdots, b_{m,k_m}\})$, $m \le p$. If $\exists t \le m$, such that $S(a_{k_1,k_2,\cdots,k_m}) \ge S(F_C(b_{t,k_t}))$, then for any set of cubes $D = \{b_{1,x_1}, b_{2,x_2}, \cdots, b_{t,k_t}, \cdots, b_{n,x_n}\}$, where $m \le n \le p$, and $x_i \ge k_i, i \le m$ and $i \ne t$, α cannot attain the maximum value using D.

Proof: See Appendix C. □

The corollary suggests that we can organize the **SelectCubes** procedure starting from the bottom of each path to compose candidates D from the nodes. When the select procedure reaches a candidate (combination) that satisfies the condition of Corollary 2, then those yet-to-be-evaluated candidates of D "above" the current combination in the lattice hierarchy can be ignored. We illustrate the pruning process with an example shown in Figure 7. The cube set C is partitioned into 3 paths containing 3, 4, and 3 nodes respectively. We select the combination for D from the bottom nodes of the paths: $b_{1,1}, b_{2,1}, b_{3,1}$. Suppose that when we evaluate the combination $D = \{b_{1,2}, b_{2,2}\}$, $S(a_{2,2}) \ge S(F_C(b_{1,2}))$ is true. According to Corollary 2, all the remaining combinations for D which include $b_{1,2}$ do not need to be evaluated. These pruned combinations are: $\{b_{1,2},\ b_{2,3}\}$, $\{b_{1,2},\ b_{2,4}\}$, and all the 9 combinations of 3 cubes: $\{b_{1,2}, b_{2,x}, b_{3,y}\}$, where $x = 2, 3, 4$, $y = 1, 2, 3$. Eleven combinations are pruned in this case.

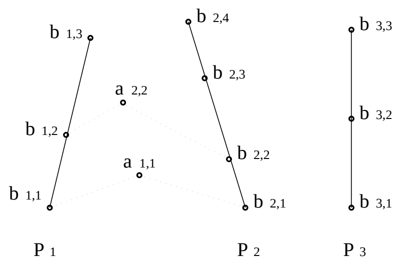

Figure 7: An example of pruning in cMP

3.2 The rMP algorithm

In this section we present the details of the procedure **SelectCubes** in cMP. Even though many combinations can be pruned while cMP is searching for the optimal α value, it may still need to consider a large number of combinations involving nodes on multiple paths. To reduce the complexity, one option is to restrict the number of nodes in a candidate combination. We remark that Theorem 2 still holds even with the size restriction. We call the search algorithm rMP when only candidates of size not larger than a certain constant r ($r > 1$) are considered. In practice, when a user is solving the schema design problem manually, the value of r would likely be very small. Therefore, it is useful to include r as a control parameter. Our performance studies show that rMP could be a good approximation of the unrestricted cMP. Obviously, the goodness depends on the value of r. When $r = p$, where p is the number of paths in C, rMP becomes cMP. When we try our algorithm on some real data sets, the results obtained by 2MP is very close to rMP, even for some large values of r. The restricted but more efficient rMP algorithm (with a small r) is thus a viable choice in many occasions. We list the procedure *SelectCubes* for rMP in Figure 8. (The main procedure of rMP is the same as that of cMP shown in Figure 5.)

In the first step of the procedure *SelectCubes*, C is partitioned into a number of paths. The loop from line 2 to line 6 evaluates all the possible combinations of D by traversing all the paths via a recursive procedure *iterate_proc*. The set of cubes to be removed (D_s) and the cube to be added (a) which attain the maximum value of α are returned at line 7.

The sequence of node traversal is constructed by the following two iterative loops:

- **Combinations of paths.** It is constructed by the loop from line 2 to line 6 and the loop from line 22 to line 26 inside the recursive procedure. Figure 7 shows an example. Suppose the size restriction r is set to 3. The sequence of path combinations considered by *SelectCubes* is $\{P_1\}, \{P_1, P_2\}, \{P_1, P_2, P_3\}, \{P_1, P_3\}, \{P_2\}$, $\{P_2, P_3\}, \{P_3\}$.
- **Traverse the nodes on a path.** This is performed at line 16 when the procedure *iterate_proc* is called, and at line 27 in each iteration of the loop between line 17 and line 28. In Figure 7, the first few combinations in the traversal sequence are $\{b_{1,1}\}$, $\{b_{1,1},\ b_{2,1}\}$, $\{b_{1,1},\ b_{2,1},\ b_{3,1}\}$, $\{b_{1,1},\ b_{2,1},\ b_{3,2}\}$, $\cdots$. During the traversal, the result in Corollary 2 is used to prune combinations that cannot attain the maximum value.

In line 18 of the procedure *iterate_proc*, the current combination is evaluated to check its α value. If one of the following two conditions occur, then the procedure can skip some iterations.

- If the condition specified at line 19 holds, then all the remaining yet-to-be-tested combinations which include the current node of each path in $rbuf.Paths - \{curPath\}$ will not be included in the final selection. Therefore, we can terminate the loop and exit the procedure.
- If the condition $S(rbuf.a) \geq S(F_C(curPath.curNode))$ is true, there is no need to add more paths to the combination in $rbuf$, because all such combinations will not be selected according to Corollary 2. Therefore, we can skip the loop between line 22 and line 26, and proceed to the next cube of $curPath$ at line 27. The other condition in line 21 $rbuf.noPaths < r$ is the size restriction on the combinations.

Let us consider the example in Figure 7 again to illustrate the above procedure. Assume the current state of $rbuf$ is: $rbuf.Paths = \{P_1,\ P_2\}$, $rbuf.curPath = P_2$, $P_1.curNode = b_{1,2}$, $P_2.curNode = b_{2,2}$. Therefore, the current combination in $rbuf$ is $\{b_{1,2},\ b_{2,2}\}$. Let $a_{2,2} = F_L(b_{1,2},\ b_{2,2})$, and assume $r = 3$.

- If $S(a_{2,2}) \geq S(F_C(b_{1,2}))$, then the loop of path P_2 is terminated, and the state of $rbuf$ is transferred to $rbuf.Paths = \{P_1\}$, $rbuf.curPath = P_1$, $P_1.curNode = b_{1,2}$. At line 27, the path P_1 will go to the next node $b_{1,3}$; therefore, the next combination to be evaluated is $\{b_{1,3}\}$;
- If $S(a_{2,2}) \geq S(F_C(b_{2,2}))$, then there is no need to add new path to the current state. The procedure will skip the loop between lines 22 and 26. At line 27, $P_2.curNode$ takes the next node $b_{2,3}$ on the path; therefore, the next combination to be evaluated is $\{b_{1,2}, b_{2,3}\}$;

```
/* Input: $L$: search space; $C$: a set of cubes; $r$: candidate combination size restriction;
   Output: $D$: cubes to be removed; $a$: a new cube to be added */
procedure SelectCubes(input: $L, C, r$; output: $D, a$)
1)   partition $C$ into paths: $path[1], path[2], \cdots, path[p\_total]$;
2)   for $i = 1$ to $p\_total$ do {
3)       add $path[i]$ to $rbuf$;
4)       call procedure $iterate\_proc(rbuf, res, r, i + 1)$;
5)       remove $path[i]$ from $rbuf$;
         /* combinations involving all paths from $path[i]$ to $path[p\_total]$ have been considered */
6)   }
7)   return result $res$;
8)
9)   procedure $iterate\_proc(rbuf, res, r, start)$
10)  $rbuf$: recursion buffer; /* all selected paths are stored in $rbuf.Paths$; on each selected
             path $path[i]$, $path[i].curNode$ is the selected cube; the set of selected cubes is the
             current candidate combination; if the combination has more than one cubes,
             then $rbuf.a$ is its smallest common ancestor, else it is null; */
11)  $res$:  /* content: the selected cube sets $D$ and $a$ which maximize $\alpha$'s value over
             all candidate combinations generated up to this point */
12)  $r$:    /* control parameter on candidate size */
13)  $start$: /* the first path not having been selected yet, i.e, the starting
               point of the recursive procedure */
14)  {
15)      $curPath = rbuf.LastAddPath$; /* $LastAddPath$ is added at line 3 or line 23 */
16)      $curPath.curNode = curPath.firstNode$;
17)      do {
18)          $evalCurrSelection(rbuf,\ res)$;
               /* evaluate the $\alpha$ value of the current combination in $rbuf$, compare it with the
               current maximum result in $res$ */
19)          if($\exists p \in rbuf.Paths - \{curPath\},\ such\ that\ S(rbuf.a) >= S(F_C(p.curNode))$)
20)              exit procedure;
21)          if($S(rbuf.a) < S(F_C(curPath.curNode))$ and $rbuf.noPaths < r$)
22)              for $i = start$ to $p\_total$ do {
23)                  add $path[i]$ to $rbuf$;
24)                  call procedure $iterate\_proc(rbuf, res, i + 1)$;
25)                  remove $path[i]$ from $rbuf$;
26)              }
27)          $curPath.curNode = curPath.nextNode$;
28)      } until ($curPath.End() == true$);
29)  }
```

Figure 8: The procedure *SelectCubes* of rMP

- Otherwise, the procedure will go into the loop between lines 22 and 26, add a new path P_3 to *rbuf*, and the state of *rbuf* is transferred to: $rbuf.Paths = \{P_1, P_2, P_3\}$, $rbuf.curPath = P_3$, $P_1.curNode = b_{1,2}$, $P_2.curNode = b_{2,2}$, $P_3.curNode = P_3.firstNode = b_{3,1}$; therefore, the next combination to be evaluated is $\{b_{1,2},\ b_{2,2},\ b_{3,1}\}$.

For illustration purpose, a complete run of *SelectCubes* on the example in Figure 7 is presented in Appendix D.

3.3 The relationship between rMP and 2MP

Although the pruning methods introduced above is very effective for rMP, its complexity is still high when r is large. It is thus interesting to see how 2MP performs comparing with the more general rMP. In particular, we want to find out the condition under which 2MP can obtain the same result as with rMP. Another important reason to study 2MP is that it may be the practitioners' choice — one practical way to solve the problem is to try all or some 2-combinations in an ad-hoc way to build up a solution.

Theorem 3 *Suppose C is in a state when rMP($r > 2$) is entering an iteration to identify new sets D and A. Assume that the maintenance cost function is MC_2. If, for all possible D's which cannot be pruned away by either Corollaries 1 or 2, D satisfies the following condition:*

$$S_a \geq \frac{1}{2(k-1)} \sum_{i=1}^{k} \sum_{j=1, j \neq i}^{k} S_{i,j} \tag{3.4}$$

where $D = \{b_1, b_2, \cdots, b_k\}$, $(2 < k \leq r)$, $S_a = S(F_L(D))$, and $\forall i, j = 1, 2, \cdots, k$, $i \neq j$, $S_{i,j} = S(F_L(\{b_i, b_j\}))$, then the (D,A) pair selected by rMP($r > 2$), and 2MP are identical.

Proof: See Appendix B. □

Theorem 3 shows a sufficient condition that rMP($r > 2$) and 2MP are equivalent. However, the condition requires the checking of an intermediate state of the algorithm. To apply the theorem to the initial state of the problem to guarantee the equivalence of the algorithms, we can use the following corollary. Its correctness is obvious.

Corollary 3 *Given L as defined in rMP and $r > 2$, and assume that the maintenance cost function is MC_2. If for all $D \subseteq L$ such that $D = \{b_1, b_2, \cdots, b_k\}$, $(2 < k \leq r)$, and $\forall i, j = 1, 2, \cdots, k$, $i \neq j$, not $b_i \prec b_j$, D satisfies the following condition:*

$$S_a \geq \frac{1}{2(k-1)} \sum_{i=1}^{k} \sum_{j=1, j \neq i}^{k} S_{i,j}$$

where $S_a = S(F_L(D))$, and $\forall i, j = 1, 2, \cdots, k$, $i \neq j$, $S_{i,j} = S(F_L(\{b_i, b_j\}))$, then the results found by rMP($r > 2$) and 2MP are identical. □

4 Performance study

We have carried out a performance study of the algorithms on a Sun Enterprise 4000 running Solaris 2.6. Our first goal is to study the "goodness" of the schemas generated by cMP and 2MP. The second goal is to study the efficiency of the two algorithms, and their pruning effectiveness.

We use the TPC-D benchmark data for the study. The database is generated with a scale factor of 0.5 [15]. The size of the database is about 0.5 GB. All the 17 DSS queries in the benchmark are frequent queries in our model, and we convert them into 17 cubes, using the approach described in Section 2.1. Hence the initial schema Q has 17 members.

4.1 Goodness of the schema generated

In the first experiment, we compare $Cost(Q, C)$ of the outputs, C, from both cMP and 2MP. We use $MC_1(C)$, the number of cubes in C, to compute the maintenance cost for simplicity.

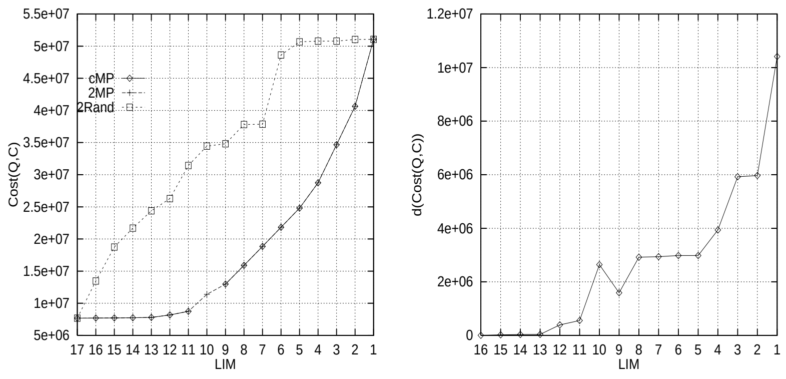

Figure 9: Query cost of the schemas generated

Figure 9 shows the result. The graph on the left shows the total response times of Q with respect to the schemas generated by cMP and 2MP. (The figure actually shows three curves with the ones from cMP and 2MP overlapping; the curve labeled 2Rand will be explained later.) The x-axis is the size (maintenance cost) limit defined in the problem. It varies from 17 down to 1. When the limit is set to 17, all the cubes in Q can be accommodated, i.e, $C = Q$, and $Cost(Q, C)$ has the minimum value. As the limit decreases, the total response time increases towards its maximum value at which the schema contains only one cube.

In our experiment, the query costs of cMP and 2MP are the same except that the curve for cMP has a gap at $LIM = 10$.[3] When $LIM = 10$, cMP at one point has reduced the schema to 11 cubes. In the next reduction, 3 cubes are selected to be replaced by one cube; hence, the size of C becomes 9. In contrast, in each step, 2MP replaces no more than 2 cubes by another. The number of cubes is thus decremented by at most one cube at a time. Therefore, 2MP has a query cost at $LIM = 10$ but not so for cMP. Furthermore, if $LIM = 10$, $Cost(Q, C)$ of

[3]The closeness of the two curves from cMP and 2MP is due to the extremely low correlations between the queries (cubes) in TPC-D.

2MP is smaller than that of cMP. However, this does not imply that 2MP has a larger α value. This behavior can also be explained because cMP does not try to find a solution which fits as close as possible into the maintenance cost limit. Instead, it tries to find a good trade-off by maximizing the decrement in the saving of the maintenance cost over the increment in query response time.

In order to obtain an insight on how good the solutions obtained by cMP are, we compared it with the algorithm that randomly merges pairs of cubes iteratively until the maintenance cost limit is not exceeded. The left graph in Figure 9 shows the result of this random selection algorithm (labeled 2Rand). It confirms that cMP is significantly better than random selection, in particular, when the cost limit is not too small so that there are more combinations for D for the algorithm to make a wise pick. We believe that an ad-hoc manual approach would be very close to the random selection if the number of cubes in the initial Q is large. Therefore, techniques such as cMP are very useful since it can produce solutions far better than what a general practitioner can do in an ad-hoc manner. In our experiment, we do not compute the optimal solutions because it is extremely time consuming due to the very large size of the TPC-D database.

Another interesting observation from the experiment is the increment rate of the query cost $Cost(Q, C)$ with respect to the reducing maintenance cost limit. It is not linear but close to a step function. To study this behavior, we plot the rate of query cost increment in the right graph of Figure 9. The rate on the y-axis is defined by $d(Cost(Q, C), m) = Cost(Q, C_m) - Cost(Q, C_{m+1})$, where m is the maintenance-cost limit on the x-axis. The graph shows that the maintenance-cost limit range can be roughly divided into several regions: (17, 11), (10, 5), (4, 2), (1). The query cost increment rate inside each region is almost the same, and there are significant gaps on the rates between the regions.

The above behavior can be explained by the correlations[4] between the cubes. According to the amount of correlation, the cubes in Q and the intermediate cubes resulted from cube-merging in cMP together form a hierarchy of clusters. Cubes in the bottom clusters of the hierarchy have high correlation (more likely to be merged); and cubes in the higher level clusters have lower correlation. Cubes that are highly correlated are merged first in cMP. The resulting query cost increment is also small. As the algorithm proceeds, however, cMP is forced to merge lesser correlated cubes. The query cost increment is thus higher in this case. Hence, the rate of query cost increases like a step function moving from levels to levels until the limit is satisfied.

The stepwise query cost rate increment can be used to select the maintenance cost limit in the data cube design. For example, a good choice of the limit in Figure 9 is 11, (a boundary point of the first region). While the response time remains small in this choice, the maintenance cost is the best fit — a further reduction on the limit would increase the response time significantly; on the other hand, increasing the limit to 12 does not help improve the query response time by much, and is thus unnecessary. If the system cannot maintain more than 10 cubes, then we have to move to the second region, (10, 5). The same selection method can be used on every region to determine a good selection.

[4]The notion of correlation here is only intuitively defined. It refers to the likelihood that a group of cubes are merged. Factors affecting the correlation may be the number of overlapping attributes or the size of the resulted cubes.

4.2 Efficiency of cMP

Our second goal is to study the efficiency of cMP. In particular, we are interested in studying their effectiveness in pruning the search space of the optimization problem.

The effect of the first pruning method that the newly added cubes A can be determined from the combination D is evident. Therefore, we only consider the other two pruning methods which are based on Corollaries 1 and 2. We measure the effectiveness by the *average pruning rate*, defined as the percentage of the pruned combinations of D over the number of all possible combinations. The results are shown in Figure 9. From the figure, we see that the average pruning rate is higher than 80% in all the cases. The pruning rate becomes smaller while the maintenance cost limit decreases. This is because, with a small limit, cMP is forced to merge cubes that are at a higher level of the lattice hierarchy (Figure 7). Hence, the chance of pruning becomes smaller. In fact, due to the design purpose of the benchmark, the correlations among the initial set of cubes in TPC-D is quite low. The high pruning rate in our experiment shows that cMP is effective even in such a not-so-favorable situation. In many general applications, we expect that the frequently asked DSS queries will have high correlation; and the pruning rate of cMP will even be better than what is shown in our experiment.

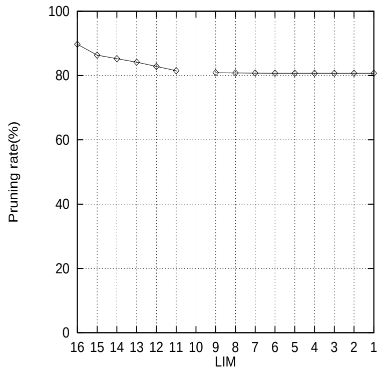

Figure 10: Average pruning rate of cMP

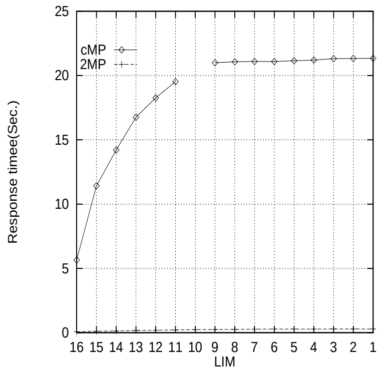

Figure 11: Response time of cMP and 2MP

Finally, we compared the efficiency of cMP and 2MP by measuring their response times. Figure 10 shows that 2MP is at least two order of magnitude faster than cMP. Considering also the effectiveness of 2MP (Figure 9), our results show that 2MP is an effective and efficient approximation solution to the data cube schema design problem.

5 Discussion and Conclusion

The basis of our 2-phase schema design approach is a set of cubes extracted from the query requirements. How valid is this approach? We have observed that some vendors have already been doing something similar. For example, Microsoft SQL OLAP server allows the users to optionally log queries submitted to it to fine tune the set of cubes [10]. From these logs, frequent queries can be identified and grouped into similar types. It is thus feasible to identify the cubes in the initial schema from the frequent queries. Currently, general practitioners design cube schema in an ad-hoc way, which is very likely far from optimal. This problem will become very serious when data cubes are required to be built on large data warehouses such as those from retail giants or Internet e-commerce shops, as their databases contain large numbers of attributes.

We have formulated the second phase of the design problem as an optimization problem, and have developed an efficient greedy algorithm to solve it. We believe that there could be other approaches for this problem. Different constraints can be set up to achieve different purposes. For example, the optimal response time achieved under a bounded maintenance cost may still be too large; instead, we can bound the response time by removing some queries (cubes) from the initial schema. The removed queries would not be answered by the data cubes but instead by the data warehouse. How to choose the queries to be removed? Can we minimize the number (size) of the queries (cubes) to be removed? These problems require further works and studies exploring different approaches.

Once a data cube schema is defined, the most imminent problem that follows is query processing. Given a DSS query submitted to the query client, the query client module needs to determine whether the query should be processed at the data cube level or at the warehouse level. If a query can be answered by the cubes, one needs to determine which cube should be used. If multiple solutions exist, one needs to determine the best choice of a cube.

We have proposed a two-phase approach to deal with the design problem in a data cube system: (1) an initial schema is derived from the user's query requirements; (2) the final schema is derived from the initial schema through an optimization process. The greedy algorithm cMP proposed for the optimization is very effective in pruning the search space of the optimal solution. Variants of cMP have been studied to reduce the search cost. Experiments on real data (TPC-D) have been performed to investigate the behavior of cMP. Results observed from the performance study confirm that cMP is an efficient algorithm. Possible future research directions also have been discussed.

References

[1] S. Agrawal, R. Agrawal, P.M. Deshpande, A. Gupta, J.F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proceedings of the International Conference on Very Large Databases*, pages 506-521, Bombay, India, September 1996.

[2] S. Chaudhuri et al. An Overview of Data Warehousing and OLAP Technology. ACM-SIGMOD Record, Vol. 26 No.1 P.65-74, March 1997

[3] S.Chauhuri and K. Shim. Including Group-By in Query Optimization. In *Proceedings of the International Conference on Very Large Databases*, pages 354-366, Santiago, Chile, 1994

[4] D. Cheung, B. Zhou, B. Kao, H. Lu, T.W. Lam, and H.F. Ting. Requirement-Based Data Cube Schema Design. In *HKU CS Technical Report TR-99-04*, 1999.

[5] T. Cormen, C. Leiserso, and R. Rivest. *Introduction to Algorithms*, The MIT press, 1990.

[6] J. Gray, A. Bosworth, A. Layman, and H. Piramish. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *Proceeding of the 12th Intl. Conference on Data Engineering*, pages 152-159, New Orleans, February 1996.

[7] A. Gupta, V. Harinarayan and D. Quass. Aggregate-Query Processing in Data Warehousing Environments. In *Proceedings of the International Conference on Very Large Databases*, pages 358-369, 1995

[8] Hyperion Essbase, http://www.hyperion.com/wired.cfm; IBM DB2 OLAP Server, http://www.software.ibm.com/data/db2/db2olap.

[9] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 205-216, Montreal, Quebec, June 1996.

[10] Microsoft SQL OLAP Services and PivotTable Service, http://www.microsoft.com/sql/70/whpprs/olapoverview.htm.

[11] P. O'Neill and G. Graefe. Multi-Table Joins Through Bitmapped Join Indexes. In *SIGMOD Record*, pages 8-11, September 1995.

[12] P. O'Neil and D. Quass. Improved Query Performace with Variant Indexes. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 38-49, Tucson, Arizona, May 1997.

[13] Seagate Info Worksheet, http://www.seagatesoftware.com/seagateworksheet.

[14] A. Shukla, P.M.Deshpande, J.F.Naughton. Materialized View Selection for Multidimensional Datasets. In *Proceedings of the International Conference on Very Large Databases*, pages 488-499, New York, USA, 1998

[15] Transaction Processing Performance Council. TPC Benchmark D(Dicision Support), Standard Specification, Revision 1.2.3. San Jose, CA , USA, 1997

[16] Y.H. Zhao, P.M. Deshpande, and J.F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 159-170, Tucson, Arizona, May 1997.

APPENDIX

A Proof of Theorem 2

Proof:

Let $C^+ = C \cup A_s - D_s$, $\alpha_m = \alpha(C, D_s, A_s) = \frac{\sum_{d\in D_s} S(d) - \sum_{a\in A_s} S(a)}{\sum_{d\in D_s} n(d)\times[S(F_{C^+}(d)) - S(d)]}$.

1. Proof by contradiction.

Since $|D_s| = 1$, let $D_s = \{b\}$, and $A_s = \{a_1, \cdots, a_m\}, m \geq 1$. If α attains its maximum value α_m with this pair of D_s and A_s, we have,

$$\alpha_m = \frac{S(b) - \sum_{a_i\in A_s} S(a_i)}{n(b)\times[S(F_{C^+}(b)) - S(b)]} \tag{A.1}$$

If $F_{C^+}(b) \notin A_s$, then $F_{C^+}(b) \in C$; also

$$\alpha_m < \frac{S(b)}{n(b)\times[S(F_{C^+}(b)) - S(b)]}.$$

Because $F_{C^+}(b) \in C$, the right side of above inequality equal to $\alpha(C, \{b\}, \emptyset)$, i.e., the α value of removing cube b from C and adding nothing. Hence, the inequality contradicts with the assumption that D_s and A_s gains the maximum α value α_m.

For the other case, if $F_{C^+}(b) \in A_s$, from A.1, we have:

$$\alpha_m \leq \frac{S(b) - S(F_{C^+}(b))}{n(b)\times[S(F_{C^+}(b)) - S(b)]} \leq \frac{-1}{n(b)}. \tag{A.2}$$

In this case, since α is evaluated to a negative value, D_s and A_s would not be chosen in the optimization.

Hence, if $|D_s| = 1$, A_s is empty.

2. We prove the second part of the theorem by proving the followings:

I. $\forall b \in D_s, F_{C^+}(b) \in A_s$;

II. $\forall a \in A_s, \exists b \in D_s$, such that $F_{C^+}(b) = a$;

III. $A_s = \{a\}$, and $a = F_L(D_s)$

IV. $\forall b \in D_s, S(F_C(b_i)) > S(a)$;

I. Proof by contradiction.

Suppose $\exists b \in D_s$ such that $F_{C^+}(b) \notin A_s$, then $F_{C^+}(b) \in C$. Note that

$$\alpha_m = \frac{S(b) + \sum_{d\in D_s-\{b\}} S(d) - \sum_{a\in A_s} S(a)}{n(b)\times[S(F_{C^+}(b)) - S(b)] + \sum_{d\in D_s-\{b\}} n(d)\times[S(F_{C^+}(d)) - S(d)]}$$

hence, one of the following two inequalities must be true: [5]

$$\alpha_m \leq \frac{S(b)}{n(b) \times [S(F_{C^+}(b)) - S(b)]} \tag{A.3}$$

$$\alpha_m \leq \frac{\sum_{d \in D_s - \{b\}} S(d) - \sum_{a \in A_s} S(a)}{\sum_{d \in D_s - \{b\}} n(d) \times [S(F_{C^+}(d)) - S(d)]} \tag{A.4}$$

If inequality A.3 is true, α_m is no larger than the α value of a combination in which only cube b is removed from C. If inequality A.4 is true, then α_m is again no larger than the α value of a combination in which b is not removed from C. All the two cases contradict with the assumption of α_m.

II. Proof by contradiction.

Suppose $\exists a \in A_s$, such that, $\forall b \in D_s, F_{C^+}(b) \neq a$, then,

$$\alpha_m = \frac{\sum_{b \in D_s} S(b) - \sum_{t \in A_s} S(t)}{\sum_{b \in D_s} n(b) \times [S(F_{C^+}(b)) - S(b)]} < \frac{\sum_{b \in D_s} S(b) - \sum_{t \in A_s - \{a\}} S(t)}{\sum_{b \in D_s} n(b) \times [S(F_{C^+}(b)) - S(b)]}$$

That is, if the cube a is not added to C^+, the corresponding α value will be larger than α_m, which is a contradiction.

III. Proof by contradiction.

Suppose $|A_s| > 1$, let $A_s = \{a_1, a_2, \cdots, a_m\}$. Define $B_1 = \{b | b \in D_s$, and $F_{C^+}(b) = a_1\}$. According to the conclusion above in II and the assumption, both B_1 and $D_s - B_1$ are nonempty. Hence α_m is equal to:

$$\alpha_m = \frac{[\sum_{b \in B_1} S(b) - S(a_1)] + [\sum_{b \in D_s - B_1} S(b) - \sum_{i=2}^{m} S(a_i)]}{[\sum_{b \in B_1} n(b) \times [S(a_1) - S(b)]] + [\sum_{b \in D_s - B_1} n(b) \times [S(F_{C^+}(b)) - S(b)]]}$$

Therefore, one of the following two inequalities must be true:

$$\alpha_m \leq \frac{\sum_{b \in B_1} S(b) - S(a_1)}{\sum_{b \in B_1} n(b) \times [S(a_1) - S(b)]} \tag{A.5}$$

$$\alpha_m \leq \frac{\sum_{b \in D_s - B_1} S(b) - \sum_{i=2}^{m} S(a_i)}{\sum_{b \in D_s - B_1} n(b) \times [S(F_{C^+}(b)) - S(b)]} \tag{A.6}$$

The right sides of the above two inequality are equal to $\alpha(C, B_1, \{a_1\})$ and $\alpha(C, D_s - B_1, A_s - \{a_1\})$, respectively. Both of the cases contradict with the assumption of D_s and A_s. Therefore, the assumption that $|A_s| > 1$ is not true. According to the conclusion above in I, there must exist an element in A_s, so $|A_s| = 1$ Let $A_s = \{a\}$, then α_m becomes:

$$\alpha_m = \frac{\sum_{b \in D_s} S(b) - S(a)}{\sum_{b \in D_s} n(b) \times [S(a) - S(b)]}$$

The right side of above equation is monotonously decreasing with $S(a)$; therefore, to reach the maximum value α_m, $S(a)$ should be at its minimum value. In addition, a must satisfy the condition: $\forall b \in D_s$, $a = F_{C^+}(b)$, i.e., a is the common ancestor of the cubes in D_s. To reach the minimum value of $S(a), a = F_L(D_s)$.

[5]If $b > 0, d > 0$ than $\frac{a+c}{b+d}$ is between $\frac{a}{b}$ and $\frac{c}{d}$.

IV. Proof by contradiction.

Let $C^+ = C \cup A_s - D_s$. According to the conclusion above in III and its proof, if $|D_s| > 1$, then $A_s = \{a\}$, $a = F_L(D_s)$, and $\forall b \in D_s$, $F_{C^+}(b) = a$. Therefore, the α value is:

$$\alpha_m = \frac{\sum_{b \in D_s} S(b) - S(a)}{\sum_{b \in D_s} n(b) \times [S(a) - S(b)]} \tag{A.7}$$

Suppose $\exists b \in D_s$, such that $S(a) \geq F_C(b)$. Let $a' = F_L(D_s - \{b\})$. According to the definition of $F_L(D)$, we have $S(a') \leq S(a)$. The equation A.7 thus becomes:

$$\begin{aligned}\alpha_m &= \frac{s(b)+\sum_{d \in D_s - \{b\}} S(d) - S(a)}{n(b) \times [S(a) - S(b)] + \sum_{d \in D_s - \{b\}} n(d) \times [S(a) - S(d)]} \\ &\leq \frac{s(b)+\sum_{d \in D_s - \{b\}} S(d) - S(a')}{n(b) \times [S(F_C(b)) - S(b)] + \sum_{d \in D_s - \{b\}} n(d) \times [S(a') - S(d)]}\end{aligned}$$

Denote the right side of the above inequality by α_x, we have one of the following two inequalities be true:

$$\alpha_x \leq \frac{s(b)}{n(b) \times [S(F_C(b)) - S(b)} \tag{A.8}$$

$$\alpha_x \leq \frac{\sum_{d \in D_s - \{b\}} S(d) - S(a')}{\sum_{d \in D_s - \{b\}} n(d) \times [S(a') - S(d)]} \tag{A.9}$$

The right sides of the above two inequalities are equal to $\alpha(C, \{b\}, \emptyset)$ and $\alpha(C, D_s - \{b\}, \{a'\})$, respectively. Therefore either $\alpha_m \leq \alpha(C, \{b\}, \emptyset)$ or $\alpha_m \leq \alpha(C, D_s - \{b\}, \{a'\})$. Both of the cases contradict with the assumption of α_m.

□

B Proof of Theorem 3

Proof by contradiction.

Suppose the result found by rMP is $D = \{b_1, b_2, \cdots, b_k\}$, then $k > 2$ (*otherwise, the combination D must be evaluated by algorithm 2MP and can be selected as result.*). Let $S_i = S(b_i)$, $n_i = n(b_i)$. According to the assumption of D, its α value is:

$$\alpha = \frac{\sum_{i=1}^{k} S_i - S_a}{\sum_{i=1}^{k} n_i (S_a - S_i)} \tag{B.10}$$

Let $\alpha_{i,j} = \alpha(C, \{b_i, b_j\}, F_L(\{b_i, b_j\})), \forall i \neq j$, $i, j = 1, \cdots, k$, then:

$$\alpha_{i,j} = \frac{S_i + S_j - S_{i,j}}{n_i (S_{i,j} - S_i) + n_j (S_{i,j} - S_j)}$$

Because D is selected by rMP, so it α value must be larger than any $\alpha_{i,j}$, that is:

$$\alpha > \alpha_{i,j}, \qquad \forall i, j$$

Multiply the two side of the above inequality with $n_i(S_{i,j} - S_i) + n_j(S_{i,j} - S_j)$, we have:

$$\alpha [n_i (S_{i,j} - S_i) + n_j (S_{i,j} - S_j)] > S_i + S_j - S_{i,j}, \qquad \forall i, j \tag{B.11}$$

Add all the inequalities of B.11 for any i, j together, we can conclude: (*note:* $S_{i,j} = S_{j,i}$)

$$\begin{aligned}
\alpha[\textstyle\sum_{i=1}^{k}\sum_{j=1,j\neq i}^{k}(n_i(S_{i,j}-S_i)+n_j(S_{i,j}-S_j))] &> \textstyle\sum_{i=1}^{k}\sum_{j=1,j\neq i}^{k}(S_i+S_j-S_{i,j}) \\
2\alpha\textstyle\sum_{i=1}^{k}[n_i(\sum_{j=1,j\neq i}^{k}S_{i,j}-(k-1)S_i)] &> 2(k-1)\textstyle\sum_{i=1}^{k}S_i-\sum_{i=1}^{k}\sum_{j=1,j\neq i}^{k}S_{i,j} \\
(\textstyle\sum_{i=1}^{k}S_i-S_a)\frac{\sum_{i=1}^{k}[n_i(\frac{1}{k-1}\sum_{j=1,j\neq i}^{k}S_{i,j}-S_i)]}{\sum_{i=1}^{k}n_i(S_a-S_i)} &> \textstyle\sum_{i=1}^{k}S_i-\frac{1}{2(k-1)}\sum_{i=1}^{k}\sum_{j=1,j\neq i}^{k}S_{i,j}
\end{aligned} \tag{B.12}$$

$\forall S_{i,j} \leq S_a$, therefore, $\forall i = 1, \cdots, k$, $\frac{1}{k-1}\sum_{j=1,j\neq i}^{k} S_{i,j} \leq S_a$, then:

$$\frac{\sum_{i=1}^{k}[n_i(\frac{1}{k-1}\sum_{j=1,j\neq i}^{k}S_{i,j}-S_i)]}{\sum_{i=1}^{k}n_i(S_a-S_i)} \leq 1 \tag{B.13}$$

Combine the two inequalities B.12 and B.13, we have:

$$\begin{aligned}
\textstyle\sum_{i=1}^{k}S_i-S_a &> \textstyle\sum_{i=1}^{k}S_i-\frac{1}{2(k-1)}\sum_{i=1}^{k}\sum_{j=1,j\neq i}^{k}S_{i,j} \\
S_a &< \textstyle\frac{1}{2(k-1)}\sum_{i=1}^{k}\sum_{j=1,j\neq i}^{k}S_{i,j}
\end{aligned} \tag{B.14}$$

The inequality B.14 contradicts with the assumption of the theorem.

□

C Proof of Corollary 2

Let $a_{x_1,\cdots,k_t,\cdots,x_n} = F_L(\{b_{1,x1},\cdots,b_{t,k_t},\cdots,b_{n,x_n}\})$. Then $\forall i \leq m, i \neq t, b_{i,x_i} \preceq a_{x_1,\cdots,k_t,\cdots,x_n}$. Since $x_i \geq k_i$, so $b_{i,k_i} \preceq b_{i,x_i}$. Therefore, $\forall i \leq m, b_{i,k_i} \preceq a_{x_1,\cdots,k_t,\cdots,x_n}$. From the definition of $F_L(\{b_{1,k_1},b_{2,k_2},\cdots,b_{m,k_m}\})$, we can conclude that $S(F_L(\{b_{1,k_1},b_{2,k_2},\cdots,b_{m,k_m}\})) \leq S(a_{x_1,\cdots,k_t,\cdots,x_n})$, or $S(a_{k_1,k_2,\cdots,k_m}) \leq S(a_{x_1,\cdots,k_t,\cdots,x_n})$. Due to the assumption: $S(a_{k_1,k_2,\cdots,k_m}) \geq S(F_C(b_{t,k_t}))$, we have $\exists t < n$, such that $S(x_1,\cdots,k_t,\cdots,x_n) \geq S(F_C(b_{t,k_t}))$. According to (2.b) of Theorem 2, $D = \{b_{1,x_1},b_{2,x_2},\cdots,b_{t,k_t},\cdots,b_{n,x_n}\}$ cannot attain the maximum value of α.

□

D An example run of the *SelectCubes* procedure

We use an example here to illustrate the *SelectCubes* procedure. Suppose that the initial C is the one shown in Figure 7. Also assume that the smallest ancestor of each node $b_{i,j}$ is on the same path, i.e $F_C(b_{i,j}) = b_{i,j+1}$, and $r = 3$. In the first step, C is partitioned into 3 paths. In the loop from line 2 to line 6, the algorithm evaluates all the possible combinations of D. At the same time, the pruning from Corollary 2 is used to remove most of combinations. We show the procedure step by step (we only present several steps here, the complete run can be found in Appendix E, and the size of related cubes are also listed there):

1. The first path P_1 is added into the recursion buffer *rbuf* at line 3, then the procedure *iterate_proc* is called to evaluate all the combinations that contain cubes on P_1. At line 15, since P_1 is the last added path in *rbuf*, it is set to *curPath*, and its *curNode* is set to P_1's first node $b_{1,1}$. We then go into the loop between line 17 and line 28. At line 18,

the current combination in *rbuf* is evaluated. The current combination consists of the *curNode* of all the paths in *rbuf*. At this point, the current combination is D=$\{b_{1,1}\}$. Its α value is 1/130. The combination is selected as the result with the maximum α value over all the selections up to this point, and is stored in *res*.

2. After the checking of the node $b_{1,1}$, a new path P_2 is added into *rbuf* at line 23. Then the algorithm calls the recursive procedure *iterate_proc* at line 24, and the combination $D = \{b_{1,1}, b_{2,1}\}$ is evaluated. Its α value is 1/110, larger than the current value 1/130 stored in *res*, so the combination is selected as the result. Up to this point, the pruning condition of Corollary 2 is not satisfied, therefore, the procedure goes into the loop between line 22 and line 26, and recursively calls the procedure *iterate_proc* again.

3. When the combination $D = \{b_{1,1}, b_{2,1}, b_{3,1}\}$ is being evaluated, the size of its smallest common ancestor $a_{1,1,1}$ is larger than $b_{1,2}$, the smallest ancestor of $b_{1,1}$ in C. Hence, the condition in line 19 becomes true. Therefore, the current calling of the procedure is exited, and at line 25, the path P_3 is removed from *rbuf*.

4. Up to this point, there are two paths P_1 and P_2 in *rbuf*, and the *curPath* is P_2. At line 27, the *curNode* of P_2 moves to the next node, i.e. $b_{2,2}$; therefore, the combination $D = \{b_{1,1}, b_{2,2}\}$ is evaluated. Since $S(a_{1,2}) > S(b_{1,1})$, the procedure exits at line 20 again to prune away all combinations which contain $b_{1,1}$, and the cubes that can derive $b_{1,2}$. The path P_2 is then removed from *rbuf*.

5. At line 23, path P_3 is added to *rbuf*, and *rbuf.Paths* now becomes $\{P_1, P_3\}$. The combination $\{b_{1,1}, b_{3,1}\}$ is checked in a new call to the procedure. The pruning condition at line 19 is satisfied. The path P_3 is removed.

6. Now the next node $b_{1,2}$ on P_1 is checked. After the evaluation, path P_2 is added to *rbuf* again to check the combinations containing $b_{1,2}$.

7. The α value of $D = \{b_{1,2}, b_{2,1}\}$ is 1/100, which is larger than the current maximum, so it is selected as the result. This is the final D selected.

E Example of a complete run of *SelectCubes*

The following table (Table 2) is a complete run of *SelectCubes* on the example presented in Appendix D. The size of the cubes involved in the example is listed in Table 3.

step	D	A	α value	action
1	$b_{1,1}$		1/130	Selected
2	$b_{1,1}$ $b_{2,1}$	$a_{1,1}$	1/110	Selected
3	$b_{1,1}$ $b_{2,1}$ $b_{3,1}$	$a_{1,1,1}$		Use pruning method 2.2
4	$b_{1,1}$ $b_{2,2}$	$a_{1,2}$		Use pruning method 2.2
5	$b_{1,1}$ $b_{3,1}$	$a_{1,0,1}$		Use pruning method 2.2
6	$b_{1,2}$		1/350	
7	$b_{1,2}$ $b_{2,1}$	$a_{2,1}$	1/100	Selected
8	$b_{1,2}$ $b_{2,1}$ $b_{3,1}$	$a_{2,1,1}$		Use pruning method 2.2
9	$b_{1,2}$ $b_{2,2}$	$a_{2,2}$	1/250	
10	$b_{1,2}$ $b_{2,2}$ $b_{3,1}$	$a_{2,2,1}$		Use pruning method 2.1
11	$b_{1,2}$ $b_{2,2}$ $b_{3,2}$	$a_{2,2,2}$		Use pruning method 2.2
12	$b_{1,2}$ $b_{2,3}$	$a_{2,3}$		Use pruning method 2
13	$b_{1,2}$ $b_{3,1}$	$a_{2,0,1}$		
14	$b_{1,2}$ $b_{3,2}$	$a_{2,0,2}$		Use pruning method 2.2
15	$b_{1,3}$		0/1	
16	$b_{1,3}$ $b_{2,1}$	$a_{3,1}$		Use pruning method 2.1
17	$b_{1,3}$ $b_{2,2}$	$a_{3,2}$		
18	$b_{1,3}$ $b_{2,2}$ $b_{3,1}$	$a_{3,2,1}$		Use pruning method 2.1
19	$b_{1,3}$ $b_{2,2}$ $b_{3,2}$	$a_{3,2,2}$		Use pruning method 2.2
20	$b_{1,3}$ $b_{2,3}$	$a_{3,3}$	1/1500	
21	$b_{1,3}$ $b_{2,3}$ $b_{3,1}$	$a_{3,3,1}$		Use pruning method 2.1
22	$b_{1,3}$ $b_{2,3}$ $b_{3,2}$	$a_{3,3,2}$		Use pruning method 2.1
23	$b_{1,3}$ $b_{2,3}$ $b_{3,3}$	$a_{3,3,3}$	2/3900	
24	$b_{1,3}$ $b_{3,1}$	$a_{3,0,1}$		Use pruning method 2.1
25	$b_{1,3}$ $b_{3,2}$	$a_{3,0,2}$		Use pruning method 2.1
26	$b_{1,3}$ $b_{3,3}$	$a_{3,0,3}$		
27	$b_{2,1}$		1/150	
28	$b_{2,1}$ $b_{3,1}$	$a_{0,1,1}$	1/110	
29	$b_{2,1}$ $b_{3,2}$	$a_{0,1,2}$		Use pruning method 2.2
30	$b_{2,2}$		1/800	
31	$b_{2,2}$ $b_{3,1}$	$a_{0,2,1}$		
32	$b_{2,2}$ $b_{3,2}$	$a_{0,2,2}$		Use pruning method 2.1
33	$b_{2,2}$ $b_{3,3}$	$a_{0,2,3}$		
34	$b_{2,3}$		0/1	
35	$b_{2,3}$ $b_{3,1}$	$a_{0,3,1}$		Use pruning method 2.1
36	$b_{2,3}$ $b_{3,2}$	$a_{0,3,2}$		Use pruning method 2.1
37	$b_{2,3}$ $b_{3,3}$	$a_{0,3,3}$	1/2000	
38	$b_{3,1}$		1/260	
39	$b_{3,2}$		1/300	
40	$b_{3,3}$		0/1	
Result	$b_{1,2}$ $b_{2,1}$	$a_{2,1}$	1/100	
The average pruning rate: 92.17%				

Table 2: Complete step of *SelectCubes*

The size of the related cubes for the example.

cube	size
$b_{1,1}$	20
$b_{1,2}$	150
$b_{1,3}$	500
$b_{2,1}$	50
$b_{2,2}$	200
$b_{2,3}$	1000
$b_{3,1}$	40
$b_{3,2}$	300
$b_{3,3}$	600
$a_{1,1}$	90
$a_{1,2}$	200
$a_{2,1}$	150
$a_{2,2}$	300
$a_{2,3}$	1200
$a_{3,3}$	1500
$a_{1,0,1}$	150
$a_{1,0,2}$	800
$a_{0,1,1}$	100
$a_{0,1,2}$	400
$a_{0,2,2}$	700
$a_{0,3,3}$	1800
$a_{1,1,1}$	200
$a_{2,2,2}$	2000
$a_{3,3,3}$	2000

Table 3: Size of the cubes