The VLDB Journal (2000) 8: 319–338

The VLDB Journal
© Springer-Verlag 2000

Optimizing multiple dimensional queries simultaneously in multidimensional databases

Weifa Liang[1], **Maria E. Orlowska**[2], **Jeffrey X. Yu**[1]

[1] Department of Computer Science, Australian National University, Canberra, ACT 0200, Australia
[2] Distributed Systems Technology Center, Department of Computer Science and Electrical Engineering, University of Queensland, St. Lucia, QLD 4072, Australia

Edited by T. Sellis. Received: July 21, 1998 / Accepted: August 26, 1999

Abstract. Some significant progress related to multidimensional data analysis has been achieved in the past few years, including the design of fast algorithms for computing datacubes, selecting some precomputed group-bys to materialize, and designing efficient storage structures for multidimensional data. However, little work has been carried out on multidimensional query optimization issues. Particularly the response time (or evaluation cost) for answering several related dimensional queries simultaneously is crucial to the OLAP applications. Recently, Zhao et al. first exploited this problem by presenting three heuristic algorithms. In this paper we first consider in detail two cases of the problem in which all the queries are either hash-based star joins or index-based star joins only. In the case of the hash-based star join, we devise a polynomial approximation algorithm which delivers a plan whose evaluation cost is $O(n^\epsilon)$ times the optimal, where n is the number of queries and ϵ is a fixed constant with $0 < \epsilon \leq 1$. We also present an exponential algorithm which delivers a plan with the optimal evaluation cost. In the case of the index-based star join, we present a heuristic algorithm which delivers a plan whose evaluation cost is n times the optimal, and an exponential algorithm which delivers a plan with the optimal evaluation cost. We then consider a general case in which both hash-based star-join and index-based star-join queries are included. For this case, we give a possible improvement on the work of Zhao et al., based on an analysis of their solutions. We also develop another heuristic and an exact algorithm for the problem. We finally conduct a performance study by implementing our algorithms. The experimental results demonstrate that the solutions delivered for the restricted cases are always within two times of the optimal, which confirms our theoretical upper bounds. Actually these experiments produce much better results than our theoretical estimates. To the best of our knowledge, this is the only development of polynomial algorithms for the first two cases which are able to deliver plans with deterministic performance guarantees in terms of the qualities of the plans generated. The previous approaches including that of [ZDNS98] may generate a feasible plan for the problem in these two cases, but they do not provide any performance guarantee, i.e., the plans generated by their algorithms can be arbitrarily far from the optimal one.

Key words: Multiple dimensional query optimization – OLAP – MDDBs – Query modeling – Data warehousing

1 Introduction

Recently, On-Line Analytical Processing (OLAP) has attracted substantial research interest. Issues such as efficient algorithms for computing datacubes [GBLP95, AAD+96, SAG96, DANR96, RS97, ZDN97], techniques for selecting group-bys for materialization [HRU96, Gupta97, GHRU97, YKL97, BPT97], and efficient incremental methods for materialized view maintenance [GM95, ZGHW95, AESY97, HZ96, RSS96] have been successfully investigated. However, there has been little attention devoted to the optimization of multiple simultaneous OLAP queries. This kind of multiple multidimensional query optimization has specific characteristics and to a large extent is different from the traditional multiple query optimization procedures [PS88, S88, SS94]. That is, typical multidimensional queries involve data aggregation across different dimensions and hierarchies, moreover, the amount of data manipulated is normally very large, e.g., gigabytes or terabytes of data. In addition to the intrinsic interest of the problem, optimizing multiple simultaneous OLAP queries is likely to be of considerably practical importance due to the recent developments in this area. Microsoft recently released its proposed ”OLE DB for OLAP” standard for interfaces to multidimensional data sources [MS]. This standard is likely to become widely accepted. One of the most interesting aspects of the standard is its definition of *Multi-Dimensional Expressions* (MDX) which provides a framework in which a user can ask several related OLAP queries in a single MDX expression. This aspect of MDX reflects the fact that OLAP-style analysis often gives rise to simultaneous related queries, and MDX intends to provide a uniform framework for a variety of data sources.

In this paper the problem considered can be informally stated as follows: given a set of materialized views in a fact table and an MDX expression which can be decomposed into a set of related multidimensional queries, find a plan for answering all the multidimensional queries (thereby answering the MDX query) such that the total response time is minimized. Therefore, our objective is to exploit the information shared (materialized views and dimensional tables) among the queries, and to fully utilize the index structures of the materialized views of some queries to reduce the total response time for all queries. In the end this reduces the response time for the given MDX. Without loss of generality, in this paper we only consider the evaluation of MDXs in a relational multidimensional database (MDDB) system. Recently Zhao et al. [ZDNS98] first exploited the problem by introducing the optimization of multiple dimensional queries simultaneously. They gave a general framework in terms of the optimization cost (the response time for an MDX) by presenting three heuristic algorithms for finding a better global plan according to the optimization cost, based on the three shared operators that they developed: `shared scan for hash-based star join`, `shared index-based join`, and `shared scan for hash-based and index-based star join`. It must be mentioned that [RSC98] also discuss a variant of this problem.

In this paper we will examine the problem in detail and propose more efficient solutions. We first consider two restricted versions of the problem, in which all queries in a set of multiple dimensional queries are either hash-based star joins or index-based star joins, by presenting a cost model for answering these queries in terms of the response time required. For the case of hash-based star joins, we devise a polynomial approximation algorithm through reducing the problem to a directed Steiner tree problem. The algorithm delivers a plan with the evaluation cost of $O(|Q|^{\epsilon})$ times the optimal, and the time for finding such a plan is $O(|Q| \cdot |MV| + (|Q| + |MV|)^{1/\epsilon} + (|Q| + |MV|)\log(|Q| + |MV|))$, where Q is the query set and $|Q|$ is the number of queries in Q, MV is the set of materialized views from which the queries in Q can be answered and $|MV|$ is the number of materialized views in MV, and ϵ is a fixed constant with $0 < \epsilon \leq 1$. We also present an exponential algorithm which delivers a plan with the optimal evaluation cost. In the case of index-based star joins, we present a heuristic algorithm for the problem by reducing this problem to a minimum weighted maximum cardinality matching problem and relaxing the constraint condition, the solution delivered by our algorithm is $|Q|$ times the optimal. We also give an exponential algorithm which delivers a plan with the optimal cost. We then deal with a general case in which the query set contains both hash-based star join and index-based star join queries. For this case we first suggest possible improvement of the algorithms of Zhao et al., based on our analysis of their approach. Subsequently, we present another heuristic algorithm for the problem. We also present an exponential algorithm which delivers a plan with the optimal cost. We should mention that the exact algorithms for the three cases are effective if the number of the queries involved is not too large (e.g., $|Q| \leq 10$). We finally conduct a performance study by implementing our algorithms. The experimental results demonstrate that the solutions delivered for the restricted cases in this paper are always within two times of the optimal, which confirms our theoretical upper bounds. Actually these experiments produce much better results than our theoretical estimates. The results in the general case are very interesting, in some cases, our algorithm outperforms the existing one, in other cases, the existing one is better than ours. To the best of our knowledge, this is the only development to provide approximation algorithms for the first two cases which are able to deliver the plans with deterministic performance guarantees in terms of the qualities of the plans generated. The previous approaches including those of [ZDNS98] may generate a feasible plan for the problem for these two cases, but they do not provide any performance guarantee (i.e., the plans generated by their algorithms can be arbitrarily far from the optimal).

The rest of the paper is organized as follows. In Sect. 2 we introduce some basic concepts and the three shared operators developed by Zhao et al. [ZDNS98]. In Sect. 3 we study a version of the problem in which case all queries are `shared scan for hash-based star join`s. For this version, we first present a cost model for it in terms of response time, then present an approximation algorithm for finding a global plan with a fixed degree approximation of the optimal cost, and finally give an exact algorithm which finds a global plan with the optimal cost. In Sect. 4 we define another version of the problem in which case all the queries are `shared scan for index-based star join`s. Following the same discussion in Sect. 3, we first model it in the time cost measure, then present a heuristic algorithm for finding a better global plan with a fixed degree approximation of the optimal cost, and finally give an exact algorithm which delivers a global plan with the optimal cost. In Sect. 5 we briefly introduce the three heuristic algorithms devised by Zhao et al. [ZDNS98], and analyze their two greedy algorithms by pointing out the drawbacks of the solutions sufficed through a running example. We then present some possible improvement to their approach. Also, we present another greedy algorithm for the problem. Finally, in Sect. 6 we present our implementation of the presented algorithms and study their performance. We conclude our discussions in Sect. 7.

2 Preliminaries

A multidimensional database is a data repository that provides an integrated environment for decision-support queries that require complex aggregations on large amounts of historic data. Here we assume that an MDDB is a relational data warehouse in which the information is organized as a star schema [Kim96].

2.1 The star schema

An MDDB consists of a *fact table* $F(D_1, D_2, \ldots, D_n, M_1, M_2, \ldots, M_r)$ and a single dimension table for each dimension D_i. Each *dimension table* D_i contains all the information that is specific only to the dimension itself, while the fact table F correlates all dimensions and contains the information on the attributes of interest for the intersection of all the dimensions, $1 \leq i \leq n$. Each M_j is called a *measure* of F, $1 \leq j \leq r$. Each tuple in F consists of a pointer

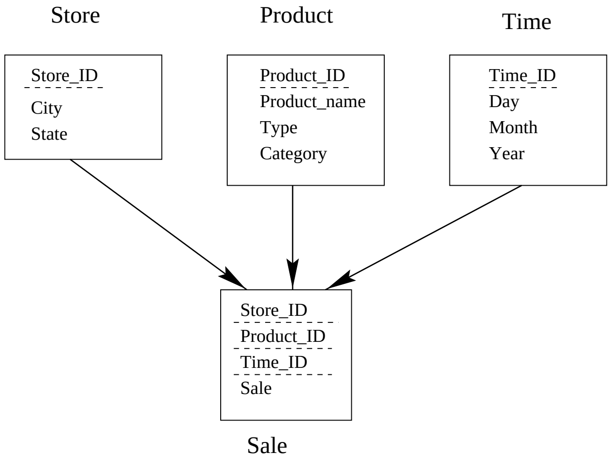


Fig. 1. A star schema

(foreign key) to each of the dimensions that provides its dimensional coordinates, and stores the numeric measures for those coordinates. Figure 1 shows an example of a star schema, where `Product, Store` and `Time` are the dimension tables, `Sale` is the fact table, and the attribute *Sale* in table `Sale` is a measure.

2.2 Data cubes

Let $F(D_1, D_2, \ldots, D_n, M)$ be a fact table where D_i is a dimension of F and M is the measure, $1 \leq i \leq n$. Obviously, there are 2^n possible combinations of the dimensions. The *data cube* of F contains all dimensional group-by aggregation tables.

In order to achieve the minimum response time for the data cube computation, the objective is to minimize the total number of operations for the computation. [HRU96] introduced a *search lattice* $\mathscr{L} = (V, E)$ for such computation, where each group-by is a node in V and there is a directed edge $\langle u, v\rangle \in E$ from group-by u to group-by v if v can be generated from u and v has exactly one dimension less than u. The weight associated with the edge is the operation cost, which usually is an estimation value by sampling. Figure 2 shows an example of the search lattice of the data cube of a fact table `R(A,B,C,D M)` and `M` is the measure of R.

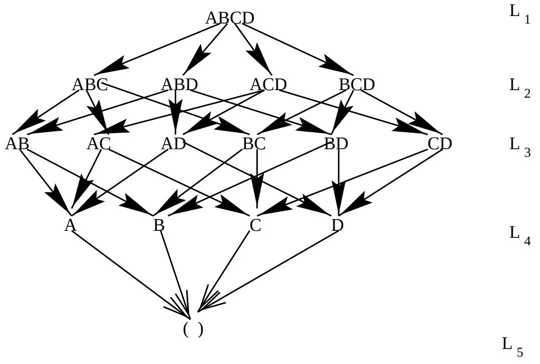


Fig. 2. A search lattice of a data cube

2.3 Multi-dimensional expressions

The main motivation behind our study is that a single MDX expression can be decomposed into several related OLAP queries. Obviously, if all the queries can be evaluated effectively, the MDX query can then be answered efficiently. In order to illustrate this concept, we give the following example which has been used by [ZDNS98]. The example also appears in [MS].

We assume that the database consists of one fact table `SalesCube` and five dimension tables `Salesperson, Geography, Quarters, Products, Years`, and one `Measure` that is `Sales`, where `Geography` is a hierarchy that has the levels states, countries, continents, and so on. `Quarters` is also a hierarchy that has the levels quarters, months, days. Then, an MDX is defined as follows.

```
NEST ({Venkatrao, Netz}, {USA_North.CHILDREN,
USA_South, Japan})
on COLUMNS {Qtr1.CHILDREN, Qtr2, Qtr3,
Qtr4.CHILDREN}
on ROWS
CONTEXT SalesCube
FILTER (Sales, [1991], Products.All)
```

where `NEST, CONTEXT, COLUMNS, CHILDREN, and FILTER` are all reserved MDX keywords. This MDX specifies the total sales for salesmen Venkatrao and Netz in all states of USA_North, USA_South, and Japan for all the months of the first quarter, the second quarter, the third quarter, and all the months of the fourth quarter in 1991.

In the relational terms, the join between the fact table and the dimension tables are defined implicitly in an MDX. Therefore, the join attributes and join conditions are not visible in the MDX. This is due to the fact that an MDX does not define the way in which an OLAP server should process the MDX query. The MDX in the example can be decomposed into the following six different group-by queries in terms of SQL statements:

1. The total sales for Venkatrao and Netz in all states of USA_North for the 2nd and 3rd quarters in 1991;
2. the total sales for Venkatrao and Netz in all states of USA_North for the months of the 1st and 4th quarters in 1991.
3. The total sales for Venkatrao and Netz in region USA_South for the 2nd and 3rd quarters in 1991.
4. The total sales for Venkatrao and Netz in region USA_South for the months of the 1st and 4th quarters in 1991.
5. The total sales for Venkatrao and Netz in Country Japan for the 2nd and 3rd quarters in 1991.
6. The total sales for Venkatrao and Netz in Country Japan for the months of the 1st and 4th quarters in 1991.

More succinctly, we have six group-bys (Salesperson, State, Quarter), (Salesperson, State, Month), (Salesperson, Region, Quarter), (Salesperson, Region, Month), (Salesperson, Country, Quarter), and (Salesperson, Country, Month). In addition, we have disjoint selection predicates for each group-by. This means that we cannot use the technique of finding common selection predicates which is widely used in multiple query optimization for general SQL queries [S88].

Therefore, from now on, unless otherwise specified we assume that an MDX has been decomposed into a set Q of queries. We will develop algorithms to reduce the total response time for answering all queries in Q.

2.4 Shared operators

As is well understood, the fundamental task in evaluating multiple related dimensional queries is identifying and exploiting common subtasks among the queries. In the star join environment, we try to make as many queries as possible share star joins. Such joins are either hash-based star joins or index-based star joins. For very selective star join queries, the index-based star join is a good alternative [OQ97]. For non-selective star join queries, the hash-based star join is a good solution [Su96]. Based on the star join environment, Zhao et al. [ZDNS98] introduce the three shared operators: `shared scan for hash-based star join`, `shared scan for index-based star join`, and `shared scan for hash-based and index-based star join`. For sake of completeness we briefly introduce these operators below.

We refer to any materialized group-by (node) in the search lattice as a *base table*. The fact table itself is a base table. Let q be a query such that q can be answered using a base table R. It follows that a hash-based star join for q involves building a hash table on each dimension table, and then probing the hash table with the tuples of the base table R. We assume that the aggregation afterwards is also done by hashing.

The `shared scan for hash-based star join` is defined as follows. Suppose we have two different queries q_1 and q_2 sharing the same base table. Clearly, q_1 and q_2 can share scanning the base table. Furthermore, if q_1 and q_2 use the same dimension tables, then they can share the hash tables too, instead of redundantly building and probing the hash tables on the same dimension tables. We refer to this sharing as `Type One Sharing`.

The `shared scan for index-based star join` is defined as follows. Assume that each dimension table has built bitmap join indexes that map the join attributes to the tuples of a base table F'. Now we consider two queries q_1 and q_2 which share the base table F' and both use the bitmaps $BT(q_1, F')$ and $BT(q_2, F')$. The `index-based star join` proceeds as follows. Form a combined bitmap $BM := BT(q_1, F') \vee BT(q_2, F')$, and use BM to probe the base table F' to extract the matching tuples. Let F'' be the set of matching tuples in F'. Each query q_i then uses its own bitmap $BT(q_i, F')$ to filter F'' to get its own matching tuples. Finally, each q_i proceeds aggregation by hashing, $i = 1, 2$. We refer to this sharing as `Type Two Sharing`.

The `shared scan for hash-based and index-based star join` operator is an operator which is a mixture of the first two. Suppose that a set of queries share scanning of the same base table. Among these queries, some of them are hash-based and the remaining are index-based star joins. Those hash-based queries also share hash-based star joins. For those index-based queries, we first use their combined bitmap to extract a set S of matching tuples from the base table. We then use the bitmap of each query to filter the matching tuples from S. We finally send these matching tuples to the corresponding group-by hash table to perform aggregation. We refer to this sharing as `Type Three Sharing`.

3 Multiple query optimization with Type One Sharing

As mentioned before, an MDX can be decomposed into a set Q of SQL queries. If all queries in Q are `Type One Sharing` queries, we show that there is an efficient algorithm for finding a query plan which delivers the guaranteed performance for this restricted version of the problem.

3.1 Problem definition

Let $\mathscr{L} = (V, E)$ be the lattice of a fact table F. Some of the nodes (aggregates) in $\mathscr{L}$ have been precomputed (materialized). Let MV be the set of all precomputed nodes. Assume that the node (table F) is always materialized and included in MV. Thus, we have a set Q of `Type One Sharing` queries.

The Multiple Dimensional Query Optimization Problem for `Type One Sharing` is to: (i) find a materialized node (base table) from MV for every query in Q from which that query can be answered; and (ii) minimize the total response time for answering all the queries.

Assume that every materialized view in MV is *useful* with respect to the queries in Q, this means, at least one query $q \in Q$ can be answered from it for every view in MV. Otherwise, those useless materialized views can be removed from MV for further consideration. Basically, the response time for an MDX consists of two parts: one is the time of loading necessary base tables (each time a fragment of a table stays in memory) and the other is the time of filtering by individual queries. We formalize the problem in terms of time cost as follows:

(i) If a materialized view $v \in MV$ is used by at least a query, then v contributes a cost

$$t_{materlized}(v) = \frac{Size(v)}{Page\_Size} \cdot t_{I/O} + t_{hash\_join}(v) \qquad (1)$$

where $Size(v)$ is the number of tuples of table v, $Page\_Size$ is the size of the unit block (e.g., 8 kB) between memory and the disk, $t_{I/O}$ is the transfer time of a single block between them, $t_{hash\_join}(v)$ is the sum of the time for star-joining the base table v with its dimension tables and the time for generating hash tables for the dimensions. Thus, $t_{materlized}(v)$ is the time related to the base table v, which is the sum of the time needed to load v to main memory and the time for hash-based star join.

(ii) If a query $q \in Q$ can be answered using $u \in MV$, then q contributes a cost

$$t_{query}(q, u) = Size(u) \cdot t_{cpu}(q, u) \qquad (2)$$

where $t_{cpu}(q, u)$ is the sum of the time needed to match the predicates in query q to a tuple in u and the time for proceeding hashing operations for aggregation which is the CPU time.

Now, for a given plan $\mathscr{P}$, every $q \in Q$ has already chosen a materialized view $mv(q) \in MV$ from which it can be evaluated. We analyze the response time for the MDX. Assume that all the base tables have been available in the main memory, then the total time for answering all the queries in Q is

$$T_{queries}(MV, Q) = \sum_{q \in Q} t_{query}(q, mv(q)). \tag{3}$$

However, the sum of the time of loading the base tables to the main memory and proceeding star-joining the base tables with their dimension tables is

$$T_{materialized}(MV, Q) = \sum_{v \in M'} t_{materialized}(v) \tag{4}$$

where $M' = \{mv(q) \mid q \in Q,\ mv(q) \in MV\}$.

Therefore, to find a better plan $\mathscr{P}$, the problem now is to minimize the following expression.

$$T_{queries}(MV, Q) + T_{materialized}(MV, Q) \tag{5}$$

It is clear that the minimum value of expression 5 is the minimum response time for answering all the queries in Q.

Since the problem involves choosing a base table from a number of base tables for every query in Q and minimizing the total response time for answering the queries, it is likely to be NP-hard. Therefore, in this paper we will focus on a polynomial approximation solution for the problem if the number of queries is large; otherwise, we will give an exact solution by presenting an exhaustive algorithm for it.

3.2 The approximation algorithm

We now give a feasible solution for the problem. The basic idea of our approach is first to reduce the problem to a directed Steiner tree problem in an auxiliary graph. We then obtain an approximation solution by solving the Steiner tree problem, and we finally transform that approximation solution back to a feasible solution of the original problem. In the following we explain the details of our algorithm.

We first construct an auxiliary graph $G = (X, Y, E_{XY}, \omega_1, \omega_2)$ which is a directed, weighted bipartite graph as follows. X is the set of base tables (materialized views), i.e., $X = MV$; Y is the set of the queries in Q, i.e., $Y = Q$. For every vertex $x \in X$, the weight associated with it $\omega_1(x)$ is $\frac{Size(x)}{Page\_Size} \cdot t_{I/O} + t_{hash\_join}(x)$. There is a directed edge $\langle x, y \rangle \in E_{XY}$ from $x \in X$ to $y \in Y$ if y can be answered using x, and the weight associated with the edge is $\omega_2(x, y) = Size(x) \cdot t_{cpu}(y, x)$. Obviously, G is an edge-weighted and partially vertex-weighted (only the vertices in X have been assigned weights) directed, bipartite graph.

The objective is to find a subset $X' \subseteq X$ in G such that: (i) every vertex in $y \in Y$ chooses a vertex $mv(y) \in X'$; and (ii) $\sum_{x \in X'} \omega_1(x) + \sum_{y \in Y} \omega_2(mv(y), y)$ is minimized. It is easy to see that this is equivalent to our objective for the original problem. However, this optimization problem is NP-hard because the minimum weighted set cover (MWSC) problem is a special case of this problem which is a well known NP-complete problem [GJ79]. Instead, we will present an approximation solution for the optimization problem below.

We transform the problem to a directed Steiner tree problem in an auxiliary graph H, defined as follows. $H = (V_H, E_H, \omega_3)$ is a directed edge-weighted graph, where $V_H = X \cup Y \cup \{s\}$ and s is a virtual vertex, $E_H = E_{XY} \cup \{\langle s, x \rangle \mid x \in X\}$, and $\omega_3(s, x) = \omega_1(x)$ for every $x \in X$, and $\omega_3(x, y) = \omega_2(x, y)$ for every $\langle x, y \rangle \in E_{XY}$. Obviously, H is a directed acyclic graph (DAG), and each other vertex in H is reachable from s. Let $S = Y$ ($= Q \subset V_H$) be the terminal set (the set consists of all query vertices). We then find a directed tree rooted at s in H including all vertices in S such that the weighted sum of the edges in the tree is minimized. It is easy to observe that this is a directed Steiner tree problem on H which is NP-hard [GJ79]. However, an approximation directed Steiner tree T rooted at s including all vertices in S can be obtained by applying the algorithm due to Zelikovsky [Zeli97]. Note that there are only two types of edges in T, one is $\langle s, x \rangle$ with $x \in X$ and another is $\langle x, y \rangle$ with $x \in X$ and $y \in Y$. We finally transform the solution on H back to a solution of our original problem. That is, for each edge $\langle x, y \rangle$ in T, query y has chosen the materialized view x as its base table, each query in Q has chosen a base table because the corresponding vertex of the query is included in T, and the weighted sum of all edges in T is the total response time for answering all the queries in Q. Now we have:

Lemma 1. *Let T_S be a directed Steiner tree of H rooted at s including all vertices in S and $W(T_S)$ be the weighted sum of the edges in T_S. Then, the minimum response time for answering all the queries in Q equals $W(T_S)$.*

Proof. Let PL be an optimal plan for all the queries in Q ($= S$), which means, evaluating this plan will lead to a minimum response time for all queries. In PL, every query $q \in Q$ has chosen a materialized view $mv(q)$ as its base table. We now build a tree T' from PL. T' has a root vertex s. We connect each materialized view which has been used by the queries in Q to s, assign the edge a direction from s to the view, and the weight to this edge as the sum of the cost of loading the view to the main memory and the cost of proceeding star-join the view with its dimension tables. Each query q connects to $mv(q)$ and assigns the edge a direction from $mv(q)$ to q and the weight of the edge by the cost of answering q. Clearly, $W(T')$ is equal to the minimum response time for answering all the queries in Q. While T' is a directed tree in H rooted at s which includes all vertices in S ($= Q$), by the definition of directed Steiner trees, $W(T_S) \leq W(T')$. Meanwhile, the Steiner tree T_S can be transformed to an assignment plan for all the queries in Q, i.e., the removal of s and all its adjacent edges from T_S will result in a forest. In this forest if query q is in a tree rooted at mv, then q will choose mv as its base table. Thus, the response time by this plan is $W(T_S)$. Note that there is no any edge between s and the vertices in S by the construction of H. Since the minimum response time for all queries is $W(T')$, so, $W(T') \leq W(T_S)$. As a result, $W(T_S) = W(T')$.

By Lemma 1, the minimum response time for our original problem now is equal to the weighted sum of the edges in the directed Steiner tree above. Since the best approximation algorithm so far for the directed Steiner tree problem delivers a solution which is $O(|S|^{\epsilon})$ times the optimal with $0 <$

$\epsilon \leq 1$. Therefore, we can give a feasible solution for our problem. That is, there is an assignment plan delivered by our algorithm, and the response time for executing this plan is $O(|Q|^{\epsilon})$ times the minimum response time, where $|X|$ is the number of elements in set X. In summary, we have the following theorem:

Theorem 1. *Let V be the set of the elements (group-bys) in the data cube of a fact table F. Given a set $MV \subset V$ of materialized views and a set Q of* `Type One Sharing` *queries, the objective is to: (i) assign every query in Q with a materialized view in MV from which the query can be answered; and (ii) minimize the total response time for all queries. There exists such an assignment plan whose response time is $O(|Q|^{\epsilon})$ times the minimum response time. The generation of the plan takes $O(|Q| \cdot |MV| + (|Q| + |MV|)^{1/\epsilon} + (|Q| + |MV|)\log(|Q| + |MV|))$ time, where $|X|$ represents the number of elements in set X and ϵ is a fixed constant with $0 < \epsilon \leq 1$.*

Proof. By the discussion above, first we construct the graph G which takes $O(|Q||MV|)$ time. The construction of H (from G) takes $O(|Q||MV| + |MV|) = O(|Q||MV|)$ time. Finding an approximation Steiner tree rooted at s including all vertices in S ($|S| = |Q|$) on H, which is ($|Q|^{\epsilon}$) times the optimal, takes $O(|Q| \cdot |MV| + (|Q| + |MV|)^{1/\epsilon} + (|Q| + |MV|)\log(|Q| + |MV|))$ time by the algorithm due to Zelikovsky [Zeli97] because H contains $|Q|+|MV|+1$ vertices and at most $|Q||MV| + |MV|$ edges. Finally, transforming this solution to a feasible solution for the original problem is straightforward which takes $O(|Q|)$ time. The theorem then follows.

Note that in practice we can use Dijkstra's single source shortest path algorithm to find a subtree rooted at s which includes all vertices in S as its leaf vertices. The tree can be found in $O(|Q||MV| + (|Q| + |MV|)\log(|Q| + |MV|)) = O(|Q||MV|)$ time. It is clear that the weighted sum of the edges in the tree is $|Q|$ times of that of the minimum Steiner tree, i.e., it is the case of Zelikovsky's algorithm when $\epsilon = 1$. From Theorem 1, we can see that the accuracy of a plan is proportional to the time spent for generating the plan, which means, the more time we spend for generating the plan, the less time required when the plan later is executed.

Despite the fact that previous approaches, including that of [ZDNS98], may generate a feasible plan for the problem, they do not provide any performance guarantee in terms of the quality of the plan generated (i.e., the plans generated by them can be arbitrarily far from the optimal one), our approach above delivers a plan which gives a deterministic bound with respect to the optimal solution.

3.3 An exact algorithm

If the number of queries $|Q|$ is small (e.g., no more than 10), we present an exact algorithm which delivers a plan with the minimum response time. In doing so, we first formalize the problem into another optimization problem. Let $Q = \{q_1, q_2, \ldots, q_{|Q|}\}$ and $MV = \{mv_1, mv_2, \ldots, mv_{|MV|}\}$. Let

$$T_{qu} = \sum_{i=1,\ldots,|Q|,j=1,\ldots,|MV|} t_{ij} x_{ij} \quad (6)$$

Associated with every query q_i, there are $|MV|$ variables $x_{i1}, x_{i2}, \ldots, x_{i|MV|}$ such that a unique $x_{ij} = 1$, and all other $x_{ij'} = 0$ if $j' \neq j$ for any j, $1 \leq j \leq |MV|$, i.e., q_i chooses the table mv_j as its base table, t_{ij} is the time for answering q_i if mv_j is already in main memory, i.e., $t_{i,j} = Size(mv_j)t_{cpu}(q_i, mv_j)$.

Let $M' = \{mv_j \mid$ if there is $x_{ij} = 1\}$ which is the set of base tables, all the materialized views in M' then must be scanned and loaded to main memory, the time for loading and star-joining for this part is

$$T_{ma} = \sum_{mv_i \in M'} Size(mv_i)/Page\_Size \cdot t_{I/O} + t_{hash\_join}(mv_i) \quad (7)$$

Therefore, the minimum response time for answering all queries in Q now is to minimize

$$T_{total\_hash} = T_{ma} + T_{qu} \quad (8)$$

We now present an exact algorithm for this optimization problem. As can be seen, associated with a query q_i there are $|MV|$ 0/1 variables $(x_{i1}, x_{i2}, \ldots, x_{i|MV|})$. By the constraints imposed in 6 there are at most $|MV|$ different assignments for these variables, i.e., $(1, 0, 0, \ldots, 0)$, $(0, 1, 0, \ldots, 0)$, …, $(0, 0, \ldots, 0, 1)$. It must be mentioned that, if q_i cannot be answered using mv_j, then the assignment of $(0, 0, \ldots, 0, 1, 0, \ldots, 0)$ with $x_{ij} = 1$ is invalid. So, the optimal solution of T_{total_hash} can be found in $O(|MV|^{|Q|})$ time.

In fact, there are at most r_i different valid assignments for a group of variables $(x_{i1}, x_{i2}, \ldots, x_{i|MV|})$, where r_i is the number of materialized views from which q_i can be answered, $1 \leq r_i \leq |MV|$. Note that r_i is usually much smaller than $|MV|$. Therefore, an assignment with the minimum response time can be found in $O(\prod_{i=1}^{|Q|} r_i)$ time. Let $\overline{r} = \sum_{i=1}^{|Q|} r_i/|Q|$, where $\overline{r}$ is the number of materialized views on average from which a query can be answered. For any k positive numbers $a_1, a_2, \ldots, a_k$, the following inequality always holds

$$\frac{a_1 + a_2 + \ldots + a_k}{k} \geq \sqrt[k]{a_1 a_2 \ldots a_k}. \quad (9)$$

By inequality 9, we have $O(\prod_{i=1}^{|Q|} r_i) \leq O(\overline{r}^{|Q|})$. Then, $O(\prod_{i=1}^{|Q|} r_i) = O(\frac{|MV|^{|Q|}}{2^{|Q|}})$ when $\overline{r} = |MV|/2$; $O(\prod_{i=1}^{|Q|} r_i) = O(|MV|^{|Q|/2})$ when $\overline{r} = \sqrt{|MV|}$. This bound is much smaller than $O(|MV|^{|Q|})$ in practice. Therefore, the approach works well when the number of queries is rather small. For example, assume that $|MV| = 1000$, $|Q| = 6$, and every query can be answered by using 32 materialized views on average. Then, the time used for finding an optimal plan takes about 10^6 unit time.

Theorem 2. *Let V be the set of the elements (group-bys) in the data cube of a fact table F. Given a set $MV \subset V$ of materialized views and a set Q of* `Type One Sharing` *queries, the objective is to: (1) assign every query in Q with a materialized view in MV from which the query can be* *answered; and (2) minimize the total response time for answering all the queries in Q. There exists an assignment plan with the minimum response time. Finding such a plan takes* $O(|MV|^{|Q|})$ *time in the worst case.*

Proof. According to our discussion above, a feasible plan is a valid assignment for all variables in Equation 6. An optimal plan is a feasible plan with minimizing T_{total_hash} in Equation **??**. However, there are $O(|MV|^{|Q|})$ valid feasible plans, choosing the one with the minimum T_{total_hash} takes $O(|MV|^{|Q|})$ time. The theorem then follows.

In practice, we believe that finding an optimal plan requires much less time than that claimed by Theorem 2. The reason is that for a query in Q, it can be answered by only a few materialized views in MV. Our experiment in Sect. 6 demonstrates that this claim is correct.

4 Multiple query optimization with Type Two Sharing

In this section we deal with the multiple dimensional query problem in which case an MDX is decomposed into a set Q of `Type Two Sharing` queries only.

4.1 Problem definition

The Multiple Dimensional Query Optimization Problem for `Type Two Sharing` is to: (1) find a materialized view (base table) from MV for every query in Q from which the query can be answered; and (2) minimize the total response time for answering all the queries in Q. Now we formalize the problem in terms of the cost model as follows:

(i) If a materialized view $mv_j \in MV$ is used by queries $q_{i_1}, q_{i_2}, \ldots, q_{i_k}$ as their base table, $q_{i_l} \in Q$, and $1 \le l \le k$, then the cost of loading those tuples of mv_j which have been indexed by the combined bitmap of the queries to main memory is

$$t_{index\_materialized}(mv_j) = \sigma_j \frac{Size(mv_j)}{Page\_Size} \cdot t_{I/O} + \sum_{l=1}^{k} t_{index}(q_{il}, mv_j) \qquad (10)$$

where σ_j is defined as follows. Let R_{il} be the set of tuples in mv_j that satisfy the conditional predicates of query q_{i_l}. Then, $|R_{il}| = \sigma_{il} \cdot Size(mv_j)$ and $0 < \sigma_{il} < 1$. The union $R' = \cup_{i=1}^{k} R_{il}$ of the sets by all the queries using mv_j will then be moved to the main memory for the purpose of query processing. Therefore, $\sigma_j = |R'|/Size(mv_j)$. $t_{index}(q_{il}, mv_j)$ is the time for searching the bitmap of q_{il}. Hence $t_{index\_materialized}(mv_j)$ is the sum of the time of loading the selected tuples of mv_j to the main memory and generating the combined bitmap from each individual bitmap of the queries (the second term in the right-hand side of expression 10).

(ii) If a query $q_i \in Q$ can be answered using $mv_j \in MV$, then q_i contributes a cost $t_{index\_query}(q_i, mv_j) = \sigma_j \cdot Size(mv_j) \cdot t_{cpu}(q_i, mv_j)$, where $t_{cpu}(q_i, mv_j)$ is the sum of the CPU time of matching a tuple in mv_j to the predicates in query q_i and the time of proceeding hashing operation for aggregation purpose.

Having the above cost measures, now for any given plan, every $q \in Q$ has already chosen a materialized view $mv(q) \in MV$ from which it can be answered finally. Therefore, if all the chosen tuples of mv_j are in the main memory, then the time taken by evaluating the plan is $T_{in\_qu}(MV, Q) = \sum_{q \in Q} t_{index\_query}(q, mv(q))$. However, by this plan the total time of loading those tuples of the materialized views to main memory, using the combined bitmap, is
$T_{in\_ma}(MV, Q) = \sum_{v \in M'} t_{index\_materialized}(v)$, where
$M' = \{mv(q) \mid q \in Q,\ mv(q) \in MV\}$.

Therefore, the response time for answering all the queries in Q is

$$T_{total\_in} = T_{in\_qu}(MV, Q) + T_{in\_ma}(MV, Q). \qquad (11)$$

The problem is then to find an assignment plan to minimize T_{total_in}, which is the minimum response time for answering all the queries in Q. Following the same argument as in Sect. 3, this version of the problem is also likely to be NP-hard. Therefore, in the following we will focus on finding an approximation solution for it.

4.2 A heuristic algorithm

Let queries $q_1, q_2, \ldots, q_k$ use $mv_j \in MV$ as their base tables, $q_i \in Q$ and $1 \le i \le k$. Then, we assume that the following inequality always holds.

$$\sum_{i=1}^{k} [\sigma_{ij} \cdot Size(mv_j)/Page\_Size \cdot t_{I/O} + \sigma_{ij} \cdot Size(mv_j) \cdot t_{cpu}(q_i, mv_j) + t_{index}(q_i, mv_j)] \ge \sigma \cdot [Size(mv_j)/Page\_Size \cdot t_{I/O} + \sum_{i=1}^{k} Size(mv_j) \cdot t_{cpu}(q_i, mv_j)] + \sum_{i=1}^{k} t_{index}(q_i, mv_j) \qquad (12)$$

where σ_{ij} is the selectivity of q_i to mv_j, $\sigma = | \vee_{i=1}^{k} F(q_i, mv_j)|/Size(mv_j)$, $\vee$ is the bit-OR operator, and $F(q_i, mv_j)$ is the bitmap of q_i to mv_j. Inequality 12 means that the cost of loading those tuples of mv_j chosen by the combined bitmap of the queries to the main memory is much cheaper than the cost of loading the tuples chosen using the bitmap of each individual query, because the bitmaps of different queries may index the same tuple. Thus, if we do not make use of the combined bitmap, then, that tuple may load many times to the main memory depending on how many queries index it. However, if we load the tuples of mv_j chosen by the combined bitmap, the cost of filtering the useless tuples for each individual query will take more CPU time. Based on the assumption in inequality 12, we present a cost estimation model in a conservative way. That is, for every query $q_i \in Q$, if it chooses a $mv_{i_j} = mv(q_i) \in MV$ as its base table, then the total cost (or response time) for q_i is

$$\sum_{i=1}^{|Q|} \sigma_{ii_j} \cdot (Size(mv_{i_j})/Page\_Size \cdot t_{I/O} + \sigma_{ii_j} \cdot Size(mv_{i_j}) \cdot t_{cpu}(q_i, mv_{i_j}) + t_{index}(q_i, mv_{i_j})). \qquad (13)$$

We now give an optimal solution for the problem based on the conservative cost model. Our approach is to reduce the problem to a minimum weighted maximum cardinality matching problem in an auxiliary weighted, bipartite graph. As a result, we obtain a feasible solution for our original problem due to inequality 12.

We first construct an undirected, weighted bipartite graph $G = (X, Y, E_{XY}, \omega_4)$ as follows. X is the set of materialized views, i.e., $X = MV$; Y is the set of Type Two Sharing queries, i.e., $Y = Q$. There is an edge $(mv_j, q_i) \in E_{XY}$ between $mv_j \in X$ and $q_i \in Y$ if q_i can be answered using mv_j, and the weight associated with the edge is

$$\begin{aligned}\omega_4(mv_j, q_i) = {} & \sigma_{ij} \cdot Size(mv_j)/Page\_Size \cdot t_{I/O} \\ & + \sigma_{ij} \cdot Size(mv_j) \cdot t_{cpu}(q_i, mv_j) \\ & + t_{index}(q_i, mv_j) \end{aligned} \qquad (14)$$

where σ_{ij} is the selectivity of q_i to mv_j.

Now, the optimization problem under the conservative cost model is to find a subset $X' \subseteq X$ in G such that every vertex in $y \in Y$ chooses a vertex $mv(y) \in X'$ and $\sum_{y \in Y} \omega_4(mv(y), y)$ is minimized. In doing so, we construct another auxiliary weighted bipartite graph $H = (X_H, Y_H, E_H, \omega_5)$ from G below.

Let $d(x)$ be the degree of vertex $x \in X$ in G. $X_H = X \cup \{x_1, x_2, \ldots, x_{d(x)-1} \mid x \in X, d(x) > 1\}$, $Y_H = Y$. $E_H = E_{XY} \cup \{(x_i, y) \mid (x, y) \in E_{XY}, 1 \le i \le d(x) - 1\}$. The weights associated with the edges in H are defined as follows: $\omega_5(x, y) = \omega_4(x, y)$ for every $(x, y) \in E_{XY}$ and $\omega_5(x_i, y) = \omega_4(x, y)$ for every $(x_i, y) \in E_H$, $1 \le i \le d(x) - 1$.

Let M' be a minimum weighted, maximum cardinality matching in H obtained, using any efficient polynomial algorithm (e.g., the algorithm by Gabow et al. [GT90]), and $W = \sum_{e \in M'} \omega_5(e)$. Then, the solution M' for H gives a corresponding solution for G which can be described as: (i) $X' = \{x' \mid x' \in X$ and $(x', y) \in M'\} \cup \{x \mid x'$ is generated from x, and $(x', y) \in M'\}$; (ii) if $(x', y) \in M'$ and $x' \in X$, then query y has chosen $mv_{x'}$ as its base table. Otherwise, $x' \not\in X$ but x' is generated from x, then, query y has chosen mv_x as its base table; and (iii) $W = \sum_{y \in Y} \omega_4(mv(y), y)$ which is the minimum among all solutions that include all vertices in Y. Thus, we have obtained such a plan $\mathscr{P}$ that, for each edge (x, y) in M', query y has chosen the materialized view x as its base table. Clearly, the solution is an optimal solution on the conservative cost model (we relax the condition). However, as mentioned before, this solution gives a feasible solution for the original problem. Now we analyze how far this feasible solution from the optimal solution of the original problem by the following theorem.

Theorem 3. *Let $\mathscr{P}$ be the plan generated by the algorithm above and let P_{opt} be an optimal plan with the minimum response time. Then, the response time of evaluating $\mathscr{P}$ is at most $|Q|$ times of that for P_{opt}.*

Proof. Without loss of generality, assume that in P_{opt}, $mv_1, mv_2, \ldots, mv_k$ have been chosen as the base tables of queries in Q where the queries indexed from 1 to i_1 use mv_1, and the queries indexed from $i_1 + 1$ to i_2 use mv_2 as their base table, ..., the queries indexed from $i_{k-1} + 1$ to n use mv_k as their base tables. We further assume that $i_l < i_{l+1}$ for all l with $1 \le l < k$. For each mv_j, let σ_j be its selectivity with respect to the queries, which is determined by the combined bitmap of those queries that use it as their base table, $1 \le j \le k$. Let $i_0 = 0$, then, by the cost model the response time for evaluating P_{opt} is

$$\begin{aligned}C_{\min} = {} & \textstyle\sum_{j=1}^{k} \sigma_i \cdot (Size(mv_j)/Page\_Size \cdot t_{I/O} \\ & \textstyle + \sum_{l=i_{j-1}+1}^{i_j} Size(mv_j) \cdot t_{cpu}(q_l, mv_j)) \\ & \textstyle + \sum_{l=i_{j-1}+1}^{i_j} t_{index}(q_l, mv_j). \end{aligned}$$

Now we consider mv_j, $1 \le j \le k$, by our assumption that the queries indexed from $i_{j-1} + 1$ to i_j use mv_j as their base tables, for each query q_l with $i_{j-1} + 1 \le l \le i_j$, there is a corresponding edge (mv_j, q_l) in G (defined before), and the weight associated with this edge in G is $\omega_4(mv_j, q_l)$ which is defined in Eq.14. It is obvious that

$$\begin{aligned}\omega_4(mv_j, q_l) \le {} & \sigma_i \cdot (Size(mv_j)/Page\_Size \cdot t_{I/O} \\ & + \sum_{l=i_{j-1}+1}^{i_j} Size(mv_j) \cdot t_{cpu}(q_l, mv_j)) \\ & + \sum_{l=i_{j-1}+1}^{i_j} t_{index}(q_l, mv_j) \end{aligned} \qquad (15)$$

We now construct a subgraph $G' = (X', Y, E'_{XY}, \omega_4)$ of G where $X' = \{mv_1, mv_2, \ldots, mv_k\}$, and an edge $(mv_j, q_l) \in E'_{XY}$ for all $1 \le j \le k$ and $i_{j-1+1} \le l \le i_j$. Then, using the construction rule of H, it is easy to see that the corresponding subgraph in H of G' is a weighted matching M'' which includes all the vertices in Y. The weight of M'' is $\sum_{e' \in M''} \omega_5(e) = \sum_{(mv_j, q_l) \in E'_{XY}} \omega_4(mv_j, q_l)$. Thus, we have $\sum_{e' \in M''} \omega_5(e') \le D \cdot C_{\min}$, where $D = \max\{i_j - i_{j-1} \mid 1 \le j \le k\} \le |Q|$. However, our algorithm is to find a minimum weighted maximum cardinality matching M' in H, which means $\sum_{e \in M'} \omega_5(e) \le \sum_{e' \in M''} \omega_5(e') \le DC_{\min}$. Therefore, the feasible solution for the original problem is at most $|Q|$ times the optimal.

In summary, we have the following theorem.

Theorem 4. *Let V be the set of the elements (group-bys) in the data cube of a fact table F. Given a set $MV \subset V$ of materialized views and a set Q of* Type Two Sharing *queries, the objective is to: (i) assign every query in Q with a materialized view in MV from which the query can be answered; and (ii) minimize the total response time for answering all queries in Q. There exists an assignment plan within $O(|Q|)$ of optimal, and this plan can be generated in time $O(\Delta|MV||Q| \log(N \cdot C)\sqrt{N\alpha(\Delta|MV||Q|, N) \log N})$, where $N = \Delta|MV| + |Q|$, $C = \max\{\omega_5(x, y) \mid (x, y) \in E_H\}$, and Δ $(\le |Q|)$ is the maximum number of queries that can be answered by a materialized view in MV.*

Proof. Following our discussion above, we know that the response time of the plan delivered by our algorithm is $|Q|$ times the optimal by Theorem 3. The dominant computational complexity of the proposed algorithm is to find a minimum weighted maximum cardinality matching which requires

$$O(\Delta|MV||Q| \log(N \cdot C)\sqrt{N\alpha(\Delta|MV||Q|, N) \log N})$$

time by the algorithm due to Gabow and Tarjan [GT90] because H contains no more than N (= $\Delta|MV|$ + $|Q|$) vertices and $|\Delta|MV||Q|$ edges in the worst case, where C = $\max\{\omega_5(x,y) \mid (x,y) \in E_H\}$ and Δ is the maximum degree of vertices in H. The theorem then follows.

4.3 An exact algorithm

Following the same framework of the exact algorithm described as in Sect. 3, we now give an exact solution for the problem in this version. The technique used here is similar to the one used in Sect. 3 except the formulae. So, we will focus on the formulas of the problem. We transform the problem into another optimization problem. Recall that $Q = \{q_1, q_2, \ldots, q_{|Q|}\}$ and $MV = \{mv_1, mv_2, \ldots, mv_{|MV|}\}$. Let

$$T_{index-qu} = \sum_{i=1,\ldots,|Q|,j=1,\ldots,|MV|} t_{ij}x_{ij} \qquad (16)$$

Associated with every query q_i, there are $|MV|$ variables $x_{i1}, x_{i2}, \ldots, x_{i|MV|}$ such that a unique x_{ij} = 1, and all other $x_{ij'}$ = 0 if $j' \neq j$ for any j, $1 \leq j \leq |MV|$, i.e., q_i chooses the table mv_j as its base table. t_{ij} is the time for answering q_i, assuming those tuples of mv_j chosen by the combined bitmap of queries are already in main memory, i.e.,

$$t_{i,j} = \sigma_j \cdot Size(mv_j) t_{cpu}(q_i, mv_j) \qquad (17)$$

where $\sigma_j = | \vee_{x_{ij}=1} F(q_i, mv_j)|/Size(mv_j)$ and $F(q,v)$ is the bitmap of q to v. But, the time of loading those tuples of mv_j by the combined bitmap to main memory is

$$T_{index\_ma} = \sum_{mv_i \in M'} \sigma_i \cdot Size(mv_i)/Page\_Size \cdot t_{I/O} + \sum_{x_{ij}=1} t_{index}(q_i, mv_j) \qquad (18)$$

where $M' = \{mv_j \mid \text{if there is } x_{ij} = 1\}$. Thus, the problem that we are concerned is now to minimize

$$T_{index} = T_{index\_ma} + T_{index\_qu}. \qquad (19)$$

By applying the technique in Sect. 3, the above optimization problem can be solved in $O(\prod_{i=1}^{|Q|} r_i)$ time in the worst case, where r_i is the number of materialized views from which q_i can be answered. Therefore, we have:

Theorem 5. *Let V be the set of the elements (group-bys) in the data cube of a fact table F. Given a set $MV \subset V$ of materialized views and a set Q of* `Type Two Sharing` *queries, the objective is to: (1) assign every query in Q with a materialized view in MV from which the query can be answered; and (2) minimize the response time for answering all queries in Q. There is an assignment plan with the minimum response time. Finding such a plan takes $O(|MV|^{|Q|})$ time in the worst case.*

Proof. The proof approach is similar to that for Theorem 2, omitted.

5 Multiple query optimization with Type Three Sharing

In the previous two sections we assume that an MDX can be decomposed into a set Q of queries, which are either `Type One Sharing` or `Type Two Sharing` but not both. In practice, the queries in Q are a mixture of the two types, namely, `Type Three Sharing` queries. In this section we will focus on how to process multiple queries of `Type Three Sharing` efficiently.

5.1 Problem definition

The Multiple Dimensional Query Optimization Problem for `Type Three Sharing` is to: (1) find a materialized view (base table) from MV for every query in Q from which the query can be answered; and (2) minimize the total response time for answering all the queries in Q.

By the previous discussion, if the set Q can be further partitioned into dis-joint subsets Q_1 and Q_2 such that Q_1 only contains the queries of `Type One Sharing` and Q_2 only contains the queries of `Type Two Sharing`, i.e., $Q = Q_1 \cup Q_2$, then two sub-plans for Q_1 and Q_2 can be found using the approximation algorithms developed in Sects. 3 and 4, respectively. Thus, a plan for Q is formed by merging the two sub-plans, which is sub-optimal. But it relies on the assumption that the queries in Q can be properly partitioned into the two subsets Q_1 and Q_2. However, for a query in Q, it is very difficult to decide to which subset it should bebeforehand. In the following we will suggest an algorithm which gives a plan with a feasible solution by exploiting data sharing further. It must be mentioned that this problem has been originally discussed by [ZDNS98]. Here, we first analyze the algorithms of Zhao et al., by pointing out some of drawbacks of the algorithms and providing some possible improvement. We then present another greedy algorithm for the problem. Unlike their algorithms, our algorithm does not require any given order of the queries added to the plan in advance. We finally design an exact algorithm which can find an optimal plan if the number of queries in Q is relatively small.

5.2 Improvements of the algorithms of Zhao et al

In order to obtain a better plan for a set of queries of `Type Three Sharing`, Zhao et al. [ZDNS98] give the following three heuristic algorithms, by utilizing the three operations defined in Sect. 2: Two Phase Local Optimal Algorithm (`TPLO`), Extended Two Phase Local Greedy Algorithm (`ETPLG`), and Global Greedy Algorithm (`GG`). For the sake of completeness, we briefly introduce these three algorithms below.

Algorithm `TPLO` is fairly simple, consisting of two phases. In the first phase, it independently picks up a base table (materialized view) from MV for each query in Q such that the cost for evaluating that query is minimal. Once the materialized view for a query is determined, it uses the SQL optimizer to generate the best plan for the query. In the second phase, it generates a global plan by merging the common

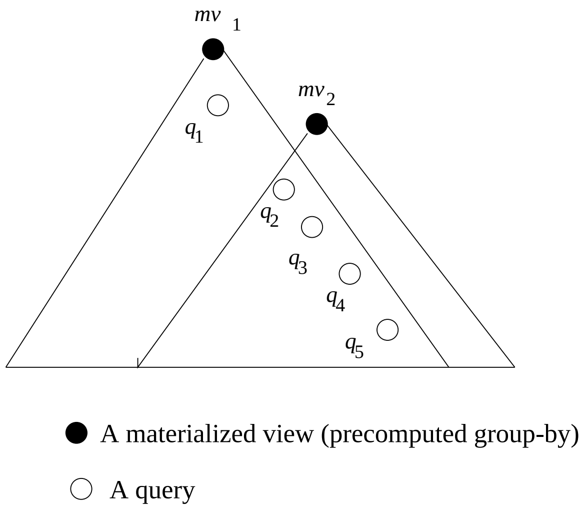


Fig. 3. An example

subtasks among individual query plans using the three operators. However, for any non-trivial optimization problem, it is usually impossible to achieve a global optimal solution by simply ”merging” individual local optimal solutions. Algorithm ETPLG explores sharing base tables (materialized views) among queries to reduce the total response time for the queries. The basic idea of ETPLG is to add queries to the global plan one by one. When a new query is added to the plan, ETPLG chooses between the best plan that allows the query to share a base table which has been used by the other queries in the plan, and the local plan for the query. The second phase is just like the second phase of Algorithm TPLO, merging the query plans at the vertex level of the query tree. So, the order of the queries added to the global plan plays a crucial role in the evaluation of the plan in terms of cost. In ETPLG, queries are added to the global plan by their *GroupbyLevel* in ascending order, where *GroupbyLevel* is the sum of the hierarchy level of the group-by on each dimension. Intuitively, the smaller the *GroupbyLevel* of a query, the more likely it shares a base table with the other queries. Algorithm GG is similar to Algorithm ETPLG, in that, GG develops the global plan by adding queries to the plan one by one, and the adding order is in ascending order of *GroupbyLevel*. The only difference between them is that GG allows some already assigned queries to change their shared base table in order to include the new query which is not allowed in ETPLG. As their performance study indicated, GG outperformed the other two.

Now let us consider the following running example (see Fig. 3), where the triangle area indicates that all queries inside can be answered by the base table (materialized view) forming the triangle. Suppose we have $MV = \{mv_1, mv_2\}$ with $Size(mv_1) = 15000$ and $Size(mv_2) = 3000$. $Q = \{q_1, q_2, q_3, q_4, q_5\}$. Assume that q_i's *GroupbyLevel* is strictly smaller than q_{i+1}'s, $1 \leq i \leq 4$, every q_i can use mv_1 as its base table, $1 \leq i \leq 5$. q_2, q_3, q_4 and q_5 except q_1 can use mv_2 as their base tables. We further assume that all the queries are using `shared scan for hash-based star join`. Assume that the sum of the time of loading mv_1 to main memory and star-joining takes 1500 unit time and mv_2 takes 300 unit time respectively.

It is obvious that the time for answering a query q_i depends on the number of tuples in its base table when its base table has been in main memory. Here we assume that it takes one unit time per processing 100 tuples. Now we run Algorithm ETPLG or GG for the example. We add the queries to the global plan S by the order $q_1, q_2, \ldots, q_5$. First we add q_1 into S, the total cost is 1500 + 15000/100 = 1650 unit time because loading and star-joining mv_1 takes 1500 unit time by the assumption, and the CPU processing takes 150 unit time. Next we consider q_2. If q_2 chooses mv_1 as its base table, the cost is 15000/100 = 150 unit time; if q_2 chooses mv_2 as its base table, the cost is 300 + 3000/100 = 330 unit time. So, by ETPLG or GG q_2 finally chooses mv_1 as its base table. Third, we consider adding q_3 to S. Following the same analysis for q_2, q_3, q_4 and q_5 will finally choose mv_1 as their base tables. So, the total cost of running the global plan is 1650 + 4 × 150 = 2250 unit time. However, the optimal plan for this example is that q_1 chooses mv_1, q_2 to q_5 choose mv_2 as their base tables, and the total cost of running the optimal plan is 1650 + 330 + 3 × 30 = 2070 unit time.

The reason is that the order of queries added to the global plan plays an important role in terms of the cost of evaluating the plan. From this example we can see that it might not be a good choice to add the queries to the global plan by the ascending order of *GroupbyLevel* because in some cases it may not be a good heuristic. Consequently, we may have an immediate alternative, i.e., adding the queries to the global plan by the descending order of *GroupbyLevel*. For this latter heuristic, we can also give an example against it. As a balance, there is a third heuristic which adds the queries to the global plan in the following order: if $|Q|$ is odd, then the order is $q_{\lfloor |Q|/2 \rfloor+1}, q_{\lfloor |Q|/2 \rfloor}, q_{\lfloor |Q|/2 \rfloor+2}, \ldots, q_1, q_{|Q|}$; otherwise, the order is $q_{|Q|/2}, q_{|Q|/2+1}, q_{|Q|/2-1}, q_{|Q|/2+2}, \ldots, q_1, q_{|Q|}$, assuming that $q_1, q_2, \ldots, q_{|Q|}$ is the query sequence by the ascending order of *GroupbyLevel*. The algorithm based on the third heuristic will deliver a better solution on average. The reason is explained as follows. For n queries, there are $n!$ different ways to give the queries in order, and it is possible that there is a case which has the optimal solution for each of the orders. The ascending and descending orders of *GroupbyLevel* heuristics are the two extremes of the spectrum of a variety of heuristics based on the order of *GroupbyLevel*, while the third heuristic above is located in the middle of the spectrum of among the heuristics. Thus, there exists an algorithm which delivers a better plan (smaller response time) than that delivered by algorithm ETPLG or GG in the general case, and the time for finding such a plan is at most three times of the time of ETPLG or GG.

5.3 A cost-based greedy algorithm

Unlike the algorithm of Zhao et al., the proposed algorithm does not set the order for queries to be added. The algorithm proceeds by adding queries to the global plan one by one, according to the cost incurred by adding that query to the solution at the current stage. So, the algorithm takes $|Q|$ times iterations. Let AQ be the set of queries which have not been assigned base tables. Suppose that the first $i-1$ queries have been assigned to their base tables. In the beginning of

```
Algorithm 1: Find_Query_Plan($MV, Q$);
$MV$ := $\{mv_1, mv_2, \ldots, mv_{|MV|}\}$;
$Q$ := $\{q_1, q_2, \ldots, q_{|Q|}\}$;
$AQ$ := $Q$;/* the set of unassigned queries */
$Cost$ := 0; /* the total cost for answering all queries in $Q$ */
for $i$ := 1 to $|MV|$ do
    $IX(mv_i)$ := $\emptyset$; /* the set of index-based queries used $mv_i$ as the base table */
endfor;
repeat
    $\Delta\_cost$ := $\infty$; /* the cost incurred by adding the $i$-th query to the solution */
    $q\_candidate$ := $\emptyset$; /* the $i$-th query will be q_candidate */
    for every $q_x \in AQ$ do
        for every $mv_y \in MV$ do
            if $q_x$ can be answered using $mv_y$ then
              if $mv_y$ is chosen in the first time then
                if $q_x$ is hash-based scan join with $mv_y$ then
                $temp$ := $\frac{Size(mv_y)}{Page\_Size} \cdot t_{I/O} + t_{hash\_join}(mv_y) + Size(mv_y) \cdot t_{cpu}(q_x, mv_y)$
                else /* $q_x$ is index-based join */
                $temp$ := $\sigma_{xy} \cdot \frac{Size(mv_y)}{Page\_Size} \cdot t_{I/O} + t_{index}(q_x, mv_y) + \sigma_{xy} \cdot Size(mv_y) \cdot t_{cpu}(q_x, mv_y)$
                endif
              else /* $mv_y$ is not chosen in the first time */
                if $mv_y$ has been used by other hash-based queries then
                  if $q_x$ is hash-based scan join then
                  $temp$ := $Size(mv_y) \cdot t_{cpu}(q_x, mv_y)$
                  else $temp$ :=Delta_Cost($q_x, IX(mv_y)$)
                  endif
                else if $mv_y$ has been used by only index-based queries then
                    if $q_x$ is hash-based scan join then
                    $temp$ := $\frac{Size(mv_y)}{Page\_Size} \cdot t_{I/O} + t_{hash\_join}(mv_y) + Size(mv_y) \cdot t_{cpu}(q_x, mv_y)$
                    else $temp$ :=Delta_Cost($q_x, IX(mv_y)$)
                    endif
                  endif
                endif
              endif
            endif;
            if $\Delta\_cost > temp$ then
              $\Delta\_cost$ := $temp$;
              $q\_candidate$ := $q_x$;
              $BT(q_x)$ := $mv_y$; /* $q_x$ uses $mv_y$ as its base table */
            endif
        endfor
    endfor;
    if $q\_candidate$ is index-based then
        $IX(BT(q\_candidate))$ := $IX(BT(q\_candidate)) \cup \{q\_candidate\}$
    endif;
    $Cost$ := $Cost + \Delta\_cost$;
    $AQ$ := $AQ - \{q\_candidate\}$
until $AQ = \emptyset$.
```

Fig. 4. A greedy algorithm for type three queries

the i-th iteration, a query q in AQ will be added to the global plan if its addition will incur the minimum increase in the total cost of the solution. The detailed algorithm is described in Fig. 4.

The function Delta_Cost is defined as follows see top of next page). Applying our algorithm to the example in Fig. 3, an optimal plan for it can then be obtained.

5.4 The exact algorithm

Following the same framework as described in Sects. 3 and 4, we give an exact solution for the problem by formalizing it as an optimization problem.

Let

$$T_{qucpu} = \sum_{i=1,\ldots,|Q|,j=1,\ldots,|MV|} t'_{ij} x_{ij}. \tag{20}$$

Associated with every query q_i, there are $|MV|$ variables $x_{i1}, x_{i2}, \ldots, x_{i|MV|}$ in which there is a unique $x_{ij} = 1$, and all other $x_{ij'} = 0$ if $j' \neq j$ for any j, $1 \leq j \leq |MV|$. For these variables x_{ij} with $1 \leq j \leq |MV|$, there are r_i different assignments, i.e., $(0, 0, \ldots, 0, 1, 0, \ldots, 0)$ with $x_{ij} = 1$ is a valid assignment if q_i uses mv_j as its base table, where r_i is the number of materialized views from which q_i can be answered. t'_{ij} is the time for answering q_i which is defined as follows. If q_i is index-based join with mv_j, then

$$t'_{i,j} = \sigma_{ij} \cdot Size(mv_j) \cdot t_{cpu}(q_i, mv_j) \tag{21}$$

```
Function Delta_Cost($q, IX(v)$);
begin
    /* $IX(v)$ is a set of index-based queries which use $v$ as their base table */
    $\sigma_{xy}^{old} := \frac{|\vee_{q' \in IX(v)} F(q',v)|}{Size(v)}$;
    $\sigma_{xy}^{new} := \frac{|\vee_{q' \in IX(v)} F(q',v) \vee F(q,v)|}{Size(v)}$;
    $\Delta\sigma_{xy} := \sigma_{xy}^{new} - \sigma_{xy}^{old} \quad (\leq \sigma_{xy})$;
    $Delta\_Cost := \Delta\sigma_{xy} \cdot \frac{Size(v)}{Page\_Size} \cdot t_{I/O} + t_{index}(q,v) + \sigma_{xy}^{new} Size(v) t_{cpu}(q,v)$
                $+ \Delta\sigma_{xy} \sum_{q' \in IX(v)} Size(v) t_{cpu}(q',v)$
end
```

where σ_{ij} is the selectivity of q_i to mv_j, $0 < \sigma_{ij} < 1$. Otherwise (q_i is hash-based join),

$$t'_{i,j} = Size(mv_j) \cdot t_{cpu}(q_i, mv_j) \tag{22}$$

T_{qucpu} is the time for processing the queries, assuming that the required tuples of materialized views have been in main memory.

Let $M' = \{mv_j \mid \text{if there is } x_{ij} = 1 \text{ in expression 20}\}$. We now calculate the cost $C_{load}(mv_j)$ of loading an mv_j to main memory and proceeding star-join. Assume that $q_{i_1}, q_{i_2}, \ldots, q_{i_k}$ are associated with mv_j, i.e., $x_{i_l j} = 1$ with $1 \leq l \leq k$. Then:
(i) if all q_{i_l} are hash-based join, then

$$C_{load}(mv_j) = Size(mv_j)/Page\_Size \cdot t_{I/O} + t_{hash\_join}(mv_j) \tag{23}$$

(ii) if all q_{i_l} are index-based join, then

$$C_{load}(mv_j) = \sigma_j \cdot Size(mv_j)/Page\_Size \cdot t_{I/O} + \sum_{l=1}^{k} t_{index}(q_{il}, mv_j) \tag{24}$$

where σ_j is defined as follows. Let R_{il} be the set of the tuples in mv_j chosen by the bitmap of query q_{i_l}. Then, $|R_{il}| = \sigma_{il} \cdot Size(mv_j)$. Then the union R' of the sets by the queries using mv_j will be moved to main memory, i.e., $R' = \cup_{i=1}^{k} R_{il}$. Therefore, $\sigma_j = |R'|/Size(mv_j)$.

Remarks. In practice before executing the query, R_{il} is not known, so, there is no way to give an exact value for σ_j. Instead we assign σ_j an approximate value. Clearly, $\max\{\sigma_{lj} \mid 1 \leq l \leq k\} \leq \sigma_j \leq \sum_{l=1}^{k} \sigma_{lj}$, and $\sigma_j \approx \frac{\sigma_{i1} + \sigma_{i2} + \ldots + \sigma_{ik}}{k}$.
(iii) otherwise, assume that the k queries are associated with mv_j in which the first r queries $q_{i_1}, q_{i_2}, \ldots, q_{i_r}$ are index-based join and the remaining $k - r$ queries $q_{i_{r+1}}, \ldots, q_{i_k}$ are hash-based join. Let $R'' = \cup_{l=1}^{r} R_{il}$. Then

$$C_{load}(mv_j) = (1 + \sigma'_j) \cdot Size(mv_j)/Page\_Size \cdot t_{I/O} + \sum_{l=1}^{r} t_{index}(q_{il}, mv_j) \tag{25}$$

where $\sigma'_j = |R''|/Size(mv_j)$.

The original problem now becomes to minimize

$$\sum_{mv \in M'} C_{load}(mv) + T_{qucpu}. \tag{26}$$

Following the approach developed in Sect. 3, we know that there are $\prod_{i=1}^{|Q|} r_i$ different valid assignments for expression 20. We then find such an assignment that minimizes expression 26, and this assignment is an optimal plan.

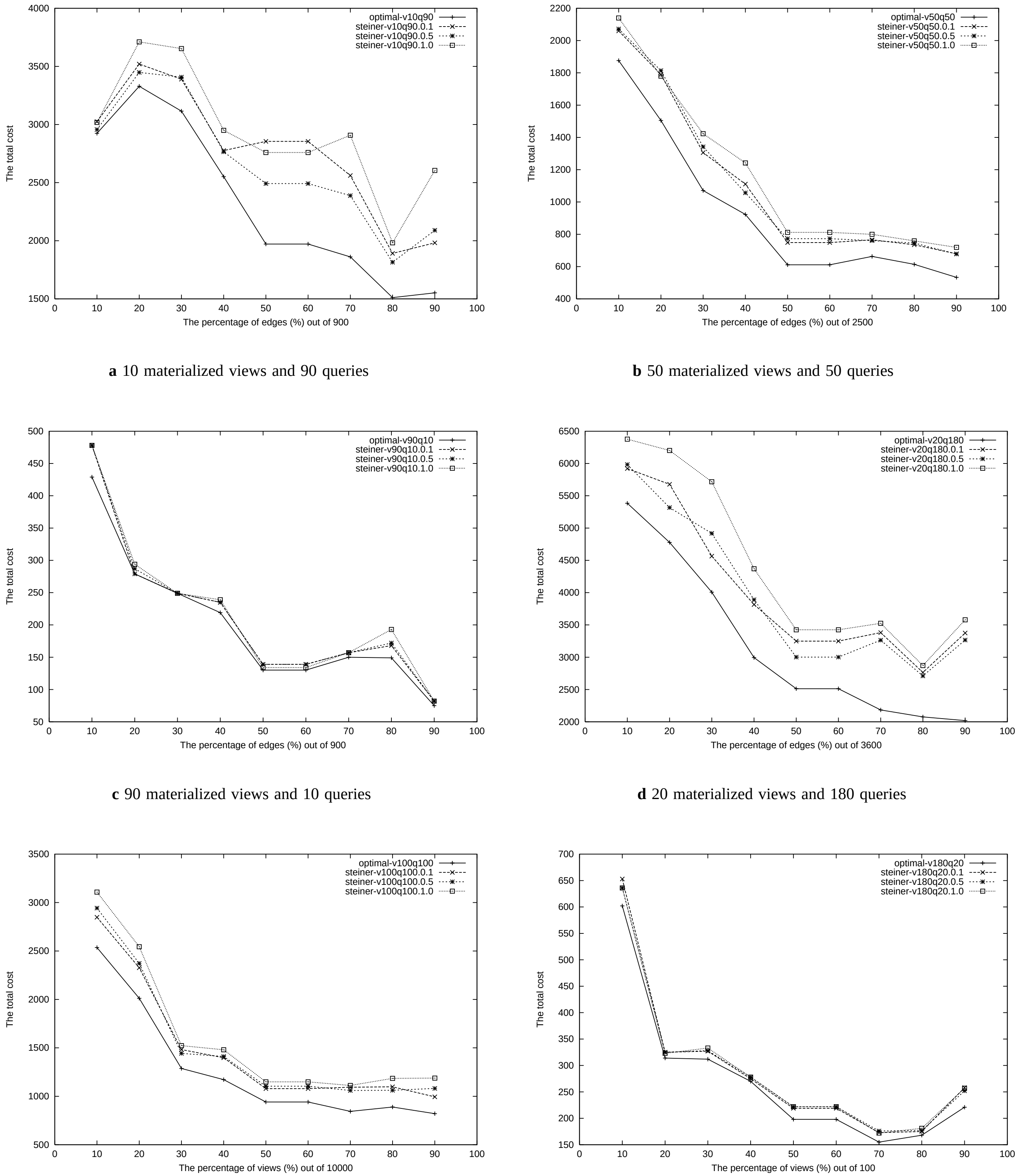
6 Performance study

In this section we conduct experiments to demonstrate the efficiency of the proposed algorithms. First, we describe how data is generated. We use `Stanford GraphBase` developed by Knuth [Knuth93] to generate random bipartite graphs. In the Stanford GraphBase, the procedure `random_bigrahp`(n_1, n_2, m, $multi$, d_1, d_2, $minlen$, $maxlen$, $seed$) is designed to produce a pseudo-random bipartite graph with $n_1 + n_2$ vertices, in which the n_1 vertices correspond to the n_1 materialized views and the other n_2 vertices correspond to the n_2 queries. Also, this graph contains m edges. There is an edge between a materialized view vertex and a query vertex if the query can be answered using the view. The parameter $multi$ permits duplicate edges if $multi \neq 0$. The d_1 and d_2 specify probability distributions on the edges. The $minlen$ and $maxlen$ bound the weights on the edges which will be uniformly distributed. The pseudo-random number used in this procedure is the $seed$. In our study, we choose $mult$ to be zero. We assume that the weights assigned to edges are uniformly distributed. The $minlen$ and $maxlen$ are 1 and 100, respectively. Here the weight associated with an edge between a materialized view vertex and a query vertex is the cost of processing the query. In addition to generating the bipartite graph, for `Type One Sharing` case, we also introduce a *virtual vertex* and the edges between the virtual vertex and the materialized view vertices. Weights (for hash-based star-join cost plus the cost of loading the materialized view to main memory) are assigned to the edges between the virtual vertex and materialized view vertices.

6.1 Type one sharing

Now we consider the `Type One Sharing` case. As we know, the problem can be reduced to a directed Steiner tree problem on a directed graph. The graph can be generated using the above approach.

We implement Zelikovsky's algorithm on the graph which is outlined below. In each iteration, we pick up a subset of terminals (queries) using a fixed factor ϵ, $0 < \epsilon \leq 1$. In theory, the smaller the ϵ is, the more accurate approximation solution Zelikovsky's algorithm can provide. The



a 10 materialized views and 90 queries

b 50 materialized views and 50 queries

c 90 materialized views and 10 queries

d 20 materialized views and 180 queries

e 100 materialized views and 100 queries

f 180 materialized views and 20 queries

Fig. 5. The approximation and exact solutions for `type one sharing` queries

Zelikovsky's Algorithm

Input: a graph $G = (V, E, \omega)$ where V_Q is the set of terminal vertices (query vertices)

Output: a directed approximation Steiner tree $\mathcal{T}_S$

```
begin
    $G' := G$; $S_m := V_Q$;
    $i := 0$;
    while $S_m \neq \emptyset$ do
        Let $S'$ be a subset of $S_m$ where $|S'| = |S_m|^\epsilon$ for $0 < \epsilon \leq 1$.
        Find an approximation directed Steiner tree $T_i$ rooted at the virtual vertex which contains
        the terminals in $S'$, using Dijkstra's shortest path algorithm.
        Let $G'$ be the graph by contracting all edges in $T_i$ from $G'$;
        $i := i + 1$;
        $S_m := S_m - S'$;
    endwhile
    The approximation Steiner tree of $G$ is obtained by recovering all $T_i$.
end
```

exact solution can be obtained by using a linear programming package called `lp_solve` developed by Berkelaar and Dirks [BD99]. Figure 5 shows the results when n vertices are used for both views and queries where $n = 100$ in Fig. 5a–c; and $n = 200$ in Fig. 5d–f. The x-axis is the percentage of edges out of number-of-views multiplied by number-of-queries which indicates how many materialized views can be chosen potentially by each query on average. The y-axis is the solution, i.e., the total cost or the response time to answer all queries. In Fig. 5, we investigate the following four different approaches:

1. The exact solution,
2. The approximation Steiner tree solution with $\epsilon = 1.0$,
3. The approximation Steiner tree solution with $\epsilon = 0.5$,
4. The approximation Steiner tree solution with $\epsilon = 0.1$,

after considering the following three cases:

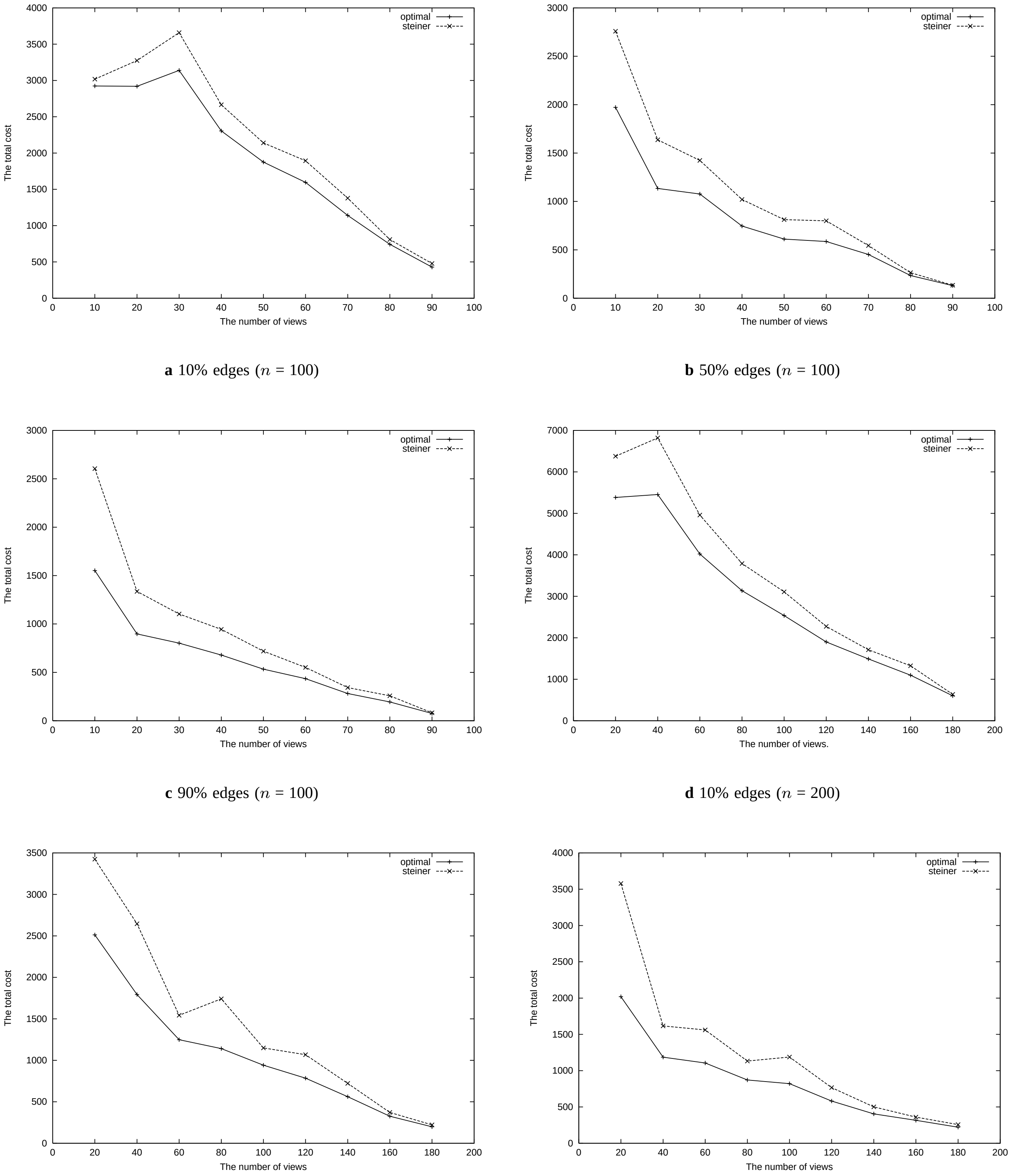
- A small number of materialized views with a large number of queries (Fig. 5a and d)
- The same numbers of materialized views and queries (Fig. 5b and e)
- A large number of materialized views with a small number of queries (Fig. 5c and f)

Figure 5a shows a case with 10 materialized views from which 90 queries can be answered. The maximum number of edges is 900 (= 10 × 90). The exact solutions show that the more edges a graph has, the cheaper cost the algorithm can achieve. This confirms that the chances of a query sharing a materialized view and the dimension tables with the other queries will increase, which will lead to save the hash-based star join time and the time of loading the shared materialized view to main memory. When there are only a small number of edges (10%), all the four approaches give the similar outcome. This is because there are only a few materialized views that can be shared among the queries. While the number of edges between the materialized views and the queries increase, the gap between the exact solution and the approximation solution obtained by using the Steiner tree approach increases as well. This phenomenon can be explained as follows. Each query can be answered by more materialized views than used to be (compared with fewer edges between the queries and the materialized views), so, the choosing strategies of materialized views can play a major role in the final solutions. But in this case, we can see that the solutions delivered by the Steiner tree approach are always within twice of the exact solution. Figure 5b shows a case where the numbers of materialized views and queries are the same; 50. When there are more materialized views to be selected for a query, it shows that the Steiner tree solutions will perform equally well regardless of the number of edges involved in the graphs. When there are a large number of materialized views to be selected as seen in Fig. 5c, the differences between the exact solutions and the approximation Steiner tree solutions are even smaller. The similar results can be seen from Fig. 5d–f when the total number of vertices for materialized views and queries is 200. One observation coming from our experimental study is that it seems not always true that, for the approximation Steiner tree solution, the more time it spends to find a solution, the more accurate solution it will deliver. One possible explanation is that in the most cases the approximation algorithm can find a nearly exact solution. So, even we further reduce ϵ (therefore increasing the time to find the solution), there is not much effect on the solution obtained.

In Fig. 6 we also use $n = 100$ for parts a–c; and $n = 200$ for parts d–f. Unlike Fig. 5, we increase the number of materialized views while fixing the number of edges as x% out of the total number of edges (the number of materialized views multiplied by the number of queries). The x-axis represents the number of materialized views. The number of queries will be the total number of vertices minus the number of materialized views. Three different cases are considered:

- A small number of edges, 10%, (Fig. 6a and d).
- A medium sized number of edges, 50%, (Fig. 6b and e).
- A large number of edges, 90%, (Fig. 6c and f).

As can be seen from Fig. 6, the approximation Steiner tree solution is within twice of the exact solution. When the number of materialized views is large, the approximation Steiner tree approach provides a solution which is near to the exact solution. The above experiments have demonstrated that the solutions obtained by our algorithms are always within twice of the exact solution, which is much better than our theoretical estimate of $|Q|^\epsilon$ times the optimal. This indicates that our estimate is very conservative. The performance of the proposed algorithm is excellent.



a 10% edges (n = 100)

b 50% edges (n = 100)

c 90% edges (n = 100)

d 10% edges (n = 200)

e 50% edges (n = 200)

f 90% edges (n = 200)

Fig. 6. The approximation and exact solutions for `type one sharing` queries with fixed edge ratios

6.2 Type two sharing

In the following we describe some experimental results for `Type Two Sharing` queries. The exact solution is obtained with the same linear programming package. For this case we reduce the original problem to a minimum weighted maximum cardinality matching problem on a bipartite graph.

First, we consider the selection of a set of materialized views in terms of the query processing and star join cost individually. If a materialized view can be used to answer m queries, we make m copies of that materialized view. Each copy will be used to answer a single query. In other words, before making a copy, a materialized view vertex might have m outgoing edges which means that there are m queries which can be answered using this view. After copying, there exist m copied materialized views vertices, and each of them has one and only one outgoing edge. Second, we find a minimum weighted maximum cardinality matching on the resulting graph after the copy operation applied. Third, we merge all those materialized view vertices to the vertex from which the vertices have been made. Finally, we find an approximation solution (the total cost) of the problem.

Figures 7 and 8 show the results when n vertices are used for both the materialized views and queries where n = 100 in parts a–c; and n = 200 in parts d–f. In Fig. 7, the x-axis is the percentage of edges out of number-of-materialized views multiplied by number-of-queries. In Fig. 8, we increase the number of materialized views while fixing the number of edges as x% out of the total number of edges (the number of materialized views multiplied by the number of queries). As can be seen from these figures, in spite of using different technique – the minimum weighted maximum cardinality matching for `Type Two Sharing`, the similar observations can be made as we did for `Type One Sharing` using Steiner tree solutions. In practice, the solutions obtained by the minimum weighted maximum cardinality matching are within twice the exact solutions in all the cases we investigated, which is much better than our theoretical estimate of $|Q|$ times the exact solutions.

6.3 Type three sharing

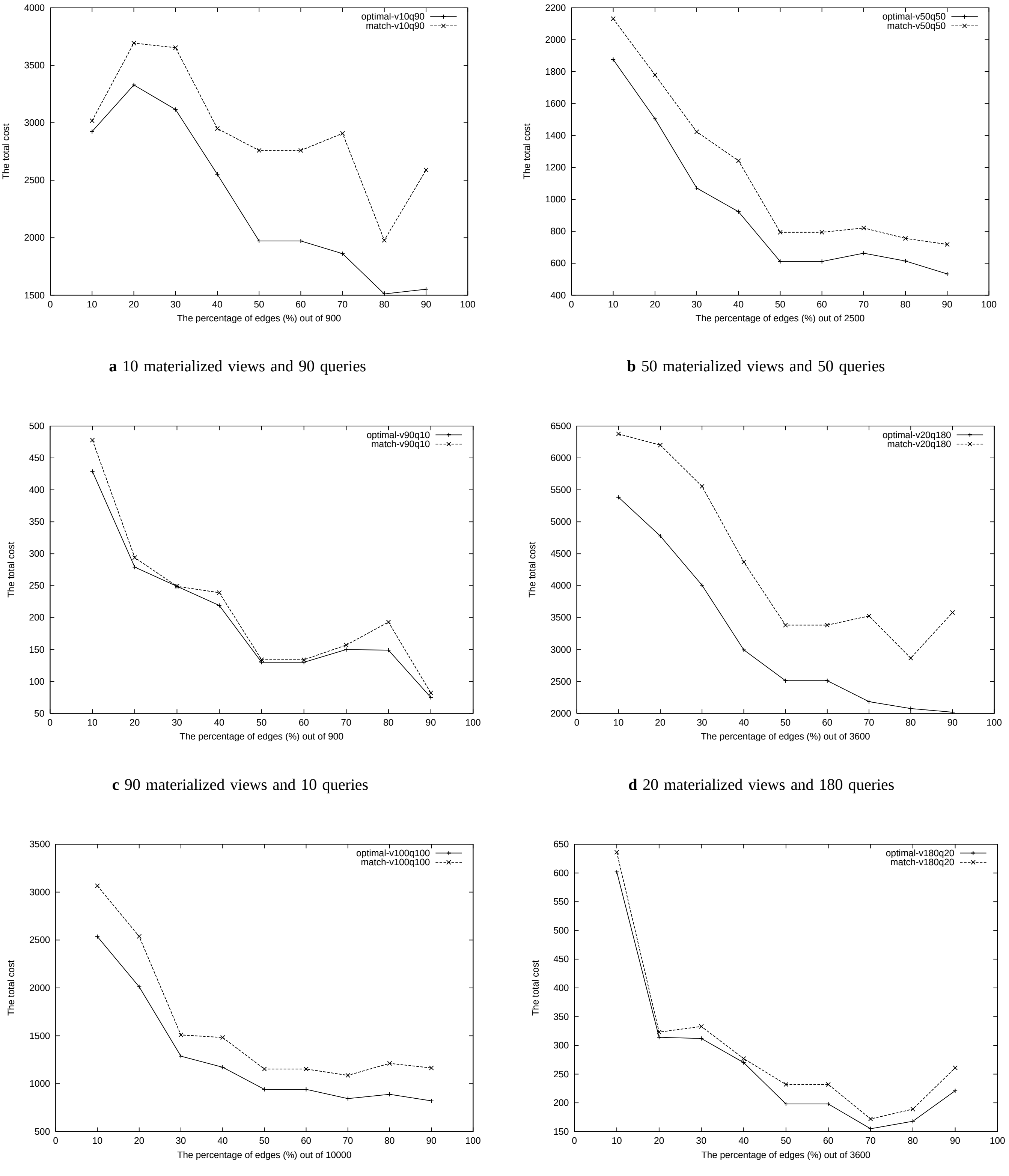
As for `Type Three Sharing`, we also conducted an extensive performance study. We compared our Steiner tree solution with the two global greedy algorithms, namely, GG-o for order-based-on *GroupbyLevel* (the algorithm of Zhao et al.) and GG-c for order-based-on cost (our algorithm for this type in this paper).

Figure 9 shows the results when the total 100 vertices are sued for both materialized views and queries. The x-axis is the percentage of edges out of number-of-views multiplied by number-of-queries. We also assume that not all queries use index-based join only and 50% of queries that can use hash-based join. The costs of using a hash-based join are randomly selected in the range between 80 and 100. In this experimental study, we consider two different scenarios. First, we assume that the index costs are fairly small – in the range between 1 and 20. The results are shown in Fig. 9a–c. When there are a small number of materialized views to be selected Fig. 9a, the Steiner tree solution outperformed the solutions delivered by the two global greedy algorithms. But when there are many materialized views to be selected, as shown in Fig. 9b and c, the solutions given by the two global greedy algorithms outperform the Steiner tree solution due to the number of choices and the small range of index costs. Second, we assume that index costs are considerably large – in the range of 20 – 40. The results are shown in Fig. 9d–f. In this case, the Steiner tree solution outperformed the solutions by the two global greedy approaches in most of cases. From Fig. 9, we can see the following facts. When there are a small number of materialized views and a large number of queries, GG-o has better performance than GG-c. When there are a large number of materialized views and a small number of queries, GG-c has better performance than GG-o. When the number of materialized views is approximately equal to the number of queries, there is no big difference in the performance between GG-c and GG-o.

7 Conclusions

In this paper we have considered the optimization of multiple dimensional queries simultaneously in an MDDB. We first considered two restricted versions of the problem by presenting both approximation and exact algorithms for finding plans within the fixed degree approximation of the optimal cost and optimal costs respectively. We also re-examined the problem discussed by Zhao et al. [ZDNS98], by presenting another greedy algorithm. Finally, we used experiments to demonstrate that the presented solutions are effective. In particular, we showed that our algorithms are efficient and the solutions delivered are within twice the optimum.

Acknowledgements. We appreciate the anonymous referees for their invaluable suggestions and comments which helped us improve the paper's quality and presentation. The work by Weifa Liang and Jeffrey X. Yu was partially supported by Australian Research Council under a small grant schema (Grant No: F00025).



a 10 materialized views and 90 queries

b 50 materialized views and 50 queries

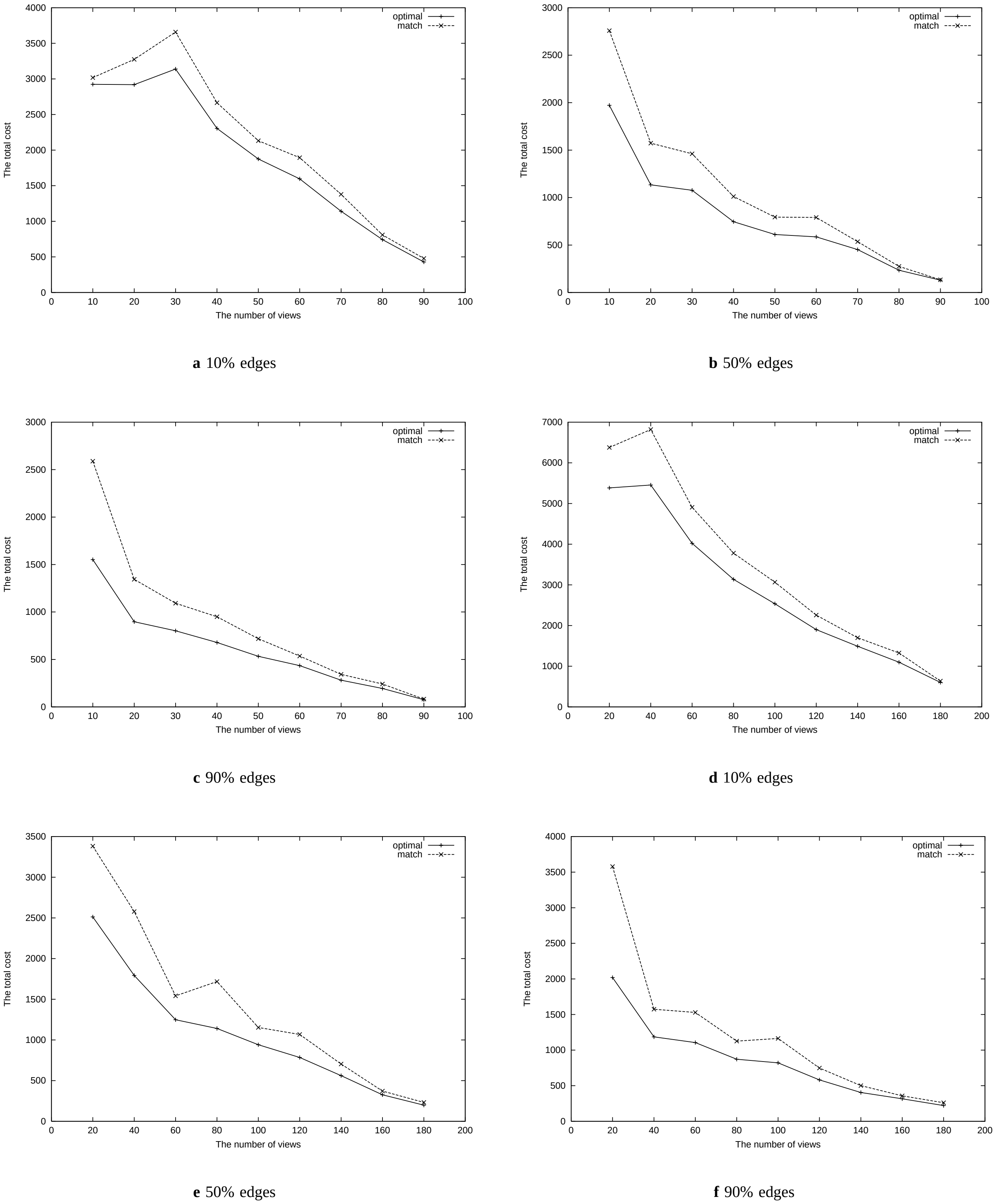
c 90 materialized views and 10 queries

d 20 materialized views and 180 queries

e 100 materialized views and 100 queries

f 180 materialized views and 20 queries

Fig. 7. The approximation and exact solutions for `type two sharing` queries



a 10% edges

b 50% edges

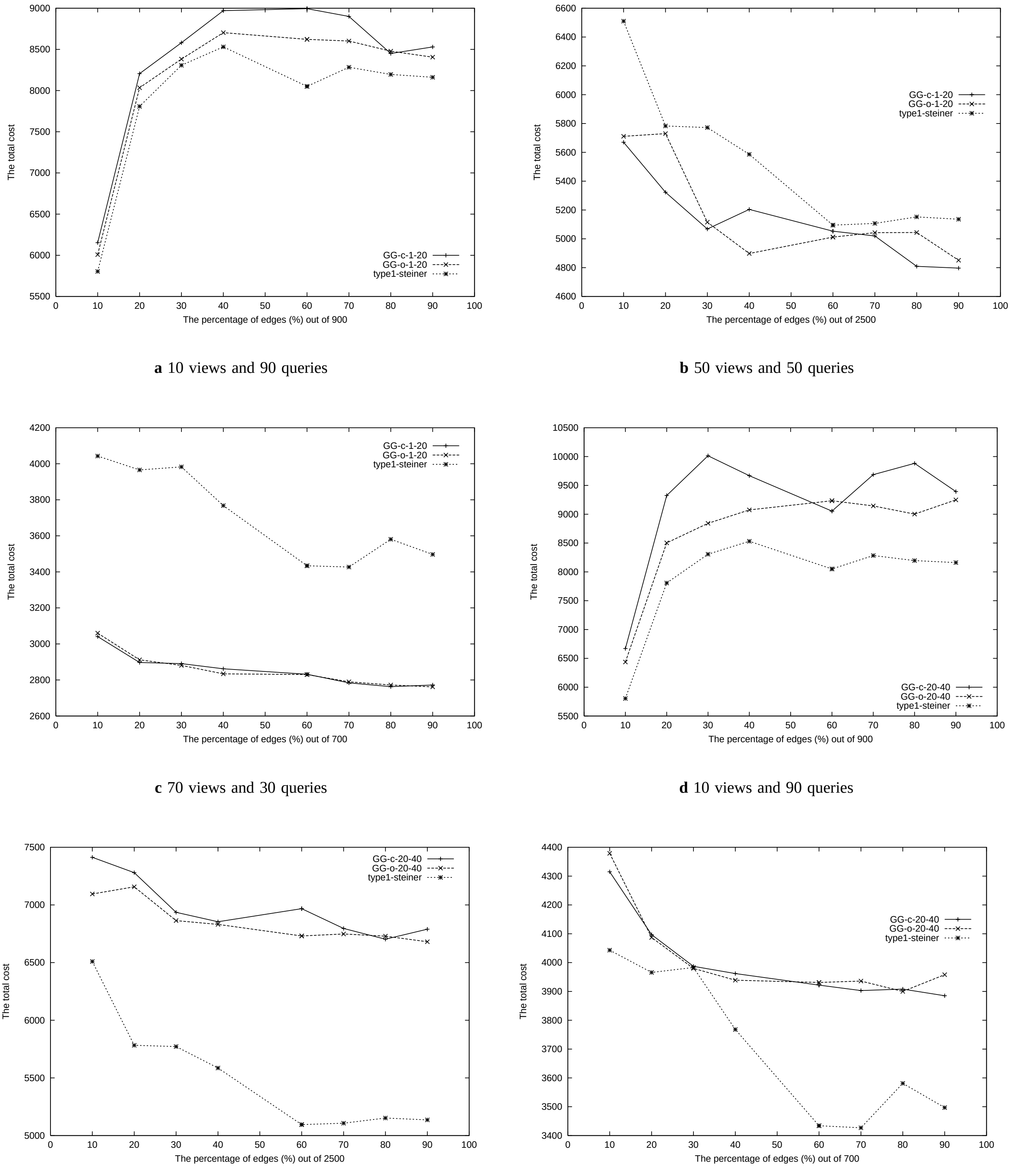
c 90% edges

d 10% edges

e 50% edges

f 90% edges

Fig. 8. The approximation and exact solutions for `type two sharing` queries



a 10 views and 90 queries

b 50 views and 50 queries

c 70 views and 30 queries

d 10 views and 90 queries

e 50 views and 50 queries

f 70 views and 30 queries

Fig. 9. The approximation solutions for `type three sharing` queries

References

[AAD+96] Agrawal S., Agrawal R., Deshpande P. M., Gupta A., Naughton J. F., Ramakrishnan R., Sarawagi S.: On the computation of multidimensional aggregates. In: Proc. 22nd VLDB Conf., Mumbai, India, 1996, pp 506–521

[AESY97] Agrawal D., El Abbadi A., Singh A., Yurek T.: Efficient view maintenance at data warehouse. In: Proc. 1997 ACM-SIGMOD Conf., 1997, pp 417–427

[BPT97] Baralis E., Paraboschi S., Teniente E.: Materialized view selection in a multidimensional database. In: Proc. 23rd VLDB Conf., Athens, Greece, 1997, pp 156–165

[BD99] Berkelaar M., Dirks J.: ftp//ftp.es.ele.tue.nl/pub/lp_solve, 1999

[DANR96] Deshpande P. M., Agarwal S., Naughton J. F., Ramakrishnan R.: Computation of multidimensional aggregates. Technical Report, No: 1314, Dept. of CS, Univ. of Wisconsin-Madison, 1996

[GT90] Gabow H. N., Tarjan R. E.: Faster scaling algorithms for general graph matching problems. J. ACM, 38: 815–853, 1990

[GJ79] Garey M. R., Johnson D. S.: In: Computers and Intractability. San Francisco, CA: W. H. Freeman, 1979

[GBLP95] Gray J., Bosworth A., Layman A., Prahesh H.: Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total. Microsoft Technical Report, MSR-TR-95-22, 1995

[GM95] Gupta A., Mumick I.: Maintenance of materialized views: problems, techniques, and applications. IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing, 18(2): 3–19, 1995

[GHRU97] Gupta H., Harinarayan V., Rajaraman A., Ullman J. D.: Index selection for OLAP. In: Proc. Intl. Conf. on Data Engineering, Birmingham, UK, 1997, pp 208–219

[Gupta97] Gupta H.: Selection of views to materialize in a data warehouse. In: Proc. Intl. Conf. on Database Theory, 1997, pp 98–112

[HRU96] Harinarayan V., Rajaraman A., Ullman J. D.: Implementing data cubes efficiently. In: Proc. 1996 ACM-SIGMOD Conf., 1996, pp 205–216

[HZ96] Hull R., Zhou G.: A framework for supporting data integration using the materialized and virtual approaches. In: Proc. 1996 ACM-SIGMOD Conf., 1996, pp 481–492

[Kim96] Kimball R.: In: The Data Warehouse Toolkit. John Wiley, 1996

[Knuth93] Knuth D. E.: In: The Stanford GraphBase: A Platform for Combinatorial Computing. Addison-Wesley/ACM Press, 1993

[MS] Microsoft Corp.: OLE DB for OLAP Design Specification–Beta 2.
http://www.microsoft.com/data/oledb/olap/prodinfo.html.

[OQ97] O'Neil P., Quass D.: Improved query performance with variant indexes. In: Proc. 1997 ACM-SIGMOD Conf., 1997, pp 38–49

[PS88] Park J., Segev A.: Using common subexpressions to optimize multiple queries. In: Proc. 4th Intl. Conf. on Data Engineering, 1988, pp 311–319

[RSC98] Ross K. A., Srivastava D., Chatziantoniou D.: Complex aggregation at multiple granularities. In: Proc. Intl. Conf. on Extending Databases Technology, 1998, pp 263–277

[RSS96] Ross K. A., Srivastava D., Sudarshan S.: Materialized view maintenance and integrity constraint checking: trading space for time. In: Proc. 1996 ACM-SIGMOD Conf., 1996, pp 447–458

[RS97] Ross K. A., Srivastava D.: Fast computation of sparse datacubes. In: Proc. 23rd VLDB Conf., Athens, Greece, 1997, pp 116–125

[SAG96] Sarawagi S., Agrawal R., Gupta A.: On the computing the data cube. Research Report, IBM Almaden Research Center, San Jose, CA, 1996

[S88] Sellis T. K.: Multi-query optimization. ACM Trans. on Database Systems, 13(1): 23–52, 1988

[SS94] Shim K., Sellis T. K.: Improvements on heuristic algorithm for multi-query optimization. Data and Knowledge Engineering, 12(2): 197–222, 1994

[Su96] Sundaresan P.: Data warehousing features in Informix Online XPS. In: Proc. 4th PDIS Conf., Miami Beach, FL, USA, 1996

[YKL97] Yang J., Karlapalem K. K., Li Q.: Fast computation of sparse databases algorithms for materialized view design in data warehousing. In: Proc. 23rd VLDB Conf., Athens, Greece, 1997, pp 136–145

[Zeli97] Zelikovsky A.: A series of approximation algorithms for the acyclic directed Steiner tree problem. Algorithmica 18: 99–110, 1997

[ZDN97] Zhao Y., Deshpande P. M., Naughton J. F.: An array-based algorithm for simultaneous multidimensional aggregates. In: Proc. 1997 ACM-SIGMOD Conf., 1997, pp 159–170

[ZDNS98] Zhao Y., Deshpande P. M., Naughton J. F., Shukla A.: Simultaneous optimization and evaluation of multiple dimensional queries. In: Proc. 1998 ACM-SIGMOD Conf., June, 1998, pp 271–282

[ZGHW95] Zhuge Y., Garcia-Molina H., Hammer J., Widom J.: View maintenance in a warehousing environment. In: Proc. 1995 ACM-SIGMOD Conf., 1995, pp 316–327