Under consideration for publication in Knowledge and Information
Systems

A Genetic Selection Algorithm for OLAP Data Cubes

Wen-Yang Lin[1] and I-Chung Kuo[2]

[1]Department of Information Management, I-Shou University, Kaohsiung 84008, Taiwan
[2]Institute of Information Engineering, I-Shou University, Kaohsiung 84008, Taiwan

Abstract. Multidimensional data analysis, as supported by OLAP (online analytical processing) systems, requires the computation of many aggregate functions over a large volume of historically collected data. To decrease the query time and to provide various viewpoints for the analysts, these data are usually organized as a multi-dimensional data model, called data cubes. Each cell in a data cube corresponds to a unique set of values for the different dimensions and contains the metric of interest. The data cube selection problem is, given the set of user queries and a storage space constraint, to select a set of materialized cubes from the data cubes to minimize the query cost and/or the maintenance cost. This problem is known to be an NP-hard problem. In this study, we examined the application of genetic algorithms to the cube selection problem. We proposed a greedy-repaired genetic algorithm, called the genetic greedy method. According to our experiments, the solution obtained by our genetic greedy method is superior to that found using the traditional greedy method. That is, within the same storage constraint, the solution can greatly reduce the amount of query cost as well as the cube maintenance cost.

Keywords: Multidimensional database; Data warehouse; OLAP; Data cubes; Genetic algorithms; Greedy method

1. Introduction

The multidimensional database (MDDB) is a new database concept dedicated to solving the demands of a decision supporting system. To understand what the data is really saying, the managers usually need to investigate data from different business perspectives and change the navigation according to the previous

Received xxx
Revised xxx
Accepted xxx

observation. Toward this purpose, data from various operational sources are reconciled and stored in a repository database using a multidimensional data model (Chaudhuri and Dayal 1997), called a *data cube*. Each cell of a data cube represents a specific view in which users are interested. Because of the space limits of MDDB, only a subset of the data cubes can be stored in the repository. The OLAP data cube selection problem is, given a set of user queries and a storage space constraint, to select a set of materialized cubes from the data cubes to minimize the query cost and/or the maintenance cost. This problem is known to be an NP-hard problem (Harinarayan et al 1996).

In this paper, unlike most of the work done so far making use of heuristics to obtain a near optimal solution, we apply genetic algorithms to solve this problem. Our genetic method is featured by an innovative greedy repair method to tackle the problem of unfeasible solutions, which occurs when a naive binary encoding scheme is used. According to the experiments, our greedy genetic algorithm is more effective than the well-known greedy algorithm (Gupta 1997, Gupta et al 1997, Harinarayan et al 1996) in finding good solutions, especially when the space constraint is more restricted.

1.1. Data Warehouse and OLAP

For many years, enterprises have accumulated a lot of data and now realize the importance of using these data for supporting their decision-making. However, these data are usually stored in different information systems, which means that the decision makers have no common data source while they make decisions. This would lead to some important faults: (1) Decision makers would not know where to get the related data. (2) While different analysts cite different data sources, it is very possible to achieve very different conclusions. (3) Because the data are distributed in different sources of information systems, analysts must integrate the data before the analysis, including format transformation and filtering. This process can be computation-intensive and makes the results out of date.

The *data warehouse* concept (Inmon and Kelley 1993) is a technique to solve these problems. According to the demands of analysts, data are extracted and transformed from each source database and saved in a repository, called a data warehouse. The data are usually organized into a relational model of multidimensional data, called a *star schema* (Chaudhuri and Dayal 1997), as depicted in Figure 1. Each *dimension table* contains all the information specific to the dimension itself, while the *fact table* correlates all dimensions and contains information of interest to the analysts. When users execute any query, the system only needs to search the data warehouse instead of the source databases.

A complete data warehousing system is composed of three primary parts: the source databases in the backend, a data warehouse and several data marts in the core, and analysis tools in the front-end. Often, the analysis that a data warehouse supports is *on-line analytical processing* (OLAP) (Chaudhuri and Dayal 1997), where the queries aggregate large volumes of data to detect trend anomalies. Typical OLAP operations include drill-down, roll-up, pivoting, and slice-and-dice (Chaudhuri and Dayal 1997). One of the design issues for supporting OLAP analysis is how to speed up the computation-intensive operations used for OLAP to facilitate on-line navigation between different abstraction views of the aggregate data. To this end, the aggregate data are pre-computed and stored in another repository dedicated for OLAP, called a multidimensional database

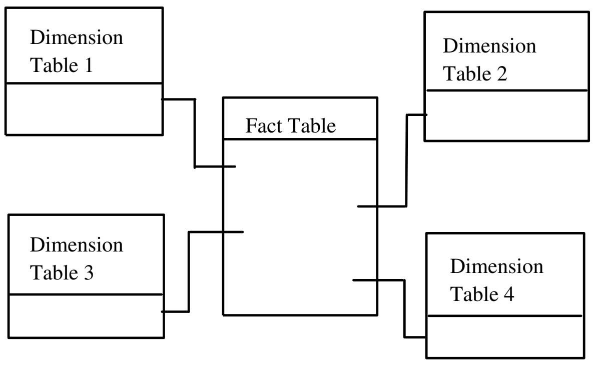

Fig. 1. A star schema.

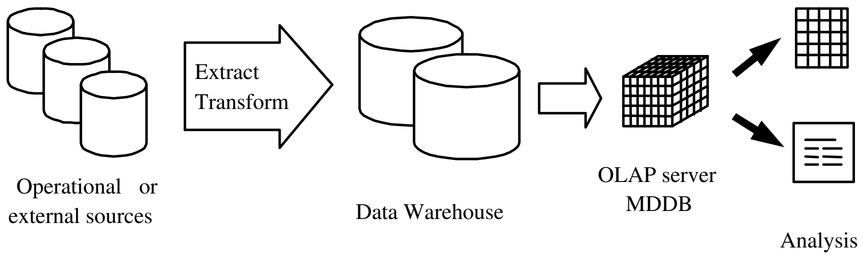

Fig. 2. A general structure of the data warehousing system with OLAP services.

(MDDB) or data mart. Figure 2 illustrates a general structure of the data warehousing system with OLAP services.

1.2. Related Work

As will be formalized later, the data in a MDDB are organized with a multidimensional data model, called a data cube. A data cube can be regarded as, in terms of database terminology, a materialized view. Harinarayan et al (1996) was the first one to consider the problem of materialized views selection for supporting multidimensional analysis in OLAP. They proposed a lattice model and provided a greedy algorithm to solve this problem. Gupta et al (1997) further extended their work to include indices selection. Ezeife (1997) also considered the same problem but proposed a uniform approach using a more detailed cost model. Recently, Shukla et al (1998) proposed a modified greedy algorithm that selects only according to the cube size. Their algorithm was shown to have the same quality as Harinarayan's greedy method but is more efficient. Soutyrina and Fotouhi (1997) proposed a dynamic programming algorithm to solve the problem, which can yield the optimal set of cubes. All of these works were based on the assumption that all aggregates are computed from a single cube. Instead of focusing on this special case, Shukla et al (2000) considered the view selection problem for multi-cube data models and proposed a multi-cube algorithm.

Baralis et al (1997) formalized the cube lattice model and proposed a method to reduce the solution space before applying the cube selection process. Qiu and Ling (2000) further investigated this issue. They proposed two methods, called the *functional dependency filter* and the *size filter*, to eliminate the redundant or insignificant views. Ross et al (1996) discussed how to materialize additional views to decrease the maintenance cost.

Gupta (1997) proposed a systematical methodology for the selection of materialized views. He extended the aggregated queries into a more general form that could be represented by an AND-OR graph. Gupta and Mumick (1999) then investigated how to select views or indices to decrease the cost of query processing under the maintenance cost constraint. The same problem was also discussed by Liang et al (2001). Though the AND-OR graph is appropriate for general view selection problem, it is not suitable for OLAP. The AND-OR graph is originally used in query evaluation, which expresses all of the alternative "useful" ways for evaluating a query from the given base relations in the presence of the other queries/views. But it is tedious to transform all of the user's OLAP activities into query expressions. Indeed, researchers have shown that for the grouping/aggregated queries used in OLAP, we only have to model the query or view dependence using the data cube lattice (or called OR-view graph) (Gupta 1997, Gupta et al 1997, Harinarayan et al 1996). In addition, the cube lattice can accommodate the dimension hierarchy information that is very important to underlie the two commonly used OLAP operations, roll-up and drill-down.

Theodoratos and Sellis (1997) supposed that all queries should be answered solely by the materialized views, with or without rewriting the users' queries. They modeled the problem as a state space optimization problem, and provided exhaustive and heuristic algorithms without concern for the storage constraint. At the same time, Yang et al (1997) proposed a different model called Multiple View Processing Plan (MVPP) for the same problem. They exploited the existence of common sub-expressions among most queries.

A general framework that incorporates all of the above issues was presented in Theodoratos and Bouzeghoub (2000). On the basis of the AND/OR expression DAGs, the framework tried to accommodate all possible design goals, such as query evaluation cost and/or view maintenance cost, and various constraints, such as space constraint and/or maintenance cost constraint. Jamil and Modica (2001) presented the design and implementation of a view selection system for multidimensional databases.

There is some work devoted to applying genetic algorithms to the view selection problem (Horng et al 1999, Zhang and Yang 1999, Zhang et al 2001). Following the AND-OR view graph used in Gupta et al (1997), Horng et al (1999) proposed a genetic algorithm to select the appropriate set of views to minimize the query cost and view maintenance cost. Their algorithm is very similar to ours. The primary difference is in the repairing scheme. We use a greedy repair method to correct the infeasible solutions while they use a *penalty function* to punish the fitness of the infeasible solutions and incorporate a local search technique to maintain local optimality. Researches have shown that the repair scheme is better in dealing with infeasible solutions than the penalty function is (Michalewicz 1994).

The problem considered in Zhang and Yang (1999) and Zhang et al (2001) is different from ours. Rather than optimize the view selection from a given query processing plan, the primary purpose in Zhang and Yang (1999) and Zhang et al (2001) is to find an optimal set of processing plans for multiple queries. A solution in their genetic algorithm thus represents a set of processing plans for the given queries.

1.3. Paper Organization

The rest of this paper is organized as follows. In Section 2, we introduce the cube lattice model and formalize the OLAP cube selection problem. Section 3 explains the concept of genetic algorithms and states how to apply GAs to the cube selection problem. The proposed greedy repair method is described as well. In Section 4, we describe our experiments and the results. Section 5 is the conclusion of this paper.

2. Problem Formulation

2.1. Data Cube and Lattice Framework

The data cube concept was first proposed by Gray et al (1997) as a generalization of the SQL groupby operator to meet users' on-line investigation of data from various viewpoints. This interactive and multidimensional analysis is usually accomplished by pre-computing aggregations over the data (Sarawagi et al 1996). For example, consider a fact table about product sales with the following schema

Sales_Fact (Product, Supplier, Customer, Sales).

Each tuple represents the sale of some product that is provided by some supplier sold to some customer at a sale price.

There are a number of queries that can be asked of this data. For example, the analysts will be interested in knowing the sales value from different viewpoints: consumer, supplier, and product. The sales data can be regarded as a data cube consisting of three dimensions: Customer, Supplier, and Product. Each cell of the data cube corresponds to a unique set of values for different dimensions. In this example, a cell represents the sales of a product provided by some supplier and sold to some customer, as shown in Figure 3. The data cube consists of four suppliers, four customers, and six products. If we want to know the sales of each product provided by every supplier, we must perform an aggregation along the customer dimension, with respect to different suppliers and products. For example, (c1, p1, s1) + (c2, p1, s1) + (c3, p1, s1) + (c4, p1, s1) yields the sales for product p1 provided by s1. This aggregated query can be described by a SQL-statement as follows:

```
SELECT Product, Supplier, SUM (Sales) AS Total Sales
FROM Sales_Fact
GROUP BY Product, Supplier
```

The result of this query indeed corresponds to a subcube of the original data cube, as shown in Figure 4. We denote this subcube as (-, p, s), where symbol '-' represents the corresponding attribute that is not in the GROUP BY statement in SQL.

In the same way, we can aggregate any combinations of the three dimensions, Customer, Product and Supplier, and obtain eight possible cubes (views). The relation of these eight cubes can be modeled as a cube lattice shown in

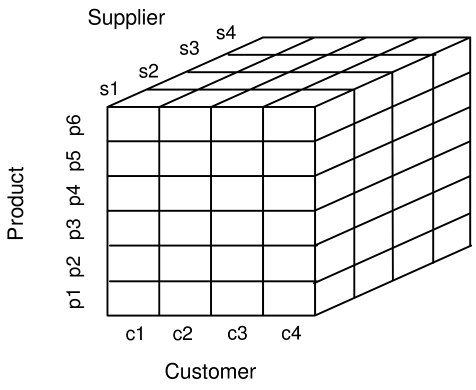

Fig. 3. An example of data cubes.

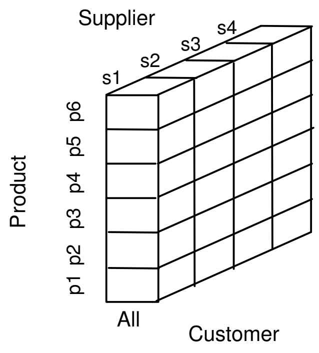

Fig. 4. The corresponding cube (-, p, s) derived from the data cube in Figure 3.

Figure 5. The numbers beside each cube indicate its size and invoking frequency, respectively. The edge between any two cubes c_i and c_j represents the dependence relation between these two cubes. We say c_i is dependent on c_j, denoted as $c_i \preceq c_j$, if any query answered by c_i can also be answered by c_j, but the reverse is not true. For example, consider (-, p, -) and (-, p, s). If we want to know the sales of each product, it is obvious that we can answer this question by using cube (-, p, -) or (-, p, s). But if we want to know the sales of each product provided by different suppliers, we can only obtain the answer from cube (-, p, s).

Given a fact relation with N dimensions, there are 2^N data cubes. Consider a subset, say M, of these 2^N data cubes. We define the ancestors and descendants of a cube c_i as follows:

$$Anc(c_i, M) = \{c_j | c_j \in M \text{ and } c_i \preceq c_j\},$$
$$Des(c_i, M) = \{c_j | c_j \in M \text{ and } c_j \preceq c_i\}.$$

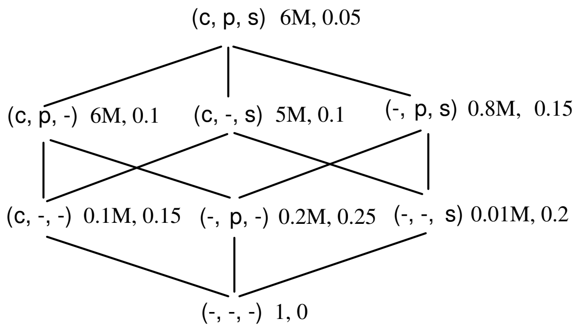

Fig. 5. A lattice with eight data cubes.

The least ancestor and greatest descendant of c_i then can be derived straightforwardly,

$$Lanc(c_i, M) = \min_{c_j \in Anc(c_i)} |c_j|,$$
$$Gdes(c_i, M) = \max_{c_j \in Des(c_i)} |c_j|.$$

For example, the ancestors of (c, -, -) in Figure 5 are (c, p, -), (c, -, s) and (c, p, s) while the least ancestor is (c, -, s).

2.2. The Cube Selection Problem

According to the above discussions, it is important that a MDDB stores suitable cubes in light of users' questions to accelerate the query processing. That is, suitable cubes are pre-computed and stored in the MDDB. The best way is to materialize all data cubes. Nevertheless, the space limitation of the MDDB would hinder us from doing this. We should instead consider users' queries to select the optimal subset of cubes for materialization in order to speed up the query response time without requiring too much work to keep the materialization consistent with any modifications to the fact table. This problem is called the data cube selection problem, which we will state formally as follows.

Given an OLAP cube-lattice L with n cubes $C = \{c_1, c_2, \ldots, c_n\}$ derived from a fact table R, a set of queries $Q = \{q_1, q_2, \ldots, q_k\}$, a set of query frequency values $F = \{f_{q_1}, f_{q_2}, \ldots, f_{q_k}\}$ associated with the queries in Q, a set of update frequency values $G = \{g_{c_1}, g_{c_2}, \ldots, g_{c_n}\}$ of the cubes in C, and a space constraint S, the cube selection problem is denoted as a seven-tuple $\Omega = (L, C, R, Q, F, G, S)$. A solution to Ω is a subset of C, say M, that can minimize the following cost function under the constraint that $\sum_{c \in M} |c| \leq S$,

$$\sum_{i=1}^{k} f_{q_i} E(q_i, M) + \sum_{c \in M} g_c U(c, M) \tag{1}$$

where $E(q_i, M)$ and $U(c, M)$ denote, with respect to the set of materialized cubes M, the cost to evaluate query q_i and the cost to update cube c, respectively.

Note that though Eq. 1 is derived from the OLAP-cube lattice model, it is valid for other general view selection models such as AND/OR view graph. The readers should refer to Gupta (1997) for the details.

2.3. The Cost Model

In this subsection, we will deliberate the cost function defined previously. There are two parts: the query cost and the maintenance cost.

For the query cost part, note that the queries considered in the OLAP analysis, such as min, max, average, and sum, are all types of aggregation functions; an entire scan of the inspected data suffices for the evaluation. Hence, we follow the cost model proposed by Harinarayan et al (1996), in which the cost of evaluating a query equals to the number of non-null cells in the aggregated cube used to answer the query. To validate this assumption, Harinarayan et al (1996) made an experiment and found that there is an almost linear relationship between the cube size and the query evaluation time.

Using the linear cost model, the evaluation cost can be expressed alternatively in terms of the cube notion. Consider an aggregation query q involving t dimensions, say $a_1, a_2, \ldots, a_t$. The cubes used to answer this query should be composed of a superset of these dimensions. We denote the very cube with just the same dimensions as query q be c_q. Hence, there is a mapping from the query set Q to the cube set C. For a cube c, we define its invoking frequency f_c as

$$f_c = \sum_{q \in Q, c = c^q} f_q.$$

Then the query cost becomes

$$\sum_{i=1}^{n} f_{c_i} E(c_i, M),$$

where $E(c_i, M)$ denotes the cost to evaluate cube c_i from the current materialization set M. In terms of the least ancestor notion,

$$E(c_i, M) = \min(|c_i|, |Lanc(c_i, M)|).$$

For the maintenance cost part, we consider only the insertion of tuples into the fact table R from which the data cubes are derived. This assumption is based on the fact that the data stored in a data warehouse are continually expanding with no modification. Furthermore, we assume that the insertions do not violate the referential integrity constraint within the star schema.

All of the materialized cubes constructed for supporting OLAP must be updated, reflecting the changes to the data warehouse. The materialized cubes can be recomputed from scratch or a deferred maintenance technique can be used depending on the design of the OLAP server (Mumick et al 1997, Ross et al 1996). In this paper, to simplify the discussion, we adopted the first approach, but the techniques addressed here can also be applied to the other approach. With this assumption, the cost to maintain a materialized cube c_i in M then equals the size of the least ancestor of c_i, e.g., $U(c_i, M) = |Lanc(c_i, M)|$. Let g_u represent the frequency of insertions to the base relation R. The total maintenance cost becomes

$$g_u \sum_{c \in M} U(c, M).$$

Combining the above derivations, the cost function in Eq. 1 then becomes

$$\sum_{i=1}^{n} f_{c_i} E(c_i, M) + g_u \sum_{c \in M} U(c, M). \tag{2}$$

Hereafter, this modified cost function is used in all discussions.

3. The Proposed Genetic Algorithm for Cube Selection

3.1. Simple Genetic Algorithms

The Genetic Algorithm (GA), first proposed by Holland (1992) in 1975, is an approach that mimics biological processes for evolving optimal or near optimal solutions to problems. Beginning with a random population (group of chromosomes), it chooses parents and generates offspring using operations analogous to biological processes, usually *crossover* and *mutation.* Adopting *the survival of the fittest principle*, all chromosomes are evaluated using a fitness function to determine their fitness values, which are then used to decide whether the chromosomes are eliminated or retained to propagate. The higher fitness chromosomes are kept and the less fitness ones are discarded in generating a new population. The new population replaces the old one and the whole process is repeated until a specific termination criterion is satisfied. The chromosome with the highest fitness value in the last population gives the solution. A general description of a simple genetic algorithm (Goldberg 1989) is given in Algorithm 3.1.

Algorithm 3.1. A simple genetic algorithm.

```
Initialize the parameters;
Generate a population $P$ randomly;
$generation \leftarrow 1$;
while $generation \leq max\_gen$ do
   Clear the new population $P'$;
   Use a fitness function $\mu(\cdot)$ to evaluate each individual in $P$;
   while $|P'| \leq population\_size$ do
      Select two parents from $P$;
      Perform crossover;
      Perform mutation;
      Place the offspring into $P'$;
   endwhile
   $P \leftarrow P'$;
   $generation \leftarrow generation + 1$;
endwhile
```

3.2. On Applying Simple Genetic Algorithms

As stated above, GA is an evolutionary process composed of many biological imitations, such as the chromosome representation, genetic operators, population selection, and fitness function. Below, we state the primary issues involved in applying GAs to the cube selection problem.

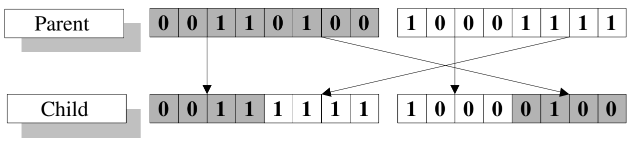

Fig. 6. One-point crossover.

Encoding scheme. The encoding scheme is the first step and the key part of using GAs. To make available the crossover and mutation operations, and to enhance the performance of the algorithm, a chromosome representation that stores the current solution is desirable. The most common encoding scheme is to transform the problem solutions into a binary string. For the cube selection problem, this scheme is extremely natural. For example, we can use an 8-bit binary string to represent the way that the cubes in Figure 5 are selected; '0' means that the cube is not selected and '1' means that it is selected. In this way, '01011010' means that the selected cubes are (c, p, -), (-, p, s), (c, -, -), and (-, -, s).

Fitness function. A fitness function evaluates the degree of fitness of each chromosome. All chromosomes with smaller fitness are discarded to increase the probability of retaining population superiority. Designing the fitness function is important in creating an effective GA. It may be the most time-consuming and is dependent on the characteristics of the problem. Different kinds of problems have different fitness function definitions. Here, we use the cost function in Eq. 2 as the fitness function.

Selection. Selection is a way of choosing individuals from a population as the parents for crossover. According to the survival of the fittest principle, we select the better individuals to generate the next generation. Nevertheless, the diversity of the population will be lost if we always focus on the best individuals. This is because that the area exploited for possible solutions is restricted, and the population will converge too quickly to evolve to a better solution. In this study, we adopted the tournament selection method (Mitchell 1996). Two individuals are randomly chosen from the current population and a random number r between 0 and 1 is generated. If r is less than a predefined value, usually set as 0.75, we choose the individual with the higher fitness value; otherwise, we choose the other one. An individual who has been selected still can be chosen the next time. Besides, we adopted *the elitism principle* (Mitchell 1996): The best individual is always selected into the next population.

Crossover. The crossover operation is used to generate offspring by exchanging bits in a pair of individuals (parents) chosen from the population. There are diverse forms of crossovers mentioned in the literature, but each has its limitations and must be considered in light of the characteristics of the problem in question. Here, we adopted the simplest one-point crossover. A crossover point is randomly selected with probability p_c (called *crossover probability*, typically $p_c = 0.7$) and the portions of the two individuals divided by this point are exchanged to form the offspring. This operation is illustrated in Figure 6.

Mutation. The mutation operator is used to randomly change some elements in selected individuals and leads to additional genetic diversity to help the search process escape from local optimal traps. The canonical bit-flip mutation is used in this study: Each locus of an individual is subject to mutation with probability p_m (called *mutation probability*) and the bit value at that locus is changed from

0 to 1 or vice versa. For example, suppose that an individual $x = 10010100$ is mutated at the sixth locus. The new individual x' will be 10010000.

3.3. Incorporated with the Greedy Repair Algorithm

A naive implementation using the simple genetic algorithm will generate infeasible solutions during the evolution. The primary reason is that the simple GA does not embed the constraint into its encoding scheme. For example, assume that the space limit is 7M and the two chromosomes selected for crossover are 10001101 and 00111110. The sizes of the materialized cubes corresponding to these two chromosomes are both under 7M. If we select the second gene as a crossover point, then the offspring generated by crossover are 10111110 and 00001101. Both exceed the space limit! There are many methods to remedy infeasible solutions (Michalewicz 1994). Some commonly used methods are (1) using an alternative encoding scheme to avoid infeasible solutions; (2) using a penalty function to make the infeasible solutions get a worse fitness, and thus decreasing their opportunities to survive; and (3) using a repair method to adjust the genes of the infeasible chromosomes to meet the problem constraint.

In the literature, using a repair method has been shown to be more effective than the others for most constraint problems (Michalewicz 1994). Hence, we adopted the third method in our study.

We proposed a greedy repair method that can be seen as a reverse version of Harinarayan's greedy method. Rather than increase the set of materialized cubes from the empty one by one, we correct the genes of an infeasible chromosome each time by choosing the least detrimental cube. That is, the elimination of this cube will increase the least cost per space unit.

For this purpose, we defined a *detriment function* $D(c, M)$ to compute, relative to some selected set of cubes M, the total cost increase caused by the elimination of a cube c, i.e.,

$$D(c,M) = \frac{1}{|c|}\left(\sum_{c_i \in C-\{c\}} f_{c_i}\left[E(c_i, M-\{c\}) - E(c_i, M)\right] + g_u \sum_{c_i \in M-\{c\}} \left[U(c_i, M-\{c\}) - U(c_i, M)\right]\right) \qquad (3)$$

Using this function, we compute the detriment of all selected cubes in M when a chromosome corresponds to an infeasible solution, and choose the one having the least detriment for repairing. Our greedy repair algorithm is described in Algorithm 3.2.

For an illustration of Algorithm 3.2, consider the previous example: An individual 10111110 needs to be repaired, and the space constraint is 7M. We assume that the cube invoked frequencies are 0.05, 0.1, 0.1, 0.15, 0.15, 0.25, 0.2, and 0, as shown in Figure 5. We also assume that when (c, p, s) is not materialized, all queries that should be answered by (c, p, s) will go back to the base relation in the data warehouse and that the size of the base relation is 200M. Table 1 lists the detriment of the six cubes, (c, p, s), (c, -, s), (-, p, s), (c, -, -), (-, p, -), and (-, -, s), where the elements in column "influenced cubes" of cube c_i refer to the cubes whose evaluation or maintenance cost must be updated when c_i is elimi-

Table 1. Detriment of subcubes during the process of greedy repair algorithm.

Cube	Influenced cubes		Detriment
	For evaluation	For maintenance	
cps	cps, cp-	cps, c-s, -ps	7.98
c-s	c-s	c-s, c–	**-0.08**
-ps	-ps	-ps, -p-, –s	1.57
c–	c–	c–	3.85
-p-	-p-	-p-	0.35
–s	–s	–s	7.8

nated. For example, when we remove cube (c, p, s), the corresponding queries for (c, p, s) and (c, p, -), currently answered by (c, p, s), should now go back to the base relation. The maintenance cost for (c, p, s) becomes zero while the cost for maintaining (c, -, s) and (-, p, s) is increased from 6M into 200M. The detriment is $((200 - 6) * (0.05 + 0.1) + (-200 + (200 - 6) * 2) * 0.1)/6 = 7.98$. According to Table 1, the cube selected for removal is (c, -, s). The size of the remaining cubes amounts to 7M, which is no larger than 7M.

Algorithm 3.2. A greedy repair method.

```
Let an individual $x$ that needs to be repaired be $(x_1, x_2, ..., x_n)$, and the
corresponding set of selected cubes be $M = \{c_i | x_i = 1, 1 \leq i \leq n\}$;
while $\sum_{c \in M} |c| > S$ do
  for each $c_i \in M$ do
    Calculate the detriment $D(c_i, M)$;
  $c_j \leftarrow$ the cube with min detriment;
  $x_j \leftarrow 0$;
  $M \leftarrow M - \{c_j\}$;
endwhile
```

Algorithm 3.3 shows our greedy genetic algorithm for OLAP cube selection.

3.4. Complexity Analysis

We first analyze the complexity of Algorithm 3.2 to facilitate the whole evaluation. It is easy to see that the computation for repairing an individual is proportional to the number of bits undergoing repair. This number is dependent on how far the total size of the materialized views represented by the individual is from the space constraint. Let the space limit be S and the total size of all sub-cubes be σ, for $\sigma = \sum_i |c_i|$. In average, we can expect an individual of size $\sigma/2$ and a sub-cube of size σ/n. The number of bits to be repaired will be $n/2 - n\,S/\sigma$. Furthermore, each bit repair involves finding the sub-cube with minimum detriment, which according to the definition of detriment function consumes approximately $O(n \log n)$ time. Therefore, each calling of Algorithm 3.2 requires $O(n^2 \log n)$ computation. Note that this is in the same order of magnitude as Harinarayan's greedy algorithm (Harinarayan et al 1996).

Let r denote the population size and g denote the max generation. The crossover and mutation operations can be completed within a constant time. Each fitness evaluation requires $O(n)$ time. In each generation there are at most r individuals undergoing crossover and needed to be re-evaluated the fitness.

The total evaluations for g generations thus require $O(grn)$ time. It is hard to estimate within a generation how many individuals will undergo repairing. In the worst case, all offspring generated by crossover become infeasible. Consequently, the cost spent on repairing for g generations is $O(grn^2 \log n)$. The complexity of Algorithm 3.3 is thus $O(grn^2 \log n)$.

Algorithm 3.3. A greedy genetic cube selection algorithm.

```
Initialize the parameters;
Generate a population $P$ randomly;
$generation \leftarrow 1$;
while $generation \leq max\_gen$ do
   Clear the new population $P'$;
   for each individual $x \in P$ do
      if $x$ is not feasible then
         Invoke Algorithm 3.2 to repair $x$;
      Evaluate the fitness of $x$;
   endfor
   while $|P'| \leq population\_size$ do
      Select two parents from $P$;
      Perform crossover;
      Perform mutation;
      Place the offspring into $P'$;
   endwhile
   $P \leftarrow P'$;
   $generation \leftarrow generation + 1$;
endwhile
```

Now that the complexity of our genetic algorithm is associated with the population size and max generations, we want to clarify the setting of these two parameters. Since genetic algorithms are stochastic search processes, there is no convincing theory that would completely justify the setting of population size and max generation for GAs to reach the optimal solution. Nevertheless, we can examine a sort of convergence in probability. Reeves (1993) adopted the principle that the initial population should contain at least one instance of every allele at every locus to ensure that a genetic algorithm is potentially able to reach every point in its search space. He showed that the probability that at least one allele is present at each locus is $(1 - (1/2)^{r-1})^n$. Let p denote the desired least probability. We can derive a lower bound on r in terms of n and p:

$$\left(1 - \left(\frac{1}{2}\right)^{r-1}\right)^n \geq p \Rightarrow r \geq 1 - \log\left(1 - \sqrt[n]{p}\right). \qquad (4)$$

Similarly, given a fixed probability p, we may ask what is the smallest number of generations required to obtain an optimal solution. Aytug and Koehler (1996) modeled the genetic algorithms with Markov chain and derived the following bound

$$\mathrm{INT}\left[\max_j \left\{\frac{\ln(1-p)}{\ln\left(\delta(Q - Q\, u_j\, u_j^T)\right)}\right\}\right] \leq g$$

$$\leq \mathrm{INT}\left[\frac{1-\ln(1-p)}{r\ln\left(1-\min\left\{p_m(1-p_m)^{n-1}, {p_m}^n\right\}\right)}\right], \tag{5}$$

where INT$[x]$ denotes the smallest integer greater than or equal to x, for $x \geq 0$, Q is the Markov chain transition matrix, u_j the jth unit vector, $\delta(A)$ the spectral radius of a matrix A and p_m the mutation probability. The lower bound is difficult to evaluate because the maximum is taken over all states of the Markov chain. Nevertheless, we can derive a lower bound for g in terms of the takeover time, i.e., the number of generations until the whole population contains the best individual of the initial population (Goldberg and Deb 1991). For the 2-tournament selection, the take over time (Goldberg and Deb 1991) is

$$\frac{\ln r + \ln(\ln r)}{\ln 2}. \tag{6}$$

Greenhalgh and Marshall (2000) showed that the upper bound can be improved further as

$$g \leq \mathrm{INT}\left[\frac{1-\ln(1-p)}{r\ln\left(1-\min\left\{p_m(1-p_m)^{n-1}, {p_m}^n\right\}\right)}\right] \approx -\frac{\ln(1-p)}{r\,{p_m}^n}. \tag{7}$$

Taking $p_m = 1/n$, as suggested by Muhlenbein (1992), and incorporating Eqs. 6 and 7, we have

$$\frac{\ln r + \ln(\ln r)}{\ln 2} \leq g \leq -\frac{\ln(1-p)n^n}{r}. \tag{8}$$

Note that the above derivation is based on the simple GAs, which does not take into account the effect of chromosomes repairing. Though very loose this bound is, it gives us the preliminary guidance in appropriate generation setting. Because the population diversity is reduced after repairing, we can expect that the population will converge more quickly. The question of what the exact bound on the number of generations is when concerning the repairing effect deserves further investigation.

4. Experimental Results

To validate the effectiveness of the proposed genetic selection method, we compared it with the leading selection method used in Harinarayan et al (1996), which is a greedy based heuristic. We used the test set in TPC-D benchmark (Raab et al 1995), which is a database generator following the proposal of the Transaction Processing Performance Council in 1995, and is dedicated to decision support applications. This database can generate up to 10000GB data. Here, we used the smallest database 1GB. We chose three dimensions from the test set, Customer c, Product p, and Supplier s. To broaden the variety of data cubes, we added additional attributes to each dimension, forming the hierarchy shown in Figure 7. Because there are four combinations for each dimension, we have 64 different data cubes, as detailed in Table 2. For simplicity, each cube is represented by the first letter of the attributes and symbol '-' means none.

We considered three different cases of cube invoking frequencies: (1) uniform frequencies set as 1. (2) random frequencies between 0 and 1. (3) linear frequencies proportional to the reciprocal of the cube size. For the space constraint, we considered ten different values set as 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%,

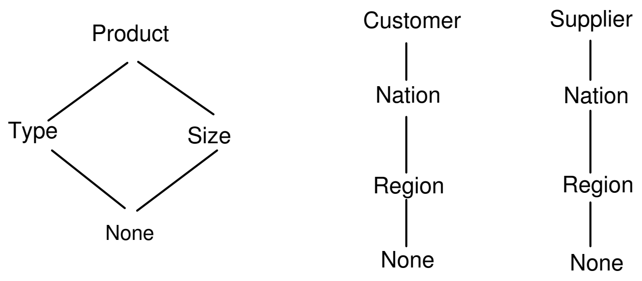

Fig. 7. The attribute hierarchy of three dimensions: Product, Customer, and Supplier.

Table 2. All data cubes derived from three dimensions: Customer, Product, and Supplier. The size is measured in K (1000) numbers of tuples.

cps	6000	nps	5000	rps	4000	-ps	800
cpn	6000	npn	5000	rpn	4000	-pn	800
cpr	6000	npr	5000	rpr	4000	-pr	800
cp-	6000	np-	5000	rp-	1000	-p-	200
css	5000	nss	500	rss	2500	-ss	500
csn	5000	nsn	30	rsn	6.25	-sn	1.25
csr	5000	nsr	6.25	rsr	1.25	-sr	0.25
cs-	5000	ns-	1.25	rs-	0.025	-s-	0.05
cts	5990	nts	800	rts	3000	-ts	1500
ctn	5990	ntn	90	rtn	18.75	-tn	3.75
ctr	5990	ntr	18.75	rtr	3.75	-tr	0.75
ct-	5990	nt-	3.75	rt-	0.75	-t-	0.15
c-s	6000	n-s	250	r-s	50	–s	10
c-n	2500	n-n	0.625	r-n	0.125	–n	0.025
c-r	500	n-r	0.125	r-r	0.025	–r	0.025
c–	100	n–	0.025	r–	0.005	—	0.001

80%, and 90%, respectively, of the total size of all cubes. In this way, we have thirty different combinations. Furthermore, if none of the cubes that are capable of answering a specific query is materialized, we assumed that the fact table R in the data warehouse is used. The cost for the evaluation or maintenance of a cube using R is set as three times that using the largest cube, i.e., 18M. The following parameter settings are used in our greedy genetic algorithm.

max generation: 100
population size: 100
crossover probability: 0.65
mutation probability: 1/64

The results are depicted in Figures 8 to 10, each corresponding to one of the three frequency assumptions. From these results, we observed the following:

(1) Regardless of the storage space and frequency of the test set, the solution obtained by our genetic greedy method is superior to Harinaryan's greedy method.
(2) The total cost reaches a minimum when the space is around 20% of the size of all cubes. Any further increase beyond this amount does not significantly

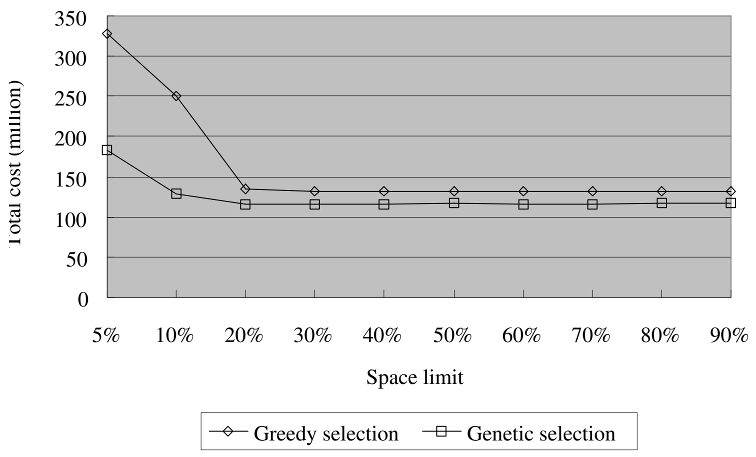

Fig. 8. Comparison of greedy selection and genetic selection with uniform frequencies.

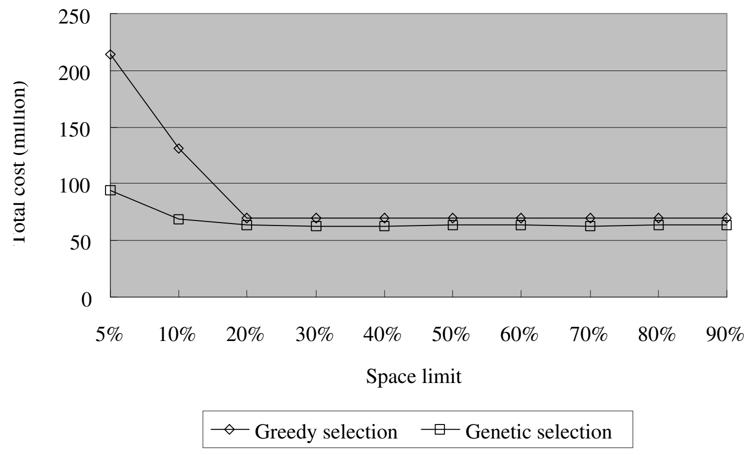

Fig. 9. Comparison of greedy selection and genetic selection with random frequencies.

decrease the cost. A similar phenomenon was also observed by Harinarayan et al (1996), whereas they only examined the query cost.

From the above observations, it can be seen that blindly increasing the amount of storage cannot decrease the cost effectively. In the test data we used, most cost-effective storage space is about 20% of the total cubes. We believe that different problems should have different characteristics. The best way is to estimate a priori the optimal storage space of the problem. When the available space is greater than the optimal space, we only have to consider the optimal space and apply the traditional greedy method. On the other hand, when the storage space is smaller than the optimal space, we should use the genetic greedy method. Note that in real world most data warehouses are extremely large and continually expanding. The available storage space thus is usually quite smaller

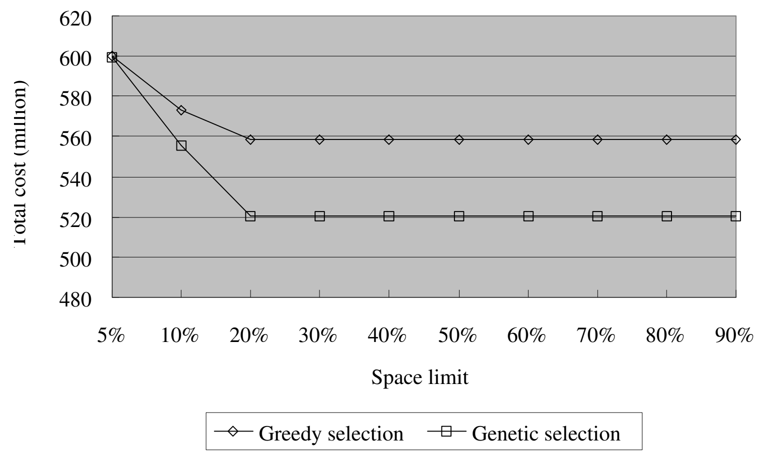

Fig. 10. Comparison of greedy selection and genetic selection with linear frequencies.

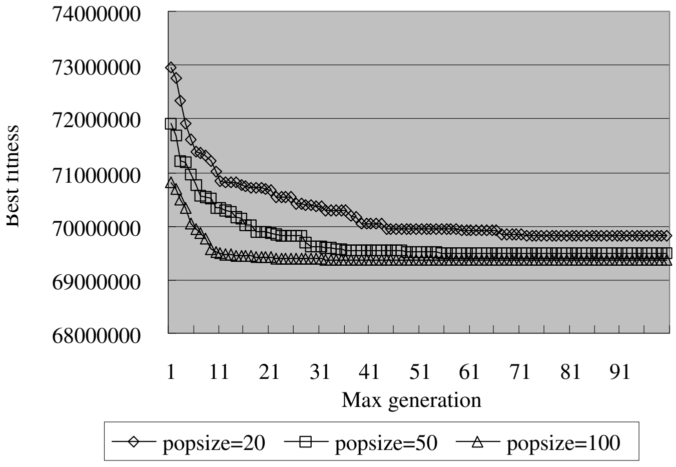

Fig. 11. Best fitness vs. max generation for random frequency with 10% space limit.

than the total cubes. From this point of view, our genetic greedy algorithm will be the best choice.

We also conducted an experiment to examine the performance distribution of solutions by changing the population size and max generation. Figure 11 depicts the results. Here we only show the case of random frequencies and 10% space limit; the results are similar for the other cases. As is indicated in Figure 11, the solution becomes better as the population size grows; the number of generations to reach the best solution decreases as well. For population size of 100, the best fitness begins to converge at generation 11, and for population sizes of 50 and 20, they begin at generation 31 and 45, respectively. It can be verified that for each of the three different population sizes, the generation at which the best fitness begins to converge conforms to the bound expressed in Eq. 8.

Figure 12 illustrates the minimum number of generations to achieve solutions

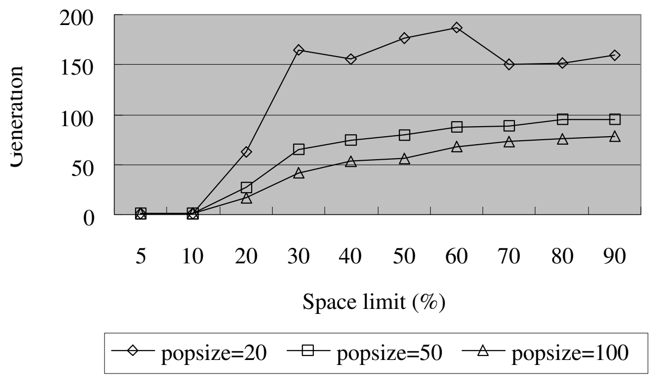

Fig. 12. Minimum number of generations to achieve solutions better than that obtained by greedy algorithm under random frequencies.

better than Harinarayan's greedy algorithm for various population sizes under the case of random invoking frequencies. We observe the following:

(1) The number of generations grows as the space limit increases. This is because that when the space limit increases, the number of feasible solutions also increases, making the genetic algorithm more difficult to locate the best solution.
(2) For space limit no more than 10% of the total cube size, the genetic greedy algorithm yields a better solution than does the greedy algorithm right after the first generation, though it is far from the best solution. This is because the greedy algorithm performs poorly under a more restricted space limit.

5. Conclusions

In this paper, we have established a selection model using a genetic algorithm in configuring OLAP data cubes. We proposed a genetic selection algorithm incorporated with a greedy repair method to avoid infeasible solutions. According to the experimental results, our genetic selection algorithm always generates a better solution than the greedy algorithm. We also observed that there is an optimal space to obtain the best cost-effective solution; increasing storage space further over this optimal space has no significant effect on reducing the cost. The problem of how to obtain the optimal cost-effective space is a critical issue and deserves advanced investigation.

Acknowledgements. We thank anonymous reviewers for their very useful comments and suggestions. This work was supported by the National Science Council of ROC with grant No. NSC89-2213-E-214-007.

References

Aytug H, Koehler GJ (1996) Stopping criteria for finite length genetic algorithms. INFORMS Journal on Computing 8(2):183–191

Baralis E, Paraboschi S, Teniente E (1997) Materialized view selection in a multidimensional database. In Jarke M, Carey MJ, Dittrich KR, et al (eds). Proceedings of the 23rd international conference on very large data bases, Athens, Greece, August 1997, pp. 156–165.

Chaudhuri S, Dayal U (1997) An overview of data warehouse and OLAP technology. ACM SIGMOD Record 26(1): 65–74

Ezeife CI (1997) A uniform approach for selecting views and indexes in a data warehouse. In Proceedings of the 2nd international database engineering and applications symposium, Montreal, Canada, August 1997, pp. 151–160

Goldberg DE (1989) Genetic Algorithms in Search, Optimization, and Machine Learning, Reading, MA, Addison-Wesley

Goldberg DE, Deb K (1991) A comparative analysis of selection schemes used in genetic algorithms. In Whitley LD (ed). Foundations of Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp. 69–93

Gray J, Chaudhuri S, Bosworth A, et al (1997) Data cube: A relational aggregation operator generalizing group-by, cross-tabs and subtotals. Data Mining and Knowledge Discovery 1(1): 29-53

Greenhalgh D, Marshall S (2000) Convergence criteria for genetic algorithms. SIAM Journal on Computing 30(1): 269–282

Gupta H (1997) Selection of views to materialize in a data warehouse. In Afrati FN, Kolaitis P (eds). Proceedings of the 5th international conference on database theory, Delphi, Greece, January 1997, pp. 98–112

Gupta H, Harinarayan V, Rajaraman A, Ullman JD (1997) Index selection for OLAP. In Gray A, Larson P (eds). Proceedings of the 13th international conference on data engineering, Birmingham, UK, April 1997. Lecture Notes in Computer Science 1186, Springer, Berlin, pp. 208–219

Gupta H, Mumick IS (1999) Selection of views to materialize under a maintenance cost constraint. In Beeri C, Buneman P (eds). Proceedings of the 7th international conference on database theory, Jerusalam, Israel, January 1999. Lecture Notes in Computer Science 1540, Springer, Berlin, pp. 453–470

Harinarayan V, Rajaraman A, Ullman JD (1996) Implementing data cubes efficiently. In Jagadish HV, Mumick IS (eds). Proceedings of ACM SIGMOD international conference on management of data, Montreal, Canada, June 1996, pp. 205–216

Holland JH (1992) Adaptation in Natural and Artificial Systems, Second Edition, MIT Press

Horng JT, Chang YJ, Liu BJ, Kao CY (1999) Materialized view selection using genetic algorithms in a data warehouse. In Proceedings of world congress on evolutionary computation, Washington DC, USA, July 1999, pp. 2221–2227

Inmon WH, Kelley C (1993) Rdb/VMS: Developing the Data Warehouse, QED Publishing Group, Boston, Massachussetts

Jamil HM, Modica GA (2001) A view selection tool for multidimensional databases. In Monostori L, Váncza J, Ali M (eds). Proceedings of the 14th international conference on industrial and engineering applications of artificial intelligence and expert systems, Budapest, Hungary, June 2001. Lecture Notes in Computer Science 2070, Springer, Berlin, pp. 237–246

Liang W, Wang H, Orlowska ME (2001) Materalized view selection under the maintenance time constraint. Data and Knowledge Engineering 37(2): 203–216

Michalewicz Z (1994) Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, New York

Mitchell M (1996) An Introduction to Genetic Algorithms, MIT press

Mühlenbein H (1992) How genetic algorithms really work: I. Mutation and hillclimbing. In Männer R, Manderick B (eds). Parallel Problem Solving from Nature 2, Brussels, Belgium, September 1992, pp. 15–25.

Mumick IS, Quass D, Mumick BS (1997) Maintenance of data cubes and summary tables in a warehouse. In Peckham J (ed). Proceedings of ACM SIGMOD international conference on management of data, Tucson, Arizona, USA, June 1997, pp. 100–111

Qiu SG, Ling TW (2000) View selection in OLAP environment. In Ibrahim MT, Küng J, Revell N (eds). Proceedings of the 11th international conference on database and expert system applications, London, UK, September 2000. Lecture Notes in Computer Science 1873, Springer, Berlin, pp. 447–456

Raab F, et al (1995) TPC Benchmark^{TM} D (Decision Support) Proposed revision 1.0, Transaction Processing Performance Council, San Jose, CA

Reeves CR (1993) Using genetic algorithms with small populations. In Forrest S (ed). Proceedings of the 5th international conference on genetic algorithms, Morgan Kaufmmann, San Mateo, CA, 1993, pp. 92–99

Ross KA, Srivastava D, Sudarshan S (1996) Materialized view maintenance and integrity constraint checking: trading space for time. In Jagadish HV, Mumick IS (eds). Proceedings of ACM SIGMOD international conference on management of data, Montreal, Canada, June 1996, pp. 447–458

Sarawagi S, Agrawal R, Gupta A (1996) On Computing the Data Cube, Research Report 10026, IBM Almaden Research Center, San Jose, California

Shukla A, Deshpande PM, Naughton JF (1998) Materialized view selection for multidimensional datasets. In Gupta A, Shmueli O, Widom J (eds). Proceedings of the 24th international conference on very large data bases, New York, USA, August 1998, pp. 488–499

Shukla A, Deshpande PM, Naughton JF (2000) Materialized view selection for multicube data models. In Zaniolo C, Lockemann PC, Scholl MH, Torsten G (eds). Proceedings of advances in database technology (EDBT '00), 7th international conference on extended database technology, Konstanz, Germany, March 2000. Lecture Notes in Computer Science 1777, Springer, Berlin, pp. 269–284

Soutyrina E, Fotouhi F (1997) Optimal view selection for multidimensional database systems. In Proceedings of 1997 international database engineering and applications symposium, 1997, pp. 309–318

Theodoratos D, Bouzeghoub M (2000) A general framework for the view selection problem for data warehouse design and evolution. In Proceedings of the 3rd ACM International Workshop on Data Warehousing and OLAP, Washington DC, USA, November 2000, pp. 1–8

Theodoratos D, Sellis T (1997) Data warehouse configuration. In Jarke M, Carey MJ, Dittrich KR, et al (eds). Proceedings of the 23rd international conference on very large data bases, Athens, Greece, August 1997, pp. 126–135

Yang J, Karlapalem K, Li Q (1997) Algorithm for materialized view design in data warehousing environment. In Jarke M, Carey MJ, Dittrich KR, et al (eds). Proceedings of the 23rd international conference on very large data bases, Athens, Greece, August 1997, pp. 136–145

Zhang C, Yang J (1999) Genetic algorithm for materialized view selection in data warehouse environments. In Mohania MK, Tjoa AM (eds). Proceedings of the 1st international conference on data warehouse and knowledge discovery, Florence, Italy, August 1999. Lecture Notes in Computer Science 1676, Springer, Berlin, pp. 116–125

Zhang C, Yao X, Yang J (2001) An evolutionary approach to materialized views selection in a data warehouse environment. IEEE Transactions on Systems, Man and Cybernetics, Part C 31(3): 282–294

Author Biographies

insert photo

Wen-Yang Lin received his B.E. and M.E. both in Computer Science and Information Engineering from National Chiao-Tung University in 1988 and 1990, respectively. He then received his Ph.D. in Computer Science and Information Engineering from National Taiwan University in 1994. In 1996, he joined the Department of Information Management at I-Shou University and now is an Associate Professor. He is primarily interested in the area of sparse matrix technology and large-scale supercomputing. Currently he is also interested in data warehousing, data mining and evolutionary computation. Dr. Lin has authored over 50 technical papers.

insert photo

I-Chung Kuo received a B.S. degree in Information Management from I-Shou University, Kaohsiung, Taiwan, in 1998 and an M.E. degree in Information Engineering from the same university, in 2000. From 2000 to 2002, he joined the army for two years' military obligation. During this time, he participated in the deployment of a human resource management system for the army headquarters. Now he works in a computer company. His research interests include data warehousing, evolutionary computation and database.

Correspondence and offprint requests to: Wen-Yang Lin, Department of Information Management, I-Shou University, Kaohsiung, Taiwan 84008, ROC. Email: wylin@isu.edu.tw