

Horizontal Partitioning by Predicate Abstraction and its Application to Data Warehouse Design

Aleksandar Dimovski¹, Goran Velinov², and Dragan Sahpaski²

¹ Faculty of Information-Communication Technologies, FON University, Skopje, 1000, Republic of Macedonia

² Institute of Informatics, Faculty of Sciences and Mathematics, Ss. Cyril and Methodius University, Skopje, 1000, Republic of Macedonia

Abstract. We propose a new method for horizontal partitioning of relations based on predicate abstraction by using a finite set of arbitrary predicates defined over the whole domains of relations. The method is formal and compositional: arbitrary fragments of relations can be partitioned with arbitrary number of predicates. We apply this partitioning to address the problem of finding suitable design for a relational data warehouse modeled using star schemas such that the performance of a given workload is optimized. We use a genetic algorithm to generate an appropriate solution for this optimization problem. The experimental results confirm effectiveness of our approach.

Keywords: Data Warehouse, Horizontal Partitioning, Predicate Abstraction.

1 Introduction

Given a database shared by distributed applications in a network, the performance of queries would be significantly improved by proper data distribution to the physical locations where they are needed most. This can be achieved by using *partitioning* (or *fragmentation*). Partitioning is the process of splitting large relations (tables) into smaller ones so that the DBMS does not need to retrieve as much data at any one time. There are two ways to partition a relation: *horizontally* and *vertically*. Horizontal partitioning involves splitting the tuples (rows) of a relation, placing them into two or more relations with the identical structure. Vertical partitioning involves splitting the attributes (columns) of a relation, placing them into two or more relations linked by the relation's primary key. Advantages that partitioning brings are the following: it can significantly impact the performance of the workload, i.e. set of queries that executes against the database system by reducing the cost of accessing and processing data; it allows parallel processing of data by locating tuples where they are most frequently accessed, etc.

Data Warehouses (DW) store active data of business value for an organization. Relational DW often contain large relations (fact relations or fact tables)

and require techniques both for managing these large relations and for providing good query performance across these large relations. Our goal is to find an optimal partitioning scheme of a data warehouse for a given representative workload by using partitioning on relations and indexes. An important issue about partitioning is to which degree should it occur. We need to find a suitable level of partitioning relations within the range starting from single attribute values or tuples to the complete relations. The space of possible physical partitioning scheme alternatives that need to be considered is very large. For example, each relation can be partitioned in many different ways.

In this paper we describe a new formal approach for horizontal partitioning and its application for optimizing data warehouse design in a cost-based manner. Horizontal partitioning is based on predicate abstraction which maps the domain of a relation to be partitioned to an abstract domain following a finite set of arbitrary predicates chosen over the whole concrete domain. To address the above optimization problem, we first choose a set of predicates to horizontally partition some (or all) dimension relations of a DW with star scheme, and then split the fact relation by using the predicates specified on dimension relations. This creates a number of sub-star fragments of the data warehouse we consider, where each sub-star fragment consists of a partition of the fact table and corresponding to it partitions of dimension relations. Then we use a genetic algorithm, known evolutionary heuristic, to find a suitable solution which minimizes the query cost. Our method does not guarantee an optimal partitioning, but the experimental results suggest that it produces good solutions in practice.

Organization After discussing related work, in Section 2 we formally present a procedure for horizontal partitioning of relations based on predicate abstraction. Section 3 contains brief review of relational DW with star scheme model. In Section 4, the optimization problem is defined and a genetic algorithm addressing it is described. We present experimental results in Section 5. Finally, in Section 6, we conclude and discuss future work.

Related Work The work on optimal partitioning of a database design for a given representative workload and its allocation to a number of processor nodes has been extensive [13, 19]. The ideas were then adapted to the setting of a data warehouse [1, 6]. In [7] is proposed a technique for materializing data warehouse views in vertical fragments, aimed to tightly fit the reference workload. In our previous works, we develop a technique for optimizing a data warehouse scheme by using vertical partitioning [16, 17], and then extend it by defining multiversion implementation scheme in order to take into account the dynamic aspect of a warehouse due to the changes of the scheme structure and queries [15].

Predicate abstraction (or boolean abstraction) has been widely used in model checking [8]. The idea of predicate abstraction is to map concrete states of a system to abstract states according to their evaluation under a finite set of predicates. Automatic predicate abstraction has been developed for verifying infinite-state systems such as software programs [2, 5].

Horizontal partitioning has also been used for optimizing the performance of queries [4, 12, 14]. However, the method has not been formalized before, and in such way, it has not been applied in a concrete algorithm. The work presented in this paper is close to [3], which also uses horizontal partitioning for selecting an optimal scheme of a warehouse. However, our work brings several benefits.

- We formally define the method of horizontal partitioning of a relation by using the notion of predicate abstraction.
- In our optimizing procedure we use arbitrary predicates which can be defined over the whole domain of a relation, not as in [3] where only atomic predicates applied to single attributes are used.
- Our partitioning method is compositional, which enables partitioning arbitrary fragments of relations with arbitrary number of predicates.
- A global index table is created which maintains pointers to each of the substar fragments.

In our experiments we use genetic algorithms that are also used for optimization of a data warehouse schema in [3] and [18]. We conducted the experiments using the Java Genetic Algorithm Framework JGAP [11].

2 Horizontal Partitioning by Predicate Abstraction

Let R be a relation, and A_1, \dots, A_n be its attributes with the corresponding domains $Dom(A_1), \dots, Dom(A_n)$. A *predicate* represents a pure boolean expression over the attributes of a relation R and constants of the attributes' domains. An *atomic predicate* p is a relationship among attributes and constants of a relation. For example, $(A_1 < A_2)$ and $(A_3 \geq 5)$ are atomic predicates. Then, the set of all predicates over a relation R is:

$$\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2$$

We define *horizontal partitioning* as a pair (R, ϕ) , where R is a relation and ϕ is a predicate, which partitions R into at most 2 fragments (sub-relations) with the identical structure (i.e. the same set of attributes), one per each truth value of ϕ . The first fragment includes all tuples t of R which satisfy ϕ , i.e. $t \models \phi$. The second fragment includes all tuples t of R which do not satisfy ϕ , i.e. $t \not\models \phi$. It is possible one of the fragments to be empty if all tuples of R either satisfy or do not satisfy ϕ . Note that, the partitioning (R, ϕ) is identical to $(R, \neg\phi)$. If we apply the predicate *true* (or *false*) to a relation, then it remains undivided.

Example 1. Let $R = (A_1 \text{ int}, A_2 \text{ int}, A_3 \text{ date})$ be a relation. It can be divided into 2 partitions by using one of the following predicates:

- $\phi = (A_1 = A_2)$, which results into a fragment where the values of A_1 and A_2 are equal for all tuples, and a fragment where those values are different.
- $\phi = (A_3 \geq '01 - 01 - 07') \wedge (A_3 < '01 - 01 - 09')$, which results into a fragment where the values of A_3 are in the range from '01 - 01 - 07' to '01 - 01 - 09', and a fragment where those values are not in the specified range. \square

Embedded horizontal partitioning is also allowed. We can apply horizontal partitioning using a predicate ϕ_2 to each of the fragments obtained by a partitioning (R, ϕ_1) , denoted as (R, ϕ_1, ϕ_2) . In this way, we can split the initial relation R into at most 4 fragments:

$$\begin{aligned} R_1 &= \{t \in R \mid t \models \phi_1 \wedge \phi_2\} \\ R_2 &= \{t \in R \mid t \models \phi_1 \wedge \neg \phi_2\} \\ R_3 &= \{t \in R \mid t \models \neg \phi_1 \wedge \phi_2\} \\ R_4 &= \{t \in R \mid t \models \neg \phi_1 \wedge \neg \phi_2\} \end{aligned}$$

Embedded horizontal partitioning can go on to an arbitrary depth m , such that in each level an arbitrary predicate is applied to the obtained fragments. Embedded horizontal partitioning of a relation R with depth m is denoted as $(R, \phi_1, \phi_2, \dots, \phi_m)$ where $\phi_1, \phi_2, \dots, \phi_m$ are arbitrary predicates. The partitioning with depth m splits the initial relation R into at most 2^m fragments.

Example 2. Let we have the relation R from Example 1. $(R, (A_3 \geq' 01 - 01 - 07') \wedge (A_3 <' 01 - 01 - 09'), A_3 <' 01 - 01 - 07')$ splits R into at most 3 fragments:

- R_1 with tuples satisfying $A_3 <' 01 - 01 - 07'$.
- R_2 with tuples satisfying $'01 - 01 - 07' \leq A_3 <' 01 - 01 - 09'$.
- R_3 with tuples satisfying $A_3 \geq' 01 - 01 - 09'$.

Note that, the final number of fragments depends on the structure of R . For example, if there are no tuples that satisfy $(A_3 \geq' 01 - 01 - 09')$ then the fragment R_3 will be empty. \square

2.1 Predicate Abstraction

Given a relation $R = (A_1, \dots, A_n)$ and a set of predicates $\mathcal{P} = \{\phi_1, \phi_2, \dots, \phi_m\}$, we define the concrete domain of R as:

$$Dom(R) = Dom(A_1) \times Dom(A_2) \times \dots \times Dom(A_n)$$

and the abstract domain of R with respect to \mathcal{P} , denoted as $AbsDom(R)_{\mathcal{P}}$, as the set of bitvectors of length m (one bit per predicate $\phi_i \in \mathcal{P}$, for $i = 1, \dots, m$):

$$AbsDom(R)_{\mathcal{P}} = \{0, 1\}^m$$

The abstraction function is the mapping from the concrete domain $Dom(R)$ to the abstract domain, assigning a tuple t in R the bitvector representing the Boolean covering of t :

$$\begin{aligned} \alpha : Dom(R) &\rightarrow AbsDom(R)_{\mathcal{P}}, \\ t = (a_1, \dots, a_n) &\mapsto (v_1, \dots, v_m), t \models v_1 \cdot \phi_1 \wedge \dots \wedge v_m \cdot \phi_m \end{aligned}$$

where $0 \cdot \phi = \neg \phi$ and $1 \cdot \phi = \phi$. The concretization function is the mapping

$$\begin{aligned} \gamma : AbsDom(R)_{\mathcal{P}} &\rightarrow Dom(R), \\ (v_1, \dots, v_m) &\mapsto \{t \mid t \models v_1 \cdot \phi_1 \wedge \dots \wedge v_m \cdot \phi_m\} \end{aligned}$$

Given a relation $R = (A_1, \dots, A_n)$ and a set of predicates $\mathcal{P} = \{\phi_1, \phi_2, \dots, \phi_m\}$, the horizontal partitioning $(R, \phi_1, \dots, \phi_m)$, or (R, \mathcal{P}) for short, splits R into at most 2^m fragments:

$$R_{(v_1, \dots, v_m)} = \{t \mid \alpha(t) = (v_1, \dots, v_m)\}$$

We form an index table with 2^m entries representing all possible bitvectors of length m :

$$\{(v_1, \dots, v_m) \mid v_i \in \{0, 1\}, i = 1, \dots, m\}$$

An index entry (v_1, \dots, v_m) specifies the tuples of R satisfying the entry value with respect to the set of predicates \mathcal{P} . So, each single entry (v_1, \dots, v_m) from the index table points to exactly one fragment $R_{(v_1, \dots, v_m)}$. If some fragment is empty, then there will be no pointer to it. Then, local index tables are created on each of the fragments.

Example 3. Let us have the relation R from Examples 1 and 2. The partitioning $(R, (A_3 \succ=' 01 - 01 - 07') \wedge (A_3 <' 01 - 01 - 09'), A_3 <' 01 - 01 - 07')$ splits R into the following fragments: $R_{(0,0)}$ with tuples satisfying $A_3 \succ=' 01 - 01 - 09'$; $R_{(0,1)}$ with tuples satisfying $A_3 <' 01 - 01 - 07'$; $R_{(1,0)}$ with tuples satisfying $'01 - 01 - 07' \leq A_3 <' 01 - 01 - 09'$; $R_{(1,1)}$ with tuples satisfying both predicates, which is an empty set. The index table contains 4 entries: $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$ pointing to the corresponding fragments. \square

2.2 Predicate Selection

We can obtain a set of predicates $\mathcal{P} = \{\phi_1, \dots, \phi_m\}$ applicable to a relation R for horizontal partitioning by extracting them from a set of given (input) queries. The predicates are specified in the selection clause of a query. As we have seen, the number of horizontal fragments is in the worst case exponential in the number of predicates involved. Therefore, it is important to use as few predicates as possible. Given \mathcal{P} and R we want to generate a set of complete and minimal predicates \mathcal{P}_{comin} and then partition R by using (R, \mathcal{P}_{comin}) . A set of predicates is *complete* if it partitions the relation into a set of mutually disjoint fragments such that the access frequency of all tuples within a fragment is uniform for all queries. A set of predicates is *minimal* if the resulting partitioning is obtained by minimal number of predicates. There might be some redundant predicates in \mathcal{P} for our horizontal partitioning algorithm, which lead to no additional fragments.

Example 4. Let we have the relation R from the previous Examples. Consider predicates $\phi_1 = (A_3 \succ=' 01 - 01 - 07') \wedge (A_3 <' 01 - 01 - 09')$ and $\phi_2 = (A_3 <' 01 - 01 - 07')$. Then (R, ϕ_1, ϕ_2) splits R into 3 fragments as in Example 2. But if we have predicate $\phi_3 = (A_3 \succ=' 01 - 01 - 09')$, then $(R, \phi_1, \phi_2, \phi_3)$ generates again the same 3 fragments as before. So, ϕ_3 is a redundant predicate. Also the partitionings (R, ϕ_1, ϕ_3) and (R, ϕ_2, ϕ_3) are identical to $(R, \phi_1, \phi_2, \phi_3)$. This means that any one of the predicates ϕ_1 , ϕ_2 , and ϕ_3 can be eliminated. \square

The procedure **ComputeMin** for computing a minimal set of predicates based on a given complete set of predicates and a relation is presented in Figure 1. It checks for each of the predicates whether it can be eliminated or not. We say that a predicate ϕ is *relevant* to a set of predicates \mathcal{P} if there are two tuples t and t' of a fragment F , where $F \in (R, \mathcal{P})$, such that $t \models \phi$ and $t' \not\models \phi$. Note that, it is still possible that some fragments produced from (R, \mathcal{P}_{min}) to be empty.

The procedure computes a set of minimal predicates \mathcal{P}_{min} for a given complete set of predicates $\mathcal{P} = \{\phi_1, \dots, \phi_m\}$ and a relation R .

- 1 Let $\mathcal{P} = \{\phi_1, \dots, \phi_m\}$ be a set of predicates. Let $i := 1$ and $\mathcal{P}_{min} := \emptyset$.
- 2 If $i > m$, return \mathcal{P}_{min} .
- 3 If $(\phi_i \in \mathcal{P}_{min})$ or $(\neg\phi_i \in \mathcal{P}_{min})$, then ϕ_i is redundant. Set $i := i + 1$, and repeat from 2.
- 4 Otherwise, if ϕ_i is relevant to \mathcal{P}_{min} , then $\mathcal{P}_{min} := \mathcal{P}_{min} \cup \{\phi_i\}$. If there exists a $\phi \in \mathcal{P}_{min}$ that is not relevant to $\mathcal{P}_{min} \setminus \{\phi\}$, then set $\mathcal{P}_{min} = \mathcal{P}_{min} \setminus \{\phi\}$. Set $i := i + 1$, and repeat from 2.
- 5 If ϕ_i is not relevant to \mathcal{P}_{min} , then ϕ_i is redundant. Set $i := i + 1$, and repeat from 2.

Fig. 1. ComputeMin procedure

2.3 Derived Horizontal Partitioning

Derived Horizontal Partitioning is defined on a relation table which refers to another relation by using its primary key as reference. Since this relationship will be used during execution of join operations over the two relations, it is of advantage to propagate a horizontal fragmentation obtained for one relation to the other relation and to keep the corresponding fragments at the same place.

Let $R = (A_1, \dots, A_n)$ and $S = (B_1, \dots, B_m)$ be relations, A_j ($1 \leq j \leq n$) be a primary key of R , and B_i ($1 \leq i \leq m$) be a foreign key of S referring to A_j . Given a horizontal fragmentation of R into R_1, \dots, R_k , then this induces the derived horizontal fragmentation of S into k fragments:

$$S_l = S \ltimes R_l, l = 1, \dots, k$$

where the semi-join operator \ltimes is defined as $S \ltimes R = \pi_{B_1, \dots, B_m}(S \bowtie R)$, i.e. the result is the set of all tuples in S for which there is a tuple in R that is equal on their common attributes.

3 Data Warehouse Schema

In the core of any data warehouse is a concept of a multidimensional data cube. The data in the cube is stored in specialized relations, called *fact* and *dimension* relations. Fact relations contain basic facts about a model, and they are

referencing any number of dimension relations. On the other hand, dimension relations contain extra information about the facts. There are two schemes of implementation: *star* and *snowflake* scheme. In the star scheme all attributes of each dimension are stored in one relation, while in the snowflake scheme attributes in each dimension are normalized and stored in different relations. In this paper we consider data warehouse with star scheme.

Let $(F, D_1, D_2, \dots, D_k)$ be a star scheme. Given a set $\mathcal{D} = \{D_1, D_2, \dots, D_k\}$ of dimension relations, let us suppose that each of them D_i ($1 \leq i \leq k$) is horizontally partitioned by using a set of predicates \mathcal{P}_i into n_i fragments. Then, a fact relation F is partitioned using derived horizontal partitioning in the following way:

$$F_j = (\dots(F \times D_{1r_1}) \times \dots \times D_{kr_k})$$

where $1 \leq r_i \leq n_i$, $1 \leq i \leq k$, and $1 \leq j \leq \prod_{i=1}^k n_i$. So the fact relation will be partitioned into $\prod_{i=1}^k n_i$ fragments. Given a fact relation partition $F_j = (\dots(F \times D_{1r_1}) \times \dots \times D_{kr_k})$, we can create a sub-star scheme fragment $(F_j, D_{1r_1}, \dots, D_{kr_k})$. If each dimension D_i is partitioned into n_i ($1 \leq i \leq k$) fragments, then there will be $\prod_{i=1}^k n_i$ sub-star schemes in the implementation scheme of a data warehouse.

More formally, if dimension relations are partitioned by sets of predicates $\mathcal{P}_i = \{\phi_{i,1}, \phi_{i,2}, \dots, \phi_{i,m_i}\}$ for $1 \leq i \leq k$, then each dimension relation D_i will be divided into at most 2^{m_i} fragments, and the fact table into at most $2^{\sum_{i=1}^k m_i}$ fragments. We can form a global index table with $2^{\sum_{i=1}^k m_i}$ entries representing all possible bitvectors of length $\sum_{i=1}^k m_i$. An index entry $(v_{1,1}, \dots, v_{1,m_1}, \dots, v_{k,m_k})$ specifies the tuples of dimension relations satisfying the entry value with respect to the corresponding set of predicates. Each single entry $(v_{1,1}, \dots, v_{1,m_1}, \dots, v_{k,m_k})$ from the index points to exactly one sub-star scheme created by dimension relations $D_{i(v_{i,1}, \dots, v_{i,m_i})}$ for $1 \leq i \leq k$, and a fact sub-relation $(\dots(F \times D_{1(v_{1,1}, \dots, v_{1,m_1})}) \times \dots \times D_{k(v_{k,1}, \dots, v_{k,m_k})})$. Then, local index tables are created on each of the sub-star schemes.

4 Optimization Problem

As we have seen the number of generated sub-star schemes grows rapidly as the number of fragments of dimensions increases. Thus, it will be difficult for the data warehouse administrator (DWA) to maintain all these sub-star schemes. We want to compute an (near) optimal number of fragments such that the performance of queries will be good and the cost of maintaining and managing so many fragments will be acceptable. The latter is addressed by allowing to choose in our procedure a maximal number of sub-star scheme fragments that DWA can maintain. We now formally define the problem of finding an optimal partitioning implementation scheme of a data warehouse.

4.1 The Optimization Problem

Let $(F, D_1, D_2, \dots, D_k)$ be a star scheme, $\mathcal{Q} = \{Q_1, Q_2, \dots, Q_l\}$ be a set of queries, and Cost be a cost evaluation function. The optimization problem is defined as follows. Find a set of sub-star fragments $\mathcal{S} = \{S_1, S_2, \dots, S_N\}$ such that the cost

$$\text{Cost}(\mathcal{S}, \mathcal{Q}) \text{ is minimal}$$

subject to the constraint $N \leq W$, where W is a threshold representing a maximal number of fragments that can be generated. The cost evaluation function is defined according to the linear cost model [9]. The cost of answering a query Q_i , denoted as $\text{Cost}(\mathcal{S}, Q_i)$, is taken to be equal to the space occupied by the fragment $S_j \in \mathcal{S}$ from which the query is answered, i.e. proportional to the total number of rows of the fragment S_j .

4.2 The Optimization Procedure

We now describe an optimization procedure for obtaining an optimal partitioning implementation scheme given a workload:

- 1 Extract all predicates \mathcal{P} used by \mathcal{Q} .
- 2 Find a complete set of predicates $\mathcal{P}_i \subseteq \mathcal{P}$ ($1 \leq i \leq k$) corresponding to each dimension relation D_i .
- 3 Use **ComputeMin**(\mathcal{P}_i, D_i) procedure to find a minimal set of predicates for each relation.
- 4 Apply a genetic algorithm to find an optimal partitioning scheme.

Genetic algorithm (GA) [10] is a search method for finding approximate solutions to optimization problems. It uses techniques inspired by evolutionary biology such as mutation, selection, crossover, and survival of the fittest. Candidate solutions to a given problem, also called *chromosomes*, are represented most commonly as bit strings, but other encodings are also possible. The algorithm starts from a population of randomly generated solutions and happens in iterations (i.e. generations). In each generation, the cost of every solution in the population is evaluated, multiple solutions are selected from the current population based on their cost, and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration. The algorithm terminates when either a maximum number of generations has been produced, or a solution with satisfactory cost has been found. We now present the design of our genetic algorithm.

Representation of Solution Let $\mathcal{P}_i = \{\phi_{i,1}, \phi_{i,2}, \dots, \phi_{i,m_i}\}$ ($1 \leq i \leq k$) be a complete and minimal set of predicates that needs to be applied to the dimension D_i for horizontal partitioning. A possible solution of our problem is a set of N ($N \leq W$) different sub-star fragments. Each fragment S_j ($1 \leq j \leq N$) is represented by a bit array (or, bit-vector).

$$(v_{1,1}, \dots, v_{1,m_1}, \dots, v_{k,1}, \dots, v_{k,m_k})$$

containing one bit for each predicate used in the partitioning. Each bit in the solution is set to 1, if the respective predicate is satisfied by all tuples in S_j ; otherwise it is set to 0. So, we have that

$$S_j = F_{(v_{1,1}, \dots, v_{1,m_1}, \dots, v_{k,m_k})} = \{t \mid \alpha(t) = (v_{1,1}, \dots, v_{1,m_1}, \dots, v_{k,m_k})\}$$

or

$$S_j = F_{(v_{1,1}, \dots, v_{1,m_1}, \dots, v_{k,m_k})} = (\dots (F \times D_{1(v_{1,1}, \dots, v_{1,m_1})}) \times \dots \times D_{k(v_{k,1}, \dots, v_{k,m_k})})$$

The entry from the local index table which points to S_j will be its bit array representation $(v_{1,1}, \dots, v_{1,m_1}, \dots, v_{k,1}, \dots, v_{k,m_k})$. In this way, we obtain that the search space of our optimization problem is $2^N \sum_{i=1}^k m_i$, or in the worst case it is $2^W \sum_{i=1}^k m_i$.

A chromosome consists of N composite genes, where each composite gene is a bit-vector representing one fragment S_j as described above. One chromosome represents one possible solution to the problem.

Genetic Algorithm Operators A single point crossover operator is used, which chooses a random bit from two parent chromosomes, i.e. solutions, and then performs a swap of that bit and all subsequent bits between the two parent chromosomes, in order to obtain two new offspring chromosomes.

The mutation operation is performed over each gene of a chromosome and mutates them with a given probability. Because the genes are represented as bit arrays, a mutation of a gene means flipping the value of every bit with the given probability.

We use a natural selection operator where a chromosome is selected for survival in the next generation with a probability inversely proportional to the cost of the solution represented by the chromosome. A strategy of elitist selection is also used where the best chromosome of the population in the current generation is always carried unaltered to the population in the next generation.

The termination of the GA is established as follows. We perform a number of GA experiments, and we determine the number of iterations that are needed for the GA, such that no significant improvement in the solution quality can be detected for a specified number of iterations.

5 Experimental Results

The experiments were performed by using four sets of 25, 50, 100 and 200 distinct queries on a star scheme with 4 dimensions, which contain 11 attributes, and 1 fact table with size of $1.25 * 10^9$ rows. Each query contains a selection clause of the form: $\phi_1 \wedge \dots \wedge \phi_n$. The sets of 25, 50, 100 and 200 queries are composed of 178, 341, 711 and 1387 predicates, respectively. The number of predicates in a query is generated using a gaussian distribution with mean 7 and standard deviation 1. The attribute and its value in a given predicate are generated using

a uniform distribution on the set of attributes and the domain of the selected attribute, respectively. The termination condition of all the experiments is set to 200 iterations and the population size is set to 200 chromosomes.

In Figure 2, we show the query execution cost for different query sets and different values of the threshold W . The query execution cost is represented in percentage relative to the worst query execution cost (on a star schema with no partitioning) for the given query set. We can see that the query execution cost on a partitioned star schema is reduced by the order of 10^3 compared to an unpartitioned schema. Also, note that the query cost reduces when the threshold increases.

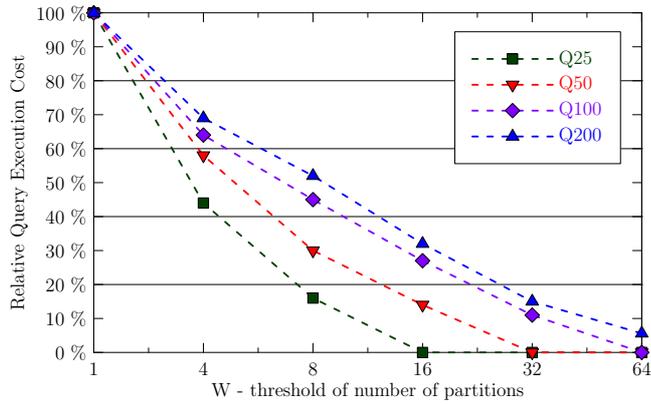


Fig. 2. The query cost for different values of the threshold W

In Figure 3, we compare the relative query execution costs on three query sets composed of 100 queries each, which contain predicates generated using a gaussian distribution with mean 3, 7 and 15 and standard deviation 1, on the same star schema used in Figure 2. The three sets of queries with 3, 7 and 15 average number of predicates per query are composed of 305, 711 and 1100 predicates, respectively. It can be seen that the query execution cost reduces more rapidly when the average number of predicates in the generated queries is smaller.

6 Conclusion

In this paper we present a novel formal approach for horizontal partitioning of relations based on predicate abstraction. Then, we show how to use this approach for finding an optimal data warehouse design which takes account of the performance of queries and the maintenance cost.

A possible direction for extension is to combine our partitioning method with vertical partitioning, and see its effects on the problem of computing an optimal

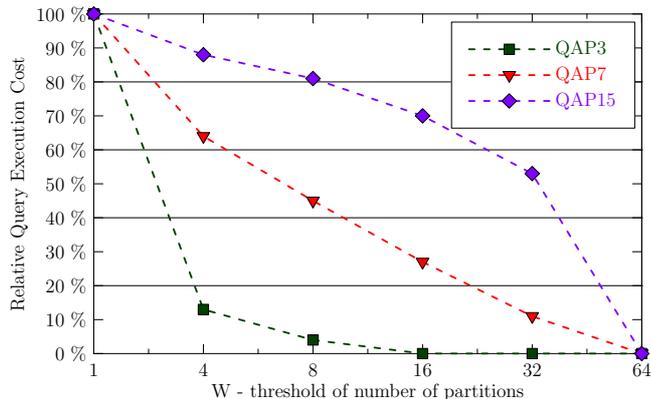


Fig. 3. The cost of query sets with different average number of predicates

data warehouse design. It is also interesting to extend the proposed approach to dynamically evolving data warehouse, which can change its scheme structures and its queries.

References

1. S. Agrawal, V. Narasayya, and B. Yang. Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design. In Proceedings of the ACM SIGMOD International Conference on Management of Data, (2004), 359–370.
2. T. Ball, A. Podelski, and S. K. Rajamani. Boolean and Cartesian Abstraction for Model Checking C Programs. In Proceedings of the International Conference on Tools and Algorithms for Construction and Analysis of Systems (*TACAS*), LNCS **2031**, (2001).
3. L. Bellatreche and K. Boukhalfa. An Evolutionary Approach to Schema Partitioning Selection in a Data Warehouse. In Proceedings of International Conference on Data Warehousing and Knowledge Discovery (*DAWAK*), LNCS **3589**, (2005), 115–125.
4. L. Bellatreche, K. Karlapalem, and A. Simonet. Algorithms and Support for Horizontal Class Partitioning in Object-Oriented Databases. In the *Distributed and Parallel Databases Journal* **8(2)**, (2000), 155–179.
5. A. Dimovski, D. R. Ghica, and R. Lazić. Data-Abstraction Refinement: A Game Semantic Approach. In Proceedings of the International Static Analysis Symposium (*SAS*), LNCS **3672**, (2005), 102–117.
6. P. Furtado. Experimental Evidence on Partitioning in Parallel Data Warehouses. Proceedings of the 7th ACM international workshop on Data warehousing and OLAP (*DOLAP*), (2004), 23–30.
7. M. Golfarelli, V. Maniezzo, S. Rizzi. Materialization of Fragmented Views in Multi-dimensional Databases. *Data & Knowledge Engineering*, Volume 49, Issue 3, (2004), 325–351.
8. S. Graf and H. Saidi. Construction of Abstract Atate Graphs with PVS. In Proceedings of the International Conference on Computer Aided Verification (*CAV*), LNCS **1254**, (1997), 72–83. Springer.

9. V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In Proceedings of the *1996 ACM SIGMOD International Conference on Management of Data*, ACM Press SIGMOD Record 25(2), (1996), 205–216.
10. J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1995.
11. K. Meffert. JGAP - Java Genetic Algorithms and Genetic Programming Package. <http://jgap.sf.net>.
12. M. T. Ozsü and P. Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, 1999.
13. D. Sacca and G. Wiederhold. Database Partitioning in a Cluster of Processors. In Proceedings of the ACM Transactions on Database Systems (*TODS*), Vol. **10(1)**, (1985), 29–56.
14. A. Sanjay, V. R. Narasayya, and V. R. Yang. Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design. In Proceedings of the *2004 ACM SIGMOD International Conference on Management of Data*, ACM Press SIGMOD Record, (2004), 359–370.
15. D. Sahpaski, G. Velinov, B. Jakimovski, and M. Kon-Popovska. Dynamic Evolution and Improvement of Data Warehouse Design. In Proceedings of Balkan Conference in Informatics, IEEE Computer Society’s Conference Publishing (*IEEE BCI*), (2009), 115–125.
16. G. Velinov, D. Gligoroski, and M. Kon-Popovska. Hybrid Greedy and Genetic Algorithms for Optimization of Relational Data Warehouses. In Proceedings of Multi-Conference: Artificial Intelligence and Applications (*IASTED*), (2007), 470–475.
17. G. Velinov, B. Jakimovski, D. Cerepnalkoski, and M. Kon-Popovska. Framework for Improvement of Data Warehouse Optimization Process by Workflow Gridification. In Proceedings of Conference on Advances in Databases and Information Systems (*ADBIS*), LNCS **5207**, (2008), 295–304.
18. J.X. Yu, X. Yao, C. Choi, and G. Gou. Materialized Views Selection as Constrained Evolutionary Optimization. In Proceedings of IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews, Volume 33(4), (2003), 458–468.
19. D. Zilio. Physical Database Design Decision Algorithms and Concurrent Reorganization for Parallel Database Systems. Ph. D. Thesis, University of Toronto, 1998.