Scientific Annals of Computer Science vol.19, 2009
"Alexandru Ioan Cuza" University of Iaşi, Romania

A Generalized Approach to Optimization of Relational Data Warehouses Using Hybrid Greedy and Genetic Algorithms

Goran VELINOV[1], Margita KON POPOVSKA[1], Danilo GLIGOROSKI [2]

Abstract

As far as we know, in the open scientific literature, there is no generalized framework for the optimization of relational data warehouses which includes view and index selection and vertical view fragmentation. In this paper we are offering such a framework. We propose a formalized multidimensional model, based on relational schemas, which provides complete vertical view fragmentation and presents an approach of the transformation of a fragmented snowflake schema to a defragmented star schema through the process of denormalization.

We define the generalized system of relational data warehouses optimization by including vertical fragmentation of the implementation schema (F), indexes (I) and view selection (S) for materialization. We consider Genetic Algorithm as an optimization method and introduce the technique of "recessive bits" for handling the infeasible solutions that are obtained by a Genetic Algorithm. We also present two novel hybrid algorithms, i.e. they are combination of Greedy and Genetic Algorithms.

Finally, we present our experimental results and show improvements of the performance and benefits of the generalized approach (SFI) and show that our novel algorithms significantly improve the efficiency of the optimization process for different input parameters.

[1]Institute of Informatics, Faculty of Sciences and Mathematics, Ss Cyril and Methodius University, Skopje, Macedonia. Email: goranv@ii.edu.mk, margita@ii.edu.mk

[2]Department of Telematics, Faculty of Information Technology, Mathematics and Electrical Engineering, Norwegian University of Science and Technology, Trondheim, Norway. Email: danilog@item.ntnu.no

1 Introduction

This paper represents a collection and extension of four papers that we have recently published [19, 20, 21, 22].

The performance of the system of RDW (Relational Data Warehouses) depends on many factors which makes its optimization a very complex and challenging problem. The main elements of a system for RDW optimization are: definition of a solution space, objective function and choice of an optimization method.

The solution space includes factors relevant for RDW performance as view and index selection and view fragmentation, i.e. partitioning. In some existing approaches the solution space for the problem of optimization of data warehouses is simplified to a great extent, and the selection of views or indexes is studied without considering other factors [1, 2, 6, 14, 15, 23]. These approaches are important for theoretical research, but are not applicable in practice. In some papers the view selection problem is generalized by including a proper set of indexes for each view and selection of views and indexes is done together [10, 20]. If the selection of views and indexes is performed separately and the set of indexes is added to the optimal set of views, then the common set might not have optimal performances. In [3] the problem of optimization of horizontal scheme partitioning was defined. The optimization problem of data warehouses as a combination of materialized views, indexes and horizontal data partitioning was introduced in [4] and the approach of vertical fragmentation was introduced in [10]. But none of them include selections of views and indexes and complete vertical fragmentation.

In this paper we present a solution of an optimization problem of RDW performance, based on a multi-dimensional model that includes complete vertical fragmentation. The model provides definition of all possible aggregate views and their data dependencies. It also includes all possible states of (de)normalization of the schema, from fully normalized snowflake to fully denormalized star schema. The solution space of the optimization problem includes aggregate views and all states of fragmentation (variants) through the process of denormalization of the views. Further, we define an indexing strategy for aggregate views, by including traditional B-tree indexes and advanced techniques that include bitmapped indexes. We name our optimization system - SFI since it includes selection of views (S), their vertical fragmentation (F) and their indexing (I) is introduced.

The objective function evaluates the quality of a solution. In [1, 2, 9,

10, 23] a simplified objective function (linear cost model) was used. They suppose that the query processing cost is the number of rows that are processed. The linear cost model is not adequate for evaluating join queries with complex selection predicates. The optimization is usually considered with a certain constraints which divide all solutions (whole solution space) into two groups of solutions: feasible and infeasible. There are two types of constraints: real system and logical constraints. System constraints are well studied in existing research prototypes [1, 2, 9, 10, 20, 22, 23]. They can be disk space or maintenance cost constraints. In [1, 2, 10] the optimization system was used under disk space constraints and formal constraints were embedded in the penalty function. In [9, 23] the objective function involved query processing cost under views maintenance (refreshment) cost constraint. In [23] the constraints were incorporated into the algorithm through a stochastic ranking procedure.

In this paper we use a non-linear cost model under the views of the maintenance cost constraint. The system was tested for complex workload, i.e. queries with projection, selection, join and grouping operations and with complex selection predicates. Also the logical constraints were considered and effects of solutions which violate logical constraints to the optimization process were analyzed.

Some types of Genetic Algorithms - GA as optimization method were used in [3, 14, 16, 20, 23, 24], and greedy algorithm with its variants and some heuristic searching techniques were used in [2, 5, 8, 9, 13]. To determine the suitability of the genetic algorithm and constraint handling to the data warehouse optimization problem, we compare it to a widely used greedy algorithm. We observe that for a generalized solution space and for large optimization problems, the greedy algorithms presented in [13] have poor performances compared to the genetic algorithm presented in [23].

We also adapt SRGA (Stochastic Ranking evolutionary (Genetic) Algorithm) introduced in [23]. The algorithm was used in SFI system, i.e. it was applied to generalized solution space, based on our multi-dimensional model. Further, two novel hybrid algorithms named GGLA (Greedy-Genetic Linear Algorithm) and GGBA (Greedy-Genetic Binary Algorithm) are presented. Both of them are combination of Greedy and Genetic algorithms. The genetic parts of the algorithms are also based on SRGA. The algorithms were applied to the generalized solution space.

The algorithms were applied to large optimization problems. For comparison, in [23] the SRGA algorithm was successfully applied to the solution

space that consists of 16 to 256 views, while in our work the algorithms were successfully applied to the solution space (views, fragmented views and indexes) up to 1110 objects. Furthermore, the system was evaluated using non-linear objective function and tested for complex workload. To show the efficiency of our novel algorithms we have compared them with SRGA. We have made many experiments which verified dramatic improvements (up to 280%) of the optimization process. The conclusion is that our optimization algorithms are much more effective and powerful then that developed in [23].

The paper is organized as follows: in Section 2 we introduce the definition of the solution space, in Section 3 we consider the optimization problem and present generalized genetic algorithm (SRGA) and novel hybrid algorithms (GGLA, GGBA) that successfully solves the aforementioned optimization problem. In Section 4 we show an experimental evidence of the efficiency of our approach and our algorithms applied to generalized solution space and finally, in Section 5 we give conclusions and post several open problems.

2 Definition of a Solution Space

In this section we define the generalized solution space of the optimization system. The solution space is based on a RDW schema which includes definition of dimension relations, all possible aggregate views, their different states of normalization, named variants of views and relational dimensions as well as all possible indexes.

2.1 Definition of an Implementation Schema

The warehouse data are multidimensional in nature, conceptually organized in a multidimensional data cube. The data are stored in specialized relations (tables), called fact and dimension relations. In the most real-life implementations the dimensions consist of more than one attribute, organized in attribute hierarchy. According to the attribute hierarchy presentation there are two schemas of implementation: star and snowflake schema.

In the star schema all attributes of each dimension are stored in one relation, i.e. the attribute hierarchy is presented implicitly. Data in the schema are denormalized which provides optimized complex aggregate query processing.

On the other hand, in the snowflake schema, attributes of each dimension are normalized and stored in different relations, i.e. the attribute hierarchy is presented explicitly. The snowflake schema offers flexibility, however, it is often at the cost of performance since more joins for each query are required.

In this subsection we define a multidimensional model which captures both schemas as extreme states and furthermore all intermediate state of (de)noramlization.

Definition 1 *The dimension D is defined as the pair $D = (R_D, H_D)$ with:*

- $R_D = \{R_1, ..., R_k\} \neq \emptyset$ *as the set of base (dimension) relations, where each dimension relation R_i is characterized by the basic set of attributes $BA_i = PA_i \cup DA_i \cup FA_i$, where $PA_i \neq \emptyset$ is the set of primary key attributes (identifying attributes), $DA_i \neq \emptyset$ is the set of descriptive attributes and FA_i is the set of foreign key attributes.*

- $H_D = \{H_1, ..., H_m\}$ *as the set of dependencies between dimension relations named dimension hierarchy, where each H_i is characterized by two dimension relations R_j, R_p, and it is presented by the pair $H_i(R_j, R_p)$, and also $PA_j \subseteq FA_p$.*

We note that for each dependence $H_i(R_j, R_p)$ the relation $PA_j \subseteq FA_p$ is satisfied. Intuitively, the dimension D is relational schema in 3th normal form - NF and no functional dependencies exist between attributes of each DA_i. However, for the dimension D the next restriction is introduced:

The graph of dimension D, obtained by interpreting dimension relations as nodes and dependencies between them as arcs, is a directed acyclic graph with the following elements:

- Exactly one root node, i.e. dimension relation R_r , satisfies $\forall H_i(R_j, R_p) \in H_D, R_r \neq R_j$;

- Non empty set of leaf nodes $\{R_l | \forall H_i(R_j, R_p) \in H_D, R_l \neq R_p\}$;

For better formal presentation the functions fR, fPA and fBA are defined as $fR(PA) = R$, $fPA(R) = PA$ ($fR = fPA^{-1}$), $fBA(R) = BA$, where R is a dimension relation and PA, BA are its primary and basic set of attributes, respectively.

Definition 2 *The extended set of attributes EA_i of dimension relation $R_i \in R_D$ of dimension $D = (R_D, H_D)$ is defined by the following algorithm:*

Find_Set (R_i)
Begin
    $EA_i = fBA(R_i)$
    For all R_k such that there exists $H_l(R_k, R_i) \in H_D$ Do
        $EA_i = EA_i \cup$ *FindSet (R_k)*
    End_For
    Return EA_i
End Find_Set

The additional set of attributes AA_i of dimension relation R_i is defined as $AA_i = EA_i \backslash BA_i$. Additional sets of attributes are important to realize our idea of denormalization of the schema of dimension relations. Actually, for each dimension relation, attributes of its additional set can be added to the basic set. Also, fEA function with $fEA(R) = EA$ is defined, where R as a dimension relation and EA as its extended set of attributes.

Definition 3 *The data cube DC is defined as the pair $DC = (DC_D, M)$, where $DC_D = \{D_1, ..., D_n\}$ is the set of dimensions and M is the set of measure attributes.*

Definition 4 *The implementation schema SC of data cube DC is defined as the pair $SC = (DC, AV)$, where $AV = \{V_1, ..., V_p\}$ is the set of aggregate views, where each V_l is characterized by:*

- *set of measure attributes $M_l \subseteq M$, where M is the set of measure attributes of DC;*
- *basic set of grouping attributes defined as $BGA_l = \cup_{D_i \in DC_D^l} fPA(R_j^i)$, where $DC_D^l \subseteq DC_D$ is a subset of dimensions of DC and $R_j^i \in R_{D_i}$;*
- *if $fPA(R_j^i), fPA(R_m^i) \subseteq BGA_l$ and $R_j^i, R_m^i \in R_{D_i} \Rightarrow j = m, \forall j, m = 1, ..., k_i$, where k_i is the number of relations of D_i dimension.*

Intuitively, aggregate views can contain different subsets of the set of measure attributes (first item) which provide vertical fragmentation of measure attributes. Between the appropriate dimension relations and the aggregate

views there are 1:M relationships, which means that the primary key attributes of dimension relations at the same time are grouping and foreign key attributes in the corresponding aggregate view (second item). Grouping ("group by") attributes of any aggregate view consists of primary key attributes from at most one relation of each dimension (third item).

The set of all dimension relations in the implementation schema SC is $R_{SC} = \cup_{D_i \in DC_D} R_{D_i}$.

To ensure that all data of data cube DC will be stored in at least one view of implementation schema SC, it is necessary to define the view with highest granularity of data and with all measure attributes.

Definition 5 *The view V_p is primary aggregate view of the implementation schema $SC = (DC, AV)$ if it is characterized by $M_p = M$ as the set of measure attributes and by $BGA_p = \cup_{D_i \in DC_D} fPA(R_r^i)$ as the set of grouping attributes, where DC_D is the set of dimensions of data cube DC and R_r^i is root relation of D_i dimension. All other aggregate views are named supporting views.*

In the next two definitions, similarly to the definitions for dimension relations, we define extended and additional set of attributes of aggregate views.

Definition 6 *The extended set of grouping attributes EGA_i of aggregate view V_i is defined as $EGA_i = \cup_{fPA(R) \subseteq BGA_i} fEA(R)$, where BGA_i is the basic set of grouping attributes of V_i.*

The additional set of grouping attributes AGA_i of aggregate view V_i is defined as $AGA_i = EGA_i \backslash BGA_i$.

Also, the definition of data dependencies between aggregate views of a data cube implementation schema is introduced:

Definition 7 *The aggregate view V_i depends on the aggregate view V_j in the implementation schema $SC = (DC, AV)$ if: 1. $M_i \subseteq M_j$ and 2. $BGA_i \subseteq EGA_j$.*

Intuitively, V_i can be created using tuples of V_j. The data of V_j are at a lower level of granularity than data of V_i. Note that using our definition of aggregate views and the second item of the definition of view dependency one can create structural lattice (directed graph) in a similar way as the lattice introduced in [8].

Lemma 1 *If the aggregate view V_i depends on the aggregate view V_j in the implementation schema $SC = (DC, AV)$, then $EGA_i \subseteq EGA_j$.*

Proof: Let assume that $\exists D_k \in DC_D^i$ with the extended set of attributes EA_k and $\exists BA_l \subseteq EA_k$ and $BA_l \subseteq EGA_i \backslash EGA_j$. Let R_r^k be the root relation of D_k, then from Definition 1 and 4, $BA_r \subseteq BGA_i$, where $BA_r = fBA(R_r^k)$ and BGA_i is the basic set of attributes of V_i. Then $BA_r \subseteq EGA_j$ and from construction of EGA_j (Definition 2 and 6) $\Rightarrow BA_l \subseteq EGA_j$, which is a contradiction. □

2.2 Vertical Fragmentation of Views

Our idea is to start from the fully normalized data cube schema and consider all possible states of denormalization. We have used a theorem which formally proves that the dimension relation of a star schema can be considered as a view defined on the respective relations of the snowflake schema (see [18]). There are two possible levels of denormalization. The first one is the denormalization of dimensional relations. For each dimension relation, it is possible to add any subset of its set of additional attributes to its basic set of attributes.

Definition 8 *The dimension relation R^i derived from the dimension relation R by adding (possible empty) subset $AA^i \subseteq AA$ to BA is a variant of original relation R, where R is characterized by BA as the basic set of attributes and by AA as the additional set of attributes.*

The relation variant R^i is characterized by its basic set $BA^i = BA \cup AA^i$ and note that $BA \subseteq BA^i \subseteq EA$. The second level is the denormalization of the aggregate views. In a similar way, for each aggregate view, it is possible to add any subset of its set of additional attributes to its basic set of grouping attributes.

Definition 9 *The aggregate view V^i derived from the aggregate view V by adding (possible empty) subset $AGA^i \subseteq AGA$ to BGA is a variant of the original view V, where V is characterized by BGA as the basic set of grouping attributes and by AGA as the additional set of grouping attributes.*

The aggregate view variant V^i is characterized by its basic set $BGA^i = BGA \cup AGA^i$ and note that $BGA \subseteq BGA^i \subseteq EGA$.

Theorem 1 *If the aggregate view V_k depends on the aggregate view V_n then any variant of V_k depends on any variant of V_n.*

Proof: All variants of the same aggregate view have the same set of measure attributes, so it is necessary to prove only the second condition of Definition 7. From the construction of extended set of attributes (Definition 2 and 6) and from Definition 9 of view variants follows that extended set of attributes is the same for all variants of the same view. So $BGA_k \subseteq BGA_k^i \subseteq EGA_k$ and $BGA_k \subseteq EGA_n$, then from Lemma 1, follows $BGA_k^i \subseteq EGA_n$. □

The previous theorem assures that data dependencies between view variants are the same as view dependencies defined in Definition 7.

We note that the number of variants of aggregate views depends exponentially on the number of elements of their additional set of grouping attributes. But, to simplify the problem, i.e. to decrease the number of variants, according to a priori defined heuristic, it is possible to group additional attributes of the same set in subsets of attributes. In this case the number of variants will depend exponentially on the number of new subsets.

2.3 Indexes

The next step in definition of solution space is determining the indexing strategy, which is very important since indexes can improve query execution time substantially.

Definition 10 *The index I on the aggregate view V_j in the implementation schema $SC = (DC, AV)$ is defined as the tuple $I = (V_j, IA_I, IA_I^k)$, where $V_j \in AV$, $IA_I \subseteq EGA_j$, $IA_I \neq \emptyset$ is the set of indexed attributes and IA_I^k is the ordered sequence of all elements of IA_I.*

We note that EGA_j is the extended set of grouping attributes of the aggregate view V_j. Intuitively, there can be several B-tree indexes on a given view, i.e. one index for every subset of the attributes of an extended set and every ordering of the subset.

Definition 11 *The index I is applicable on the variant V_j^i of aggregate view V_j if $IA_I \subseteq BGA_j^i$, where $IA_I \neq \emptyset$ is the set of indexed attributes and $BGA_j^i \subseteq EGA_j$ is the basic set of attributes of V_j^i and EGA_j is the extended set of grouping attributes V_j.*

The index I is named as fat index on V_j^i if $IA_I = BGA_j^i$ and index I is named as slim (one-attribute) index if $|IA_I| = 1$, i.e. cardinality of IA_I is 1. The set of indexes in implementation schema SC is SI.

According to the theory of indexing, the novel bitmapped indexes are very suitable for processing of data cube based queries. A bitmap index is usually defined for one attribute. However it is easy to process complex conditions involving more than one attribute. Thus, in experiments of this work one-attribute bitmap indexes were considered. Note that the number of one-attribute bitmap indexes of aggregate view V_j is equal to the number of elements of its extended set of grouping attributes EGA_j. In rest of this paper terms (one-attribute) bitmap index and index will be used simultaneously. The sequence of all indexes of aggregate view V_j is disposed in advance.

Definition 12 *The view length L_{V_j} of aggregate view V_j is defined as $L_{V_j} = 1 + k + m + n$, where k is the number of elements of the additional set of grouping attributes of the view AGA_j, m is the number of elements of the set of measure attributes and n is the number of bitmap indexes of the view, i.e. number of elements of the extended set of grouping attributes EGA_j.*

The parameters L_{V_j} are necessary to calculate the number of bits (genes) for representation of the solutions in our algorithms.

2.4 Workloads

A DW responds to a large number of data cube based ad-hoc queries, i.e. to dynamic and in principle unpredictable queries. However, lot of them can be determined a priori and can be formalized. In [17] processing of grouping queries on DW was researched, while in [8] all possible slice queries based on data cube views were considered. Selection ("where clause") attributes are disjoint from grouping attributes. That means they first select data by a certain set of attributes and after that they group them by another set of attributes. But in [8] different data operators, i.e. predicates in selection expressions are not considered. The predicates in selection expressions are also important for query processing time, i.e. different predicates returns different numbers of tuples. Therefore, in this paper, in our query definition, a set of different predicates is included.

Definition 13 *The query Q in the implementation schema $SC = (DC, AV)$ is defined as the tuple $Q = (M_Q, GA_Q, P_Q, F_Q)$, where $M_Q \subseteq M$ is the set*

of measure attributes, $GA_Q \subseteq EGA_p$ is the set of grouping attributes, P_Q is the selection expression and F_Q is the expected frequency of the query.

The selection expression P_Q of the query Q is in the form: $p_1 AND p_2 AND ... AND p_m$, where $p_i : A_i \theta Value_i$, $\theta \in \{=, <, >, \leq, \geq\}$ is a data operator, $Value_i \in Dom(A_i)$. The set SA_Q defined as $SA_Q = \{A_1, ..., A_m\}$ is the set of selection attributes and $SA_Q \subseteq EGA_p$ (EGA_p is the extended set of attributes of the primary view). The data operators are divided in two groups: equality and inequality data operators. Inequality operators usually return result which contains more then one tuple, so after selection by certain set of attributes it is reasonable to group resulting data by the same set of attributes. By this reason, sets of grouping and selection attributes are not disjoint. The set of all possible queries in the implementation schema SC is SQ.

Definition 14 *The ratio of equality operator ER in the selection expression of query Q is defined as k/n, where $n > 0$ is the number of data operators, i.e. the number of elements of the set of selection attributes SA_Q and k is the number of equality operators.*

The ratio of equality operator is necessary to estimate the number of tuples returned in each step of query execution, i.e. to calculate query execution cost. This is important to develop a realistic query evaluator.

Definition 15 *The query Q is computable from (can be answered by) the aggregate view V in the implementation schema $SC = (DC, AV)$ if: 1. $M_Q \subseteq M_V$ and 2. $GA_Q \cup SA_Q \subseteq EGA_V$, where M_V, EGA_V is the set of measures and the extended set of grouping attributes of V, respectively.*

The previous definition is necessary to determine the set of views from which each query can be answered. The next step is to determine such view from which query is answered by the lowest execution cost.

Theorem 2 *If the query Q is computable from the aggregate view V then Q is computable from any variant of V.*

Proof: The first condition of Definition 15 is satisfied because all variants of the same aggregate view have the same set of measure attributes and proof of the second condition directly follows from Definition 9 of variants of views. □

The important issue is how to select such a variant of each dimensional relation, such a set of views for materialization (each view presented with its proper variant) in order to minimize the total query processing time of queries with a certain constraint.

3 Definition of an Optimization System

3.1 A Practical Example

For better representation of our formal model the practical example of sale database system is considered. The three dimensional data cube $SALE$ with $SALE_D = \{I, S, D\}$ and $M = \{S\_quantity, S\_amount, S\_price\}$ is considered. The dimension relations, organized in dimension hierarchies, are shown in Figure 1. Simplified schema of the data cube, with all dimension relations and only with two views (the primary view and one supporting view of the schema) formally is described by:

```
Item (I_id, It_id, I_name);
Item_type(It_id, It_name)
Supplier (S_id, C_id, S_name);
City (C_id, Co_id, C_name);
Country (Co_id, Co_name);
Date (D_id, W_id, M_id, D_date);
Week (W_id,  D_week);
Month (M_id, Y_id, D_month);
Year (Y_id, D_year);
Sale_ISD (I_id, S_id, D_id, S_quantity, S_amount, S_price);
Sale_ItS (It_id, S_id, S_quantity, S_price);
```

and graphically is shown in Figure 2.

Set of relations of dimension I is $R_I = \{Item, Item\_type\}$. Relation *Item* is characterized by PA=$\{I\_id\}$, DA=$\{I\_name\}$, FA=$\{It\_id\}$, BA=$\{I\_id,$ $I\_name,$ $It\_id\}$, EA=$\{I\_id,$ $I\_name,$ $It\_id,$ $It\_name\}$ and AA=$\{It\_name\}$. Number of variants of *Item* relations is $2^1 = 2$. Dimension hierarchy of dimension I is defined as $H_I = \{It - I\}$, where $It - I : Item\_type \rightarrow Item$, i.e. $It - I(Item\_type, Item)$. According to the dimension hierarchies, the number of aggregate views of schema of the data cube $SALE$ is 60. Primary aggregate view is $Sale\_ISD$ and is characterized by set of measure attributes $M_{Sale\_ISD} = \{S\_quantity, S\_amount, S\_price\}$ and by $BGA_{Sale\_ISD} = \{I\_id, S\_id, D\_id\}$ as basic set of grouping attributes.

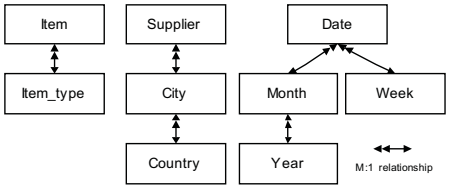

Figure 1: Dimension hierarchies for *I, S* and *D* dimensions

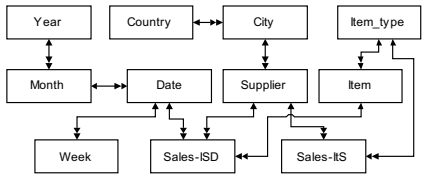

Figure 2: Simplified schema of data cube *SALE*

Extended set of grouping attributes is EGA_{Sale_ISD}={*I_id, I_name, It_id, It_name, S_id, S_name, C_id, C_name, Co_id, Co_name, D_id, D_date, W_id, D_week, M_id, D_month, Y_id, D_year*}, i.e. all attributes. Number of variants of aggregate view *Sale_ISD* is $2^{15} = 32768$. Supporting aggregate view *Sale_ItS* is characterized by set of measure attributes $M_{Sale\_ItS} = \{S\_quantity, S\_price\}$ and by $BGA_{Sale\_ItS} = \{It\_id, S\_id\}$ as basic set of grouping attributes. Extended set of grouping attributes is EGA_{Sale_ItS}= {*It_id, It_name, S_id, S_name, C_id, C_name, Co_id, Co_name*}. Number of variants of *Sale_ItS* is $2^6 = 64$.

Note that *Sale_ItS* is computable from *Sale_ISD*. Actually, if *Sale_ISD* is given by its zero variant $Sale\_ISD^0$, this means that the view *Sale_ItS* can be created by the following SQL statement:

```
CREATE MATERIALIZED VIEW Sale_ItS AS
SELECT I.It_id, F.S_id, SUM(F.S_quantity) S_quantity,
       AVG(F.S_price) S_price
FROM Sales_ISD F, Item I
WHERE I.I_id=F.I_id
GROUP BY I.It_id, F.S_id;
```

To create (compute) *Sale_ItS* it is necessary to join aggregate view *Sale_ISD* and dimension relation *Item*. But if *Sale_ISD* is given by its variant $Sale\_ISD^1$ characterized by $BGA^1_{Sale\_ISD}$ = {*I_id, It_id, It_name, S_id, S_name, D_id*} then the view *Sale_ItS* can be created by the following SQL statement:

```
CREATE MATERIALIZED VIEW Sale_ItS AS
SELECT F.It_id, F.S_id, SUM(F.S_quantity) S_quantity,
       AVG(F.S_price) S_price
FROM Sales_ISD F
```

```
GROUP BY F.It_id, F.S_id;
```

Note that maintenance cost of $Sale\_ISD$ is increased by adding the new attributes, but at the same time maintenance cost of $Sale\_ItS$ is decreased. Also note that view $Sale\_ItS$ is presented by its zero variant $BGA_{Sale\_ItS} = BGA^0_{Sale\_ItS} = \{It\_id, S\_id\}$.

If we assume that the view *Sales_ItS* is presented by its zero variant, i.e. materialized by above SQL statement, then to process the following query, named *QSale* (sales of type items "Milk Products" by supplier names), it is necessary to join proper dimension relations to the materialized view:

```
SELECT F.It_id, F.S_id, S.S_name, SUM(F.S_quantity) S_quantity
FROM Sales_ItS F, Item_type It, Supplier S
WHERE It.It_name="Milk Products" AND F.It_id=It.It_id
      AND F.S_id=S.S_id
GROUP BY F.It_id, F.S_id, S.S_name;
```

Formally, the query *QSale* can be described by following sets: $M_{QSale} = \{S\_quantity\}$ as set of measure attributes, $GA_{QSale} = \{It\_id, S\_id, S\_name\}$, $P_{QSale} : It\_name = "MilkProducts"$ as selection expression and set of selection attributes $SA_{QSale} = \{It\_name\}$. Expected query frequency id $F_{QSale} = 1$. To reduce the processing time required for joining views and dimension relations, frequently accessed attributes can be stored into materialized views. Thus, another possible way to create view *Sales_ItS*, from variant of *Sales_ISD* with $BGA^2_{Sale\_ISD} = \{I\_id, It\_id, It\_name, S\_id, S\_name, D\_id\}$), i.e. another variant of *Sales_ItS* is:

```
CREATE MATERIALIZED VIEW Sale_ItS AS
SELECT F.It_id, F.It_name, F.S_id, F.S_name,
       SUM(F.S_quantity) S_quantity, AVG(F.S_price) S_price
FROM Sales_ISD F
GROUP BY I.It_id, It.It_name, F.S_id, S.S_name;
```

The second variant $Sales\_ItS^1$ of the view *Sales_ItS* is characterized by the set $BGA^1_{Sale\_ItS} = \{It\_id, It\_nameS\_id, S\_name\}$ of grouping attributes. It is easy to note that the number of tuples is the same as the first variant of the view. To process previous query now it is not necessary to join dimension relations to fact relation and the SQL statement for the query is:

```
SELECT F.It_id, F.It_name, F.S_id, F.S_name, F.S_quantity
FROM Sales_ItS F
WHERE F.I_tname="Milk Products";
```

The advantage of using the second variant is decreasing the processing time of the queries, but the disadvantage is increasing the storage space, i.e. maintenance cost. In [5] some frequently accessed dimension keys and attributes are stored in various materialized views. However, a serious problem is to consider set of all possible variants of the data cube views and to find the optimal one with largest benefit of query processing and minimal maintenance cost.

3.2 An Objective Function

In this section we will propose a suitable evaluation function of the optimization process. Let SC_M be the state of the data cube schema SC with the set $AV_M \subseteq AV$ of candidate views for materialization where each of them is presented by exactly one of its variants and the set $SI_M \subseteq SI$ of its candidate indexes. Let also all dimension relations be presented by their appropriate variants. Then maintenance-cost constrained optimization problem is the following one: Select a state SC_M of data cube schema SC that minimizes

$$\tau(SC, SC_M, SQ) = \sum_{Q \in SQ} F_Q * P(Q, SC_M),$$

under the constraint $U(SC, SC_M) \leq S$, where SQ is the set of predefined queries, F_Q is query frequency and $P(Q, SC_M)$ denotes the minimum processing cost of the query Q in the SC_M state of implementation schema SC.

Let $U(SC, SC_M)$ be total maintenance cost defined as:

$$U(SC, SC_M) = \sum_{R \in R_{SC}} G_R * m(R, SC_M) + \sum_{V \in AV_M} G_V * m(V, SC_M) + \sum_{I \in SI_M} G_I * m(I, SC_M),$$

where G_R, G_V and G_I is update frequency of relations, views and indexes, respectively. Let $m(R, SC_M)$, $m(V, SC_M)$ and $m(I, SC_M)$ be the minimum cost of maintaining relations, views and indexes, respectively in presence of state SC_M.

We note that $P(Q, SC_M)$ is objective function of the problem. To calculate values of the functions $P(Q, SC_M)$, $m(R, SC_M)$, $m(V, SC_M)$ and $m(I, SC_M)$ we developed algorithms based on common query execution (processing) theory, presented in [7] and also on some ideas from [1, 2, 11, 14], as well as [17].

The common method for dealing with constrained optimization problems is to introduce a penalty function to the objective function. We defined the penalty function as $\phi(SC, SC_M) = Max\{U(SC, SC_M) - S, 0\}$. The function is used in the *Algorithm 2* for comparing pairs of two adjacent chromosomes.

3.3 Handling of Logical Constraints

As we said, each view is presented by exactly one bit, followed by bits used to present its additional attributes (view variants), its measure attributes and its indexes. If a view is not selected for materialization, all its bits which present view variants are irrelevant for evaluation of the current solution, i.e. adding additional attributes to the view is logically infeasible. We have named those bits as *recessive bits*. In this case the bits which present measure attributes and indexes of the view are also recessive bits. Thus, there are three types of recessive bits: for representations of view variants, for the measures attributes and for the indexes, named as variant, measure and index bits, respectively. Note that the number of recessive bits is very large in regard of other bits. They are irrelevant for evaluation of the current solution, but they are important for optimization process in the next generations, i.e. they are important for GA operations: crossover and mutation. This means that recessive bits comprise large genetic material and by their handling we can change the direction and we can improve the performances of the optimization process.

For example, if the view is not chosen for materialization in the first generation and all recessive bits have value 0, then if in the second generation a bit that presents a view mutate to 1, then that view will be chosen for materialization and it will be presented by a basic variant, without additional attributes and without indexes. For the opposite situation, if all recessive bits have value 1, then after a mutation, the view will be presented by variant with all additional attributes and with all indexes. In this paper we examine three different strategies for recessive bits handling. By the first strategy (RBR) recessive bits are generated randomly and can be changed by operation of mutation. By next two strategies all recessive bits have fixed value 1 (RB1) and 0 (RB0), respectively and can not be changed by operation of mutation. Effects of each strategy to the optimization process, according to the different parameters, are shown in next section.

3.4 The Generalized Genetic Algorithm

In this paper SRGA (Stochastic Ranking Genetic Algorithm) is adapted and applied to the solution space generalized by including view fragmentation and index selection, so also we named the algorithm as GGA (Generalized Genetic Algorithm).

First, we present the solution space for SRGA by an array of bits, i.e. by a chromosome. We start with the representation of dimension relations (as special types of views for materialization), including their different variants. The number of bits that are needed to present each dimension relation with all its variants is $\log_2 n + 1$, where the n is the number of all possible variants and $\log_2 n$ is equal to the number of elements of an additional set of attributes. Thus, the first bit is used to present a dimension relation and other n bits to present additional attributes. As we said, all dimension relations must be materialized, thus each of them has a value of 1. If an attribute of the additional set is to be added to its dimension relation, it has a value of 1 and if it is not added it has a value of 0. Aggregate views are presented in similar way, i.e. for each view, one bit is used for its representation (by 1 if it is selected or by 0 if it is not selected for materialization) and n bits are used to present its additional attributes. The attributes of additional sets of aggregate views are presented in the same way as attributes of additional sets of dimension relations. Finally, for each aggregate view, the measure attributes are presented. A measure attribute, if it is added to the appropriate view has a value of 1 and if it is not added, then it has a value of 0. Note that for each view at least one measure attribute must be added. The representation of each view is followed by representation of its possible indexes. Each index is presented exactly by one bit, i.e. by 1 if it is selected or by 0 if it is not selected. The number and sequence of all indexes are disposed in advance.

For example, the view length of aggregate view *Sales_ItS* is $L_{Sale\_ItS} = 1 + 6 + 2 + 8 = 17$. If *Sales_ItS* is presented by the 10101001010101000, then the first bit is used for representation of the view - it is selected for materialization. The next 6 bits are used for representation of the additional attributes - the view is presented by its variant with 2 additional attributes (*S_name, C_name*). The next 2 bits are used for representation of the measure attributes - the view has 1 measure attribute (*S_quantity*) The last 8 bits are used for representation of the indexes - 3 indexes for *It_id, S_id and C_id* attributes are selected for materialization. If the view is not materialized, all its variant, measure and index bits are irrelevant for

evaluation of the current solution.

Input parameters of the SRGA algorithm are: NC - Number of Chromosomes (population size), NG - Number of Generations, LC - Length of Chromosome (number of bits needed to present whole solution space), PC - Probability of mutation of Chromosomes, PG - Probability of mutation of Genes, R_{SC} - set of dimension Relations, AV - set of Aggregate Views (including dimension relations as special types of views), SI - Set of Indexes, SQ is Set of Queries and PF is a Probability for rank-based selection, introduced in [23]. $POP(i)$ presents i^{th} generation of the population.

The initial population is randomly generated by the procedure *Create_population* according to the rules presented above. Each of its chromosomes is evaluated for a predefined workload, i.e. set of queries SQ by the procedure *Evaluate_population*. We note that *Evaluate_population* is implementation of the function $\tau(SC, SC_M, SQ)$ and by $POP(i)$ are presented NC particular states of implementation schema SC, each chromosome of the population presents a different state. The procedure *Evaluate_population_materialization* evaluates maintenance solution cost for each chromosome of the population $POP(i, LC)$ - it is implementation of the function $U(SC, SC_M)$.

Algorithm 1: $SRGA(NC, NG, LC, PC, PG, AV, SI, SQ, PF)$

Begin
  Create_population$(POP(1), NC, LC, AV, SI)$;
  Evaluate_population$(POP(1), SQ)$;
  Evaluate_population_materialization$(POP(1))$;
  For $i = 2$ *to* NG *Do*
    Perform_crossover$(POP(i-1), NC)$;
    Perform_mutation$(POP(i), PC, PG, NC)$;
    Evaluate_population$(POP(i), SQ)$;
    Evaluate_population_materialization$(POP(i))$;
    $POP(i) := Merge_populations(POP(i-1), POP(i))$;
    Perform_selection$(POP(i), NC, SQ, PF)$;
  End For;
End GGLA;

In order to obtain a population with better characteristics, the procedures *Perform_crossover*, *Perform_mutation* and *Perform_selection* perform GA operations - crossover, mutation and selection, respectively. We note that by applying the *Perform_crossover*$(POP(i-1), NC)$ procedure to the $(i-1)$-th generation of population, NC new chromosomes of i-th generation of the population are generated. We used a special case of multi-point

crossover operation where the number of crossover points is $n-1$, where n is the number of genes, i.e. blocks with length of 1 gene. This crossover is known as uniform crossover. The procedures *Perform_mutation*, *Evaluate_population* and *Evaluate_population_materialization* are applied to the new i-th generation. Probability to choose a chromosome for mutation is given by the parameter PC, and probability to mutate genes inside of a chosen chromosome is given by the parameter PG. Formally, the procedure *Merge_populations* adds chromosomes from previous $(i-1)$-th to the newest i-th generation.

The key difference from most GAs is the *Perform_selection* procedure, based on the stochastic ranking algorithm presented in [23]. The algorithm is presented in the next subsection. It is similar to bubble-sort algorithm used for ranking the union of chromosomes of last two generations. But in [23] the algorithm is used just for selection of views and it is applied to small solution space. In this paper the algorithm is applied to the generalized solution space which includes selection of views and indexes and selection of view fragments. Also, the algorithm is used to find a near optimal model of data cubes consist of a large number of objects.

3.5 The Stochastic Ranking Procedure

Finding a penalty coefficient optimal value is difficult and the penalty methods setting a static or dynamic penalty coefficient value do not work well for the constrained optimization problems. In [23] is presented a new constraint handling technique, named stochastic ranking, to balance the dominance of the objective and penalty functions for the view selection problem. The novel idea of this technique is the introduction of a probability P_f for rank-based selection. During the course of ranking, it is necessary to compare pairs of two adjacent individuals. If they are both feasible solutions, naturally, they will be compared according to the objective function. However, when either of them is infeasible, the probability of comparing them according to the objective function is P_f, while the probability of comparing them according to the penalty function will be $1-P_f$. Since is a probability, it gives an opportunity for both the objective and penalty functions to rank a pair. When $P_f > 1/2$, the ranking is biased toward the objective function. When $P_f < 1/2$, the ranking is biased toward the penalty function. So P_f can balance the objective and penalty functions more directly, explicitly and conveniently. Moreover, we do not have any extra computing cost for setting penalty coefficient values since we do not use any penalty terms.

In this paper the stochastic ranking algorithm is implemented by the *Perform_selection* procedure (see Algorithm 2). The probability P_f is presented by the parameter PF. In our experiments we set the parameter value dynamically during the optimization process. For example, we usually set $PF < 1/2$ to reduce the ratio of infeasible solutions to the whole in the several final generations.

Finally, we note that the constraints of the optimization process are incorporated into the algorithm through a stochastic ranking procedure.

The input population $POP(i)$ consists of $2*NC$ chromosomes as union set of chromosomes from two generations. The procedure *Delete_Chromosome* eliminates NC worse chromosomes from the population. To select the chromosomes for elimination we use the negative (elimination) selection method presented in [12]. The reasons for that are: the population consists of $2*NC$ chromosomes, so in this case elimination selection is faster than generation selection; elitism is inherently involved in elimination selection.

Algorithm 2: *Perform_Selection*$(POP(i), NC, SQ, PF)$

```
Begin
  For k = 1 to NC Do
    Swap:=False;
    For j = 1 to NC * 2 − 1 Do
      If (φ(POP(i, j)) = φ(POP(i, j + 1)) = 0) or
        (Random(0, 1) > PF) Then
        If τ(POP(i, j), SQ) > τ(POP(i, j + 1), SQ) Then
          Swap_Chromosomes(POP(i, j), POP(i, j + 1));
          Swap:=True;
        End If
      Else
        If φ(POP(i, j)) > φ(POP(i, j + 1)) Then
          Swap_Chromosomes(POP(i, j), POP(i, j + 1));
          Swap:=True;
        End If;
      End If;
    End For;
    If Swap=False Then
      Exit For;
    End If;
  End For;
  Delete_Chromosome(POP(i), NC);
End Perform_Selection;;
```

3.6 The Novel Greedy-Genetic Algorithms

In this section we will present two novel hybrid algorithms for optimization of RDW. Our algorithms are hybrid because they are combination of greedy and genetic algorithms.

The Algorithm 3 is named GGLA - Greedy-Genetic Linear Algorithm. The parameter NF represents Number of Fragments of the solution space, which is equal to the number of steps of greedy procedure. All other input parameters of the algorithm are the same as in Algorithm 1.

Parameters of the Algorithm 4 are: NL - New Length of chromosome (input), CL - Current Length of the chromosome (input/output), AV_B - set of chosen views (input/output), AV_C - set of chosen views in current step (output), AV - set of Aggregate Views (input).

Algorithm 3: $GGLA(NC, NG, LC, PC, PG, NF, AV, SQ)$

```
Begin
  AV_B := ∅; CL := 0;
  Choose_views(Round(LC/NF), CL, AV_B, AV_C, AV);
  Extend_chromosome(POP(1), AV_C, NC);
  Evaluate_population(POP(1), SQ);
  Evaluate_population_materialization(POP(1));
  j := 2;
  For i = 2 to NG Do
    If Mod(i, Round(NG/NF)) = 1 Then
      Choose_views(Round(LC * j/NF), CL, AV_B, AV_C, AV);
      Extend_chromosome(POP(i), AV_B, NC);
      j := j + 1;
    End If;
    Perform_crossover(POP(i − 1), NC);
    Perform_mutation(POP(i), PC, PG, NC);
    Evaluate_population(POP(i), SQ);
    Evaluate_population_materialization(POP(i));
    Perform_selection(POP(i), NC);
  End For;
End GGLA;
```

The GGLA algorithm is graphically shown in Figure 3. Before the optimization process starts, we order all aggregate views by their ratio of usability (UR), i.e their importance. The general idea is in certain steps (NF) by greedy procedure to choose the subsets of views with all their bits (Algorithm 4, i.e. procedure *Choose_views*) and to add them to already selected ones, i.e. to concatenate their randomly generated bits to the

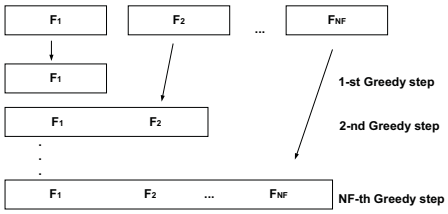

Figure 3: GGLA - Greedy-Genetic Linear Algorithm

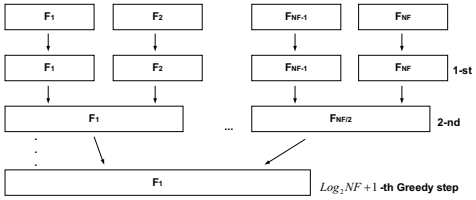

Figure 4: GGBA - Greedy-Genetic Binary Algorithm

already created chromosomes (*Extend_chromosome* procedure). By those procedures we roughly modulate the solution space.

After each step of greedy procedure, we perform fine optimization by using GA (few generations on current solution space). The procedure *Evaluate_population* evaluates the quality of solutions and the procedure *Evaluate_population_materialization* evaluates the maintenance solution cost. The procedures *Perform_crossover*, *Perform_mutation* and *Perform_selection* perform GA operations - crossover, mutation and selection, respectively.

Agorithm 4: *Choose_views*(NL, CL, AV_B, AV_C, AV)

Begin
  $AV_C := \emptyset$;
  While $NL > CL$ *Do*
    $AV_B := AV_B \cup V_i$, where V_i has maximal UR_i in $AV \backslash AV_B$;
    $AV_C := AV_C \cup V_i$;
    $CL := CL + L_{V_i}$
  End While;
End Choose_views;

The next algorithm is named GGBA - Greedy-Genetic Binary Algorithm. All input parameters are the same as in Algorithm 3.

The GGBA algorithm graphically is shown in Figure 4. $POP(i, AV_k)$ presents the i^{th} generation of the population and consists of a fragment of chromosomes represented by AV_k subset of views. $POP2(i)$ presents i^{th} generation of the population and consists of whole chromosomes (created by concatenation of all fragments of POP population), represented by set of all aggregate views AV. The population $POP2$ is necessary to evaluate the maintenance solution cost. The variable NS is the number of steps of the greedy procedure.

Algorithm 5: $GGBA(NC, NG, LC, PC, PG, NF, AV, SQ)$

```
Begin
  Divide_views(NF, AV);
  For k = 1 to NF Do
    Create_population(POP(1, AV_k), NC);
    Evaluate_population(POP(1, AV_k), SQ);
    Concat_whole_chr(POP(1, AV_k), POP2(1));
  End For;
  Evaluate_population_materialization(POP2(1));
  NS := log_2 NF + 1;
  For i = 2 to NG Do
    If Mod(i, Round(NG/NS)) = 1 Then
      For k = 1 to NF/2 Do
        Concat_chr(POP(i, AV_k), AV_{2*k-1}, AV_{2*k});
      End For;
      NF := NF/2;
    End If;
    For k = 1 to NF Do
      Perform_crossover(POP(i - 1, AV_k), NC);
      Perform_mutation(POP(i, AV_k), PC, PG, NC);
      Evaluate_population((POP(i, AV_k), SQ);
      Concat_whole_chr(POP(i, AV_k), POP2(i));
    End For;
    Evaluate_population_materialization(POP2(i));
    For k = 1 to NF Do
      Perform_selection((POP(i, AV_k), NC);
    End For;
  End For;
End GGBA;
```

In the procedure *Concat_chr* we define a new set of aggregated views $AV_k := AV_{2*k-1} \cup AV_{2*k}$ and new population $POP(i, AV_k)$ which consists of chromosome fragments created by concatenation of chromosome fragments of $POP(i, AV_{2*k-1})$ and $POP(i, AV_{2*k})$ populations. The chromosome fragments of both populations are ordered from the best to the worst evaluated and concatenated fragments at the same position. We named this concatenation strategy *best-to-best*. In similar way, the procedure *Concat_whole_chr* creates $POP2(i)$ as concatenation from populations of all subsets of views $POP(i, AV_k)$. Here we also use best-to-best concatenation strategy. All procedures with the exception of *Evaluate_population_materialization* can be parallelized for different fragments of chromosomes, i.e. subsets AV_k of AV which gives the total improvement of the performances of the algorithm.

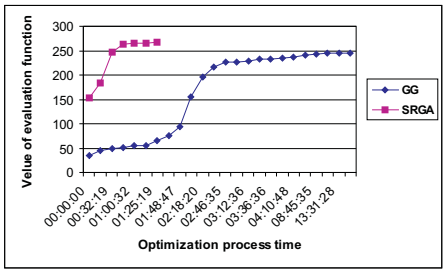

Figure 5: SRGA is better than GG for large optimizations problems

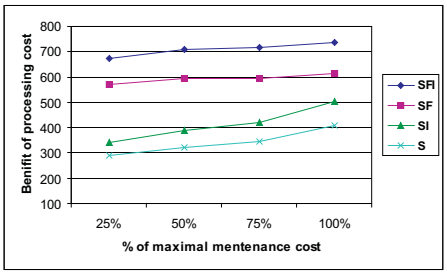

Figure 6: Benefit of processing cost versus maintenance cost constraint

4 Experimental Results

In this section we present our experimental system and some of the experimental results obtained by it. For the efficiency of the optimization process using GA several input parameters are important. In our previous work we have already described some experiments with a wide range of their values. In this paper we fixed the parameters to the following values: Number of Chromosomes (Population Size)- $NC = 20$, Probability of Chromosome mutation - $PC = 0.5$ and Gene mutation - $PG = 0.05$, so the overall probability of mutation is 2.5%.

To determine the suitability of the GA and constraint handling to the RDW optimization problem, we compare it to a widely used greedy algorithm (see Figure 5). The SRGA was compared to the greedy algorithm presented in [13] - GG (Greedy by Gupta). On the x-axis, the optimization process time is shown, even on the y-axis, the benefit of query processing cost (value of the evaluation function) is shown. We observe that for the generalized solution space and for large optimization problems, greedy algorithms have poor performances compared to the SRGA.

In order to show the efficiency of vertical view fragmentation of the approach of view selection with vertical view fragmentation and indexes (SFI), we compared it to the approaches without fragmentation (SI), without indexes (SF) and without fragmentation and indexes (S). Exactly 1280 experiments were performed with SRGA. For all experiments the termination condition of optimization process was 50 generations.

A comparison of the benefit of processing cost of all approaches, on different levels of maintenance cost constraints, is shown in Figure 6. The optimization process is considered with four different values of maintenance

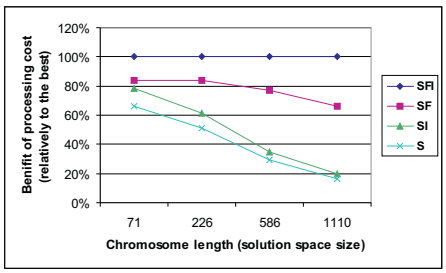

Figure 7: Benefit of processing cost versus solution space size

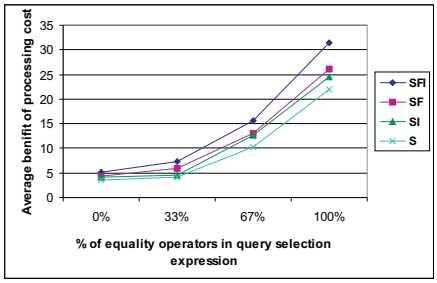

Figure 8: Benefit of processing cost versus % of equality operators

cost from 25% to 100% of the maximal maintenance cost. The next important parameter of the optimization process is the solution space size, i.e. chromosome length. Comparison of different approaches according to the solution space size is shown in Figure 7. On y-axis the benefit of query processing cost relatively to the best approach is shown. Evidently, in all cases SFI approach has the highest benefit of processing cost.

In our experiments we considered four different parameters in query definition: number of grouping (projection) attributes, number of selection attributes, ratio of the equality operators and number of measure attributes. For all parameters the SFI approach has the highest benefit of processing cost. The average benefit of processing cost of queries with 3 attributes in the selection expression for different ratios of the equality operators in the expression is shown in Figure 8.

As we have described in the previous section, recessive bits are irrelevant for evaluation of the current solution, but they could be important for optimization process in the next generations. In this work we examine three different strategies for recessive bits generation. By the first strategy (RBR), recessive bits are generated randomly and can be changed by operation of mutation. In the next two strategies, all recessive bits have fixed value 1 (RB1) and 0 (RB0), respectively and can not be changed by mutation operation. The benefit of the processing cost for each strategy according to the different values of maintenance cost is shown in Figure 9. Comparison of different strategies according to the solution space size is shown in Figure 10. In both cases the best strategy is RB1, i.e. to fix recessive to value 1. This doesn't mean that RB1 leads to the extreme solution in which all additional attributes and all indexes are selected, because

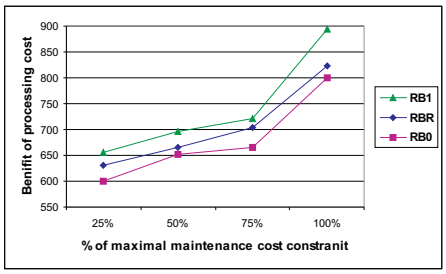

Figure 9: Benefit of processing cost versus maintenance cost constraint

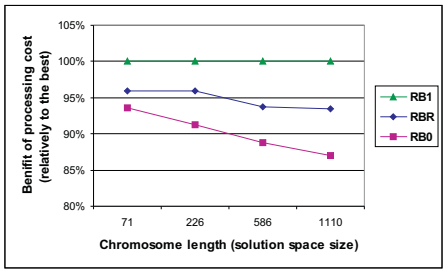

Figure 10: Benefit of processing cost versus solution space size

the maintenance cost constraints in next generations eliminate infeasible solutions that would be otherwise obtained by RB1.

To compare our novel algorithms (GGLA and GGBA) with SRGA, exactly 192 experiments were performed. For all experiments termination condition of optimization process was 64 generations. All algorithms were applied to the generalized solution space based of our system SFI.

We experimented with different values of the parameter NF - Number of Fragments. For GGLA the number of fragments is equal to the number of steps of the greedy procedure, while for GGBA the number of steps of the greedy procedure is given by $log_2 NF + 1$. A comparison of the optimization process execution time of all three algorithms, for different number of fragments NF, is shown in Figure 11. The optimization process was considered with four different values of NF parameter, from 4 to 32. Improvements of GGLA and GGBA over SRGA are evident when we increase the values of the parameter. The value of the parameter is limited by the number of generations of the optimization process (in our case 64). However, usually the number of generations increases by the increasing of the complexity of the problem, i.e. its solution space size.

A comparison of the different algorithms according to the solution space size is shown in Figure 12. For better representation of the performances of the algorithms on the same chart, we scaled values of the optimization process execution time. On the y-axis the benefit of optimization process execution time relatively to the worst algorithm is shown. Evidently, in all cases GGBA algorithm has the highest improvement of the optimization time. Note that improvements of both GGLA and GGBA algorithms rise by increasing the solution space size, which is another important feature:

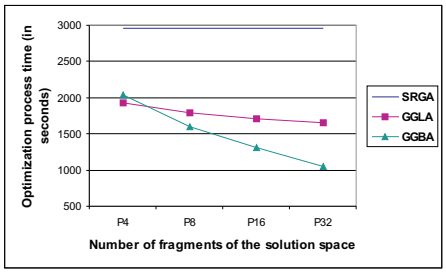

Figure 11: SRGA vs GGLA vs GGBA (time vs number of fragments)

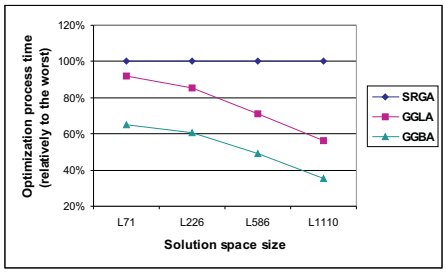

Figure 12: SRGA vs GGLA vs GGBA (optimization time vs solution space)

the scalability of our algorithms and their appropriateness for practical implementation.

In our algorithms we use a priori knowledge about usability of aggregate views and implement that knowledge as a heuristic in the greedy procedure for choosing views. Using greedy algorithm in the first step enables us to decrease the genetic material, i.e. the number of calculation in genetic part of GGLA algorithm and the possibility to parallelize some segments of genetic part in the GGBA algorithm. Finally, we note that values of given solutions by using GGLA and GGBA are in the range of 98%-101% of solutions given by using SRGA.

5 Conclusion and Open Problems

The performance of the system of RDW depends on several factors and the problem of its optimization is very complex and making a perfect system is still a challenge. In this paper we have focused on generalization of the optimization problem and improvement of the optimization process of RDW. We fully analyzed the problem by including lot of factors relevant for optimization of the system, i.e. view selection, vertical view fragmentation and index selection. Further we have focused on improvement of the efficiency of the optimization process.

By doing so we have achieved the following:

- We introduced a generalized model of optimization of RDW, named SFI, which is comprise of selection of views (S), their vertical fragmentation (F) and their indexing (I).

- By using a GA we have addressed and solved the problem of generalized model of optimization of RDW.
- We have introduced a new technique of "recessive bits" and we have analyzed the effects of those bits to the optimization process.
- We have performed a large set of experiments and confirmed the benefits of the SFI model according to several parameters.
- We have defined two novel algorithms GGLA and GGBA and we have successfully applied them for fast finding optimal solutions in the generalized solution space.

We have achieved significant performance improvements of the optimization process compared to the stochastic ranking genetic algorithm and we have verified those improvements by performing a large set of experiments. The result is an optimization algorithm that is much more effective and powerful than that developed in [23]. For comparison, the GA algorithm developed in [23] was successfully applied to a solution space that consists of 16 to 256 views, while our algorithm can be successfully applied to the solution space (views, fragmented views and indexes) of up to 1110 objects. Furthermore, the system was tested for complex queries with projection, selection, join and grouping operations and with complex selection predicates.

In the future, we plan to extend our multidimensional model by including horizontal partitioning and definition of a clustering strategy and to define an optimization process by applying the algorithms to the extended space. There is also the possibility to analyze different strategies for handling the recessive bits and the possibility to parallelize the algorithms on different levels such as: evaluation of chromosome within populations, parallelization of populations or parallel optimization of different data cubes. In this paper static algorithms are considered. Our plan is to develop dynamic algorithms for RDW design optimization.

Acknowledgements

This work was supported by NATO ICS.EAP.CLG.983334 project.

References

[1] K. Aouiche, J. Darmont, O. Boussaid, F. Bentayeb. Automatic Selection of Bitmap Join Indexes in Data Warehouses. *Proc. of the 7th International Conference on Data Warehousing and Knowledge Discovery, DAWAK 05*. Copenhagen, Denmark, pp. 64-73, 2005.

[2] K. Aouiche, P. Jouve, J. Darmont. Clustering-Based Materialized View Selection in Data Warehouses, *Proc. of the 10th East-European Conference on Advances in Databases and Information Systems, ADBIS'06*. Thessaloniki, Greece, pp. 81-95, 2006.

[3] L. Bellatreche, K. Boukhalfa. An Evolutionary Approach to Schema Partitioning Selection in a Data Warehouse. *Proc. of the 7th International Conference on Data Warehousing and Knowledge Discovery, DAWAK'05*, Copenhagen, Denmark, pp. 115-125, 2005.

[4] L. Bellatreche, M. Schneider, H. Lorinquer, M. Mohania. Bringing Together Partitioning, Materialized Views and Indexes to Optimize Performance of Relational Data Warehouses. *Proc. of the 6th International Conference on Data Warehousing and Knowledge Discovery DAWAK'04*, Zaragoza, Spain, pp. 15-25, 2004.

[5] G.K.Y. Chan, Q. Li, L. Feng. Optimized Design of Materialized Views in a Real-Life Data Warehousing Environment. *International Journal of Information Technology*, vol. 7, no. 1, pp. 30-54, 2001.

[6] R. Chirkova, Y.A. Halevy, D. Suciu. A formal perspective on the view selection problem. *Proc. of the 27th International Conference on Very Large Data Bases VLDB'02*, Hong Kong, China, pp. 216 - 237, 2002.

[7] R. Elmasri, S.B. Navathe. *Fundamentals of Database Systems*. Fourth Edition, Addison-Wesley Publishing Company Inc., 2003.

[8] H. Gupta, V. Harinarayan, A. Rajaraman, J.D. Ullman. Index Selection for OLAP. *International Conference on Data Engineering ICDE*, Birmingham, England, 1997.

[9] H. Gupta, S. Mumich. Selection of Views to Materialize Under a Maintenance Cost Constraint. *Proc. of the 7th International Conference on Database Theory, ICDT'99*, Jerusalem, Israel, pp. 453-470, 1999.

[10] M. Golfarelli, V. Maniezzo, S. Rizzi. Materialization of fragmented views in multidimensional databases. *Data & Knowledge Engineering*, Volume 49, Issue 3, pp. 325-351, 2004.

[11] M.Golfarelli, S.Rizzi, E.Saltarelli. Index Selection Techniques In Data Warehouse Systems. *Proc. of the International Workshop on Design and Management of Data Warehouses DMDW'02*, Toronto, pp.33-42, 2002.

[12] M. Golub. Improving the Efficiency of Parallel Genetic Algorithms. *Ph.D. thesis, Zagreb University*, Croatia, 2001.

[13] H. Gupta, Selection And Maintenance of Views in a Data Warehouse, *Ph.D. dissertation, Stanford University*, USA, 1999.

[14] J. Kratica, I. Ljubic, D. Tosic. A Genetic Algorithm for the Index Selection Problem. *Proc. of the Applications of Evolutionary Computing: EvoWorkshops 2003*, Essex, UK, pp. 280-290, 2003.

[15] Y. Kotidis, N. Roussopoulos. DynaMat: A Dynamic View Management System for Data Warehouses. *Proc. of the ACM SIGMOD International Conference on Management of Data*, Philadelphia, Pennsylvania, USA, 1999.

[16] M. Lee, J. Hammer, Speeding Up Warehouse Physical Design Using A Randomized Algorithm, *Proc. of the International Workshop on Design and Management of Data Warehouses DMDW'99*, Heidelberg, pp.1-9,1999.

[17] A. Tsois, N. Karayannidis, T. Sellis, D. Theodoratos. Cost-based optimization of aggregation star queries on hierarchically clustered data warehouses. *Proc. of the International Workshop on Design and Management of Data Warehouses DMDW'02*, Toronto, Canada, pp. 62-71, 2002.

[18] P. Vassiliadis, Formal Foundations for Multidimensional Databases (extended version). *NTUA Technical Report*, 1998.

[19] G. Velinov, M. Kon-Popovska, D. Gligoroski. Vertical Fragmentation in Relational Data Warehouses. *Proc. of the ROSYCS 2006, Database Theory and Practice in the context of (Semantic) Web Technologies*, Iasi, Romania, pp. 109-122, 2006.

[20] G. Velinov, M. Kon-Popovska, D. Gligoroski. Optimization of Relational Data Warehouses. *Proc. of the 4th European Conference on Intelligent Systems and Technologies, ECIT2006*, Iasi, Romania, 2006.

[21] G. Velinov, D. Gligoroski, M. Kon-Popovska. Recessive Bits in Genetic Algorithm for Some Optimization Problems in Relational Data Warehouses. *Proc. of the Third International Bulgarian-Turkish Conference - Computer Science'06*, Istanbul, Turkey, pp. 101-106, 2006.

[22] G. Velinov, D. Gligoroski, M. Kon-Popovska, Hybrid Greedy and Genetic Algorithms for Optimization of Relational Data Warehouses, *Proc. of the 25th IASTED International Multi-Conference: Artificial intelligence and applications*, Innsbruck, Austria, pp. 470-475, 2007.

[23] J.X. Yu, X. Yao, C. Choi, G. Gou. Materialized Views Selection as Constrained Evolutionary Optimization. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, Volume 33, No. 4, pp. 458-468, 2003.

[24] C.Zhang, X. Yao, J. Yang. An Evolutionary Approach to Materijalized Views Selection in a Data Warehouse Environment, *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, Volume 31, No. 3, pp. 282-294, 2001.