

Chapter 3

OPTIMIZING MATERIALIZED VIEWS

3.1 Introduction

The evaluation of materialized views (queries) may require considerable computational effort because some materialized views (1) can share some intermediate results (views) with other materialized views, or (2) are complex enough to justify some intermediate pre-computed views. To reduce the effort to maintain views updated, some intermediate views can be materialized. However, how many and which intermediate views will be materialized will depend on many factors, such as view maintenance costs, response time, available storage, etc.

The materialized view (query) optimization problem has been studied in different contexts, such as: query optimization, view maintenance, and data warehouse design and configuration. However, none of these works have provided an explicit problem formulation in terms of materialized view interrelations, neglecting the possibility to take advantage from it.

In this chapter we consider the optimizing materialized views problem, i.e., selecting views to materialize in order to optimize a criterion, subject to a set of materialization constraints (maintenance time, available storage, etc.). We propose an *optimization framework* to decide the optimum way to materialize views, i.e., which additional views need to be materialized. We use an *expression-DAG* to express equivalent view evaluation plans. Then, we formulate an optimization problem to make a decision.

This chapter is organized as follows: Section 3.2 presents related work. Section 3.3 presents the contributions. Section 3.4 presents the problem characterization, where the view optimization is formulated based on shortest (cheapest) path in an expression-DAG. In Section 3.5 we present a linear-time algorithm when all possible view evaluation alternatives are available, and a local search strategy for the general case. Finally, Section 3.6 presents some experiments and the implementation of the general shortest path algorithm.

3.2 Related Work

The view (query) selection problem has been studied in different contexts, such as: query optimizations [NKOD95], view maintenance [CW91, GL95, GMS93, GM95, RSS96], and data warehouse design [Gup97, HRU96, GHRU97, YKL97, BPT97] and configuration [TS97]. However, in terms of the solution, current research provides *near-optimal* heuristics (without guarantee of the solutions' quality), very expensive optimal

exhaustive search algorithms, or they just address to special cases.

In particular, works [BPT97, RSS96, TS97, YKL97] provide frameworks, heuristics, and an exhaustive search algorithm in order to optimize the sum of response and maintenance time without any constraints. Ross [RSS96] has proposed view selection based on a minimization of maintenance cost. However, [RSS96] presents an exhaustive search algorithm, which is exponential (double exponential) in the number of possible views to be materialized. Work [YKL97] formulates the problem as one of *integer programming* in terms of the view evaluation plan. However, the number of evaluation plans is exponential in terms of the number of possible views to be materialized.

The results reported in [Gup97, HRU96, GHRU97, GM98] provide a formulation with storage constraint and time evaluation constraint. Three of them provide *near-optimal* heuristics (greedy algorithm). In particular, [Gup97, GHRU97] extend [HRU96], and present a formulation as an optimization problem. Works [Gup97, GHRU97] provide polynomial-time heuristics (in terms of the number of possible views to be materialized) for two special cases (AND and OR graphs), and *near-optimal* exponential-time greedy algorithm for AND-OR graphs. However, the solution quality is not guaranteed. Finally, [GM98] extends previous works, which minimizes the response and maintenance time of selected views, subject to a maximum maintenance time. However, the heuristics and algorithms still present the same previous behavior.

3.3 Contributions

This chapter addresses the optimal view materialization problem, where for a given set of materialized views (queries), one must decide which additional (intermediate) views should be materialized in order to reduce the overall maintenance effort, under some materialization constraints. A standard mechanism to represent intermediate views corresponds to AND-OR graphs [Gup97, HRU96, GHRU97, GM98]. However, there is no work in which the structural properties of such graphs are exploited. In general, this problem is NP-hard [RSS96, Gup97], because it corresponds to selecting a subset of elements from the set of all intermediate views, where the number of subsets is exponential in the number of additional views.

This research exploits the structure of the representation mechanism for intermediate views. More specifically, the contributions are as follows. First, it extends the expression-DAG (Direct Acyclic Graphs) [RSS96] as mechanism to represent compactly intermediate views (queries) using equivalence and operation nodes. It shows that equivalence nodes correspond to nodes in an AND-OR graph, and operation nodes correspond to AND arcs. It characterizes an expression-DAG in terms of its size and expression-paths ¹ (i.e., complete view evaluation plans). However, while the size of a standard AND-OR graph is defined in terms of its nodes and arcs, the size of an expression-DAG is defined in terms of the cardinality of its operational nodes.

Second, an optimization framework is formulated in terms of the expression-DAG

¹Note that, under certain conditions, AND-paths are equivalent to expression-paths.

structure. Thus, under certain objective function conditions, the problem of optimal selection of materialized views can be formulated as the constrained shortest path in an expression-DAG, i.e., a complete expression-path or AND-path. For this case, a linear-time algorithm (in terms of the expression-DAG size) is presented. Note that this special case can be found in important applications such as the case when the evaluation time is the critical variable, and therefore, while more intermediate views are materialized, the complete evaluation should be more efficient. A set of experiments was run, and the results suggest that our approach is feasible.

Third, for the general optimization problem and if the expression-DAG has all possible view evaluation plans, then the linear-time algorithm can be applied to obtain a solution. Note that this algorithm does not evaluate all possible equivalent evaluation plans, because all those which are subsumed by others are eliminated earlier. Finally, if the expression-DAG with all possible view evaluation plans is not available, a local search algorithm is presented. However, further research is necessary in this area.

3.4 View Materialization Characterization

In this section we characterize the optimization problem to support an efficient and cost effective view materialization and maintenance. First, we present the basic definitions based on the relational model. Then, we formulate the optimization problem, characterizing the search space and the objective function. Finally, we present an equivalent

formulation and we discuss their effective solutions.

3.4.1 Definitions

This subsection describes basic definitions and concepts based on the relational database model [Ull88]. In particular, view and view evaluation plan are described.

Definition 18. *A database DB is a collection of n -relations (r_1, \dots, r_n) over relational schemes (R_1, \dots, R_n) . The set of relational schemes is called a database schema. Each relational schema is formed by a finite set of attributes names, $A_i = \{A_{i1}, \dots, A_{im_i}\}$, and each attribute has a set of its possible values called domain.*

We consider the set \mathbb{A} as the set of all attributes in DB, i.e., $\mathbb{A} = A_1 \cup \dots \cup A_n$. The n -relations (r_1, \dots, r_n) is called an extensional database or simply database.

Definition 19. *A view \mathcal{V} over a database DB is an expression of the form:*

DEFINE VIEW \mathcal{V} AS

SELECT \mathbb{B}

FROM T_1, T_2, \dots, T_M

WHERE \mathbb{C}

where \mathbb{B} is a subset of the set of attributes \mathbb{A} , T_i could be either a relational name (R_i 's) or other view name (V_i 's) on DB, called base relations, and \mathbb{C} is a constraint called selection condition. We will denote view \mathcal{V} as $\mathcal{V}(T_1, \dots, T_M)$.

Note that a view \mathcal{V} is a derived relation, i.e., it is not included in the database schema. Therefore, in order to keep view \mathcal{V} consistent with the data sources, it has to be maintained or re-evaluated from any relevant change produced at T_1, \dots, T_M .

Definition 20. *Let \mathcal{V}_1 and \mathcal{V}_2 be two views. We say that \mathcal{V}_2 is subsumed by \mathcal{V}_1 , denoted by $\mathcal{V}_2 \models \mathcal{V}_1$, if $\mathcal{V}_2 \subseteq \mathcal{V}_1$, where \subseteq refers to view containment. \mathcal{V}_1 is equivalent to \mathcal{V}_2 , $\mathcal{V}_1 \equiv \mathcal{V}_2$, if $\mathcal{V}_1 \models \mathcal{V}_2$ and $\mathcal{V}_2 \models \mathcal{V}_1$.*

Intuitively, view \mathcal{V}_1 subsumes view \mathcal{V}_2 if for any legal database instance \mathcal{V}_2 can be derived from \mathcal{V}_1 , i.e., \mathcal{V}_2 is contained by \mathcal{V}_1 . In general, a view \mathcal{V} can be decomposed into a set of equivalent views $\{\mathcal{V}_0, \mathcal{V}_1, \dots, \mathcal{V}_n\}$, such that $\mathcal{V} \equiv \mathcal{V}_0(\mathcal{V}_1, \dots, \mathcal{V}_n)$. Then, we can use either $\{\mathcal{V}_0, \mathcal{V}_1, \dots, \mathcal{V}_n\}$ or \mathcal{V} to answer \mathcal{V} . Note that \mathcal{V} can be decomposed recursively, i.e., each one of its \mathcal{V}_i , $1 \leq i \leq n$, can be decomposed in $(\mathcal{V}_{i0}, \mathcal{V}_{i1}, \dots, \mathcal{V}_{in_i})$, and so on.

We will say that a view decomposition is a view *evaluation plan* if a view \mathcal{V} is decomposed recursively until all its base relations are reached. Note that two or more different view evaluation plans may share one or more intermediate views.

3.4.2 Optimal View Materialization Problem

In this subsection we formulate and characterize the problem of selecting a set of materialized views as an optimization one. Informally, given a set of materialized views (queries) \mathcal{V} , defined over a set of base-relations or views \mathcal{R} , we have to decide what additional views \mathcal{V}^* should be materialized in order to optimize a criterion (maintenance

costs, response time, etc.), satisfying a set of materialization constraints (maintenance time, available storage, budget, etc.). Selecting one subset of additional views (evaluation plan) or another requires that the following issues be considered.

1. **Incremental Evaluation Tradeoff:** Selecting an evaluation plan with few views may spoil the performance, because materializing views will require more effort. On the other hand, evaluation plans with many views will require, when some data sources change (due to update, insert or delete) occurrences, more materialized view recalculations.
2. **Design Constraints:** There are some constraints that restrict the solutions, and some evaluation plans may not satisfy them. For instance, available storage, processing and view maintenance time, a limited budget, and any other resource constraints.
3. **Selection Criteria:** When several alternatives satisfy our design constraints, we need to define some criteria to select one among all possible. For example, time, cost, and storage measures of view materialization and maintenance could be the selected criterion.

The following definition is a general characterization of the optimization criterion and search space for the optimization problem formulation.

Definition 21. *Let \mathcal{V} be a view. We say that the set \mathbb{V} is the set of all equivalent*

evaluation plans (subset of materialized views) for \mathcal{V} , the function $f : \mathbb{V} \rightarrow \mathbb{R}$ is a real function that characterizes our selection criterion, and $\Psi(v), v \in \mathbb{V}$ is a constraint.

Therefore, the view selection problem corresponds to select a set of views (\mathcal{V}^*), i.e., an evaluation plan for \mathcal{V} , among all feasible alternatives (i.e., those $v \in \mathbb{V}$ that satisfy $\Psi(v)$), such that a criterion f is optimized. We will assume that the optimization is a minimization, and then the following optimization problem formulates the view selection problem.

$$\begin{aligned} \min_v \{f(v)\} \\ \text{s.t. } \Psi(v), v \in \mathbb{V} \end{aligned} \tag{3.1}$$

Note that [RSS96] uses a similar formulation to (3.1) but without constraints. Before we discuss how problem (3.1) can be solved effectively, we will concentrate on the more precisely characterization of set \mathbb{V} , function f , and constraint $\Psi(v)$. The following subsections address these characterizations.

3.4.3 Expression DAG

In this subsection we introduce *expression-DAGs* as a mechanism to represent equivalent view (query) evaluation plans, i.e., a characterization of the set \mathbb{V} . Expression-DAGs were originally introduced in [RSS96], we adopt the original definition of [RSS96], and extend the expression-DAG concept with some useful properties.

Definition 22. An expression DAG E_{DAG} is a directed acyclic graph (acyclic digraph), represented by the pair $E_{DAG} = (E, O)$, where $E = \{e_1, \dots, e_n\}$ is the set of equivalence nodes, and $O = \{O_1, \dots, O_m\}$ the set of operation nodes, with the following properties:

1. An equivalence node has edges to one or more operation nodes.
2. An operation node contains an operator, has edges to one or more equivalence nodes, and its parent is an equivalence node.

We denote by $|O_j|$ the *cardinality* of the operation node O_j , i.e., the total number of incoming and out-coming edges on O_j . In addition to that, we denote by $C(O_j)$ and $C(e_i)$ the children set (equivalence nodes and operation node respectively) of the operation node O_j and equivalence node e_i , respectively. Finally, we denote by $P(O_j)$ the parent (equivalence node) of O_j , and $L(E_{DAG})$ the set of all leaf nodes, i.e., those nodes without children.

Definition 23. Let $E_{DAG} = (E, O)$ be an expression DAG, with equivalence node $E = \{e_1, \dots, e_n\}$ and operation nodes $O = \{O_1, \dots, O_m\}$. We denote the size of E_{DAG} by $size(E_{DAG})$ and the average cardinality by \bar{E}_{DAG} , defined as follow:

$$size(E_{DAG}) = \sum_{O_i \in O} |O_i| \quad \text{and} \quad \bar{E}_{DAG} = \frac{1}{m} \sum_{O_i \in O} |O_i|$$

Expression-DAGs are used to compactly represent the space of equivalent view (query) *evaluation plans* in [RSS96], where equivalence nodes represent views, operation

nodes represent relational equivalence between parent and children, and the leaves of an expression -DAG correspond to the base relations. Furthermore, expression-DAGs are equivalent to AND-OR graphs (used in [Gup97, RSS96]) if the equivalence nodes are equivalent to the nodes in the AND-OR graph, and for each AND arc there exists an operation node and vice versa.

Definition 24. A path P_{st} of length q , in an expression-DAG $E_{DAG} = (E, O)$, is a sequence of equivalence and operations nodes, $P_{st} = (e_1 = s, O_{i_1}, e_2, O_{i_2}, \dots, O_{i_q}, e_{q+1} = t)$, where:

$$s \in P(O_{i_1}), t \in C(O_{i_q}), \text{ and } e_j \in C(O_{i_{j-1}}) \cap P(O_{i_j}), j = 2, \dots, q$$

nodes s and t are the origin and destination respectively, and we say that t is connected to s .

Note that a path is simply a sequence of equivalence and operation nodes. However, we would like to extend that concept to one that provides the notion of view evaluation plans. This extension is as follows.

Definition 25. Let E_{DAG} be an expression-DAG, we say that an expression-Path, denoted by $\pi = (E_\pi, O_\pi)$, is a rooted expression-DAG, where for each $e_i \in E_\pi$ there exists only one $O_j \in O_\pi$ selected.

It is easy to see that an expression-Path, from the root of E_{DAG} to the base relations,

represents a view evaluation plan, because each view is materialized using only one operation node.

An expression-DAG can be subdivided in sub expression-DAGs, which are a portion of an expression-DAG. We formalize this concept as follows.

Definition 26 (Sub-expression-DAG). *Let $E_{DAG} = (E, O)$ be an expression-DAG, we say that $E_{SDAG} = (E', O')$ is a sub-expression-DAG of E_{DAG} rooted in node e_i if E_{SDAG} is an expression-DAG with the following sets:*

$$E' = \{e_j \mid e_j \in E \wedge \exists P_{e_i e_j} \neq \emptyset\} \quad \text{and} \quad O' = \{O_j \mid O_j \in O \wedge \exists P_{e_i O_j} \neq \emptyset\}$$

The following example shows an expression-DAG to represent view \mathcal{V} . Note that we use a different notation than [RSS96], where the equivalence nodes are represented as a circle (views), and the operation nodes as an inverted black triangle.

Example 2. *Consider the relation schema (AB) , (CDE) , and (FG) , where A , B , C , D , F , G are the attribute names, and a view \mathcal{V} defined over this schema as follows:*

$$\mathcal{V} = \pi_{A,D,G}(\sigma_{(A=C) \wedge (B < C \vee B < D) \wedge (G < E) \wedge (F=f) \wedge (B=b)}(AB \times CDE \times FG))$$

Figure 3.1 presents an expression-DAG $E_{DAG} = (E, O)$ for view \mathcal{V} , where the set of equivalence nodes is $E = \{V, V_1, V_2, V_3, V_4, AB, CDE, FG\}$, and the set of operation nodes $O = \{O_1, O_2, O_3, O_4, O_5\}$, each one associated with a specific operation.

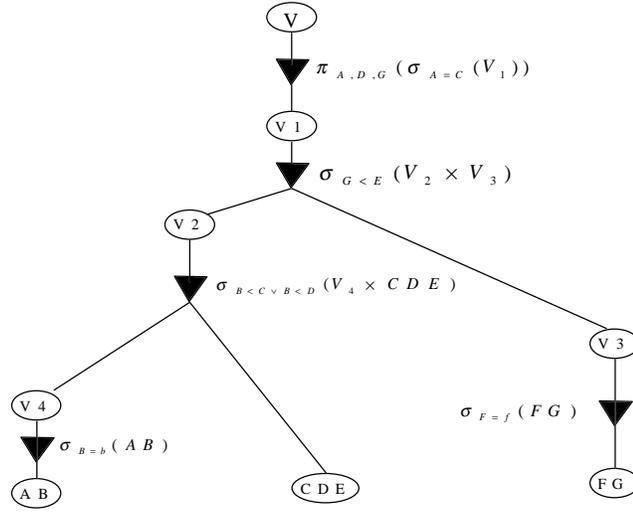


Figure 3.1: An Expression-DAG of \mathcal{V}

An expression-DAG represents equivalent view evaluation plans for a given view. However, there are some of those evaluation plans that are implicitly represented. Thus, from Figure 3.1, view V_1 can directly be evaluated from base relations AB, CDE, and FG, and in the same sense, view V_2 from AB and CDE. We formalize those additional and implicit ways to evaluate view V as follows.

Definition 27. Let $E_{DAG} = (E, O)$ be an expression-DAG for a view \mathcal{V} . Then, we say that $E_{DAG}^+ = (E, O^+)$ is the transitive closure of E_{DAG} if E_{DAG}^+ is an acyclic digraph and set O^+ is defined as follows:

$$\begin{aligned}
 O^+ &= \{O_j^+ \mid \forall e_i \neq e_j \in E, \exists P_{e_i e_j} \neq \emptyset \wedge \\
 &O_j^+ \in C(e_i) \wedge C(O_j^+) = \cup_{O_k \in P_{e_i e_j}} C(O_k)\}
 \end{aligned} \tag{3.2}$$

The transitive closure E_{DAG}^+ of an E_{DAG} has all possible equivalent evaluation plans for a given view \mathcal{V} , i.e., it characterizes the search space (\mathbb{V}) for the view selection problem. However, the main drawback in this concept is that $size(E_{DAG}^+)$ grows exponentially in the number of equivalence nodes (views) in E_{DAG} .

3.4.4 Objective Function

In this subsection we characterize the optimization criterion by means of a function f . To characterize f we adopt the same optimization function proposed in [Gup97]. We consider that a set of views (queries) \mathcal{V} need to be answered and that a set of views \mathcal{V}^* have been selected to be additionally materialized. Then, function f has the following items.

Requirement Cost: this cost is associated to answer views in \mathcal{V} , i.e., each time that a view $\mathcal{V}_i \in \mathcal{V}$ is required, the system will compute it from the additional materialized views \mathcal{V}^* (if there are some) or from the base relations. Therefore, the cost incurred by such operations represent the *requirement cost*. We use $C_A(\mathcal{V}_i, \mathcal{V}^*)$ to denote this cost.

Maintenance Cost: this cost is associated with the marginal maintenance of views in \mathcal{V}^* . Since views in \mathcal{V}^* are all materialized views, when a base relation changes (update, delete, or insert), views in \mathcal{V}^* have to be maintained. The cost of such maintenance is captured in the *maintenance cost*. We use $C_M(\mathcal{V}_i^*, \mathcal{V}^*)$ to denote

the maintenance cost of materialized view \mathcal{V}_i^* .

Finally, we consider that for each view $\mathcal{V}_i \in \mathcal{V}$ there exists a frequency β_i representing the number of times that view \mathcal{V}_i is required per unit time. In the same way, for each materialized view $\mathcal{V}_j^* \in \mathcal{V}^*$ there exists a frequency λ_j representing the number of times that view \mathcal{V}_j^* is maintained per unit time. Therefore, the objective function f is expressed as follows.

$$f = \sum_{i \in \mathcal{V}} \beta_i C_A(\mathcal{V}_i, \mathcal{V}^*) + \sum_{j \in \mathcal{V}^*} \lambda_j C_M(\mathcal{V}_j^*, \mathcal{V}^*) \quad (3.3)$$

Function f models the trade off between the number of materialized views and the total cost. Thus, when more additional views are materialized, the cost to answer views in \mathcal{V} is reduced, but the cost to maintain materialized views increases. The tradeoff associated between these costs forms the crux of the materialization view problem. The next subsection presents two additional, but equivalent, formulations for our problem.

3.4.5 Optimization Problem

In this section we re-formulate problem (3.1) in two equivalent formulations. One of them is based on selecting a subset of views \mathcal{V}^* among all candidate views to be materialized, the other, is based on recursive programming, taking advantage from the E_{DAG} representation. Before we present such formulations, we define the *resource*

constraint concept as follows.

Definition 28 (Resource Constraint). *Let \mathcal{V}^* be the set of selected materialized views, and T be the maximum resource available. Then, a resource constraint is a constraint saying that the resource used by views in \mathcal{V}^* must be lower or equal to T .*

In general, this type of constraint has been associated with storage (hard disk capacity), processing time, or budget [Gup97, HRU96, GM98]. Independently of which meaning the constraint has, we formulate the optimization problem in terms of that type of constraint.

Definition 29. *Let \mathcal{V} be a set of views, and \mathcal{V}^* be the set of materialized views. We say that the function $S : \mathcal{V}^* \rightarrow \mathbb{R}$ represents the resource used by views in \mathcal{V}^* .*

Proposition 12. *Let $\mathcal{V} = \{\mathcal{V}_1, \dots, \mathcal{V}_m\}$ be a set of views (queries) to be answered, $E_{DAG} = (E, O)$ be their expression-DAG, \mathcal{V}^* be a subset of (views) E , and $S(\mathcal{V}^*)$ be the resource utilization by \mathcal{V}^* . Then, the following optimization problem*

$$\begin{aligned}
 \min_{\mathcal{V}^*} \quad & \sum_{i \in \mathcal{V}} \beta_i C_A(\mathcal{V}_i, \mathcal{V}^*) + \sum_{j \in \mathcal{V}^*} \lambda_j C_M(\mathcal{V}_j^*, \mathcal{V}^*) \\
 \text{s.t.} \quad & S(\mathcal{V}^*) \leq T \\
 & \mathcal{V}^* \subseteq E
 \end{aligned} \tag{3.4}$$

is equivalent to (3.1), if $\Psi(v), v \in \mathbb{V}$ is equivalent to $S(\mathcal{V}^*) \leq T$ and $\mathcal{V}^* \subseteq E$.

Proof. The proof follows directly from the fact that an evaluation plan of \mathcal{V} (i.e., an element $v \in \mathbb{V}$) is a subset \mathcal{V}^* of E . Then, if $\Psi(v), v \in \mathbb{V}$ is equivalent to $S(\mathcal{V}^*) \leq T$ and

$\mathcal{V}^* \subseteq E$, both search spaces are equivalent, and the objective function are equivalent too. This completes the proof. \square

Optimization problem (3.4) is the same problem formulated in [Gup97, GHRU97, GM98], and it is clearly NP-hard problem, because (3.4) selects a subset of elements (views) \mathcal{V}^* from the set E on E_{DAG} , and the number of subsets is exponential in the number of elements in E . However, [Gup97, GHRU97, GM98] provide near-optimal greedy heuristics for some special E_{DAG} cases. These heuristics perform the following three steps: (1) select a subset $\mathcal{V}^* \subseteq E$, (2) check if it is feasible (i.e., if it satisfies the constraints), and (3) evaluate the objective function and compare with previous solutions.

In general, these heuristics have been applied to some particular cases (AND and OR graphs), and they do not guarantee quality in their solutions. We extend formulation (3.4) taking advantage of the E_{DAG} structure, i.e., the interrelation between equivalence and operation nodes, as follows.

Proposition 13. *Let $E_{DAG} = (E, O)$ be an expression-DAG, and E_{DAG}^+ be its transitive closure. Then, for every solution $\mathcal{V}^* \in E$ of (3.4), there exists an expression-Path $\pi = (E_\pi, O_\pi)$ in E_{DAG}^+ , such that $\mathcal{V}^* \equiv E_\pi$.*

Proof. Let \mathcal{V}^* be a solution of (3.4), i.e., it is an evaluation plan. Then, as E_{DAG}^+ has all possible of those plans, there exists an expression-Path with those selected views, i.e., $\pi = (\mathcal{V}^*, O_\pi)$. \square

Therefore, selecting an optimal subset of views \mathcal{V}^* using (3.4) is equivalent to find the cheapest expression-Paths in E_{DAG}^+ that satisfies the resource constraint.

Definition 30. Let $\mathcal{V} = \{\mathcal{V}_1, \dots, \mathcal{V}_m\}$ be a set of views (queries) to be answered, and $E_{DAG} = (E, O)$ be the expression-DAG of \mathcal{V} , and $E_{DAG}^+ = (E, O^+)$ its transitive closure. Then, we say that \mathcal{V}_0 is a dummy view of \mathcal{V} , if its expression-DAG $E'_{DAG} = (E', O')$ and transitive closure $E'^+_{DAG} = (E', O'^+)$ are defined by $E' = E \cup \{\mathcal{V}_0\}$, $O' = O \cup \{O_0\}$, $O'^+ = O^+ \cup \{O_0\}$, $C(\mathcal{V}_0) = \{O_0\}$, and $C(O_0) = \{\mathcal{V}_1, \dots, \mathcal{V}_m\}$.

A dummy view is a concept to create an expression-DAG with a unique root (\mathcal{V}_0). All the following results are based on this concept.

Definition 31. Let $E_{DAG} = (E, O)$ be an expression-DAG rooted at e_0 , $\{e_1, \dots, e_{n_j}\}$ the set of all children of O_j , where $O_j \in C(e_0)$, and $E_{DAG}^k = (E_k, O_k)$ be a sub-expression-DAG rooted at e_k . Then, we say that $f : E \rightarrow \mathbb{R}$ is an additive function if there exists $f_i : E_i \rightarrow \mathbb{R}$, $1 \leq i \leq n_j$, a nondecreasing function $\mathcal{F}_j : \mathbb{R}^{n_j} \rightarrow \mathbb{R}$, for all $j \in C(e_0)$, and $\nu : O_j \rightarrow \mathbb{R}$, such that $f(e_0) = \nu(O_j) + \mathcal{F}_j(f_1(e_1), \dots, f_{n_j}(e_{n_j}))$.

Additive functions allow us to write functions recursively in terms of an expression-DAG structure. In particular, we are interested in rewriting optimization problem of (3.4).

Proposition 14. Let \mathcal{V}_0 be a view with closure expression-DAG $E_{DAG}^+ = (E, O^+)$ rooted at e_0 , $L(E_{DAG}^+)$ be the set of base relations of E_{DAG}^+ , $\{e_1, \dots, e_{n_j}\}$ the set of all children of O_j , where $O_j \in C(e_0)$, $E_{DAG}^{k+} = (E_k^+, O_k^+)$ be a sub-expression-DAG

of E_{DAG}^+ rooted at e_k , and $\pi = (E_\pi, O_\pi)$ be an expression-Path on E_{DAG}^+ , such that $O_j \in O_\pi$. Then,

1. Let T be the available resource, $S(E_\pi)$ be a resource utilization function, and $\hat{S}(e_0) = T - S(E_\pi)$ be the available resource at e_0 . Then, if $S(E_\pi)$ is an additive resource utilization function, with $r(O_j)$ the resource utilization of operation node O_j , constraint $S(E_\pi) \leq T$ is equivalent to

$$\hat{S}(e_0) = \min_k \left\{ \hat{S}(e_k) \right\} - r(O_j) \geq 0$$

$$\hat{S}(e_k) = T, e_k \in L(E_{DAG}^+)$$

2. Let $\eta(e_0, \pi) = \beta_0 C_A(\mathcal{V}_0, E_\pi) + \sum_{e_i \in E_\pi} \lambda_i C_M(e_i, E_\pi)$ the cost of expression-Path π . Then, if function $C_A(\mathcal{V}_0, E_\pi)$ and $C_M(e_i, E_\pi)$ are additive with $C_A(\mathcal{V}_0, e_0) = \mu(O_j)$ and $C_M(e_0, E_\pi) = \psi(O_j)$,

$$\eta(e_0, \pi) = \mu(O_j) + \psi(O_j) + \sum_{k \in C(O_j)} \eta(e_k, \pi_k)$$

Proof. (1) If operational node $O_j \in \pi$, and as $S(E_\pi)$ is an additive function, the resource utilization at e_0 is the maximum resource utilization at $\{e_1, \dots, e_{n_j}\}$ plus the

resource utilization at node O_j (i.e., $r(O_j)$). Therefore,

$$\begin{aligned}
S(E_\pi) \leq T &\Leftrightarrow r(O_j) + \max_k \{S(E_{\pi_k})\} \leq T \\
&\Leftrightarrow 0 \leq -r(O_j) + \min_k \{T - S(E_{\pi_k})\} \\
&\Leftrightarrow \min_k \{\hat{S}(e_k)\} - r(O_j) \geq 0
\end{aligned}$$

This completes this part of the proof.

(2) From the $\eta(e_0, \pi)$ definition, and since C_A and C_M are additive functions,

$$\begin{aligned}
\eta(e_0, \pi) &= \beta_0 C_A(\mathcal{V}_0, E_\pi) + \sum_{e_i \in E_\pi} \lambda_i C_M(e_i, E_\pi) \\
&= \beta_0 C_A(\mathcal{V}_0, e_0) + \lambda_0 C_M(e_0, E_\pi) + \sum_{k \in C(O_j)} \{\eta(e_k, E_{\pi_k}) + \beta_0 C_A(\mathcal{V}_0, E_{\pi_k})\} \\
&= \mu(O_j) + \psi(O_j) + \sum_{k \in C(O_j)} \eta(e_k, \pi_k)
\end{aligned}$$

This completes the proof. □

Theorem 7. *Let $\mathcal{V} = \{\mathcal{V}_1, \dots, \mathcal{V}_m\}$ be a set of views (queries) to be answered, \mathcal{V}_0 be its dummy view with expression-DAG transitive closure $E_{DAG}^+ = (E, O^+)$, and $\eta(e_i)$ be the minimum cost at equivalence node e_i , i.e., $\eta(e_i) = \min_{\pi_k} \{\eta(e_i, \pi_k)\}$. Then, the*

following optimization problem is equivalent to (3.4)

$$\begin{aligned} \eta(e_i) = \min_j & \left\{ \mu(O_j) + \psi(O_j) + \sum_{k \in C(O_j)} \eta(e_k) \mid O_j \in C(e_i) \right\} \\ \text{s.t. } & e_i \in E^+ - L(E_{DAG}^+) \end{aligned} \quad (3.5)$$

$$\hat{S}(e_i) \geq 0$$

$$\eta(e_u) = 0, C_A(\mathcal{V}_0, e_u) = 0, S(e_u) = T, u \in L(E_{DAG}^+)$$

If functions C_M and C_A are additive.

Proof. First, problem (3.1) can be expressed as follows, $\min_{\pi} \{\eta(\mathcal{V}_0, \pi) \mid S(E_{\pi}) \leq T\}$,

i.e., finding the cheapest evaluation plan for \mathcal{V}_0 , subject to the resource constraint.

From Proposition 13 and using the fact that C_A and C_M are additive functions, then

we can re-write (3.1) as follows:

$$\min_j \left\{ \mu(O_j) + \psi(O_j) + \sum_{e_k \in C(O_j)} \min_{\pi_k} \{\eta(\mathcal{V}_0, \pi_k) \mid \hat{S}(e_k) \geq 0\} \mid \hat{S}(\mathcal{V}_0) \geq 0 \right\}$$

Now, renaming $\min_j \left\{ \mu(O_j) + \psi(O_j) + \sum_{k \in C(O_j)} \eta(e_k) \mid O_j \in C(e_i) \right\}$ by $\eta(e_i)$, we get

(3.5). This complete the proof. \square

Problem (3.5) finds the cheapest evaluation plan for \mathcal{V}_0 , when there exist resource constraints. In the next section we present two algorithms to solve (3.5).

3.5 Solution and Algorithms

In this section we propose two algorithms to solve (3.5). We first consider that the solution is a complete expression-Path in an expression-DAG, and present a linear time algorithm (in terms of the expression-DAG size). Then, we consider the case where the solution is a subset of views, and it is not necessarily a complete expression-Path. We present a local search algorithm to solve (3.5), where we use a special case of the unconstrained shortest path algorithm as global search.

3.5.1 Shortest Path Algorithm in an Expression-DAG

The algorithm is a general one in the sense that it finds the shortest path in an expression-DAG. However, to guarantee that the algorithm solves (3.5), we need some additional conditions. The algorithm uses as input a general expression-DAG E_{DAG} , the available resource T , and a vector \vec{y} with value $y_i = 1$ if view e_i can be materialized and $y_i = 0$ otherwise. The output is a set $\{P_i \mid i \in E\}$ indicating, for each equivalence node (view) i , which operation node has been selected. Therefore, selected views are found starting from \mathcal{V}_0 and recursively selecting views from P_i 's. The algorithm assumes that E_{DAG} is ordered in inverse topological order, i.e., its nodes are enumerated, such that the following condition is satisfied:

$$(P(O_k) = \{i\}) \wedge (j \in C(O_k)) \Rightarrow j < i. \quad (3.6)$$

Since E_{DAG} is ordered in inverse topological order, the root \mathcal{V}_0 has the highest order. Finally, we consider that the set $BS(e_i)$ is the set of all operation nodes having node e_i as parent, i.e., $BS(e_i) = \{O_j \mid i \in P(O_j)\}$. The algorithm is as follows.

ViewSelection($E_{DAG} = (E, O), T, \vec{y}$)

for each $i \in E$ **do**

$$P_i = 0$$

if ($i \in L(E_{DAG})$) **then**

$$\eta(i) = 0, \hat{S}(i) = T$$

else $\eta(i) = \infty$

for each $j \in O$ **do** $k_j = 0$

for $i = 1$ **to** $|E| - 1$ **do**

for each $O_j = (\{z\}, C(O_j)) \in BS(i)$ **do**

$$k_j = k_j + 1$$

if $k_j = |C(O_j)|$ **then**

$$f = G(\{\eta(e_k), \vec{y} \mid e_k \in C(O_j)\})$$

$$\hat{S}^* = \min_k \{\hat{S}(e_k) \mid e_k \in C(O_j)\} - r(O_j)$$

if ($\eta(z) > f \wedge \hat{S}^* \geq 0$) **then**

$$\eta(z) = f$$

$$P_z = O_j$$

$$\hat{S}(z) = \hat{S}^*$$

Function $G(\{\eta(e_k), \vec{y} \mid e_k \in C(O_j)\})$ evaluates the cost from all children of operation node O_j . Note that vector \vec{y} modifies this evaluation, allowing some equivalence nodes not to be considered. In general, when $\vec{y} = \vec{1}$, all nodes are considered to be materialized. The next theorem provides the correctness of our algorithm.

Theorem 8. *Let \mathcal{V}_0 be a view, and E_{DAG}^+ be the transitive closure of \mathcal{V}_0 expression-DAG. Then, algorithm $ViewSelection(E_{DAG}^+, T, \vec{1})$ produces an optimal solution \mathcal{V}^* to the optimization problem (3.5).*

Proof. Since E_{DAG}^+ has all possible \mathcal{V}_0 evaluation plans, and $ViewSelection(E_{DAG}^+, T, \vec{1})$ finds the *shortest* (cheapest) path that satisfies the resource constraint, then, all views belonging to this path are the solution of (3.5), because they satisfy the constraint and produce the cheapest cost. This completes the proof. \square

To analyze the complexity of the $ViewSelection()$ algorithm, we first note that each node and each edge is selected at most once, and for each operation node O_j we evaluate f using all nodes belonging to its children, i.e., $C(O_j)$, we consider this evaluation has a complexity $O(f)$. Therefore, the overall complexity is $O(O(f) \times size(E_{DAG}))$.

Although $ViewSelection(E_{DAG}^+, T, \vec{1})$ produces the optimal solution of (3.5), the complexity of E_{DAG}^+ is still exponential in the number of views. However, if the solution of (3.5) is a complete expression-Path in E_{DAG} , then, $ViewSelection()$ algorithm provides the optimal solution. The next subsection presents a local search algorithm to solve (3.5).

3.5.2 Local Search Algorithm for Expression-DAG

Algorithm $ViewSelection(E_{DAG}^+, T, \vec{1})$ requires an exponential time to solve problem (3.5), because $size(E_{DAG}^+)$ is exponential in terms of the number of views. We propose a local search algorithm [LH96, Glo89, Jea91] to overcome this drawback. The local search algorithm is structured as follows: *a number of local searches are performed, where for each one, the algorithm checks if the local optimum is better than the current objective function value.* This procedure is repeated until there is no acceptable neighborhood possible. Before describing the algorithm, we define the concept of materialization vector.

Definition 32. *Let \mathcal{V}_0 be a view, and $E_{DAG} = (E, O)$ be the expression-DAG of \mathcal{V}_0 , where $E = (e_1, \dots, e_n)$ and $O = (O_1, \dots, O_m)$. We say that a vector $\vec{y} = (y_1, \dots, y_n)$ is a materialization vector if*

$$y_i = \begin{cases} 1 & \text{if view } e_i \in E \text{ can be materialized,} \\ 0 & \text{otherwise} \end{cases}$$

Materialization vectors represent a mechanism to define neighborhoods, i.e., initial solutions for each local search. The local search framework is described as follows.

Step 0. Assign 0 to k , $\vec{y}_k = (1, \dots, 1)$, and $f^* = \infty$.

Step 1. Perform a local search, solving the following problem

$$\begin{aligned} \min_{\vec{x}} f(\vec{x}/\vec{y}_k) \\ s.t. \Psi(\vec{x}/\vec{y}_k) \end{aligned}$$

Step 2. if $f(\vec{x}) < f^*$, then $f^* = f(\vec{x})$, and $\vec{y}^* = \vec{x}$.

Step 3. Increase k by 1, select a new \vec{y}_k , and go to step 2. If there is no \vec{y}_k additional, go to step 4.

Step 4. Report objective function f^* and solution \vec{y}^* .

We have to explain how procedures local search (step 1) and selection of a new \vec{y}_k are performed (step 4). First, we explain local search.

Local-Search We iteratively solve $ViewSelection(E_{DAG}, \infty, \vec{y})$ and put the solution in vector \vec{x} . Then, we evaluate if the constraint is satisfied. If it is not, then we select a *victim* and eliminate it. We solve $ViewSelection(E_{DAG}, \infty, \vec{y})$ again. The procedure is as follows.

Local-Search($E_{DAG} = (E, O), T, \vec{y}$)

$$\vec{x} = ViewSelection(E_{DAG}, \infty, \vec{y})$$

repeat until ($S(\mathcal{V}_0) \leq T$)

$$EliminateMaterialized(\vec{y}, \vec{x})$$

$$\vec{x} = ViewSelection(E_{DAG}, \infty, \vec{y})$$

Procedure *EliminateMaterialized*(\vec{y}, \vec{x}) selects a *victim*, i.e., a view k with the highest resource occupancy index. When the victim has been selected, we assign $y_k = 0$.

Neighborhood Selection Vector \vec{y} is used to defined a neighborhood, because selecting some $y_i = 0$ we do not allow some solutions. We propose an aggressive strategy to move from neighborhood to neighborhood. This strategy eliminates a certain number N of views with the lowest resource occupancy index.

Although we do not offer a complexity analysis for this algorithm, the global complexity will depend of the number of internal iteration in *ViewSelection*(E_{DAG}, ∞, \vec{y}) ($|E|$ in the worse case), and the number of neighborhoods visited ($2^{|E|}$ in the worse case). However, there are many of those alternatives that are subsumed by others, and therefore, they do not need to be visited. A more exhaustive work is needed in this area to create an efficient mechanism to select neighborhoods.

3.6 Implementation and Experiments

This section presents the main results of our *ViewSelection*(E_{DAG}, T, \vec{y}) algorithm. Table 3.1 summarizes the results for 25 experiments, where *Size* is the expression-DAG size, \bar{E} is the average cardinality, $\sigma_{\bar{E}}$ is its standard deviation, and Time is the

running time measured in milliseconds. The algorithm was implemented using visual C++ 4.0, and it was run on a 120 Mhz PC compatible.

In general, expression-DAGs with a lower number of equivalence and operation nodes than those shown in Table 3.1, reported a time less than 10 milliseconds.

Table 3.1: Empirical Results

Size	E	σ_E	Time	Size	E	σ_E	Time
15818	3.7	3.0	170	6168	4.3	5.2	110
10089	5.2	5.7	110	9504	4.3	4.2	110
7716	2.6	1.9	50	16950	2.5	1.2	110
23759	2.9	2.1	280	4475	5.1	4.4	60
20128	2.4	0.8	160	39777	8.4	13.6	770
39921	4.1	3.5	550	19856	3.2	1.8	220
21881	2.8	1.3	280	27548	4.3	3.4	440
24890	2.3	0.8	270	25496	2.7	1.1	280
18226	3.1	1.6	170	30355	3.2	2.1	440
21188	2.6	1.4	270	33875	2.6	1.2	390
37047	2.5	1.2	490	39163	5.5	4.8	820
35241	2.4	1.1	440	27549	8.5	7.7	660

The experiments were generated randomly, where the minimum and maximum number of equivalence nodes were approximately 2,000 and 15,000 respectively, and operation nodes were 1,400 and 15,000 respectively.

Note that in this case (when all possible views are candidates to be materialized), function $G(\{\eta(e_k), \vec{y} \mid e_k \in C(O_j)\})$ corresponds to a simple summation of all $\eta(e_k)$, for all $e_k \in C(O_j)$, which reduces the complexity of our algorithm.

Figure 3.2 shows that the running time against the expression-DAG size. This time presents a positive linear behavior when the expression-DAG size increases. The

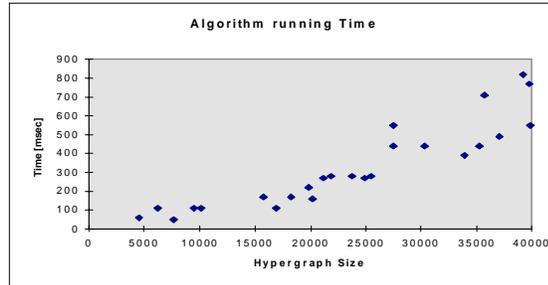


Figure 3.2: Experimental Run Time

variance produced in this relation is due to different expression-DAG structures, i.e., expression-DAG depth and average density (number of children at each node).

The main conclusion of these experiments, is that for a given view expression-DAG, the time to find the shortest (cheapest) path between the root and all base relations is a linear function of the expression-DAG size. Therefore, if the transitive closure E_{DAG}^+ is available, the algorithm is a good alternative, and when E_{DAG}^+ is not available, we can apply our local search algorithm in a reasonable searching time.