Using Association Rules to Add or Eliminate Query Constraints Automatically

Agathoniki Trigoni Ken Moody
University of Cambridge
Computer Laboratory
{at263,km10}@cl.cam.ac.uk

Abstract

Much interesting work has been done on the use of semantic associations for optimizing query execution. Our objective is to study the use of association rules to add or eliminate constraints in the `where` *clause of a* `select` *query. In particular, we take advantage of the following heuristics presented by Siegel et al. [6]: i) if a selection on attribute A is implied by another selection condition on attribute B and A is not an index attribute, then the selection on A can be removed from the query; ii) if a relation R in the query has a restricted attribute A and an unrestricted cluster index attribute B, then look for a rule where the restriction on A implies a restriction on B. The contribution of our work is twofold. First, we present detailed algorithms that apply these heuristics. Hence, our ideas are easy to implement. Second, we discuss conditions under which it is worth applying these optimization techniques, and we show the extent to which they speed up query execution.*

1. Introduction

Rules that correlate values of data attributes in large databases have been investigated for two main purposes. The first is to aid management and decision-making in large companies with great amounts of data. The second relates to query optimization, by exploiting the semantic knowledge expressed in rules [2, 3, 6, 9, 10]. Siegel et al. [6] have proposed several heuristics to guide rule generation and so help to transform queries to more efficient forms. In this paper we investigate algorithms for applying two of these heuristics to optimize range queries. We express queries using OQL (Object Query Language); however, our algorithms and optimization techniques are also applicable in any relational or object-relational database system. The form of queries under consideration is:

select $\mathtt{e(x)}$
from Extent_X as x
where $\mathtt{p_1(x)}$ and ... and $\mathtt{p_n(x)}$

The first heuristic (H1) aims to eliminate as many predicates (or constraints) $\mathtt{p_i(x)}$ as possible, when they are implied by other existing predicates. The second heuristic (H2) finds implications from one or more existing predicates to a new predicate on a cluster index attribute. If the latter is added to the `where` clause of the query and is tested first, then the existing predicates will be tested on a smaller number of objects. In order to take advantage of either of these heuristics, we use association rules that are relevant to `Extent_X`.

In our context, association rules express semantic correlations between the values of the attributes of all objects belonging to a certain extent. They consist of two parts, the antecedent, which has one or more constraints, and the consequent, which has just one constraint: $\mathtt{constr_1} \wedge \ldots \wedge \mathtt{constr_n} \Rightarrow \mathtt{constr_0}$. The constraints $\mathtt{constr_i}$ are of the form $\mathtt{class\_name.attribute_i}\ \mathtt{comp\_op_i}\ \mathtt{value_i}$, where for all $0 <= \mathtt{i} <= \mathtt{n}$, $\mathtt{comp\_op_i} \in \{<, >, <=, >=, =, \neq\}$. Association rules are typically ranked by two measures of interest: confidence and support. Confidence expresses the fraction of all objects satisfying the antecedent that also satisfy the consequent. Support expresses the percentage of objects of the extent satisfying both the antecedent and the consequent; hence it indicates how often the rule occurs in the extent of a certain class. A number of algorithms have been proposed for mining association rules over categorical or quantitative data [1, 7, 4, 5, 8]. In the remainder of the paper, we assume that the process of mining rules has been completed and that we already have a warehouse with a set of rules for each extent in the database. Further, in this warehouse we store the exceptions to each rule, i.e. those objects that satisfy its antecedent but not its consequent. We also assume that the rules and their exceptions are correctly maintained when the database is updated.

Automatic use of these rules by the OQL optimizer raises many issues. The first concerns mapping the predicates in the `where` clause of a `select` query to constraints that could potentially be found in the antecedent (or the consequent) of association rules. This is a step towards identifying the templates of association rules that could help in our optimizations; this task is slightly different for heuristics H1 and H2, as discussed in section 2. In section 3, we present a common graph algorithm that goes through all existing association rules and discovers the ones that fit the templates along with their exceptions. In section 4, another graph algorithm combines these rules to discover the possible optimizations, i.e. the sets of constraints that could be omitted from (H1) or added to (H2) the query. In section 5 we present the actual transformations of OQL queries for each of the two optimization techniques. Finally, in section 6 we investigate the conditions under which it is worth applying the optimizations and show the extent to which they are expected to speed up query execution.

2. Converting query predicates to rule constraints

The first step towards applying either of the heuristics is to map the predicates found in the `where` clause of a `select` query to constraints found in the existing association rules. Consider the following OQL query:

$$\begin{array}{l}\texttt{select x from Employees as x}\\ \texttt{where x.salary} > 35,000 \texttt{ and x.salary} < 55,000 \texttt{ and}\\ \texttt{x.position} = \text{“associate”} \texttt{ and}\\ \texttt{x.year\_of\_birth} <= 1975 \texttt{ and x.year\_of\_birth} >= 1955\\ \texttt{and x.year\_of\_employment} <= 1990 \texttt{ and}\\ \texttt{x.year\_of\_employment} > 1970\end{array} \quad (1)$$

We first need to gather all predicates that refer to the same attribute into one predicate, and form constraints of the form `attribute_name in range`. The resulting constraints in our example are given below:

`salary in` $\{(35,000, 55,000)\}$	C_A
`position in` $\{$“associate”$\}$	C_B
`year_of_birth in` $\{[1955, 1975]\}$	C_C
`year_of_employment in` $\{(1970, 1990]\}$	C_D

Based on these query constraints, the next step is to search for *similar* constraints in the association rules. Suppose that the warehouse contains the association rules illustrated in figure 1, which are relevant to the class `Employee`. The constraints in the antecedent and the consequent of the rules are labeled with short names, e.g. $C_{k.R_{kj}}$ is a constraint limiting the values of attribute k to the range R_{kj}. Here a *range* is a specified set of values of some attribute. The set of ranges associated with a particular attribute is partially ordered under inclusion in the usual way.

To distinguish constraints that are derived from the `where` clause of our query from those in the antecedent or consequent of the association rules, we refer to the former as *query constraints*, and to the latter as *rule constraints*. For each heuristic we look for rule constraints corresponding to the query constraints.

2.1. Heuristic H1

We first discuss rule constraints relevant to the constraint elimination heuristic. For each constraint C on `attribute_name` with range R, identify the following rule constraints:

Category 1 - relaxed constraints First, identify the constraints whose ranges are the next more general than (or the same as) the range of the original constraint C. If there exists such a constraint C_0 with a range R_0, then for all other constraints C_i $i \neq 0$ on `attribute_name` with ranges R_i, if $R \subseteq R_i$ then $\mathtt{not}(R_i \subseteq R_0)$. That is, R_0 is one of the least supersets of R among all ranges of constraints referring to the same attribute.

Category 2 - relaxed combinations of constraints Second, identify minimal combinations of constraints which have the property that the union of their ranges is a superset of the range of the original constraint C. The constraints in any combination must satisfy the following conditions: firstly, none of the constraints can be the same as (or with a wider range than the range of) any constraint identified previously in category 1; secondly, no combination must include all the constraints of a smaller combination; thirdly, if either of two constraints on the same attribute may take part in a combination, and the range of the one is a subset of the range of the other, then the one with the smaller range is selected.

Category 3 - tightened constraints Third, identify the constraints whose ranges are the next more specific than (or the same as) the range of the original constraint. If there exists such a constraint C_0 with a range R_0, it means that for all other constraints C_i $i \neq 0$ on `attribute_name` with ranges R_i, if $R \supseteq R_i$ then $\mathtt{not}(R_i \supseteq R_0)$. That is, R_0 is one of the greatest subsets of R among all ranges of constraints referring to the same attribute.

Category 4 - tightened combinations of constraints Fourth, identify maximal combinations of constraints which have the property that the intersection of their ranges is a subset of the range of the original constraint C. The constraints in any combination must satisfy the following conditions: firstly, none of the constraints can be the same as (or with a range contained in the range of) any constraint identified previously in category 3: secondly, no combination must include all the constraints of a smaller combination: thirdly, if either of two constraints on the same attribute may take part in a combination, and the range of the one is a superset of the range of the other, then the one with the wider range is selected.

$$
\begin{array}{rcl}
\texttt{year\_of\_birth in}\ \{[1955,1975]\}(\texttt{C}_{1.\texttt{R}_{11}}) & \Rightarrow & \texttt{salary in}\ \{[35,000,50,000]\}(\texttt{C}_{2.\texttt{R}_{21}}) \\
\texttt{position in}\ \{\text{“}\texttt{associate}\text{”}\}(\texttt{C}_{4.\texttt{R}_{41}}) & \Rightarrow & \texttt{salary in}\ \{[30,000,60,000]\}(\texttt{C}_{2.\texttt{R}_{22}}) \\
\texttt{position in}\ \{\text{“}\texttt{manager}\text{”}\}(\texttt{C}_{4.\texttt{R}_{42}}) & \Rightarrow & \texttt{year\_of\_birth in}\ \{[1955,1980]\}(\texttt{C}_{1.\texttt{R}_{12}}) \\
\texttt{year\_of\_birth in}\ \{[1950,1970]\}(\texttt{C}_{1.\texttt{R}_{13}}) & \Rightarrow & \texttt{position in}\ \{\text{“}\texttt{associate}\text{”}\}(\texttt{C}_{4.\texttt{R}_{41}}) \\
\texttt{year\_of\_employment in}\ \{[1975,1990]\}(\texttt{C}_{5.\texttt{R}_{53}}) & \Rightarrow & \texttt{position in}\ \{\text{“}\texttt{associate}\text{”}\}(\texttt{C}_{4.\texttt{R}_{41}}) \\
\texttt{year\_of\_employment in}\ \{[1965,1985]\}(\texttt{C}_{5.\texttt{R}_{52}}) & \Rightarrow & \texttt{efficiency in}\ \{[3,6]\}(\texttt{C}_{3.\texttt{R}_{31}}) \\
\left.\begin{array}{r}\texttt{salary in}\ \{[30,000,65,000]\}(\texttt{C}_{2.\texttt{R}_{23}}) \\ \texttt{position in}\ \{\text{“}\texttt{manager}\text{”}\}(\texttt{C}_{4.\texttt{R}_{42}})\end{array}\right\} & \Rightarrow & \texttt{year\_of\_employment in}\ \{[1970,1985]\}(\texttt{C}_{5.\texttt{R}_{51}}) \\
\left.\begin{array}{r}\texttt{salary in}\ \{[30,000,60,000]\}(\texttt{C}_{2.\texttt{R}_{22}}) \\ \texttt{efficiency in}\{[3,6]\}(\texttt{C}_{3.\texttt{R}_{31}})\end{array}\right\} & \Rightarrow & \texttt{year\_of\_birth in}\ \{[1955,1975]\}(\texttt{C}_{1.\texttt{R}_{11}}) \\
\left.\begin{array}{r}\texttt{salary in}\ \{[30,000,60,000]\}(\texttt{C}_{2.\texttt{R}_{22}}) \\ \texttt{efficiency in}\{[3,6]\}(\texttt{C}_{3.\texttt{R}_{31}})\end{array}\right\} & \Rightarrow & \texttt{position in}\ \{\text{“}\texttt{associate}\text{”}\}(\texttt{C}_{4.\texttt{R}_{41}}) \\
\left.\begin{array}{r}\texttt{year\_of\_birth in}\ \{[1955,1980]\}(\texttt{C}_{1.\texttt{R}_{12}}) \\ \texttt{salary in}\ \{[35,000,50,000]\}(\texttt{C}_{2.\texttt{R}_{21}})\end{array}\right\} & \Rightarrow & \texttt{efficiency in}\{[3,6]\}(\texttt{C}_{3.\texttt{R}_{31}})
\end{array}
$$

Figure 1. Association rules relevant to the class Employee

In the example of query 1, the query constraints $\texttt{C}_\texttt{A}$, $\texttt{C}_\texttt{B}$, $\texttt{C}_\texttt{C}$ and $\texttt{C}_\texttt{D}$ correspond to the following rule constraints:

	Relaxed Constraints
$\texttt{C}_\texttt{A}$	$\texttt{salary in}\ \{[30,000,60,000]\}(\texttt{C}_{2.\texttt{R}_{22}})$
$\texttt{C}_\texttt{B}$	$\texttt{position in}\ \{\text{“}\texttt{associate}\text{”}\}(\texttt{C}_{4.\texttt{R}_{41}})$
$\texttt{C}_\texttt{C}$	$\texttt{year\_of\_birth in}\ \{[1955,1975]\}(\texttt{C}_{1.\texttt{R}_{11}})$
	Relaxed Combinations of Constraints
$\texttt{C}_\texttt{D}$	$\{\texttt{year\_of\_employment in}\ \{[1970,1985]\}(\texttt{C}_{5.\texttt{R}_{51}})$
	$\texttt{year\_of\_employment in}\ \{[1975,1990]\}(\texttt{C}_{5.\texttt{R}_{53}})\}$
	Tightened Constraints
$\texttt{C}_\texttt{A}$	$\texttt{salary in}\ \{[35,000,50,000]\}(\texttt{C}_{2.\texttt{R}_{21}})$
$\texttt{C}_\texttt{B}$	$\texttt{position in}\ \{\text{“}\texttt{associate}\text{”}\}(\texttt{C}_{4.\texttt{R}_{41}})$
$\texttt{C}_\texttt{C}$	$\texttt{year\_of\_birth in}\ \{[1955,1975]\}(\texttt{C}_{1.\texttt{R}_{11}})$
$\texttt{C}_\texttt{D}$	$\texttt{year\_of\_employment in}\ \{[1970,1985]\}(\texttt{C}_{5.\texttt{R}_{51}})$
	$\texttt{year\_of\_employment in}\ \{[1975,1990]\}(\texttt{C}_{5.\texttt{R}_{53}})$
	Tightened Combinations of Constraints
$\texttt{C}_\texttt{C}$	$\{\texttt{year\_of\_birth in}\ \{[1955,1980]\}(\texttt{C}_{1.\texttt{R}_{12}})$
	$\texttt{year\_of\_birth in}\ \{[1950,1970]\}(\texttt{C}_{1.\texttt{R}_{13}})\}$

For each tightened combination of constraints we generate a new constraint whose range is the intersection of the ranges of the constraints in the combination. We add this new constraint to the category of tightened constraints and create a new temporary rule with the initial constraints in the antecedent and the new tightened constraint in the consequent. For instance, based on the tightened combination $\{\texttt{year\_of\_birth in}\ \{[1955,1980]\}(\texttt{C}_{1.\texttt{R}_{12}})$, $\texttt{year\_of\_birth in}\ \{[1950,1970]\}(\texttt{C}_{1.\texttt{R}_{13}})\}$, we create the constraint $\{\texttt{year\_of\_birth in}\ \{[1955,1970]\}(\texttt{C}_{1.\texttt{R}_{1.23}})$. Repeating this procedure for all tightened combinations of constraints, we generate a new set of tightened constraints.

The rationale behind finding the constraints in categories 1 to 3 is the following. By definition, the query constraints imply their corresponding relaxed constraints. Likewise, the tightened constraints imply the initial query constraints. Hence if we can find association rules that relate constraints in the first two categories to those in the third, it is equivalent to finding rules that correlate the initial query constraints with each other.

2.2. Heuristic H2

For the purposes of applying the constraint introduction heuristic, we identify the following rule constraints:

Category 1 - relaxed constraints These are identical to the relaxed constraints described in section 2.1.

Category 2 - relaxed combination of constraints This category is the same as its counterpart in section 2.1.

Category 3 - index constraints If the extent has a cluster index on attribute $\texttt{attr}_\texttt{i}$, then this category includes all the constraints on this attribute that occur as consequents in the existing rules. Assume that the cluster index of the extent `Employees` is the attribute `efficiency`. Based on the rules of figure 1, the only index constraints is the constraint $\texttt{efficiency in}\{[3,6]\}$.

Rule constraints imply relaxed (combinations of) constraints, by definition. If we succeed in finding paths from relaxed (combinations of) constraints to an index constraint, then we can add the latter to the `where` clause of the query.

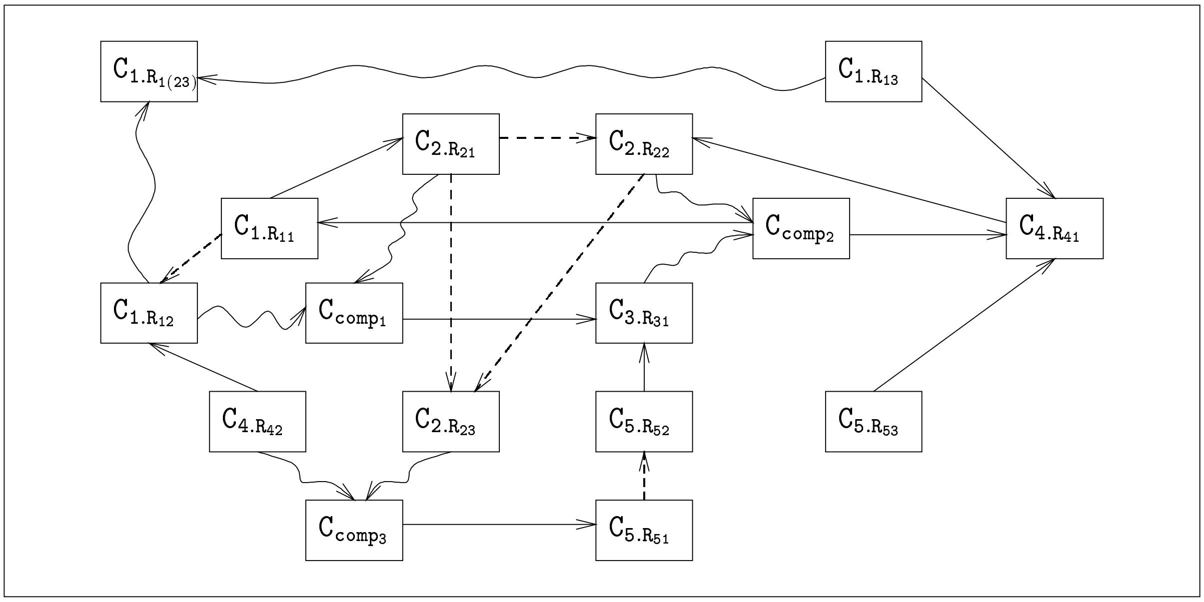

Figure 2. Graph of Constraints

This allows us to reduce the disc access overhead without altering the results of the query.

2.3. Discussion

Constraints in categories 1 and 2 play an identical role for heuristics H1 and H2. However, the other categories play a different role in the two cases. Tightened constraints are similar to existing query constraints that can potentially be eliminated. Index constraints are new constraints that can potentially be added to the `where` clause of the query. In the next section we present a graph algorithm that finds associations from the relaxed constraints to the tightened (H1) or the index (H2) constraints respectively. Since the algorithm is common to both heuristics, tightened and index constraints are hereafter referred to as *target constraints*.

3. Identifying associations between constraints

In this section, we present an algorithm that goes through the association rules in the warehouse and identifies direct or indirect correlations between the relaxed and the target constraints discussed in the previous section. Before presenting this algorithm, we discuss an alternative graphical representation of the association rules. Consider for example the association rules in figure 1. They can be represented using a directed graph of constraints, see figure 2.

A rule having a single constraint in its antecedent (e.g. $C_{1.R_{11}} \Rightarrow C_{2.R_{21}}$) is represented by two vertices, labeled with the constraints ($C_{1.R_{11}}$ and $C_{2.R_{21}}$) and linked together by an edge from the antecedent constraint ($C_{1.R_{11}}$) to the consequent constraint ($C_{2.R_{21}}$). A rule with more than one constraint in its antecedent uses an additional vertex to denote the conjunction. The link $\leadsto$ denotes the fact that a constraint in the antecedent of a rule implies the consequent only in combination with the other conjuncts forming the antecedent. The dashed edges link pairs of nodes that represent constraints on the same attribute, with the source constraint implying the destination constraint.

The notion of path in our context requires one extension to the standard notion for a directed graph. A path from a constraint C_{source} to a constraint C_{dest} is any sequence of vertices linked by directed edges, provided that no vertex represents a composite constraint. If a path contains a composite constraint, say $C = C_1 \wedge \ldots \wedge C_n$, then there must be n paths from C_{source}, one to each of the conjuncts C_i. These subpaths merge at the node representing the composite constraint, and from there the path continues towards C_{dest}.

We consider also paths from a set of constraints $\{C_1, \ldots C_n\}$ to a single constraint. Such a path consists of subpaths starting from the source constraints $\{C_1, \ldots C_n\}$ which merge at various composite constraints before reaching the destination constraint.

We can now develop an algorithm that navigates over a graph of constraints, finds paths linking relaxed (combinations of) constraints to target constraints and combines them to derive association rules useful for the purposes of optimization. It consists of three basic steps as shown in figure 3. The algorithm is discussed in detail in section 3.1.

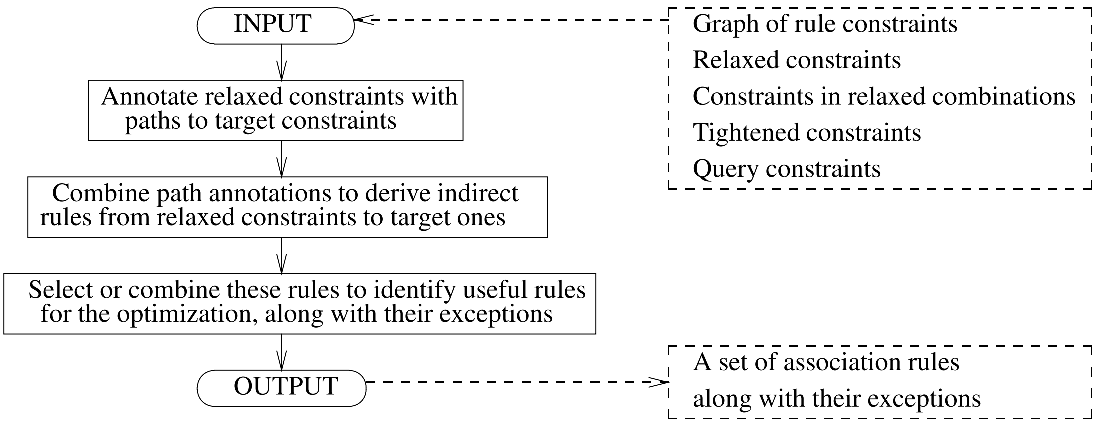

Figure 3. An abstract view of the algorithm

At the end of each step we give an example to show how it can be applied for heuristic H1. It is equally applicable to heuristic H2.

Relaxed constraints and constraints in relaxed combinations often play the same role in the following algorithm, and we use the term *source constraint* in such contexts.

3.1. Algorithm

For each constraint $C_{k.R_{kj}}$ in the target constraints take the following steps:

Step 1

Navigate backwards from $C_{k.R_{kj}}$ in the graph of constraints, i.e. navigate against the direction of the edges and annotate any source constraints encountered with information about the reverse path traversed so far. On encountering a composite constraint navigate backwards from *all* of its conjuncts. If a particular constraint is encountered more than once, or if there is no incoming link, then backtracking from the current constraint terminates.

Step 1 may be implemented as a recursive method `backtrack` on a class `Constraint`. We demonstrate its functionality by an example. Consider the target constraint $C_{4.R_{41}}$ in the constraint graph (figure 2). We recursively apply `backtrack` to all the constraints leading to it, i.e. $C_{1.R_{13}}$, $C_{5.R_{53}}$ and C_{comp_2}. The argument for the method is the path traversed so far, i.e. $\{C_{4.R_{41}}\}$. $C_{1.R_{13}}$ is not a source constraint, so no path annotation is assigned to it. No constraint leads to it (there is no incoming link), so there is no further recursive call from it. However, $C_{5.R_{53}}$ is a constraint in a relaxed combination, so it is annotated with the path $\{C_{4.R_{41}}\}$. No recursive call occurs from it either. C_{comp_2} is a composite constraint, so we recursively call `backtrack` on each of the conjuncts $C_{2.R_{22}}$ and $C_{3.R_{31}}$, passing as argument the updated path traversed $\{C_{comp_2} \rightarrow C_{4.R_{41}}\}$. Recursive calls of `backtrack` continue until the constraint on which a call is made has no incoming link or the constraint appears in the argument path annotation. In figure 4, we present the path annotations assigned to source constraints by the time Step 1 is completed for the target constraint $C_{4.R_{41}}$.

Discussion of step 2

The objective of step 2 is to combine the annotations of source constraints to identify *complete* paths from sets of these constraints to a target constraint $C_{k.R_{kj}}$. The first step of the algorithm annotated the source constraints with complete or incomplete paths from them to a particular target constraint. For instance, $C_{1.R_{11}}$, which is a relaxed constraint, is annotated with four paths, as shown in figure 4. Paths that include $\rightsquigarrow$ links are called incomplete; they are useful only in combination with other incomplete paths that include the related conjuncts. One of the objectives of step 2 is to combine incomplete paths in order to identify (complete) paths from a source constraint like $C_{1.R_{11}}$ to a target constraint like $C_{4.R_{41}}$. In fact, we can combine paths from more than one source constraint to a destination constraint.

Each edge $\rightarrow$ stands for an association rule which possibly has a list of exceptions E. A series of consecutive edges $\rightarrow$ corresponds to an indirect association from the source constraint to the destination one. The exceptions to this rule are evaluated from the exceptions of the association rules involved. In particular, if constraint A leads to constraint B based on a rule with exceptions $E_{A\Rightarrow B}$, and B leads to C based on a rule with exceptions $E_{B\Rightarrow C}$, then the exception set of the indirect path $A \rightarrow C$ (or the indirect rule $A \Rightarrow C$) is:

$$\begin{aligned} E \;=\; & \{exc | exc \leftarrow E_{A\Rightarrow B},\ \mathrm{not}\ C(exc)\} \\ & \cup \{exc | exc \leftarrow E_{B\Rightarrow C},\ A(exc)\} \end{aligned} \tag{2}$$

Note that $E_{A\Rightarrow B} \cap E_{B\Rightarrow C} = \{\}$, since all exceptions of $E_{A\Rightarrow B}$

Relaxed Constraints	*Path Annotations*
$C_{2.R_{22}}$	$\{C_{2.R_{22}} \rightsquigarrow C_{comp_2} \to C_{4.R_{41}}\}$ $\{C_{2.R_{22}} \to C_{2.R_{23}} \rightsquigarrow C_{comp_3} \to C_{5.R_{51}} \to C_{5.R_{52}} \to C_{3.R_{31}} \rightsquigarrow C_{comp_2} \to C_{4.R_{41}}\}$
$C_{4.R_{41}}$	
$C_{1.R_{11}}$	$\{C_{1.R_{11}} \to C_{2.R_{21}} \to C_{2.R_{22}} \rightsquigarrow C_{comp_2} \to C_{4.R_{41}}\}$ $\{C_{1.R_{11}} \to C_{2.R_{21}} \to C_{2.R_{22}} \to C_{2.R_{23}} \rightsquigarrow C_{comp_3} \to C_{5.R_{51}} \to C_{5.R_{52}} \to C_{3.R_{31}} \rightsquigarrow C_{comp_2} \to C_{4.R_{41}}\}$ $\{C_{1.R_{11}} \to C_{2.R_{21}} \rightsquigarrow C_{comp_1} \to C_{3.R_{31}} \rightsquigarrow C_{comp_2} \to C_{4.R_{41}}\}$ $\{C_{1.R_{11}} \to C_{1.R_{12}} \rightsquigarrow C_{comp_1} \to C_{3.R_{31}} \rightsquigarrow C_{comp_2} \to C_{4.R_{41}}\}$

Constraints in Relaxed Combinations	*Path Annotations*
$C_{5.R_{51}}$	$\{C_{5.R_{51}} \to C_{5.R_{52}} \to C_{3.R_{31}} \rightsquigarrow C_{comp_2} \to C_{4.R_{41}}\}$
$C_{5.R_{53}}$	$\{C_{5.R_{53}} \to C_{4.R_{41}}\}$

Figure 4. Paths from relaxed constraints to the target constraint $C_{4.R_{41}}$

A(e)	B(e)	C(e)	e in $E(A \Rightarrow B)$	e in $E(B \Rightarrow C)$	*e in* $E(A \Rightarrow C)$
T	T	T	F	F	F
T	T	F	F	T	T
T	F	T	T	F	F
T	F	F	T	F	T
F	T	T	F	F	F
T	T	F	F	T	F
F	F	T	F	F	F
F	F	F	F	F	F

Figure 5. Truth table which shows the correctness of formula 2

satisfy A, but do not satisfy B. Therefore they are not exceptions of $E_{B \Rightarrow C}$. The correctness of formula 2 is proved by the truth table in figure 5.

Step 2

Path annotations derived from step 1 are combined to form complete paths by repeating steps 2.1 and 2.2:

Step 2.1 For all path annotations we omit all the initial constraints, until the first constraint which is followed by a $\rightsquigarrow$ edge (this constraint is maintained). To omit these initial constraints, we must first evaluate the exceptions involved in the rules until the $\rightsquigarrow$ link (equation 2). The resulting paths are either empty or start with subconstraints leading to composite constraints.

When this step is executed for the first time the resulting paths in our example are the following:

$$\begin{array}{ll}
C_{2.R_{22}} & P_1: \{C_{2.R_{22}} \rightsquigarrow C_{comp_2} \to C_{4.R_{41}}\} \\
 & E_1 = \{\} \\
C_{2.R_{22}} & P_2: \{C_{2.R_{23}} \rightsquigarrow C_{comp_3} \to C_{5.R_{51}} \to C_{5.R_{52}} \to C_{3.R_{31}} \rightsquigarrow \\
 & \quad C_{comp_2} \to C_{4.R_{41}}\} \\
 & E_2 = E(C_{2.R_{22}} \Rightarrow C_{2.R_{23}}) = \{\} \\
C_{1.R_{11}} & P_3: \{C_{2.R_{22}} \rightsquigarrow C_{comp_2} \to C_{4.R_{41}}\} \\
 & E_3 = \{e|e \leftarrow E(C_{1.R_{11}} \Rightarrow C_{2.R_{21}}),\ !C_{2.R_{22}}(e)\} \\
C_{1.R_{11}} & P_4: \{C_{2.R_{23}} \rightsquigarrow C_{comp_3} \to C_{5.R_{51}} \to C_{5.R_{52}} \to C_{3.R_{31}} \rightsquigarrow \\
 & \quad C_{comp_2} \to C_{4.R_{41}}\} \\
 & E_4 = \{e|e \leftarrow E(C_{1.R_{11}} \Rightarrow C_{2.R_{21}}),\ !C_{2.R_{22}}(e),\ !C_{2.R_{23}}(e)\} \\
 & \quad = \{e|e \leftarrow E(C_{1.R_{11}} \Rightarrow C_{2.R_{21}}),\ !C_{2.R_{23}}(e)\} \\
C_{1.R_{11}} & P_5: \{C_{2.R_{21}} \rightsquigarrow C_{comp_1} \to C_{3.R_{31}} \rightsquigarrow C_{comp_2} \to C_{4.R_{41}}\} \\
 & E_5 = E(C_{1.R_{11}} \Rightarrow C_{2.R_{21}}) \\
C_{1.R_{11}} & P_6: \{C_{1.R_{12}} \rightsquigarrow C_{comp_1} \to C_{3.R_{31}} \rightsquigarrow C_{comp_2} \to C_{4.R_{41}}\} \\
 & E_6 = \{\} \\
C_{5.R_{51}} & P_7: \{C_{3.R_{31}} \rightsquigarrow C_{comp_2} \to C_{4.R_{41}}\} \\
 & E_7 = \{\} \cup \{e|e \leftarrow E(C_{5.R_{52}} \Rightarrow C_{3.R_{31}}), C_{5.R_{51}}(e)\} \\
C_{5.R_{53}} & P_8: \{\} \\
 & E_8 = E(C_{5.R_{53}} \Rightarrow C_{4.R_{41}})
\end{array}$$

If a path annotation is empty, it means that there is a complete path from the source constraint (e.g. $C_{5.R_{53}}$) to the destination one ($C_{4.R_{41}}$). A new rule is created from the former to the latter ($C_{5.R_{53}} \Rightarrow C_{4.R_{41}}$) and the exceptions evaluated so far for this path annotation (E_8) are assigned to the new association rule. The path annotation and its exceptions are deleted. If an empty annotation corresponds to more than one source constraint then the rule that is generated has a composite antecedent, i.e. it is of the form: $C_1 \wedge \ldots \wedge C_n \Rightarrow C_0$.

When using equation 2 to determine exceptions during step 2.1 we must take note of the exception sets arising from earlier iterations of steps 2.1 and 2.2.

Step 2.2 For each composite constraint C_{comp} at the second position of some path annotation, we try to find a set of path annotations having at their first positions the component constraints of $\mathtt{C}_{\mathtt{comp}}$. These path annotations are combined into new path annotations as follows: i) the subconstraints at their first positions are omitted; ii) the resulting exceptions of the remaining paths are the union of the exceptions evaluated so far for each combined path that satisfy the source constraints of the other combined paths.

$$\begin{aligned}&\mathtt{E}(\mathtt{C_A} \wedge \mathtt{C_B} \Rightarrow \mathtt{C}) = \\ &\qquad \{\mathtt{e}|\mathtt{e} \leftarrow \mathtt{E}(\mathtt{C_A} \Rightarrow \mathtt{C}), \mathtt{C_B}(\mathtt{e})\} \cup \\ &\qquad \{\mathtt{e}|\mathtt{e} \leftarrow \mathtt{E}(\mathtt{C_B} \Rightarrow \mathtt{C}), \mathtt{C_A}(\mathtt{e})\}\end{aligned} \tag{3}$$

When step 2.2 is executed for the first time in our example, the following path annotations are generated:

$$\begin{array}{ll}
\mathtt{C}_{2.\mathtt{R}_{22}}, \mathtt{C}_{5.\mathtt{R}_{51}} & \mathtt{P}_9: \{\mathtt{C}_{\mathtt{comp}_2} \rightarrow \mathtt{C}_{4.\mathtt{R}_{41}}\} \\
& \mathtt{E}_9 = \{\mathtt{e}|\mathtt{e} \leftarrow \mathtt{E}_7,\ \mathtt{C}_{2.\mathtt{R}_{22}}(\mathtt{e})\} \\
\mathtt{C}_{1.\mathtt{R}_{11}}, \mathtt{C}_{5.\mathtt{R}_{51}} & \mathtt{P}_{10}: \{\mathtt{C}_{\mathtt{comp}_2} \rightarrow \mathtt{C}_{4.\mathtt{R}_{41}}\} \\
& \mathtt{E}_{10} = \{\mathtt{e}|\mathtt{e} \leftarrow \mathtt{E}_3,\ \mathtt{C}_{5.\mathtt{R}_{51}}(\mathtt{e})\} \\
& \quad \cup \{\mathtt{e}|\mathtt{e} \leftarrow \mathtt{E}_7,\ \mathtt{C}_{1.\mathtt{R}_{11}}(\mathtt{e})\} \\
\mathtt{C}_{1.\mathtt{R}_{11}} & \mathtt{P}_{11}: \{\mathtt{C}_{\mathtt{comp}_1} \rightarrow \mathtt{C}_{3.\mathtt{R}_{31}} \rightsquigarrow \mathtt{C}_{\mathtt{comp}_2} \rightarrow \mathtt{C}_{4.\mathtt{R}_{41}}\} \\
& \mathtt{E}_{11} = \{\mathtt{e}|\mathtt{e} \leftarrow \mathtt{E}_5,\ \mathtt{C}_{1.\mathtt{R}_{11}}(\mathtt{e})\} \\
& \quad = \mathtt{E}(\mathtt{C}_{1.\mathtt{R}_{11}} \Rightarrow \mathtt{C}_{2.\mathtt{R}_{21}})
\end{array}$$

If two path annotations which are combined do not have the same paths after the composite constraint (after the second position), then two combinations of paths should be generated, one for each annotation.

We retain path annotations $\mathtt{P}_1$ to $\mathtt{P}_8$ in order to combine them with new annotations at a later execution of step 2.2.

Substeps 2.1 and 2.2 are repeated until neither of them has any effect on the path annotations. This happens when all paths that have not been converted to rules in step 2.1 cannot be further combined in step 2.2.

Step 3

Not all the rules derived in step 2.1 are useful for optimization purposes. The aim of this step is to filter and combine the rules derived from the previous step, referred to as $\mathtt{R_A}$, in order to generate rules relating query constraints with each other (H1), or query constraints to new index constraints (H2). The resulting set of rules is referred to as $\mathtt{R_B}$. We first give a detailed account of step 3 in the context of heuristic H1. We then point out a detail that should be changed so that the step is also applicable for H2.

We look for (sets of) rules in $\mathtt{R_A}$ of the following kinds:

- A rule from a relaxed constraint to the target constraint $\mathtt{C_i}$, i.e. of the form $\mathtt{C} \Rightarrow \mathtt{C_0}$. Such a rule is used to generate a new rule $\mathtt{C}' \Rightarrow \mathtt{C}'_0$, such that $\mathtt{C}'$, $\mathtt{C}'_0$ are the query constraints that refer to the same attributes as $\mathtt{C}$ and $\mathtt{C_0}$ respectively, $\mathtt{C}$ is one of the relaxed constraints of $\mathtt{C}'$ and $\mathtt{C_0}$ is one of the target constraints of $\mathtt{C}'_0$. Indeed,

$$\left.\begin{array}{l}\mathtt{C}' \Rightarrow \mathtt{C} \\ \mathtt{C} \Rightarrow \mathtt{C_0} \text{ (existing rule)} \\ \mathtt{C_0} \Rightarrow \mathtt{C}'_0\end{array}\right\} \mathtt{C}' \Rightarrow \mathtt{C}'_0$$

If the rule $\mathtt{C} \Rightarrow \mathtt{C_0}$ has exceptions $\mathtt{E}$, the exceptions of the new rule $\mathtt{C}' \Rightarrow \mathtt{C}'_0$ are $\mathtt{E}' = \{\mathtt{e}|\mathtt{e} \leftarrow \mathtt{E},\ \mathtt{C}'(\mathtt{e})\}$.

- A set of rules of the form $\mathtt{C_i} \Rightarrow \mathtt{C_0}, \mathtt{i} = 1, \ldots, \mathtt{n}$, such that $\mathtt{C_i}$ are all constraints in the same relaxed combination of size $\mathtt{n}$. This set of rules is used to generate the new rule $\mathtt{C}' \Rightarrow \mathtt{C}'_0$, in which $\mathtt{C}'$, $\mathtt{C}'_0$ are the query constraints that refer to the same attribute as the $\mathtt{C_i}$ and $\mathtt{C_0}$ respectively. If the corresponding exception lists of the rules $\mathtt{C_i} \Rightarrow \mathtt{C_0}, \mathtt{i} = 1, \ldots, \mathtt{n}$ are $\mathtt{E_i}$, the exceptions of the new rule are $\mathtt{E} = \{\mathtt{e}|\mathtt{e} \leftarrow \mathtt{E_1} \cup \ldots \cup \mathtt{E_n},\ \mathtt{C}'(\mathtt{e})\}$.

- A rule from a set of relaxed constraints on different attributes to a target one, i.e. $\mathtt{C_1} \wedge \ldots \wedge \mathtt{C_n} \Rightarrow \mathtt{C_0}$. This rule is converted to the corresponding rule $\mathtt{C}'_1 \wedge \ldots \wedge \mathtt{C}'_n \Rightarrow \mathtt{C}'_0$, such that $\mathtt{C}'_0, \ldots, \mathtt{C}'_n$ are the query constraints on the same attributes as $\mathtt{C_0}, \ldots, \mathtt{C_n}$ respectively. The exceptions of the new rule are those of the initial rule that satisfy $\mathtt{C}'_1 \wedge \ldots \wedge \mathtt{C}'_n$, i.e. $\mathtt{E}' = \{\mathtt{e}|\mathtt{e} \leftarrow \mathtt{E},\ \mathtt{C}'_1(\mathtt{e}),\ \ldots,\ \mathtt{C}'_n(\mathtt{e})\}$.

- A set of rules whose antecedents contain both relaxed constraints and constraints in relaxed combinations is useful if it consists of all rules of the form:

$$\overbrace{\mathtt{C_1} \ \ldots \ \mathtt{C_k}}^{\text{Relaxed constraints}} \quad \overbrace{\mathtt{C}_{(\mathtt{k}+1).\mathtt{j}_1} \ \ldots \ \mathtt{C}_{(\mathtt{k}+\mathtt{m}).\mathtt{j}_\mathtt{m}}}^{\text{Constraints in relaxed combinations}} \Rightarrow \mathtt{C_0}$$

where $1 \leq \mathtt{j_i} \leq \mathtt{n_i}$ for all $1 \leq \mathtt{i} \leq \mathtt{m}$.

Constraints $\mathtt{C}_{(\mathtt{k}+\mathtt{i}).1}, \ldots, \mathtt{C}_{(\mathtt{k}+\mathtt{i}).\mathtt{n_i}}$ form the relaxed combination $\mathtt{i}$. The set contains $\mathtt{N} = \mathtt{n_1} \times \ldots \times \mathtt{n_m}$ rules, such that each constraint in a relaxed combination occurs in all possible combinations with constraints from the other relaxed combinations. In some of the rules some (or all) of the relaxed constraints $\mathtt{C_1} \ldots \mathtt{C_n}$ may not be present.

If such a set of rules occurs in $\mathtt{R_A}$ we may derive the new rule $\mathtt{C}'_1 \ \ldots \ \mathtt{C}'_k \ \mathtt{C}'_{k+1} \ \ldots \ \mathtt{C}'_{k+m} \Rightarrow \mathtt{C}'_0$, such that $\mathtt{C}'_1, \ldots, \mathtt{C}'_{k+m}, \mathtt{C}'_0$ are the query constraints that refer to the same attributes as $\mathtt{C_1}, \ldots, \mathtt{C}_{(\mathtt{k}+\mathtt{m}).\mathtt{j_m}}, \mathtt{C_0}$ respectively. If the original rules have exception lists $\mathtt{E_1}, \ldots, \mathtt{E_N}$, then the resulting rule has exceptions $\mathtt{E} = \{\mathtt{e}|\mathtt{e} \leftarrow \mathtt{E_1} \cup \ldots \cup \mathtt{E_N},\ \mathtt{C}'_1(\mathtt{e}),\ \ldots,\ \mathtt{C}'_{k+m}(\mathtt{e})\}$.

Evaluating exceptions appears very expensive in this case. However, in the common situation that $\mathtt{m} = 1$, the cost of finding exceptions to the new rule is similar to the cost in the second case of step 3.

Note that in the context of heuristic H2, $\mathtt{C_0}$ is a constraint on the cluster index attribute; it does not correspond to any of the existing query constraints. Hence, step 3 is applicable to H2, provided that in all four cases above, $\mathtt{C}'_0 = \mathtt{C_0}$.

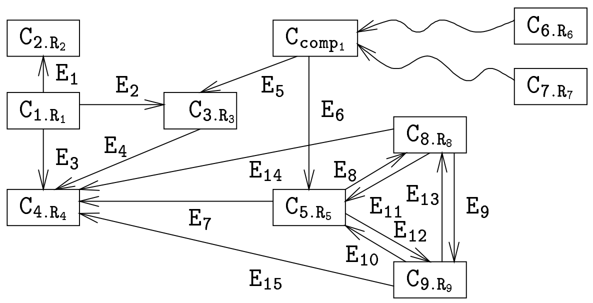

Figure 6. Correlations of query constraints

4. Identifying optimization solutions

In the previous section, we gave an algorithm that navigates over a constraint graph and extracts a set of useful association rules along with their exceptions. This section combines these rules to identify all possible solutions to the constraint elimination or constraint introduction problem.

Consider an OQL `select` query with say 15 predicates. We combine all predicates referring to the same attribute into a single constraint $\mathtt{attr_i}$ in $\mathtt{range_i}$ ($C_{i.R_i}$). Assume that we derive the following set of constraints: $C_{1.R_1}, \ldots, C_{12.R_{12}}$, say. Since there is just one range for each attribute, ranges are presented as R_i instead of R_{ij}. We first identify the rule constraints of possible interest for H1 or H2, then apply the graph algorithm presented in section 3. The next step is discussed separately for each heuristic.

4.1. Heuristic H1

Say that the graph algorithm (section 3) results in the associations illustrated in figure 6. Not all query constraints need be related; in our example only the first nine are. Note that the association graph (figure 6) is transitive; if there is a direct link from constraint A to constraint B, and another one from B to C, then A and C are also directly connected. This is a property of the graph algorithm in section 3. We now define an algorithm that produces a set of constraint elimination solutions $\{(M_i, E_i)\}$, where M_i, E_i stand for Maintained Constraints and Exceptions respectively. Each pair (M_i, E_i) implies a solution of the following form:

> We may eliminate all but the constraints M_i from the `where` clause of the query, given that we take into account the exceptions E_i.

Algorithm

Step 1 Identify all constraints that do not have any incoming links. These constraints should be maintained (not eliminated), since no other constraint implies them. In figure 6, these constraints are $C_{1.R_1}, C_{6.R_6}, C_{7.R_7}$.

Step 2 Identify all cyclic paths that have no incoming link from any external constraint. If two cyclic paths have at least one common constraint, they are considered as a single cyclic path. This case does not occur in our example. Form combinations of constraints by choosing one constraint from every cyclic path identified. In our example there is just one such path containing the constraints $C_{5.R_5}, C_{8.R_8}, C_{9.R_9}$; therefore the combinations consist of just one element: $\{C_{5.R_5}\}$, $\{C_{8.R_8}\}$, $\{C_{9.R_9}\}$. Extend each of these combinations with the constraints without incoming links, identified in step 1. The resulting combinations are $\{C_{5.R_5}, C_{1.R_1}, C_{6.R_6}, C_{7.R_7}\}$, $\{C_{8.R_8}, C_{1.R_1}, C_{6.R_6}, C_{7.R_7}\}$ and $\{C_{9.R_9}, C_{1.R_1}, C_{6.R_6}, C_{7.R_7}\}$. Each combination is a minimal set of constraints that implies all the remaining constraints in the graph (figure 6).

Step 3 For each combination evaluate the exceptions that are involved in removing the implied constraints. If a constraint is implied by more than one constraint in a combination, then consider the exceptions of the strongest implication (the implication with the fewest exceptions). For example, the exceptions corresponding to the combination $\{C_{5.R_5}, C_{1.R_1}, C_{6.R_6}, C_{7.R_7}\}$ are evaluated by forming the union of the exception lists: E_1 for the elimination of C_2; $\mathtt{min\_set}\{E_2, E_5\}$ for the elimination of C_3; $\mathtt{min\_set}\{E_3, E_6, E_7\}$ for the elimination of C_4; E_8 for the elimination of C_8; E_{12} for the elimination of C_9.

Step 4 Form the final solutions to the elimination problem by adding to each combination the remaining query constraints that are not related to each other ($C_{10.R_{10}}, C_{11.R_{11}}, C_{12.R_{12}}$). The exceptions corresponding to each solution are filtered so that they satisfy all the maintained constraints, i.e. all the constraints in the combination.

Note that all the solutions identified in the algorithm have the same number of constraints. We assume that the cardinality of the extent against which the query is run is far greater than any of the exception lists annotating the optimization solutions. Therefore, there is no advantage in eliminating only a subset of the constraints in a solution in order to decrease the number of exceptions involved.

4.2. Heuristic H2

Assume that in the context of H2 the graph algorithm (section 3) results in the associations illustrated in figure 7. The constraints of the form $C_{i.R_i}$ are existing query constraints, while the constraints $C_{0.R_{0j}}, j = 1, \ldots, 3$ are new constraints on the cluster index attribute. The additional subscript j is needed because the constraints on

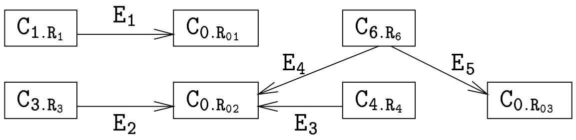

Figure 7. Correlations between query constraints and new indexed constraint

the index attribute may have different ranges. We identify all possible constraint introduction solutions as follows:

Algorithm

For each constraint on the index attribute $\mathtt{C_{0.R_{0j}}}$

Step 1 find the least expensive association from a query constraint to $\mathtt{C_{0.R_{0j}}}$, i.e. find the incoming link $\mathtt{C_{i.R_i}} \Rightarrow \mathtt{C_{0.R_{0j}}}$ with the fewest exceptions. For example if the cardinalities of $\mathtt{E_1}, \ldots, \mathtt{E_5}$ are $\mathtt{n_1}, \ldots, \mathtt{n_5}$ and $\mathtt{n_4} \leq \mathtt{n_2} \leq \mathtt{n_3}$, then the least expensive association for $\mathtt{C_{0.R_{0j}}}$ is $\mathtt{C_{6.R_6}} \Rightarrow \mathtt{C_{0.R_{02}}}(\mathtt{E} = \mathtt{E_4})$.

Step 2 filter out all the exceptions that do not satisfy at least one of the query constraints $\mathtt{C_{1.R_1}}, \ldots, \mathtt{C_{12.R_{12}}}$. We do not need to test the exceptions for the antecedent of the corresponding rule, since they satisfy it by definition.

The constraint introduction solutions identified in our example are the following:

Introduced Constraints	*Exceptions*
$\mathtt{C_{0.R_{01}}}$	$\{\mathtt{e}\mid\mathtt{e} \leftarrow \mathtt{E_1}, \mathtt{C_{2.R_2}(e)}, \ldots, \mathtt{C_{12.R_{12}}(e)}\}$
$\mathtt{C_{0.R_{02}}}$	$\{\mathtt{e}\mid\mathtt{e} \leftarrow \mathtt{E_4}, \mathtt{C_{1.R_1}(e)}, \ldots, \mathtt{C_{5.R_5}(e)}, \mathtt{C_{7.R_7}(e)}, \ldots, \mathtt{C_{12.R_{12}}(e)}\}$
$\mathtt{C_{0.R_{03}}}$	$\{\mathtt{e}\mid\mathtt{e} \leftarrow \mathtt{E_5}, \mathtt{C_{1.R_1}(e)}, \ldots, \mathtt{C_{5.R_5}(e)}, \mathtt{C_{7.R_7}(e)}, \ldots, \mathtt{C_{12.R_{12}}(e)}\}$

5. Optimizing OQL queries

In the previous sections, a series of algorithms were given to find a collection of constraint elimination or constraint introduction solutions. In this section, we show how the original query is transformed to its optimized form using the optimal solution. Consider the OQL query:

```
select x
from Extent_X as x
where C1.R1 and ... and Cn.Rn
```

5.1. Heuristic H1

Let $< \{\mathtt{C_{i1}}, \ldots, \mathtt{C_{im}}\}, \mathtt{E_i} >$ be the maintained constraints and the exceptions of a solution `i`. Only the maintained constraints of the optimization solution should be tested on the objects of the whole extent; however, all the constraints should be tested on the exception cases and the objects that satisfy the maintained constraints but not the ones omitted should be removed from the result. Hence, the original query should be converted to the following one:

```
(select x from Extent_X as x
 where Ci1 and ... and Cim)
   except Ei
```

If we assume that tests on the query constraints take roughly the same time, the optimal solution is the one with fewest exceptions $\mathtt{E_i}$.

5.2. Heuristic H2

Let $< \mathtt{C_{0.R_{0i}}}, \mathtt{E_i} >$ be the index constraint and the corresponding exceptions of a constraint introduction solution. Instead of testing the query constraints $\mathtt{C_{1.R_1}}, \ldots, \mathtt{C_{n.R_n}}$ on the entire extent `Extent_X`, we apply them only on the results of the subquery

```
select x
from Extent_X as x          (Q0)
where C0.R0i
```

Since $\mathtt{C_{0.R_{0i}}}$ is a constraint on a cluster index attribute, the select operation is expected to be quite fast. The original query is transformed to its more efficient form:

```
(select x
 from Q0 as x
 where C1.R1 and ... and Cn.Rn)
   union Ei
```

The exceptions $\mathtt{E_i}$ are merged to the result because they satisfy the query constraints, but not the new index constraint $\mathtt{C_{0.R_{0i}}}$. Since the `union` operation is relatively cheap, the optimal solution is the one that introduces the index constraint with the highest selectivity.

6. Discussion

We now look at two different scenarios, and estimate the extent to which heuristics H1 and H2 speed up query execution.

The first scenario concerns frequently executed queries. Assume that the association rules which are used by algorithm 3 are not modified. We may optimize a query once at compilation time, then execute its optimized form. It is worth optimizing provided that the execution time of the optimized query is less than the execution time of the original query. For heuristic H1 this happens only if the time saved by omitting some constraints is greater than the time needed to remove the exceptions from the result (`except` operation). The more the eliminated constraints and the fewer the exceptions, the better the optimization. As one of the referees pointed out, the time saved by the elimination of constraints is CPU-related. Since query execution is dominated by data access time, this optimization is not expected to alter performance significantly. It would help only in contexts rich in associations with few exceptions, in which users express many constraints in their queries.

Heuristic H2 is expected to bring more significant benefits. Firstly, this optimization involves a `union` operation, which is much cheaper than the `except` operation used in H1. Secondly, instead of retrieving all the objects of an extent from the database, we need only look at the subset retrieved through an indexed constraint. Hence, we save a considerable amount of data access time, spending a negligible amount of CPU time in evaluating the additional constraint.

The second scenario concerns queries which are executed only once. In this case, the time required for optimization is significant. This time depends on the algorithm that finds associations between relaxed constraints and tight constraints (see section 3), since this is the most expensive step in the optimization process. This algorithm finds paths in a directed graph, and combines the exceptions associated with each edge of the path to derive the total exceptions for the path. Therefore its complexity is a function of i) the average number of exceptions in the existing association rules and ii) the number of different constraints found in the antecedents and the consequents of the rules.

We have already implemented the algorithms for applying H1; the next step is to implement the corresponding algorithms for H2. This should not be difficult, since the main algorithm - finding associations between rule constraints - is common to the two heuristics. We intend to set up an experimental model in order to evaluate H1 and H2 in the scenarios discussed above.

7. Conclusion

The use of association rules for query optimization is relevant to both relational and object-oriented database systems. There has been a lot of research on generating association rules and maintaining them in the presence of updates. Research has also focused on finding heuristics that take advantage of rules in order to optimize a query. Most of this work ([2, 3]) has considered integrity rules, rather than association rules with exceptions. Semantic optimization heuristics were also applied without considering indirect associations. In this paper, we implement algorithms that apply two optimization heuristics presented by Siegel et al., taking account of both exceptions and indirect associations. We show how to use these heuristics to optimize an OQL query. The complexity of the optimization process is closely related to the complexity of the constraint graph, which represents the set of association rules in the data. It also depends on the number of exceptions associated with each rule. We have designed an experimental framework to evaluate the two optimization techniques, both in the context of queries repeated frequently over a period of time, and in the context of ad-hoc queries executed once only. The results of this experimental work will be presented in a later paper.

8. Acknowledgements

We are grateful to the anonymous referees, who read the paper carefully and critically and made many helpful suggestions. Agathoniki Trigoni is supported by a scholarship from the Greek Scholarships Foundation, and is deeply obliged to the National Bank of Greece.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD Intl Conference on Management of data*, pages 207–216, 1993.

[2] U. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, 1990.

[3] J. Grant, J.Gryz, J. Minker, and L. Raschid. Semantic query optimization for object databases. In *ICDE*, pages 444–453, 1997.

[4] R. Miller and Y. Yang. Association rules over interval data. In *ACM SIGMOD*, 1997.

[5] J. Park. An effective hash-based algorithm for mining association rules. In *ACM SIGMOD*, pages 175–186, 1995.

[6] M. Siegel, E. Sciore, and S. Salveter. A method for automatic rule derivation to support semantic query optimization. *ACM Transactions on Database Systems*, 17(4):563–600, December 1992.

[7] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *ACM SIGMOD Intl Conference on Management of data*, pages 1–12, 1996.

[8] D. Tsur, J. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: a generalization of association-rule mining. In *ACM SIGMOD Intl Conference on Management of data*, pages 1–12, 1998.

[9] S. Yoon. Intelligent query answering in deductive and object-oriented databases. In *Fourth ACM Intl Conference on Information and Knowledge Management*, pages 244–251, 1994.

[10] S. Yoon, I. Song, and E. Park. Semantic query processing in object-oriented databases using deductive approach. In *Intl Conference on information and knowledge management*, pages 150–157, 1995.