Semantic Query Optimization for Tree and Chain Queries

Wei Sun and Clement T. Yu

Abstract—**Semantic query optimization, or knowledge-based query optimization, has received increasing interest in recent years. This paper provides an effective and systematic approach to optimizing queries by appropriately choosing semantically equivalent transformations. Basically, there are two different types of transformations: transformations by eliminating unnecessary joins, and transformations by adding/eliminating redundant *beneficial/nonbeneficial* selection operations (restrictions). A necessary and sufficient condition to eliminate a single unnecessary join is provided. We prove that it is $\mathfrak{NP}$-Complete to eliminate as many unnecessary joins as possible for various types of acyclic queries with the exception of the *closure chain* queries whose query graphs are chains and all equi-join attributes are distinct. An algorithm is provided to minimize the number of joins in tree queries. This algorithm has an important property that, when applied to a closure chain query, it will yield an optimal solution with the time complexity $O(n * m)$, where n is the number of relations referenced in the chain query, and m is the time complexity of a *restriction closure computation*. (A restriction closure consists of all deducible restrictions of a query qualification under a given set of constraints. An algorithm to compute it was given in [46]).**

Index Terms—**Chain, complexity, join, NP-complete, query transformation, restriction, semantic query optimization, tree.**

I. Introduction

QUERY optimization is a classical problem of database systems, and has been attacked in different ways [18], [19], [25], [33], [34], [36], [37], [40]. Semantic query optimization or knowledge-based query optimization proposed since early 1980's [1]–[3], [9], [12], [14]–[17], [20], [26]–[29], [46] has been increasingly gaining attention from database researchers in recent years.

Semantic integrity constraints (constraints for brief) assert permissible or consistent database states (a database state is the stored data at a given instance). For example, in a university database, the university/department regulation requires that all graduate students maintain a minimum GPA of 3.0; in a payroll database, the age of the youngest employee must be, by law, greater than or equal to 18.

The general concern of this paper is semantic query optimization, that is, given a query and a set of constraints, transform the original query into another semantically equivalent one such that the transformed query can be evaluated more efficiently. The basic transformations are: eliminating unnecessary joins, adding redundant beneficial restrictions, and eliminating redundant nonbeneficial restrictions. By adding restrictions, it is possible that a contradiction in a query qualification under given semantic knowledge is detected. A contradiction in a query qualification implies that the answer to the query is null without physically accessing the database.

The contributions of this paper are as follows.

- The complexity of semantic query optimization in terms of eliminating unnecessary joins has not been addressed before. We prove that this problem is $\mathfrak{NP}$-Complete for various types of acyclic queries with the exception of the *closure chain queries* where all equi-join attributes are distinct and whose query graphs are chains.
- An algorithm to semantically optimize tree queries is provided. This algorithm has the property that when applied to a closure chain query, an optimal solution can be obtained in time $O(n * m)$, where n is the number of relations involved in the qualification of a chain query, and m is the time complexity of a *restriction closure* computation (which will be briefly discussed in Section II-C; for detail please see [46]). Since the set of closure chain queries is a natural subclass of the general chain queries (whose semantic query optimization is shown to be $\mathfrak{NP}$-Complete in Section 3.2), it is not likely that our algorithm for the closure chain queries can be generalized for a larger class of queries while retaining its polynomial complexity for an optimal solution.
- A necessary and sufficient condition to eliminate a single unnecessary join in a tree query is provided, while essentially only a sufficient condition was provided in the literature [17], [26]. The condition we provide is less restrictive in eliminating a single join.
- The important operations in semantic query optimization are: the detection of a contradiction, the elimination of as many unnecessary joins as possi-

Manuscript received July 27, 1990; revised June 6, 1991. This work was supported in part by the National Science Foundation under Grant IRI-8901789, by the Office of Naval Technology (Navy Ocean System Center), and by the Florida High Technology and Industrial Council.

W. Sun was with the Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, Chicago, IL 60680. He is now with the School of Computer Science, Florida International University—The State University of Florida at Miami, University Park, Miami, FL 33199.

C. Yu is with the Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, Chicago, IL 60680.

IEEE Log Number 9211309.

ble, and the addition/elimination of beneficial/nonbeneficial redundant restrictions. A systematic approach of how these actions are integrated has not received sufficient investigation. We provide a procedure to integrate these actions and explain why our method is reasonable.

The rest of the paper is organized as follows. In Section II, assumptions are stated and semantic integrity constraints are formally defined. Some of our earlier results are also briefly reviewed. Section III addresses semantic query optimization. Conditions that a single unnecessary join in a tree query can be eliminated are discussed first. In Section III-B, the complexity classification of the semantic query optimization problems is presented and $\mathfrak{NP}$-Completeness for semantic query optimization of certain types of queries are proved. An effective algorithm to semantically optimize tree queries is provided in Section III-C. In Section III-D, we show that when the above algorithm for optimizing tree queries is applied to a closure chain query, an optimal solution is obtained in polynomial time complexity. Finally, in Section III-V, we show how *beneficial* restrictions are generated, and *nonbeneficial* restrictions are eliminated. Section IV briefly discusses the generalization of cost models, indicates future research directions, and concludes this paper.

II. Related Works and Terminology

This section provides some background information. Section II-A states assumptions and defines semantic integrity constraints. In Section II-B, join closures, query graphs, and closure query graphs are introduced. Section II-C briefly reviews some of our earlier results, which will be used in subsequent discussions.

A. Semantic Integrity Constraints and Assumptions

In this section, we first state our assumptions, then formally define *semantic integrity constraints*. The relational model [4] is assumed. A query $Q[T:q]$ has a target (output) consisting of an attribute list T, and a conjunctive qualification $q \equiv p_1 \wedge p_2 \wedge \cdots \wedge p_n$, or denoted as a set $\{p_1, p_2, \cdots, p_n\}$, where each $p_{k, 1 \le k \le n}$ is a restriction (selection) or an equi-join predicate. An equi-join predicate is of the form $(R.X = S.Y)$, namely, relations R and S are joined on attributes X and Y, respectively; a restriction (selection) is of the form $(R.X \ \mathbf{op} \ c)$, where $\mathbf{op} \in \{>, \geq, <, \leq, \neq, =\}$ and c is a constant in the domain of $R.X$. Without loss of generality, restrictions in negated form are not allowed, because negated restrictions can always be put in nonnegated form, namely $\neg(R.X\ [>, \geq, <, \leq, =, \neq]\ c)$ is equivalent to $(R.X\ [\leq, <, \geq, >, \neq, =]\ c)$. In the following discussion, let $JP(q)$ and $Res(q)$ be the set of *cross-product free* join predicates and the set of restrictions in q, respectively. q or $JP(q)$ is *cross product free* if joins specified in q do not yield a cross (Cartesian) product of relations. $UR(q)$ is called the *underlying relation of* q, which is the relation obtained by performing all joins specified in q [i.e., $JP(q)$] and all restrictions specified in q.

Definition 1: A **semantic integrity constraint** (briefly, a constraint) is a logical implication $JC \wedge A \Rightarrow B$, where JC is a set of cross-product free equi-join predicates in conjunction, and A and B are restrictions in conjunction. A and B must be evaluable on $UR(JC)$, which is the relation obtained by performing all join operations specified by JC. The interpretation is: after $UR(JC)$ is formed, then whenever a tuple in $UR(JC)$ satisfies the restriction(s) A, it must satisfy the restriction(s) B. JC, B and A are called the **join precondition**, the **right-hand side (RHS)** and the **left-hand side (LHS)** of the constraint, respectively. Join predicates are not allowed to appear on the right-hand side of a constraint by this interpretation. JC and/or A can be null. If $JC = \varnothing$, the underlying relation is the relation on which restrictions A or B are defined (since no cross product is allowed, A and B must be evaluable on the same base relation); if $A = \varnothing$, then the evaluation of B on the underlying relation becomes unconditionally true (i.e., the *RHS* should always be enforced). □

Definition 2: A constraint $JC \wedge A \Rightarrow B$ is **normalized** if the *RHS* B is a single restriction only. □

Since the RHS of a constraint is a set of restrictions in conjunction, thus the constraint can be equivalently decomposed into a number of normalized constraints. For example, a constraint of the form $(LHS \Rightarrow P_1 \wedge P_2 \wedge \cdots \wedge P_m)$ is equivalent to m separate *normalized* constraints $(LHS \Rightarrow P_i,\ 1 \le i \le m)$. It should be noted that if the RHS of a constraint is a disjunction, the constraint can also be put into normalized form. For example, $JC \wedge A \Rightarrow B_1 \vee B_2$ is equivalent to either $JC \wedge A \wedge B_1 \Rightarrow B_2$ or $JC \wedge A \wedge \neg B_2 \Rightarrow B_1$.

The following is a sample university database consisting of three relations:

student(s) [s.ss#, s.GPA, s.Name, s.Status],
student-course(sc) [sc.ss#, sc.c#, sc.Mark],
course(c) [c.c#, c.Level, c.Instructor, c.Room, c.Time].

where attributes c#, c.Level, and s.Status represent the identity of the *offering (call number)* of a course, the *degree* of difficulty of a course and the *status* of a student (whose domain consists of "Graduate" or "Undergraduate"). Other attributes are self-explanatory. In the above database, there also exist *referential integrity constraints* [5], [31] from sc.c# to c.c# and from sc.ss# to s.ss#. $R.X \sqsubseteq S.Y$ is said to be a referential integrity constraint from $R.X$ to $S.Y$ if every value appears under $R.X$ also appears under $S.Y$ for any database instance. The following are four sample constraints that exist in the above database.

1) The semantics: "The lowest acceptable GPA for a student is 2.0."
   The constraint: $\Rightarrow (s.GPA \geq 2.0)$.
2) The semantics: "The lowest acceptable GPA for a graduate student is 3.0."
   The constraint: $s.Status = \text{"}Grad\text{"} \Rightarrow s.GPA \geq 3.0$.

3) The semantics: "Course at the 500 level or above are only for graduate students."
The constraint: $s.ss\# = sc.ss\# \wedge sc.c\# = c.c\# \wedge c.Level \geq 500 \Rightarrow s.Status = \text{"}Grad.\text{"}$
4) The semantics: "Tom only teaches courses of 600 level."
The constraint: $c.Instructor = \text{"}Tom\text{"} \Rightarrow c.Level = 600$.

Constraints can be classified into *domain assertions (the above constraint 1), restrictions within a relation (the above constraints 2 and 4), and restrictions across relations (the above constraint 3)*. Semantic integrity constraints are generally application-dependent, and are usually specified by database administrators or application users. However, an automatic scheme to acquire constraints is proposed in [46], where the maintenance of a set of constraints (i.e., nonredundancy and consistency) is also addressed.

B. Query Graphs and the Closure of Query Graphs

Two different sets of join predicates may be equivalent in the sense that they can yield the same join operation. For example, $(R_1.X = R_2.Y \wedge R_2.Y = R_3.Z)$ is equivalent to $(R_1.X = R_3.Z \wedge R_2.Y = R_3.Z)$. It is obvious that equi-join induces an equivalence relationship. Therefore, any set of equi-join attributes in a set of join predicates P can be partitioned by the equi-join relationship into equivalence classes.

Definition 3: The **join closure**, $JP^*(P)$, of a set of join predicates P is the partition of all join attributes involved in P into **equivalence classes** induced by the equi-join relation. $JP^*(P) = \{Equi\_Set_1, Equi\_Set_2, \cdots, Equi\_Set_k\}$, where each $Equi\_Set_i$, $1 \leq i \leq k$, is an equivalence class. In other words, $\forall R.X, \forall S.Y \in Equi\_Set_i$ for some i, $1 \leq i \leq k$, $(R.X = S.Y)$ is in P or implied by P. If $R.X \in Equi\_Set_i$, $S.Y \in Equi\_Set_j$ and $i \neq j$, then $(R.X = S.Y)$ is neither in P nor implied by P. □

The following lemma is direct.

Lemma 1: Two sets of join predicates JP_1 and JP_2 specify the same join operation **if and only if** $JP^*(JP_1) = JP^*(JP_2)$. □

Example 1: $JP^*(\{R_1.A = R_2.B, R_2.B = R_3.C, R_1.E = R_4.F\}) = \{\{R_1.A, R_2.B, R_3.C\}, \{R_1.E, R_4.F\}\}$. □

Example 2: Suppose $P_1 = \{R_1.X = R_2.Y, R_2.Y = R_3.Z\}$ and $P_2 = \{R_1.X = R_3.Z, R_2.Y = R_3.Z\}$. $JP^*(P_1) = JP^*(P_2) = \{\{R_1.X, R_2.Y, R_3.Z\}\}$. Thus, P_1 and P_2 specify the same join operation, though in different forms. □

Definition 4: $G = (V, E)$ is a **query graph** of a query qualification q, where $V = \{R|$ relation R is referenced in $q\}$, and $E = \{(X, Y)|\exists A, \exists B, (X.A = Y.B) \in JP(q)\}$. E specifies the join condition of q.

$G^* = (V^*, E)$ is the **closure of a query graph** G, where $V^* = V$, and $E^* = \{(X, Y)|\exists A, \exists B, \exists Equi\_Set \in JP^*(q), \{X.A, Y.B\} \subseteq Equi\_Set\}$. E^* gives all join conditions, either explicitly specified in q or implied by the transitivity of equality of equi-join predicates in q.

If G is a chain (star, tree), the corresponding query is called a **general chain (star, tree) query**, or simply a **chain (star, tree) query**. If G^* is a chain (star, tree), then the corresponding query is called a **closure chain (closure star, closure tree) query**. Clearly, closure chain, closure star, and closure tree queries are restricted types of chain, star, and tree queries, respectively. □

Example 3: The following are three samples queries, their query graphs and their closure query graphs. Q_1 is a chain and a closure chain. Q_2 is a chain, but not a closure chain. Q_3 is a star and a closure star rooted at R_1.

$$Q_1[R_1.A_4 : R_1.A_1 = R_2.A_1 \wedge R_2.A_2 = R_3.A_2 \wedge R_3.A_3 = R_4.A_3]$$

$$Q_2[R_1.A_4 : R_1.A_1 = R_2.A_1 \wedge R_2.A_1 = R_3.A_1 \wedge R_3.A_2 = R_4.A_2]$$

$$Q_3[R_1.A_4 : R_1.A_1 = R_2.A_1 \wedge R_1.A_2 = R_3.A_2 \wedge R_1.A_3 = R_4.A_3].$$

Lemma 2: A query $Q[T:q]$ is a closure chain query, or a closure star query, or a closure tree query **if and only if** all equi-joins in Q are specified on distinct attributes, and the query graph is a chain, a star, or a tree. In other words, a join attribute can only appear at most once in the join predicates of Q.

Proof: Sufficiency: Assume all equi-joint attributes are distinct in query qualification whose query graph is a chain (a star, or a tree). By hypothesis, no edge can be added in the closure graph, and therefore, its closure query graph is also a chain (a star, or a tree).

Necessity: Suppose a join attribute $R.X$ appears in two equi-join predicates, say $(R.X = S.Y)$ and $(R.X = T.Z)$. Thus, $(S.Y = T.Z)$ is implied, and a cycle containing nodes R, S, and T is formed in the closure graph, which implies that the closure graph can neither be a chain, nor a star, nor a tree, where no cycle is allowed. It is also clear that if the query graph is not a chain (a star, or a tree), then the closure of the query graph can not be a chain (a star, or a tree) either. □

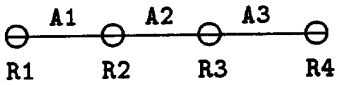
Example 4: In Example 3, equi-joins in Q_1 and Q_3 are specified on distinct attributes, thus they are also closure chain and closure star queries. Two equi-joins in Q_2 involve A_1, thus Q_2 is not a closure chain query by the above lemma. □

C. The Restriction Closure

In this section, we briefly review some of our earlier results [46]. One of the essential problems in semantic query optimization is

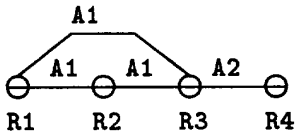
> **Given** a set of restrictions R, a set of equi-join predicates P, and a set of constraints $\mathbb{S}$.
> **Find** all possible restrictions implied or deduced from R under P and $\mathbb{S}$.

The set of all these restrictions are called the *restriction closure* of R under P and $\mathbb{S}$, denoted as $C^*(R, P, \mathbb{S})$. In



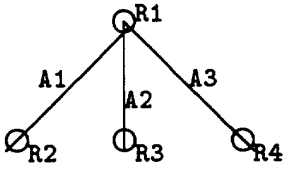
Q1 & Q1*

Fig. 1. Closure chain query.



Q2 & Q2*

Fig. 2. Chain, but not closure chain query.



Q3 & Q3*

Fig. 3. Closure star query.

[46], an efficient procedure is provided, which will return $C^*(R, P, \mathbb{S})$ if there is no contradiction in R under P and $\mathbb{S}$; otherwise, return FALSE. The time complexity of the closure algorithm is shown to be $O(|P| \log |P| + |R| \log |R| + |S_1| \log (|R| + |P| * |S_1|) + L)$, where S_1 is the set of rules in $\mathbb{S}$ whose join preconditions are satisfied by P, L is the number of restrictions in LHS's of all constrains in $\mathbb{S}$, and $|X|$ is the cardinality of a set X. The procedure TESTING(R, P, $\mathbb{S}$, x) will return true if a set of restrictions x is redundant with respect to (wrt) another set of restrictions R under P and $\mathbb{S}$. The algorithm TESTING is essentially the closure computation of R under P and $\mathbb{S}$.

III. Semantic Query Optimization

Queries can be optimized by using constraints. The following are three sample queries on our university database that can be semantically optimized.

Example 5: The query is "Find all the undergraduate students (by social security numbers ss#) who take courses at 600 level or above."

$$Q_1[s.ss\# : s.ss\# = sc.ss\# \wedge sc.c\# = c.c\# \wedge s.Status = \text{"}Undergraduate\text{"} \wedge c.Level \geq 600].$$

Suppose the following is a constraint in our university database (See Section II-A):

$$(s.ss\# = sc.ss\# \wedge sc.c\# = c.c\# \wedge c.Level \geq 500 \Rightarrow s.Status = \text{"}Grad\text{"}).$$

Then, $s.Status = \text{"}Grad\text{"}$ is deduced, which, when added to the qualification, leads to a contradiction. Thus, the answer to the query is null without physically accessing the database. □

Example 6: The query is "Find the names of the students who take courses at 600 level or above."

$$Q_2[s.Name : s.ss\# = sc.ss\# \; sc.c\# = c.c\# \wedge c.Level > 600].$$

By the given constraint (see Section II-A)

$$(s.ss\# = sc.ss\# \wedge sc.c\# = c.c\# \wedge c.Level \geq 500 \Rightarrow s.Status = \text{"}Grad\text{"})$$

the query can be semantically equivalently transformed into

$$Q_2'[s.Name: s.ss\# = sc.ss\# \wedge sc.c\# = c.c\# \wedge s.Status = \text{"}Grad\text{"} \wedge c.Level > 600].$$

Relation s can be reduced by applying ($s.Status$ = "*Grad*") before executing the join in Q_2', so that the cost of the join involving relation s may be reduced. We call the restriction ($s.Status$ = "*Grad*") a *beneficial* restriction (see Section III-E for details). This is also referred to as *scan* reduction [3], [17], [26], [27]. Furthermore, if $s.Status$ is an indexed attribute, then this fast access path can be used to speed up accessing the relations s. □

Example 7: The query is "Find the graduate students by social security numbers who take Tom's courses."

$$Q_3[s.ss\# : s.ss\# = sc.ss\# \wedge sc.c\# = c.c\# \wedge c.Instructor = \text{"}Tom\text{"} \wedge s.Status = \text{"}Grad\text{"}].$$

Assume that we know: *Tom only teaches EECS 600* (see Section 2.1), then Q_3 can be equivalently transformed into (by adding the RHS):

$$Q_3'[s.ss\# : s.ss\# = sc.ss\# \wedge sc.c\# = c.c\# \wedge c.Instructor = \text{"}Tom\text{"} \wedge s.Status = \text{"}Grad\text{"} \wedge c.Level = 600].$$

By shifting the target via the equi-join without affecting the final answer, we obtain

$$Q_3''[sc.ss\# : s.ss\# = sc.ss\# \wedge sc.c\# = c.c\# \wedge c.Instructor = \text{"}Tom\text{"} \wedge s.Status = \text{"}Grad\text{"} \wedge c.Level = 600].$$

Assume we have the knowledge that courses at 500 level or above are only for graduate students, i.e., ($s.ss\# = sc.ss\# \wedge sc.c\# = c.c\# \wedge c.Level \geq 500 \Rightarrow s.Status =$ "*Grad*"), the following equivalent transformations can be easily obtained:

$$Q_3'''[sc.ss\# : s.ss\# = sc.ss\# \wedge sc.c\# = c.c\# \wedge c.Instructor = \text{"}Tom\text{"} \wedge c.Level = 600].$$

$Q_3''''[sc.ss\#: sc.c\# = c.c\# \wedge c.Instructor$

$= \text{``}Tom\text{''} \wedge c.Level = 600]$.

Q_3'''' is semantically equivalent to Q_3''' since the join ($s.ss\# = sc.ss\#$) is unnecessary due to the referential integrity constraint ($s.ss\# \supseteq sc.ss\#$) [5], [31] (that is, values under the attribute *ss#* of the base relation **sc** are contained in or equal to the set of values under *ss#* of the base relation **s**). Q_3'''' only involves one join, while the original query Q involves two joins.

Furthermore, since ($c.Instructor = \text{``}Tom\text{''} \Rightarrow c.Level = 600$), Q_3'''' is semantically equivalent to Q_3'''''.

$Q_3'''''[sc.ss\#: sc.c\# = c.c\# \wedge c.Instructor = \text{``}Tom\text{''}]$.

Clearly, it is less costly to evaluate Q_3''''' than to evaluate Q_3'''' if the attribute *c.Level* is not a join attribute nor an indexed attribute. The restriction (selection) ($c.Level = 600$) is said to be non-beneficial (see Section 3.5 for detail). Non-beneficial restrictions, if redundant, should be eliminated. □

Basically there are two different types of transformations:

1) the transformation which eliminates unnecessary joins
2) the transformation that adds/eliminates beneficial/nonbeneficial redundant restrictions.

It is known that the cost of performing joins is rather expensive, and usually is more significant than the cost involved in performing other operations like selections and projections in evaluating a query. Therefore, it is very desirable that as many unnecessary joins as possible are eliminated. Based upon this assumption, the following is a sketch of our approach for semantically optimizing a user's query $Q[T: q]$ under the given semantic knowledge $\mathcal{S}$.

Step 1: Compute the *restriction closure* of q under $\mathcal{S}$, and detect a contradiction in q under $\mathcal{S}$.

Step 2: Eliminate as many unnecessary joins as possible.

Step 3: Eliminate all redundant nonbeneficial restrictions.

The top priority of our method is to detect a contradiction if there is any, since a contradiction implies a null answer to the query. The restriction closure helps to detect such a contradiction. Then, we will try to eliminate as many unnecessary joins as possible. Finally, we will try to eliminate all redundant nonbeneficial restrictions. It should be noted that the restriction closure provides all implied restrictions including all beneficial restrictions.

A. Eliminating an Unnecessary Join

In this section, we present a necessary and sufficient condition of eliminating a single unnecessary join in a tree query. Example 7 given above illustrates a typical situation where an unnecessary join can be eliminated. In the following, we first define the semantic equivalence between two queries $Q[T: q]$ and $Q'[T': q']$. The equivalence in the definition does not require that Q and Q' yield the same underlying relation (clearly, if the underlying relations for both Q and Q' are the same, then the two identical underlying relations will yield the same answer for the same target T). It is possible that answers to two queries may be the same although the two targets are not identical (see the transformation from Q_3' to Q_3'' in Example 7). Since joins may be eliminated, as shown in Example 7, two query qualifications may yield different schemes of their underlying relations. In defining the equivalence, the above two factors should be taken into consideration. It is sufficient that both queries provide the same answer.

Definition 5: Let $Q[T: q]$ be a query, $\Pi_A B$ be the projection of the relation (or an underlying relation) B on attribute(s) A, and $Scheme(q)$ be the set of attributes involved in the relations referenced by the qualification q. $Q'[T': q']$, $T' \subseteq Scheme(q')$, which involves a subset of relations referenced in Q is semantically equivalent to $Q[T: q]$ for every database instance satisfying the given set of constraints, denoted by ($Q \sim^T Q'$), if

1) $\Pi_{scheme(q')} UR(q) = UR(q')$, and
2) $\forall R.X \in T$, $\exists S.Y \in T'$, $\exists Equi\_Set \in JP^*[JP(q)]$, $R.X$ and $S.Y \in Equi\_Set$. □

Condition 2) says that target specified on equi-join attributes can be *shifted* via equi-joins (see the transformation from Q_3' to Q_3'' in Example 7) such that if $R.X \in T$, then it is equivalent to replace it by any $S.Y$ that is in the same equivalence class containing $R.X$. Condition 1) says that the projection of the underlying relation of q on the scheme of q' is the underlying relation of q'. Therefore, projecting on T', which is a subset of $Scheme(q')$, will provide the same answer under both underlying relations $\Pi_{scheme(q')} UR(q)$ and $UR(q')$. This condition allows that the two query qualifications may involve two different number of joins (thus possibly different schemes of their underlying relations). Conditions 1) and 2) will ensure that Q and Q' yield the same answer.

Example 8: In Example 7, Q_3 and Q_3'''' are semantically equivalent by the above definition. T is *s.ss#* and T' is *sc.ss#*. q' is ($sc.c\# = c.c\# \wedge c.Instructor = \text{``}Tom\text{''}$), which only involves one join while the original q involves two joins. □

In specifying conditions to eliminate an unnecessary join, referential integrity constraints are needed. However, it is not necessary that a referential integrity constraint be explicitly specified. For example, if $R.X \sqsubseteq S.Y$ and $S.Y \sqsubseteq T.Z$ are present, then $R.X \sqsubseteq T.Z$ is implied. It can be determined whether a referential integrity is implied by the original user's specification by drawing a directed graph $G_{ref} = (V_{ref}, E_{ref})$, where $V_{ref} = \{X.A | X.A$ is an attribute referenced in the original specification of referential integrity constraints$\}$. If $R.X \sqsubseteq S.Y$ or $R.X \sqsubseteq S.Y$ are initially specified, then we draw a directed edge from $R.X$ to $S.Y$, i.e., $(R.X, S.Y) \in E_{ref}$. Then, we compute the transitive closure $G_{ref}^* = (V_{ref}^*, E_{ref}^*)$ of G_{ref}, that

is, initially $G^*_{ref} = G_{ref}$, $\forall R.X, S.Y, T.Z \in V^*_{ref}$, if $(R.X, S.Y)$, $(S.Y, T.Z) \in E^*_{ref}$, then $(R.X, T.Z) \in E^*_{ref}$. Clearly, $(R.X \sqsubseteq S.Y)$ is implied **if and only if** $(R.X, S.Y) \in E^*_{ref}$. In the following, when we refer to a referential integrity, the above testing is always implied.

Let q^c be the restriction closure of $Res(q)$ under $JP(q)$ and $\mathcal{S}$, i.e., $q^c = C^*(Res(q), JP(q), \mathcal{S})$ (see Section II-C), and $Attributes(R)$ be the set of attributes defined on the relation R. The following is a necessary and sufficient condition that a single unnecessary join $R_0 \bowtie R_1$ can be eliminated, where R_0 only joins with the relation R_1 in a tree graph (namely, R_0 is a leaf node in the graph). Therefore, eliminating the join implies the removal of R_0 from the graph.

Proposition 1: Let R_1 be the only relation that R_0 joins in a tree query $Q[T:q]$. $Q[T:q] \sim^T Q'[T': q^c - Res_{R_0}(q^c) - \{R_0.A = R_1.A\}]$, **if and only if**

1) $R_0.A \sqsupseteq R_1.A$,
2) $Res_{R_0}(q^c) = \varnothing$ or $Res_{R_0}(q^c)$ are redundant wrt $[q^c - Res_{R_0}(q^c)]$ under $\mathcal{S}$, and
3) $T \cap Attributes(R_0) \subseteq \{R_0.A\}$,

where T' is T if $R_0.A \notin T$; otherwise T' is $(T - \{R_0.A\}) \cup \{R_1.A\}$, and $Res_{R_0}(q^c)$ is the set of restrictions of specified on R_0 from q^c.

Proof: Sufficiency: Considering condition 3), $T \cap Attributes(R_0) \subseteq \{R_0.A\}$ implies $T \cap Attributes(R_0) = \varnothing$ or $T \cap Attributes(R_0) = \{R_0.A\}$. For the latter case, the target attribute $R_0.A$ in T can be *shifted* to $R_1.A$ by the equi-join, namely, T is equivalent to $(T - \{R_0.A\} \cup \{R_1.A\})$. Thus, R_0 is target free. If condition 2) is satisfied, then $Res_{R_0}(q^c)$ can be equivalently eliminated. Hence, there is no restriction specified on R_0. Consequently, by condition 1), this join is clearly an unnecessary one and can be removed.

Necessity: That is, if the answers are identical by removing the join, then the above three conditions are true for any instance of database that satisfies the set of constraints. First, consider condition 3). Suppose $Q \sim^T Q'$, but there is a target attribute B in R_0 and $B \neq A$. It is easy to see that there is a database instance satisfying all constraints such that $Q[T:q]$ outputs some value $b \in B$, but the other query Q' does not reference B at all. Now consider the first condition. Suppose $Q \sim^T Q'$, but $R_0.A \not\sqsupseteq R_1.A$. We now show that there exists a database instance satisfying the set of constraints, but the answers to the queries Q and Q' are not the same. Consider the case that the qualification q consists of one join only, $(R_0.A = R_1.A)$. The case involving more relations and joins can be analyzed in a similar manner. By condition 3), the target T is defined on R_1 after possible target shifting. Since $R_0.A \not\sqsupseteq R_1.A$, there exists a tuple t in R_1 such that $t[A] \notin R_0.A$. We now construct a database instance consisting of R_0 and R'_1, where R'_1 only consists of one tuple t. Since t is a tuple in R_1, and R_1 and R_0 satisfy the constraints, therefore, the new database instance also satisfies all the constraints. However, the answers to the two queries (the one with the join and the one without the join) are clearly different. The answer to the query with the join will yield a null answer due to $t[A] \notin R_0.A$ where A is the equi-join attribute, while the answer to the query without the join will give a nonnull answer. Hence, it is concluded that $R_0.A \sqsupseteq R_1.A$ must be true. In a similar manner, it can be proved that the condition 2) will hold if $Q \sim^T Q'$ due to the completeness of the restriction closure computation. □

In order to eliminate a join, the redundancy of restrictions on the relation to be eliminated should be tested against the closure of q (i.e., q^c) instead of q itself. The essential point is that there may exist certain restrictions on relations other than R_0 that are semantically equivalent to those specified on R_0 in the closure (however these restrictions may not be specified in q), such that restrictions on R_0 may become redundant (and hence can be eliminated) wrt these restrictions. This can be illustrated by the following example.

Example 9: Suppose the query is $Q[R_1.C: R_0.A = R_1.A \wedge R_1.B = R_2B \wedge Restriction_{R_0} \wedge Restriction_{R_1}]$, and we have two constraints:

$$R_0.A = R_1.A \wedge R_1.B = R_2.B \wedge Restriction_{R_0} \wedge Restriction_{R_1} \Rightarrow Restriction_{R_2}$$

$$R_0.A = R_1.A \wedge R_1.B = R_2.B \wedge Restriction_{R_2} \Rightarrow Restriction_{R_0}$$

and $R_0.A \sqsupseteq R_1.A$, where $Restriction_R$ is a restriction specified on R. By computing q^c, $Restriction_{R_2}$ is added to the qualification by the first constraint. And in turn, this added restriction makes the restriction on R_0, $Restriction_{R_0}$, redundant. Thus, Q is semantically transformed into the following: $Q[R_1.C: R_0.A = R_1.A \wedge R_1.B = R_2.B \wedge Restriction_{R_2} \wedge Restriction_{R_1}]$. In turn, the join $(R_0.A = R_1.A)$ can be removed. □

Example 9 shows why we need to compute the restriction closure first. By computing the restriction closure in our approach, as many restrictions as possible are generated so that the redundancy testing of restrictions needed for a join elimination and the detection of a contradiction can be conducted to a maximum degree. For example, the restriction on R_0 is not redundant wrt the restrictions from q on other relations, namely, $Restriction_{R_0}$ is not redundant wrt $(q - Restriction_{R_0})$. However, $Restriction_{R_0}$ is redundant wrt $[q^c - Res_{R_0}(q^c)]$. It should be noted that our result is a necessary and sufficient condition to eliminate a single unnecessary join; while there is neither a statement nor a proof for a necessary condition of eliminating an unnecessary join in the literature.

The elimination of as many unnecessary joins as possible has not been systematically addressed in the literature. This will be discussed in following sections.

B. NP-Completeness for Semantic Query Optimization of Chain and Closure Star Queries

In this section, we prove that the semantic query optimization for general chain queries and closure star queries

are $\mathcal{NP}$-Complete in terms of eliminating unnecessary joins. Therefore, the problems of semantic query optimization for general star queries, general tree queries, closure tree queries, and general queries are $\mathcal{NP}$-Complete.

Let $Q[T:q]$ be a user's query, $Q'[T':q']$ be a semantically equivalent query of Q, and RS' be the set of relations referenced by Q'. RS' is called a **relation cover (RC)** of Q if $Q' \sim^T Q$.

Definition 6: The semantic query optimization problem (SQO) is defined as follows:

> **Given** a query $Q[T:q]$, a set of referential integrity constraints [5], a set of constraints $\mathbb{S}$, and an integer k, $k > 0$.
> **Is there** a $Q'[T':q']$, $Q' \sim^T Q$, such that $|RS'| \leq k$, where RS' is the set of relations referenced in q'. □

We will prove that the SQO problems for general chain queries or closure star queries are $\mathcal{NP}$-Complete. The *vertex cover* (VC) problem of a connected graph $G = (V, E)$ is reduced to SQO problems. The vertex cover problem was shown to be $\mathcal{NP}$-Complete [8]:

> **Given** A connected graph $G = (V, E)$, an integer k, $k > 0$.
> **Is there** a subset V' of V, $|V'| \leq k$, such that $\forall(a, b) \in E$, $a \in V'$ or $b \in V'$?

Proposition 2: The SQO for general chain queries is $\mathcal{NP}$-Complete.

Proof: Given a connected graph $G = (V, E)$, $|V| = n$, we construct an instance of a database (with relations, rules, referential integrity constraints) and a query as follows:

For each $v_i \in V$, $0 \leq i \leq n-1$, there is a relation R_i. The join specification $JOIN$ of the query is that all relations are joined on the same attribute X:

$$JOIN \equiv (R_0.X = R_1.X) \wedge (R_1.X = R_2.X) \wedge \cdots \wedge (R_{n-2}.X = R_{n-1}.X).$$

Clearly, its query graph is a chain, but the closure of its query graph is a *complete graph of n vertices*, i.e., the join closure is $JP^*(JOIN) = \{\{R_0.X, R_1.X, \cdots, R_{n-1}.X\}\}$. Each relation R_i has n^2 tuples, and the values under the attribute $R_i.X$ are distinct and are $\{1, 2, 3, \cdots, n^2\}$. The query is

$$Q[T:q] = Q[R_0.X: JOIN \wedge (\text{all RHS's of constraints in } \mathbb{S})]$$

where the constraints are constructed as follows: for each $(v_i, v_j) \in E$, $i < j$, two constraints are formed:

$$RULE_{ij}: JOIN \wedge REST(i, j) \Rightarrow REST(j, i), \text{ and}$$

$$RULE_{ji}: JOIN \wedge REST(j, i) \Rightarrow REST(i, j),$$

where $REST(Z, Y)$ is a restriction specified on an attribute of R_Z other than $R_Z.X$. For $Z < Y$, $REST(Z, Y)$ will disqualify precisely the tuple of R_Z whose value under $R_Z.X$ is $(Z * n + Y)$. For $Z > Y$, $REST(Z, Y)$ will disqualify precisely the tuple of R_Z whose value under $R_Z.X$ is $(Y * n + Z)$. Clearly, this database instance satisfies these constraints.

Now we prove the $\mathcal{NP}$-Completeness by showing that G has a vertex cover V' of size k if and only if Q has a semantically equivalent query Q' having a relation cover RS' of size k.

$\Rightarrow$ Let V' be a VC of G of size k. Consider any vertex v_i, $v_i \in V - V'$. We now show that R_i (corresponding to v_i) can be eliminated, i.e., $R_i \notin RS'$.

Let $Adj(v_i) = \{v | (v_i, v) \in E\}$ and $v_j \in Adj(v_i)$. Since $v_i \notin V'$ and V' is a VC, $v_j \in V'$. This implies $Adj(v_i) \subseteq V'$. Clearly,

$$\begin{aligned} q &\equiv JOIN \wedge (\text{all RHSs of constraints in } \mathbb{S}) \\ &\equiv JOIN \wedge (\wedge_{\forall(v_x, v_y) \in E} REST(x, y)) \\ &\equiv JOIN \wedge (\wedge_{\forall v_j \in Adj(v_i)} REST(i, j)) \wedge (\wedge_{\forall x \neq i, (v_x, v_y) \in E} \cdot REST(x, y)) \\ &\sim^T JOIN \wedge B. \end{aligned}$$

where B is $\wedge_{\forall x \neq i, (v_x, v_y) \in E} REST(x, y)$. The last step is true because $\forall v_j \in Adj(v_i)$, $RULE_{ji}$: $JOIN \wedge REST(j, i) \Rightarrow REST(i, j) \in \mathbb{S}$, and $REST(j, i) \in B$, which implies $REST(i, j)$. Thus, all restrictions on R_i can be eliminated.

Consider the target $T \equiv R_0.X$, if $R_0 \notin RS'$, then T can be modified to be $R_y.X$ for any $R_y \in RS'$. The referential integrity constraint in Proposition 1 is also satisfied. Thus, all the three conditions in Proposition 1 are satisfied. Hence, R_i can be eliminated. In conclusion, RS' is a RC of Q of size k.

$\Leftarrow$ Let RS' be an RC of Q of size k. We now establish that V' corresponding to RS' is a VC of G of size k.

Let $R_i \notin RS'$. By correspondence, $v_i \notin V'$. Consider any edge in E incident on v_i. Let it be (v_i, v_j). We now claim that $R_j \in RS'$, which implies $v_j \in V'$. Hence V' is a VC of G. Suppose $R_j \notin RS'$. We will prove a contradiction that RS' is not a RC of Q. Since $(v_i, v_j) \in E$, by the construction of $\mathbb{S}$ and Q, $RULE_{ij}$, $RULE_{ji} \in \mathbb{S}$, $REST(i, j) \in q$ and $REST(j, i) \in q$. Assume $i < j$ (the situation that $j < i$ can be analyzed similarly). $REST(i, j)/REST(j, i)$ will precisely disqualify the tuple of R_i/R_j with the value $(i * n + j)$. However, since $R_i \notin RS'$ and $R_j \notin RS'$, and $REST(i, j)$ and $REST(j, i)$ are the only restrictions that will disqualify the tuple with value $(i * n + j)$ under X, therefore, in $UR(q')$ there does exist a tuple such that its value under X is $(i * n + j)$. This contradicts that RS' is an RC of Q (i.e., $Q' \sim^T Q$). This establishes that $v_j \in V'$. □

Closure chain, closure star, and closure tree queries are restricted types of queries as discussed in Section II-B. However, the following result shows that for a closure star query, the SQO problem is $\mathcal{NP}$-Complete. In Section III-D, an efficient yet optimal algorithm is provided for closure chain queries.

Proposition 3: The SQO for a closure star query is $\mathcal{NP}$-Complete.

Proof: Given a connected graph $G = (V, E)$, $|V| = n$, we construct an instance of a database and a query as follows. Each $v_i \in V$, $0 \le i \le n - 1$, is assigned a relation R_i. The join specification *JOIN* of the query is

$$JOIN \equiv (R_n.X_0 = R_0.X_0) \wedge (R_n.X_1 = R_1.X_1) \wedge \cdots \wedge (R_n.X_{n-1} = R_{n-1}.X_{n-1})$$

and $Q[T:q]$ is $Q[R_n.X_n : JOIN \wedge$ (all RHS's of constraints in $\mathcal{S}$)], where $X_0, X_1, \cdots, X_{n-1}, X_n$ are distinct attributes on R_n. Clearly, this is a closure star query. There are $n + 1$ relations referenced in Q. Each relation R_i, $0 \le i \le n$, has n^2 tuples, and the values under the attribute $R_i.X_i$, $R_n.X_i$, $0 \le i \le n$ are distinct and are $\{1, 2, 3, \cdots, n^2\}$. For each $(v_i, v_j) \in E$, $i < j$, two constraints $RULE_{ij}$ and $RULE_{ji}$ are formed, which are exactly the same as those constraints constructed in the proof of Proposition 2. It can be easily verified that this database instance satisfies these constraints.

Now we can prove the $\mathfrak{NP}$-Completeness by showing that G with n vertices has a vertex cover V' of size k if and only if Q involving $n + 1$ relations has a semantically equivalent query Q' having a relation cover $RS' \cup \{R_n\}$ of size $k + 1$. This proof is similar to that in Proposition 2 after the above instance of database and the set of constraints have been constructed. □

Based on the above two propositions, the following lemma is direct.

Corollary 1: The SQO for a general star query and a closure tree query is $\mathfrak{NP}$-Complete. □

Now, the time complexity of an SQO algorithm for a closure chain query remains unknown. In the next section, an effective and systematic algorithm is proposed to semantically optimize tree queries. This algorithm has an important property, when applied to a closure chain query, an optimal solution can be obtained in polynomial time. Due to the $\mathfrak{NP}$-Completeness of semantically optimizing general chain queries and closure star queries, it is unlikely that an optimal algorithm with polynomial time complexity can be found for a class of queries which is a superclass of closure chain queries. In this sense, our result is unlikely to be incomplete.

C. Semantic Query Optimization of Tree Queries

In this section, we will provide an effective algorithm to semantically optimize general tree queries. A query is a *tree query* if it is equivalent to a query with a tree query graph [44]. Tree queries are known to be a very important and widely submitted class of queries. Because of the $\mathfrak{NP}$-Completeness of SQO for closure star queries and general chain queries, heuristics have to be used in a non-enumerative algorithm. The proposed algorithm has an important property that if it is applied to a closure chain query, optimality can be achieved in time complexity $O(n * m)$, where n is the number of relations involved in the chain query and m is the time complexity for a restriction closure computation. (The time complexity of a restriction closure computation was provided in Section II-C; for details please also see [46]).

1) Labels and Tree Query Graphs: Given a tree query Q, we can properly rename attributes such that the names of the *attributes* of all *relation.attributes* in an equivalence class of the join closure will be the same, and attributes in different equivalence classes have different names. Then, a *labeled query graph* $G = (V, E)$ for the three query Q can be constructed, where V is the set of relations $\{R_1, R_2, \cdots, R_n\}$, and each edge $(R_i, R_j, A) \in E$ denotes the equi-join attribute(s) A between relations R_i and R_j after proper renaming of attributes. The join attribute(s) A is also called the *label* of the edge and A is allowed to be compound (i.e., a set of equi-join attributes).

Example 10: If the query is

$$Q: [R_1.A_4: R_1.A_6 = R_2.A_7 \wedge R_1.A_1 = R_3.A_3 \wedge R_2.A_2 = R_3.A_3 \wedge R_2.A_5 = R_4.A_2].$$

Then, a *labeled query graph* of Q is

$$G = (\{R_1, R_2, R_3, R_4\}, \{(R_1, R_2, D), (R_1, R_3, B), (R_2, R_3, B). (R_2, R_4, C)\})$$

where A_1 in $R_1.A_1$, A_2 in $R_2.A_2$ and A_3 in $R_3.A_3$ are renamed to be B; A_5 in $R_2.A_5$ and A_2 in $R_4.A_2$ (which is different from $R_2.A_2$, although A_2 is used in both cases) are renamed to be C, and A_6 in R_1 and A_7 in R_2 are renamed to be D. The above graph is not a tree. However, it can be equivalently transformed into a tree: $G = (\{R_1, R_2, R_3, R_4\}, \{(R_1, R_2, BD), (R_2, R_3, B). (R_2, R_4, C)\})$. The label between R_1 and R_2 is compound, which consists of attributes B and D. □

In the following discussion, we assume that all attributes have been properly renamed, and the query graph G of a tree query Q refers to its *labeled query graph*, which is a tree [44], [45]. Given a tree graph G representing a tree query Q involving n relations, there are only $n - 1$ joins to be performed (equi-joins represented by a compound edge is considered as one join). Each edge will represent a join to be performed. Restrictions specified on a relation will be associated with that relation (node). Eliminating an edge from Q is corresponding to eliminating a join, and will result in eliminating a node (a relation along with its associated restrictions) from the tree graph. In our later discussion, query graph and query qualification, edge and join, node and relation, are used interchangeably.

A graph $G_s = (V_s, E_s)$ is a connected *subgraph* of a connected graph $G_t = (V_t, E_t)$, denoted by $G_s \subseteq G_t$, if $V_s \subseteq V_t$, $E_s \subseteq E_t$ and G_s is connected. Obviously, if G_t is a tree, then G_s must be a tree. Like the mapping between a tree graph and a query qualification, a connected tree subgraph $G_s = \{V_s, E_s\}$ is corresponding to a query qualification where fewer number of joins are involved, which can be interpreted as a qualification by joining all relations in V_s as specified in E_s, together with restrictions from the restriction closure q^c on V_s, $Res_{V_s}(q^c)$. We define

the cost for processing a qualification to be the number of joins. In other words, an important objective of semantic query optimization is to minimize the number of joins, or eliminate as many unnecessary joins as possible. In the case of tree queries, the cost of a tree (sub)graph G_s becomes $|E_s|$, i.e., $COST(G_s) = |E_s|$. It should be noted that the above cost function assumes that all joins are equally costly. However, in practical situations, different joins may be associated with different costs. For example, joining two large relations is very likely to be more costly than joining two small relations. In Section IV, we will briefly discuss the optimization problem under a more general cost model that will take relation sizes and other factors into consideration.

Example 11: In Example 7, the labeled query graph for Q_3 is $G_{Q_3} = (\{s, sc, c\}, \{(s, sc, ss\#), (sc, c, c\#)\})$. The labeled query graph for Q_3'''' is $G_{Q3}'''' = (\{sc, c\}, \{(sc, c, c\#)\})$, which is a connected subgraph of G_{Q3}. Node (relation) s together with the restriction ($s.Status$ = "*Grad*") specified on s is eliminated from G_{Q3}. □

2) Core Graph: As discussed above, eliminating an edge (join) in a tree graph G will result in eliminating a node (relation) from G. By Proposition 1, it is clear that if R can be eliminated as a result of eliminating a join, then all restrictions specified on a relation R, $Res_R(q^c)$, are redundant. Therefore, it is clear that if restrictions on a relation R are not redundant wrt all other restrictions from q^c, then R must be retained in any query graph G, that is semantically equivalent to G. It is also obvious that if there exists a target attribute on R and the target attribute is not an equi-join attribute (hence the target attribute on R cannot be "*shifted*" around), then R must be retained in any semantically equivalent query graph in order that the answer be defined. The following definition and lemma are obvious.

Definition 7: A relation R is called a **core relation** if there is a target attribute specified on R and the target attribute is not an equi-join attribute; or if restrictions specified on R from the closure $Res_R(q^c)$ are not redundant wrt the restrictions on other relations $[q^c - Res_R(q^c)]$. □

Lemma 3: If $G_s(V_s, E_s) \sim^T G$, then $CORER \subseteq V_s$, where $CORER$ is the set of core relations. □

The above lemma says that if G_s and G are semantically equivalent, then V_s must contain $CORER$. Clearly, $CORER$ can be computed in $O(n)$ restriction closure computations by testing if restrictions from q^c on each relation is redundant, where n is the number of relations involved in the query.

Example 12: In Example 7, relation/node c is a core relation due to that $c.Instructor$ = "*Tom*" is not implied by anything else (i.e., not redundant). □

Definition 8: Let G be a tree graph of a tree query Q, and G' be a tree graph which is equivalent to G, namely $G'^* = G^*$ (see Definition 4). A **core graph** G_{core} is a connected subgraph of G' (therefore, joins represented by G_{core} are implied by G), and $CORER \subseteq V_{core}$. □

Any query that is semantically equivalent to the original query Q must correspond to a connected graph, say G_c. If Q is represented by a graph G, then G_c should be a subgraph of G', which is equivalent to G (namely, $G'^* = G^*$). G_{core}, as defined above, is connected, and is a subgraph of G'. Its query graph may or may not be semantically equivalent to Q. Ideally, we would like G_c as small as possible. Thus, we proceed as follows.

Consider a minimal G_{core}. If the query corresponding to G_{core} is semantically equivalent to Q, then $G_c = G_{core}$; otherwise, G_{core} is expanded by having additional edges (joins) and relations such that the query corresponding to the expanded G_{core} is semantically equivalent to Q.

Now we proceed to discuss how G_{core} and G' as defined in Definition 8 can be computed. Initially, $G_{core} = G$. Then, G_{core} will be "*shrunk*" iteratively based on the condition stated below. (The lemma stated below also shows how G_{core} and G' can be formed.) If no further shrinking is possible, then a *minimal core graph* is obtained. Let $label(R)$, $R \in V_{core}$, be the set of distinct labels on edges in G_{core} incident to R. The following lemma states a condition that G_{core} shrinks by one node.

Lemma 4: Let $R \in V_{core}$, $R \notin CORER$. If for every two edges incident on R in E_{core} with labels X and Y, either $X \subseteq Y$ or $Y \subseteq X$ is true (i.e., elements in $label(R)$ can be completely ordered by the comparator $\subseteq$), then G_{core} can be shrunk by excluding R.

Proof: We prove it by induction on the cardinality of label (R). Since G_{core} is connected, thus $|label(R)| \geq 1$ for any $R \in V_{core}$.

Induction Basis ($|label(R)| = 1$): There are two cases when $|label(R)| = 1$:

1) Only one edge in E_{core} is incident to R. In other words, R is a leaf node in G_{core}. Since $R \notin CORER$, then the new G_{core} is obtained by excluding R and the edge in E_{core} incident to R. G' remains to be G.

2) There are more than one edge in E_{core} incident to R, and all the edges have the identical label, say X. Let the nodes in V_{core} adjacent to R be $\{R_1, R_2, \cdots, R_k\}$. These k edges are $\{(R, R_i, X) | 1 \leq i \leq k\}$. Then, we can replace any $k - 1$ out of the k edges by the $k - 1$ edges $\{(R_i, R_{i+1}, X) | 1 \leq i \leq k - 1\}$. Let G' and G'_{core} be the graphs obtained from G and G_{core}, respectively, by applying the above transformation. It can be easily shown that $G'^* = G^*$, G' is a tree, and G'_{core} is a subgraph of G'. But now R is a leaf node in G'_{core}, namely the only edge in E'_{core} incident to R is (R, R_1, X). This is exactly the situation in 1). Consequently, R can be excluded from G_{core}.

Induction Hypothesis: Suppose for any such $label(R)$, $|label(R)| = k$, $k > 1$, R can be excluded from G_{core}.

Induction: Now we show that for any such $label(R)$, $|label(R)| = k + 1$, R can be excluded from G_{core}. Let the $k + 1$ elements in $label(R)$ be $\{X_1, X_2, \cdots, X_{k+1}\}$ such that $X_1 \subseteq X_2 \subseteq \cdots \subseteq X_{k+1}$. Let the nodes in V_{core} adjacent to R and that have edge label X_1 be $\{R_1, R_2, \cdots, R_m\}$; namely there are m edges in E_{core} that are labeled X_1 and incident to R. When $m = 1$, the only edge labeled X_1

in E_{core}, (R, R_1, X_1), can be replaced by a newly inserted edge (R_1, R_x, X_1), where R_x, $x \neq 1$ is any relation in V_{core} that is adjacent to R. When $m > 1$, then all the above m edges can be replaced by the m edges of $\{(R_i, R_{i+1}, X_1) | 1 \leq i < m\}$ and (R_m, R_Y, X_1), where R_Y is any node in V_{core} that is adjacent to R with the connecting edge whose label is not X_1. Let G' and G'_{core} be the graphs obtained from G and G_{core}, respectively, by applying the above transformation. It can be easily shown that $G'^* = G^*$, G' is also a tree and G'_{core} is a subgraph of G'. But now $|label(R)| = k$ in G'_{core} due to that those edges labeled X_1 have been eliminated. By the induction hypothesis, we conclude that R can be excluded from G_{core}. □

The detection of the above condition to shrink G_{core} by one node is direct and simple by Lemma 4. After each iteration, G and G_{core} are modified. As a result, $label(R)$'s should be changed accordingly. This procedure should be repeated until the condition can not be satisfied. In each iteration, one node together with one edge will be excluded from G_{core}. Upon the termination of the algorithm, a minimally connected core graph G_{core} is obtained. The algorithm to find G_{core}, when given *CORER* and G, is obvious and is given below:

Algorithm FIND_CORE(G(V,E), CORER)
begin
$\forall R \in V$, $R \notin CORER$, Compute label(R);
if ($\exists R \in V$, $R \notin CORER$) and (conditions in Lemma 4 are satisfied)
then {$V' = V - \{R\}$;
E' is reconfigured by Lemma 4;
FIND_CORE($G' = (V', E')$, CORER);
}
else RETURN(G);
end

The following is an example that shows how G_{core} and G' are constructed by applying the above algorithm using Lemma 4, recursively.

Example 13: Given G: $(\{R_1, R_2, R_3, R_4, R_5\}, \{(R_1, R_3, ABC), (R_1, R_2, AB), (R_1, R_4, A), (R_4, R_5, A)\})$. Suppose *CORER* is $\{R_2, R_3, R_5\}$. Initially, $G_{core} = (V_{core}, E_{core})$ is G. Consider $R_4 \notin CORER$. $Label(R_4) = \{A\}$. There are two edges in E_{core} incident to R_4. This is situation 2) in the proof of Induction Basis. Therefore, R_4 can be converted into a leaf node by replacing the edge (R_4, R_5, A) by (R_1, R_5, A). Then, $R_4 \notin CORER$, can be removed from G_{core}. Therefore, G and G_{core} become

$$G' = (\{R_1, R_2, R_3, R_4, R_5\}, \{(R_1, R_3, ABC), (R_1, R_2, AB), (R_1, R_4, A), (R_4, R_5, A)\})$$

$$G'_{core} = (\{R_1, R_2, R_3, R_5\}, \{(R_1, R_3, ABC), (R_1, R_2, AB), (R_1, R_5, A)\})$$

Now consider $R_1 \notin CORER$. Since $label(R_1) = \{A, AB, ABC\}$, the hypothesis of Lemma 4 is satisfied. First consider the edge with the smallest label (R_1, R_5, A). This is the case when $m = 1$ in the Induction of the proof. Since R_2 is adjacent to R_1, this edge can be replaced by (R_5, R_2, A). Now, $label(R_1) = \{AB, ABC\}$. And similarly, the edge (R_1, R_2, AB) is replaced by (R_2, R_3, AB). After the above transformations, $R_1 \notin CORER$ becomes a leaf node, and in turn R_1 can be removed from G'_{core}. Thus, the following G'' and G''_{core} are obtained:

$$G'' = (\{R_1, R_2, R_3, R_4, R_5\}, \{(R_1, R_3, ABC), (R_3, R_2, AB), (R_2, R_5, A), (R_4, R_5, A)\})$$

$$G''_{core} = (\{R_2, R_3, R_5\}, \{(R_3, R_2, AB), (R_2, R_5, A)\}).$$

G''_{core} is the final minimal core graph such that G''_{core} is a subgraph of G'', and $G''^* = G^*$. □

The following lemma and proposition establish that the G_{core} computed by the above algorithm is minimal.

Lemma 5: Let $G = (V, E)$ be a tree graph, and $G^* = (V, E^*)$ be the closure query graph of G. $(R_1, R_2, A) \in E^*$ **if and only if** the label of every edge in the path from R_1 to R_2 in G is either A or a superset of A.

Proof: The result is obvious if $(R_1, R_2, A) \in E$. If $(R_1, R_2, A) \in (E^* - E)$, since G^* is constructed from G by transitivity of edges, the lemma can be directly verified. □

When the algorithm terminates, we obtain the G'_{core} and G' such that $G'_{core} \subseteq G'$, $G'^* = G^*$, and G' is a *minimal* tree.

Proposition 4: The core graph G'_{core} computed by the above algorithm is **minimal**, namely, upon the termination of the algorithm, no more node R, $R \in V'_{core}$ and $R \notin CORER$ can be excluded from G'_{core}.

Proof: We prove this by contradiction. Suppose $V''_{core} = V'_{core} - \{R\}$ has one fewer node than V'_{core}, G'' is a tree, $G''^* = G^*$, $G''_{core} \subseteq G''$. We now show that G'' contains a cycle, which contradicts that a tree contains no cycle.

When the algorithm terminates, R, $R \notin CORER$, cannot be a leaf node in G'_{core}. Furthermore, since the hypothesis of the Lemma 4 is not satisfied, there are at least two labeled edges (R_1, R, X), $(R_2, R, Y) \in E'_{core}$ such that neither $(X \subseteq Y)$ nor $(Y \subseteq X)$ is true. Thus, let $\beta = X \cap Y$, $X' = X - \beta$, $Y' = Y - \beta$, $X' \neq \emptyset$ and $Y' \neq \emptyset$. We want to establish that there must be a cycle in G''.

Since G''_{core} is a connected, there is a path from R_1 to R_2 within G''_{core}. Let this path be called path A. By Lemma 5, $(R_1, R, X' \cup \beta) \in E'_{core}$, $(R_2, R, Y' \cup \beta) \in E'_{core}$, and $E'_{core} \subseteq E'$ imply that $(R_1, R, X' \cup \beta) \in E^*$ and $(R_2, R, Y' \cup \beta) \in E^*$. Since $G''^* = G^*$, again by Lemma 5 there is a path in G'' from R_1 to R [let it be called path $B(R_1, R)$], and from R to R_2 [let it be called path $B(R_2, R)$], and the label of every edge in the path form R_1 to R must be X' or a superset of X', and the label of every edge in the path from R to R_2 must be Y' or a superset of Y'. Note that R is outside G''_{core}. Let the path $B(R_1, R)$ concatenated with path $B(R_2, R)$ be path B.

Let $\alpha_1, \alpha_2, \cdots, \alpha_k$ be the labels of edges in the path from R_1 to R_2 in G''_{core}, and $\gamma = \cap_{i=1}^{k} \alpha_i$. We first claim that $\gamma \subseteq \beta$. The path implies that there is an edge labeled

γ in G^* by Lemma 5. Since $G'^* = G^*$ and by Lemma 5, there is a path from R_1 to R_2 in G' such that the label of every edge in the path must be γ or the superset of Γ. Since G'_{core} and G' are trees, therefore $(R_1, R, X' \cup \beta)$, $(R, R_2, Y' \cup \beta)$ is the only path from R_1 to R_2 in G'. This establishes the above claim.

We now establish that there is a cycle in G''. Suppose there does not exist a cycle in G''. Then there are only two cases that the above two paths can be: path B from R_1 to R_2 consists of path A and path $B(R_2, R)$ or path B from R_1 to R_2 consists of path A and path $B(R_1, R)$. In the first case, since the label of every edge must be X' or a superset of X', namely, $\gamma \supseteq X'$. By $\gamma \subseteq \beta$ established above, this contradicts that $\beta \cap X' = \varnothing$ as given. In a similar manner, the second case is impossible. Hence, we concluded that there must exist a cycle in G''.

3) Tree Optimizer: In this section, we present an algorithm for semantically optimizing a tree query. Based on the discussion in above sections, given a tree query as represented by a graph $G = (V, E)$, an overall architecture of the algorithm is as follows:

1) **Compute** the set of core relations *CORER* and the restriction closure q^c
2) **Compute** G_{core} and its corresponding tree graph G' by applying the algorithm **FIND_CORE**
3) **Initialization:** The optimal solution $G_{opt} = G_{core}$
4) **Repeat**
   generate a connected subgraph $G_s = (V_s, E_s)$ of G', $G_{core} \subseteq G_s$
   if G_s is semantically equivalent to G' and the processing cost for G_s is the cheapest so far $[Cost(G_s) < Cost(G_{opt})]$
   then $G_{opt} = G_s$
   Until no more such connected subgraph G_s can be enumerated
5) **Apply** the algorithm ELIMINATE_REDUNDANCY on the optimal G_{opt} to eliminate nonbeneficial redundant restrictions specified on V_{opt} (to be discussed in Section III-E).

The final optimized query is G_{opt}, namely joining all relations in V_{opt} by E_{opt}, together with the conjunctions of all restrictions of q^c-projected on V_{opt}, $Res_{V_{opt}}(q^c)$. This is an enumerative algorithm. In other words, this algorithm enumerates all possible connected subgraphs of G containing G_{core}, and choose a semantically equivalent subgraph with the minimum cost (the minimum number of edges/joins) to be the final solution. By using G_{core}, the number of connected subgraphs to be enumerated (the search space) may be significantly reduced (see Example 15 in the next section). In the above algorithm, we need to test whether G_s and G are semantically equivalent, i.e., $G_s \sim^T G$. The corollary below, which directly follows Proposition 1, states a necessary and sufficient condition that a connected subgraph G_s of G is semantically equivalent to G.

Corollary 2: $G_s \sim^T G$, where G_s is a connected subgraph of G, **if and only if**

1) The target T of the query $Q[T; q]$ is defined on V_s after possible *target shifting* via equi-join edges, and
2) $\forall R$, $R \notin V_s$, $Res_R(q^c)$ is redundant wrt $Res_{G_s}(q^c)$, and
3) for every two adjacent relations in each path from each leaf node of G to G_s (say $R_1.A_1 = R_2.A_1$, $R_2.A_2 = R_3.A_2$, $\cdots$, $R_k.A_k = R_{k+1}.A_k$, where R_1 is a leaf node in G, $R_k \notin V_s$, and $R_{k+1} \in V_s$), $\forall i$, $1 \le i < k$, $R_i.A_i \supseteq R_{i+1}.A_{i+1}$ is implied. □

Given the cost function $COST(G_s) = |E_s|$, the above algorithm can be fine-tuned in accordance with this cost function as follows. Starting from G_{core}, the algorithm will then enumerate all connected subgraphs of G containing G_{core}. The subgraph enumeration would be conducted in such a way that in the ith, $0 \le i \le (|E| - |E_{core}|)$ iteration, all connected subgraphs of G containing G_{core} with exactly $(|E_{core}| + i)$ edges are enumerated. For each enumerated subgraph, Corollary 2 is applied to test whether it is semantically equivalent to G; if yes, then an optimal answer is found (we do not need to continue the solution search in light of this specific cost function. However, if the cost function is different, this may not be the best solution); otherwise, the enumeration (or the search) continues. Let $G(k)$ be the set of connected subgraphs of G containing G_{core} with exactly k edges. The following algorithm **TREE_OPTIMIZER** will take a query Q, a set of constraints $\mathcal{S}$, and a set of referential integrity constraints REF, and return the optimized semantically equivalent query qualification q'.

TREE_OPTIMIZER(Q[T : q], $\mathcal{S}$, REF)
begin
Compute $q^c = CLOSURE(Res(q), JP(q), \mathcal{S})$;/*
  Compute the restriction closure*/
Compute *CORER* and G_{core};/*Compute a minimal core subgraph*/
for $i = |E_{core}|$ **to** $|E|$ **do** /*iterate over the number of edges*/
  repeat Generate a subgraph $G_j(i)$ of G, $G_j(i) \in G(i)$;
    if $G_j(i) \sim^T G$ /*Test the equivalence*/
    then $\{G' =$ **ELIMINATE_REDUNDANCY** $(G_j(i), \mathcal{S})$; RETURN$(G')\}$
  until no more such connected subgraph in G can be enumerated
endfor
end

where ELIMINATE_REDUNDANCY will eliminate all redundant nonbeneficial restrictions specified on the optimal subgraph $G_j(i)$ (to be discussed in Section III-E). It should be noted that at the ith iteration, if we store the restriction closures for all $G_j(i)$'s, $G_j(i) \in G(i)$, then for the restriction closure computation in the $(i + 1)$th iteration, we do not have to start the restriction closure computation completely from scratch. It is clear that for each

$G_j(i+1) \in G(i+1)$, we can find at least one corresponding $G_{j'}(i) \in G(i)$ such that $G_{j'}(i)$ is a subgraph of $G_j(i+1)$. By the definition of the restriction closure, $CLOSURE(Res_{G_{j'}(i)}, JP, \mathbb{S}) \subseteq CLOSURE(Res_{G_j(i+1)}, JP, \mathbb{S})$. Hence, in computing the restriction closure of $G_j(i+1)$, the restriction closure of $G_{j'}(i)$ can be used as the initial set with newly added restrictions on the relation R, $R \in V_j(i+1)$ and $R \notin V_{j'}(i)$ to continue the closure computation so that substantial computational cost is saved. An example to illustrate this mechanism is given in the next section. Clearly, in the worst case, the time complexity of this algorithm on a tree is exponential in terms of the number of relations involved, since there may exist exponential number of connected subgraphs when $G_{core} = \varnothing$. However, for a nonnull G_{core}, fewer connected subgraphs are to be enumerated. In the next section, we will analyze the time complexity when we apply the above general algorithm to a closure chain query. A detailed example is also provided.

D. Closure Chain Optimizer

In this section, we show that the above tree algorithm can yield an optimal result when applied to a closure chain query. The time complexity of the algorithm is linear in the number of restriction closure computations, whose time complexity, in turn, is provided in Section II-C and [46].

More specifically, the problem in this section is as follows. Given a closure chain query $Q[T:q]$ involving n joins (hence $n+1$ relations) of $R_0.A_1 = R_1.A_1$, $R_1.A_2 = R_2.A_2$, $\cdots$, $R_{n-1}.A_n = R_n.A_n$, a set of semantic constraints $\mathbb{S}$, and a set of referential integrity, where A_i's, $1 \le i \le n$ are different, find a semantically equivalent qualification q' such that q' involves the smallest number of joins. Recall that a closure chain implies that all join attributes are distinct. We first note that the closure of G, G^c, is the same as G.

Let $R_{i,j, 0 \le i \le j \le n}$ be the resultant relation that R_i, R_{i+1}, $\cdots$, R_j join together. When $i = j$, then $R_{i,i}$ is defined to be R_i. In order to avoid a cross product in the optimal q', q' must be one of the $R'_{i,j}$'s. In other words, joins/relations to be eliminated should be from both ends of the chain, otherwise a cross product yields. They are all possible connected subgraphs of G, which construct the search space of the algorithm. Apparently, there are $\Sigma_{i=1}^{n+1} i = (n+1)(n+2)/2$ different $R'_{i,j}$'s. The following corollary is a direct application of Corollary 2 to a closure chain case, which states a necessary and sufficient condition that a subchain is semantically equivalent to the original closure chain.

Corollary 3: $R_{i,j}$ is semantically equivalent to the original query, (namely, the edges $R_k.A_{k+1} = R_{k+1}.A_{k+1}$, $0 \le k \le (i-1)$ or $j \le k \le (n-1)$, are to be eliminated, where $A_1, A_2, \cdots, A_n$ are distinct attributes), **if and only if** the following conditions are true:

1) $R_k.A_{k+1} \supseteq R_{k+1}.A_{k+1}$, $0 \le k \le (i-1)$ or $j \le k \le (n-1)$,
2) The output $T \subseteq \{R_{i-1}.A_i\} \cup Attributes(R_i), \cdots, Attributes(R_j) \cup \{R_{j+1}.A_{j+1}\}$, and
3) $Res_{R_{0,i-1}}(q^c) \cup Res_{R_{j+1,n}}(q^c)$ are redundant wrt $Res_{R_{ij}}(q^c)$. □

$R_{i,j}$'s, $0 \le i \le j \le n$, are all the subchains to be enumerated (i.e., the complete search space, or all possible connected subgraphs). When $i = j$, it represents a single relation R_i, which is also considered a chain of length zero. By the definition of core relations, a relation R_i is a *core relation* in a closure chain if $Res_{R_i}(q^c)$ is not redundant wrt $Res_{R_{1,i-1}}(q^c)$ under the join condition $R_0 \bowtie R_1 \bowtie \cdots \bowtie R_i$, and $Res_{R_i}(q^c)$ is not redundant wrt $Res_{R_{i+1,n}}(q^c)$ under the join condition $R_i \bowtie R_{i+1} \bowtie \cdots \bowtie R_n$. Since G_{core} is a connected subgraph of G, therefore it must be a subchain of the form $R_{coremin, coremax}$, which can be computed as follows.

Core Relations: Let R_{cmin} and R_{cmax} be the first core relation from the left end and the right end of the chain, respectively. The optimal equivalent qualifications are restricted $R_{i,j}$'s, where $0 \le i \le cmin$, $cmax \le j \le n$.

Target T Restriction: Let R_{tarmin} and R_{tarmax} be the first relation from the left/right end of the chain such that there is at least a target attribute defined that cannot be shifted right/left any more via equi-join edges, respectively. The optimal equivalent qualifications are also restricted to $R'_{i,j}$'s, where $0 \le i \le tarmin$, $tarmax \le j \le n$.

In summary, the final search space is $R_{i,j}$'s, $0 \le i \le coremin$, $coremax \le j \le n$, where

$$coremin = min\{cmin, tarmin\};$$

$$coremax = max\{cmax, tarmax\}$$

Example 14: In Example 7, let relations s, sc, and c be R_0, R_1, and R_2, respectively. Suppose R_0 and R_1 are identified to be core relations in a query, then $coremax = 1$ and $coremin = 0$. Only $R_{0,1}$ and $R_{0,2}$ will be generated, which are possible candidates semantically equivalent to the original query. □

$G(k)$ is the set of connected subgraphs of G with exactly k edges, which is defined to be $R'_{i,j}$ s, $0 \le i \le coremin \le coremax \le j \le n$, and $j - i = k$ for a closure chain query. The following **CHAIN_OPTIMIZER (Q[T:q], $\mathbb{S}$, REF)** will take $Q[T:q]$, $\mathbb{S}$, and a set of referential integrity REF as its input, and return the optimal $Q'[T':q']$.

CHAIN_OPTIMIZER(Q[T:q], $\mathbb{S}$, REF)
begin
q^c = CLOSURE(Res(q), JP(q), S);
Compute $G_{core} = R_{coremin, coremax}$;
$corelen = coremax - coremin$;
for *iter* = *corelen* **to** n **do** /*iterate over the number of edges*/
  for i = *coremax* − *iter* **to** coremax **do** /*enumerate all subchains with *iter* edges*/

if ($R_{i,i+iter} \sim^T R_{0n}$) /*test the equivalence between a subchain and the query*/
then{G' = **ELIMINATE_REDUNDANCY** ($R_{i,i+iter}$, $\mathcal{S}$); RETURN(G');}
endfor
endfor
end

where ELIMINATE_REDUNDANCY will eliminate all redundant nonbeneficial restrictions specified on $R_{i,i+iter}$ (to be discussed in Section III-E). In the worst case, G_{core} is empty. Then, the following subchains of $R_{i,j}$'s will be tested row by row by applying Corollary 3 to determine whether a subchain is semantically equivalent to $R_{0,n}$. Consequently, either a semantically equivalent subchain is found, or no join can be eliminated.

The first row of Table I, the 0th iteration, is for testing whether a single relation is equivalent to the original query or not. It is important to observe that at each iteration, say kth iteration, the restriction closures for all $R'_{i,j}$, $j - i = k$, are stored. In the next iteration, the restriction closures are computed from the restriction closures obtained in the last iteration. For example, in the fourth iteration, the restriction closure of $R_{1,5}$ can be computed by initiating it to be the restriction closure of $R_{1,4}$ (i.e., starting from the restriction closure of $R_{1,4}$), and continue the deduction with additional restrictions on R_5 from q^c, $Res_{R_5}(q^c)$, instead of starting the deduction from the very beginning using $Res_{R_{15}}(q^c)$. Now we consider how to determine whether each of the relations $R_{0,i}$, $0 \le i \le n$, in the diagonal is semantically equivalent to the original query. First the restriction closure of the qualification q, q^c is obtained. This is projected on $R_{0,0}$, i.e., $Res_{R_{0,0}}(q^c)$, and the restriction closure algorithm is applied to $Res_{R_{0,0}}(q^c)$ to yield, say $R'_{0,0}$. If $R'_{0,0}$ is q^c, then $R_{0,0}$ is semantically equivalent to the original query if the other conditions in Corollary 3 are also satisfied; otherwise, the restriction closure algorithm is applied to ($R'_{0,0}$ union q^c projected on R_1). This is equivalent to applying the restriction closure algorithm to q^c projected on R_1, after obtaining $R'_{0,0}$. Let the result be $R'_{0,1}$. If $R'_{0,1}$ is q^c, then $R'_{0,1}$ is semantically equivalent to the original query if the other conditions in Corollary 3 are also satisfied; otherwise, the process continues in which, in general, the restriction closure is applied to ($R'_{0,i-1}$ union q^c projected on R_i), $0 \le i \le n$, to yield $R'_{0,i}$. Since this is equivalent to applying the restriction closure algorithm to q^c projected on R_i after obtaining $R'_{0,i-1}$, all $\{R'_{0,i}, 0 \le i \le n\}$ can be obtained by applying the restriction closure algorithm to the projection q^c on R_0, followed by applying the restriction closure algorithm to the projection q^c on R_i, $1 \le i \le n$. This entire process is equivalent to one restriction closure computation.

A similar argument can be used to determine whether the off-diagonal relations $\{R_{j,j}, R_{j,j+1}, \cdots, R_{j,n}\}$ are semantically equivalent to the original query. Again, this process is no more than one restriction closure computation. There are altogether $(n + 1)$ diagonal or off-diagonal lines. Thus, the amount of work is no more than $(n + 1)$ restriction closure computations.

TABLE I
SUBCHAINS ENUMERATED

Iteration	R'_{ij}'s
$k = 0 (j - i = 0)$	$R_{0,0}$ $R_{1,1}$ $R_{2,2}$ $R_{3,3}$ $\cdots$ $R_{n,n}$
$k = 1 (j - i = 1)$	$R_{0,1}$ $R_{1,2}$ $R_{2,3}$ $\cdots$ $R_{n-1,n}$
$k = 2 (j - i = 2)$	$R_{0,2}$ $R_{1,3}$ $\cdots$ $R_{n-2,n}$
$k = 3 (j - i = 3)$	$R_{0,3}$ $\cdots$ $R_{n-3,n}$
$\vdots$	$\vdots$ $\vdots$
$k = n (j - i = n)$	$R_{0,n}$

Example 15: Suppose $n = 10$ and $G_{core} = \varnothing$. Then, 55 subchains will be generated for equivalence testing. However, if R_0 and R_7 are core relations, then only four subchains, namely, $R_{0,7}$, $R_{0,8}$, $R_{0,9}$, and $R_{0,10}$, need to be generated for consideration. In this case, the saving by usingG_{core} is substantial. □

E. Eliminating Nonbeneficial Restrictions

In this section, we will discuss how nonbeneficial restrictions can be eliminated. Situations that a restriction is *beneficial* or *nonbeneficial* are pointed out, and three different types of restriction redundancy are revealed. In this section, let $Q[T:q]$ be the query where unnecessary joins have been eliminated.

It is common that indexes can be used to speed up accessing relations, and restrictions specified on join attributes can be evaluated while joins are performed. In such cases, restrictions specified on indexed attributes and/or join attributes may reduce the cost of query processing significantly. It should be noted that the following definition provides us an intuitive feeling of the types of restrictions that are likely to lessen the cost of processing a query. However, having all *beneficial* restrictions and eliminating all *nonbeneficial* restrictions will not necessarily yield the least cost of processing the query, as also pointed out by [21].

Definition 9: A restriction ($R.X$ **op** c) is **beneficial** if 1) $R.X$ is a join attribute, or 2) $R.X$ is an indexed attribute, or 3) this restriction is not redundant wrt other restrictions specified on R. If a restriction does not satisfy 1), 2), or 3), this restriction is **nonbeneficial**. □

The restrictions of type 3) above are called *the scan reduction* [27]. They are useful because the restriction of this type specified on R, even though they may be redundant wrt restrictions on other relations, can be applied first to reduce the size of R before a subsequent join involving R occurs. Hence, the cost for the subsequent join may be reduced.

Example 16: In Example 6, the restriction ($c.Level >$ 600) is a beneficial one by 3). The restriction ($c.Level =$ 600) in Q_3''' in Example 7 is a nonbeneficial one. □

There are two goals to be achieved: add all beneficial redundant restrictions, and eliminate all nonbeneficial redundant restrictions. As we have discussed before, beneficial restrictions have been generated in the restriction closure, and a contradiction if any is also detected. Thus,

we will focus on the elimination of redundant nonbeneficial restrictions. We begin with a classification of redundant restrictions.

1) Redundancy within an Attribute: For example, if both restrictions ($s.GPA < 2.0$) and ($s.GPA < 2.5$) are present, then ($s.GPA < 2.5$) is redundant and should be eliminated unconditionally. There does not exist this type of restrictions in the restriction closure due to the *minimum representation* of restriction closure (for details, please see [46]).

2) Redundancy Among Attributes within a Relation: For example, if both restrictions ($s.GPA > 4.5$) and ($s.Rank < 3$) are present, and ($s.Rank < 3 \Rightarrow s.GPA \geq 4.6$) $\in \mathcal{S}$, then ($s.GPA > 4.5$) is redundant. If it is not beneficial (i.e., $s.GPA$ is not an indexed/join attribute), then it should be eliminated.

3) Redundancy Across Relations: For example, if the qualification is ($s.ss\# = sc.ss\# \wedge sc.c\# = c.c\# \wedge s.Status$ = "*Undergraduate*" $\wedge$ $c.Level < 600$), and we know ($s.ss\# = sc.ss\# \wedge sc.c\# = c.c\# \wedge s.Status$ = "*Undergraduate*" $\Rightarrow c.Level < 500$) $\in \mathcal{S}$, then ($c.Level < 600$) is redundant in the qualification. However, this type of restrictions is always beneficial by definition (*scan reduction*), and should be retained, as shown in Example 5.

In summary, it is sufficient to eliminate the above second type of redundant restrictions specified on nonindex and nonjoint attributes. The following procedure SIMPLIFY eliminates all such restrictions from a set of restrictions $\{p_1, p_2, \cdots, p_n\}$ specified on the same relation.

SIMPLIFY($\{p_1, p_2, \cdots, p_n\}$, JP(q), $\mathcal{S}$)
begin
if $\exists p_i$, NONINDEX(p_i) $\wedge$ NONJOIN(p_i) $\wedge$ TESTING($\{p_i, \cdots, p_{i-1}, p_{i+1}, \cdots, p_n\}$, JP(q) $\mathcal{S}$, $\{p_i\}$)
then SIMPLIFY($\{p_1, \cdots, p_{i-1}, p_{i+1}, \cdots, p_n\}$, JP(q), $\mathcal{S}$)
else RETURN$\{p_1, p_2, \cdots, p_n\}$
end

where procedures NONINDEX(p_i) and NONJOIN(p_i) will return true if p_i is not specified on an indexed attribute and a join attribute, respectively; otherwise false; *TESTING(REST, JP, $\mathcal{S}$, p)* will return true if a restriction p is redundant wrt *REST* under *JP* and $\mathcal{S}$ (see Section II-C). The recursive algorithm will find a restriction, if any, that is redundant wrt to the remaining restrictions, and then eliminate it. This procedure will be repeated until no change occurs. For each relation involved in Q, the above SIMPLIFY algorithm is applied. Thus, the procedure ELIMINATE_REDUNDANCY(Q, $\mathcal{S}$) is established to eliminate all redundant nonbeneficial restrictions in Q.

IV. Discussions and Conclusions

In this paper, the following complexity classification of semantic query optimization problems is presented.

Query Type	General	Closure
Chain	$\mathcal{NP}$-Complete	$O(n)$
Star	$\mathcal{NP}$-Complete	$\mathcal{NP}$-Complete
Tree	$\mathcal{NP}$-Complete	$\mathcal{NP}$-Complete

A systematic approach is proposed to semantically optimize tree queries. The algorithm has the property that when applied to a closure chain query, an optimal solution is obtained in polynomial time. A necessary and sufficient condition is established to eliminate a single unnecessary join. In addition, our approach systematically integrates the elimination of unnecessary joins, the addition of beneficial restrictions, the elimination of nonbeneficial restrictions, and the detection of contradiction in a query qualification.

In this paper, the processing cost of a query is assumed to be the number of joins to be performed, which may not be realistic in certain situations. For example, it is likely that joining two large relations is more costly than joining two small ones. Thus, it is meaningful if the optimization problem is defined as "eliminating unnecessary joins such that the sum of sizes of relations to be joined is minimum" (or equivalently "eliminating unnecessary joins such that the sum of sizes of relations eliminated is maximum"). Therefore, a query graph can be further generalized in the following two ways.

Edge-Weighted Labeled Query Graph: That is, each edge in G is weighted, and the cost of G is defined as $COST(G) = \Sigma_{e_i \in E}\ weight(e_i)$. This cost function makes it possible that different joins are associated with different costs. Furthermore, the interpretation of an edge weight is flexible and could be user- or application-dependent. For example, it can represent the communication distance between two relations residing at different sites in a distributed environment. In this case, the semantic optimization problem becomes to eliminate unnecessary joins involving the longest communication distance. When all edge weights are the same, the problem is reduced to what we have attacked in earlier sections.

Node-Weighted Labeled Query Graph: That is, each node is weighted, and the cost of G is defined as $COST(G) = \Sigma_{v_i \in V}\ weight(v_i)$. For example, the weight of a node can be the size of the corresponding relation. In this case, the semantic optimization problem is to remove the set of relations having the largest sum of sizes by eliminating unnecessary joins. Another weight assignment scheme could be that in a distributed environment, the load factor of a computer containing a relation is the weight of the relation, so that the optimization will yield a strategy that will not favor accessing relations on heavily-loaded computers/sites.

With the above cost models, the TREE_OPTIMIZER

and CHAIN_OPTIMIZER can be used with slight modification. The difference only lies in that the enumeration of subgraphs will continue until exhausted for the above newly defined cost functions, while the algorithm will be terminated when the first semantically equivalent one is found in TREE_OPTIMIZER and CHAIN_OPTIMIZER using the previous cost function. But how the above cost models will perform deserves further investigation.

For further research, we may generalize our result by allowing non-equi-joins and disjunctive restrictions in query qualification and semantic integrity constraints. Many problems dealing with optimizing queries in a database system have been shown to be $\mathfrak{NP}$-Complete [13], [24], [38], [39], where heuristics have been proposed for solving these problems. We are interested in identifying a set of good heuristics for semantic query optimization and testing their performance in a realistic environment. Furthermore, we hope to explore the types of constraints that may convey the data application semantics in a distributed system or an object-oriented database system, and study how these constraints can be used to improve the performance of query processing.

REFERENCES

[1] U. Chakravarthy, D. H. Fishman, and J. Minker, "Semantic query optimization in expert systems and database systems," in *Expert Database System*, L. Kerschberg, Ed. Culver City, CA: Benjamin/Cummings Publishing Co., Inc., 1986, pp. 659-674.

[2] U. Chakravarthy, J. Minker, and J. Grant, "Semantic query optimization: Additional constraints and control strategies," in *Expert Database System*, L. Kerschberg, Ed. Culver City, CA: Benjamin/Cummings Publishing Co., Inc., 1987, pp. 345-379.

[3] U. Chakravarthy, J. Grant, and J. Minker, "Logic-based approach to semantic query optimization," *ACM Trans. Database Syst.*, pp. 162-207, June 1990.

[4] E. F. Codd, "A relational model for large shared data banks," *Commun. Assoc. Comput. Mach.*, vol. 13, no. 6, pp. 377-387, 1970.

[5] C. J. Date, *An Introduction to Database Systems, Vols. I & II*. Reading, MA: Addison-Wesley, 1986.

[6] H. Decker, "Integrity enforcement on deductive databases," in *Expert Database System*, L. Kerschberg, Ed. Culver City, CA: Benjamin/Cummings Publishing Co., Inc., 1987, pp. 381-395.

[7] H. Ehrich, U. Lipeck, and M. Gogolla, "Specification, semantics, and enforcement of dynamic database constraints," in *Proc. 10th Int. Conf. VLDB*, 1984, pp. 301-308.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability—A Guide to the Theory of NP*-Completeness. New York: W. H. Freeman, 1979.

[9] G. Graefe, "Research problems in database query optimization," in *Proc. Workshop Database Query Optimization*, G. Graefe, Ed., May 1989, pp. 1-11.

[10] L. Haas, J. Freytag, G. Lohman, and H. Pirahesh, "Extensible query processing in Starburst," in *Proc. 1989 ACM SIGMOD*, Portland, OR, June 1989, pp. 377-388.

[11] M. M. Hammer and D. J. McLeod, "Semantic integrity in relational database systems," in *Proc. 1st Int. Conf. Very Large Databases*, Sept. 1975, p. 25-47.

[12] M. M. Hammer and S. B. Zdonik, "Knowledge-based query processing," in *Proc. 6th Int. Conf. Very Large Databases*, Sept. 1980, pp. 137-147.

[13] A. Hevner, "Query processing in distributed database systems," Ph.D dissertation, Univ. Minnesota, 1980.

[14] M. Jarke, J. Clifford, and Y. Vassiliou, "An optimizing prolog front-end to a relational query system," *Proc. ACM SIGMOD*, Boston, MA, vol. 14, no. 2, pp. 296-306, June 1984.

[15] M. Jarke, "External semantic query simplification: A graph-theoretic approach and its implementation in prolog," in *Expert Database System*, L. Kerschberg, Ed. Culver City, CA: Benjamin/Cummings Publishing Co., Inc., 1986, pp. 675-692.

[16] J. J. King, "Quist: A System for semantic query optimization in relational databases," in *Proc. 7th Very Large Databases Conf.*, 1981, pp. 510-517.

[17] ——, *Query Optimization by Semantic Reasoning*. Ann Arbor, MI: UMI Res. Press, 1984.

[18] G. Lohman, C. Mohan, L. Haas, B. Lindsay, P. Selinger, P. Wilms, and D. Daniels, "Query processing in R*," in *Query Processing in Database Systems*, W. Kim, D. Batory, and D. Reiner, Eds. New York: Springer-Verlag, 1985.

[19] W. Luk and P. Black, "On cost estimation in processing a query in a distributed system," in *Proc. IEEE 5th Int. Comput. Software Applicat. Conf.*, Chicago, IL. New York: IEEE, Nov. 1981, pp. 24-32.

[20] C. V. Malley and S. B. Zdonik, "A knowledge-based approach to query optimization," in *Expert Database System*, L. Kerschberg, Ed. Culver City, CA: Benjamin/Cummings Publishing Co., Inc., 1987, pp. 329-343.

[21] K. Ono and G. Lohman, "Extensible enumeration of feasible joins for relational query optimization," IBM Res. Rep. RJ 6625, Dec. 1988.

[22] X. Qian and D. Smith, "Integrity constraint reformulation for efficient validation," in *Proc. 13th VLDB*, Brighton, England, Sept. 1987, pp. 417-425.

[23] X. Qian and R. Waldinger, "A transaction logic for database specification," in *Proc. SIGMOD'88*, Chicago, IL, June 1988, pp. 243-250.

[24] A. Segev, "Global heuristics for distributed query optimization," *IEEE INFOCOM'86*, 1986, pp. 388-394.

[25] P. Selinger *et al.*, "Access path selection in a relational database system," *ACM SIGMOD 79*, 1979, pp. 23-34.

[26] S. T. Shenoy and Z. M. Ozsoyoglu, "A system for semantic query optimization," in *Proc. 1987 ACM-SIGMOD Conf. Management Data*, San Francisco, CA, May 27-29, 1987, pp. 181-195.

[27] ——, "Design and implementation of a semantic query optimizer," *IEEE Trans. Knowledge Data Eng.*, pp. 344-361, Sept. 1989.

[28] M. Siegel, "Automatic rule derivation for semantic query optimization," in *Proc. 2nd Int. Conf. Expert Database Syst.*, L. Kerschberg, Ed. The George Mason Foundation, Fairfax, Virginia, 1988, pp. 371-386.

[29] M. Stonebraker, A. Jhingran, J. Goh, and S. Potamianos, "On rules, procedures, caching and views in data base systems," Tech. Rep. UCB/ERL M89/119, Univ. California, Berkeley, Oct. 1989.

[30] W. Sun, "A Prolog implementation of the closure computation and domain knowledge for semantic knowledge base," Tech. Rep., Dept. EECS, Univ. Illinois, Chicago, Dec. 1988.

[31] J. D. Ullman, Principles of Database and Knowledge-Base Systems, Vol. 1 and 2. Reading, MA: Comput. Sci. Press, 1982.

[32] B. Wah, "File placement on distributed computer systems," *IEEE Trans. Comput.*, vol. 17, pp. 23-32, Jan. 1984.

[33] B. Wah and Y. Lien, "The file-assignment and query-processing problems in local multiaccess networks," in *Proc. IEEE Int. Conf. Data Eng.*, Los Angeles, CA, Apr. 1984, pp. 228-235.

[34] K.-Y. Whang and R. Krishnamurthy, "Query optimization in a memory-resident domain relational calculus database system," *ACM TODS*, vol. 15, no. 1, pp. 67-95, Mar. 1990.

[35] G. Wiederhold and X. Qian, "Modeling asynchrony in distributed database," in *Proc. 3rd Int. Conf. Data Eng.*, Los Angeles, CA, Feb. 1987, pp. 246-250.

[36] E. Wong and K. Youssef, "Decomposition—A strategy for query processing," *ACM Trans. Database Syst.*, vol. 1, no. 3, Sept. 1976.

[37] S. B. Yao, "Optimization of query evaluation algorithm," *ACM Trans. Database Syst.*, vol. 4, no. 2, 1979, pp. 133-155.

[38] C. Yu *et al.*, "Two Surprising Results in Processing Simple Queries in Distributed Databases," *IEEE COMPSAC*, Nov. 1982, pp. 377-384.

[39] ——, "A Promising Approach to Distributed Query Processing," in *Proc. Berkeley Conf. Distribut. Databases*, Feb. 1982, pp. 363-390.

[40] C. Yu and C. Chang, "Distributed query processing," *ACM Computing Surveys*, 1984.

[41] C. Yu, C. Chang, M. Templeton, D. Brill, and E. Lund, "On the design of a distributed query processing strategy," in *Proc. ACM SIGMOD*, San Jose, CA, May 1983, pp. 30-39.

[42] ——, "Mermaid: An algorithm to process queries in a fragmented database environment," *IEEE Trans. Software Eng.*, pp. 795-810, Aug. 1985.

[43] C. Yu, K. Guh, W. Zhang, M. Templeton, D. Brill, and A. Chen, "Algorithms to process distributed queries in fast local network," *IEEE Trans. Comput.*, vol. C-36, pp. 1153-1164, Oct. 1987.

[44] C. Yu and M. Ozsoyoglu, "An algorithm for tree query membership for a distributed query," *IEEE COMPSAC*, Nov. 1979, pp. 306–312.
[45] ——, "On determining tree query membership of a distributed query," *Canadian J. Oper. Res. Inform. Process.*, pp. 211–218, Aug. 1984.
[46] C. Yu and W. Sun, "Automatic knowledge acquisition for semantic query optimization," *IEEE Trans. Knowledge Data Eng.*, pp. 362–375, Sept. 1989.

Wei Sun received the Ph.D degree in electrical engineering and computer science from the University of Illinois at Chicago, in 1990.

In August 1990, he joined the School of Computer Science, Florida International University—The State University of Florida at Miami, where he is currently an Assistant Professor. His interests are in data and knowledge base management systems.

Dr. Sun is a member of Association for Computing Machinery, SIGMOD, SIGIR, and the IEEE Computer Society.

Clement T. Yu received the B.Sc. degree in applied mathematics from Columbia University, New York, NY, in 1970 and the Ph.D. degree in computer science from Cornell University, Ithaca, NY, in 1973.

He is currently a Professor with the Department of Electrical Engineering and Computer Science, University of Illinois at Chicago. He has published in various journals and conference proceedings, including the *Journal of the ACM*, *Communication of the ACM*, *ACM Transactions on Database Systems*, *ACM Computing Surveys*, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, IEEE TRANSACTIONS ON PATTERN ANALYIS AND MACHINE INTELLIGENCE, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEEERING, IEEE TRANSACTIONS ON COMPUTERS, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, *ACM SIGMOD*, *VLDB*, *ACM SIGIR*, *IFIP*, etc. He has been serving as a consultant for various corporations. His research interests are database management and information retrieval.

Dr. Yu has served as Chairman of the ACM Special Interest Group on Information Retrieval, as an advisory committee member for the National Science Foundation, as Program Committee Chairman for annual ACMS-IGIR conference, as Program Chairman for the First Workshop on Heterogeneous Database Systems, and was the Chairman of the 1992 International Workshop on Research Issues on Data Engineering.