SEARCH CONTROL IN SEMANTIC QUERY OPTIMIZATION

G. Ding XU

TR# 83-09

Computer and Information Science
University of Massachusetts
Amherst, MA 01003

ABSTRACT

An analysis of semantic query optimization for a subclass of relational query language called SJP expressions is presented. By incorporating the probablistic model and the concept of constraint strength into semantic query processing we can use the semantic rules in the knowledge data base to provide the most efficient access paths to the relations in the query. The problem of eliminating the join operations in SJP expressions is discussed. A possible implementation of semantic query optimization is given.

1. INTRODUCTION

Since the introduction of the relational data model [Codd70] there has been considerable research in the area of relational query optimization. This is largely due to the pragmatic importance of the subject. A relational database management system ([Codd82], [Kim79]) shields the users from knowledge of the physical organization used to represent the database, and provides them with a query language that is far removed from the primitive access operations employed to locate and retrieve the desired data. Consequently, the users formulate their retrieval requests without any concern for implementation details and delegate the responsibility for efficient implementation of their requests to the relational database management system (DBMS). The query optimization research is concerned with techniques to fulfill this responsibility.

Much of the research in query optimization has focused on a class of relational queries known as SJP expressions. This class is composed of queries expressed with the relational operators: select, join, and project. Although SJP expressions cannot express all retrieval requests against a relational data base, they occur often enough both by themselves and as subqueries to justify their place in our optimization efforts.

Given an SJP expression the optimization problem is to determine an execution plan specifying both the order and the implementation method for the relational operators in the expression, which, it is estimated, will perform most efficiently. The efficiency measure is generally taken to be the number of page transfers between primary and secondary memory which reflects the dominating influence of secondary storage access in query evaluation.

In general, an exhaustive examination of possible execution plans for an arbitrary SJP expression is highly impractical. A reasonable solution to this problem is to employ heuristics to reduce the search space, (e.g., by constraining the order of operations) so that a detailed evaluation of alternative execution plans becomes more practical. In one such approach, the query expression is subjected to a series of algebraic transformations in order to produce what is considered to be the optimal reformulation of the query in view of certain heuristics ([Hall76],[Smith75]). In another approach, the query expression is heuristically decomposed into its simpler constituent subexpressions for which an exhaustive examination of possible execution plans is then undertaken ([Selinger79], [Wong 76]).

The conventional query optimization suffers from an inherent limitation. The efficient implementation of a query is predicated, to a large extent, on how well the favorable access paths to the underlying database are matched to the requirements

of the query. For example, in order to select tuples in a relation based on the value of one of their attributes, a complete scan of the relation would be unavoidable if the selection attribute happens to be one for which no favorable access paths (such as an index) have been implemented.

The semantic query optimization, as proposed in ([Hammer80], [King81]), seeks to overcome part of this limitation. The principal idea is that specific knowledge about the real-world application which the database pertains to model, can be used to paraphrase a given query in semantically equivalent yet syntactially different forms that are unobtainable by mere algebraic transformations. The hope is that by introducing alternative statements of the orginal query, efficient implementations may be discovered which would not have been considered by conventional query optimization alone.

The knowledge base upon which semantic query optimization relies is available in the form of the integrity constraints that are part of the definition of a relational database. The key issue in semantic query optimization is first to identify possible semantic transformations for the query, and second to choose among them those that merit detailed implementation consideration by a conventional query optimizer. The difficulty lies with the fact that the real value of a transformation cannot be fully ascertained without costing the resulting evaluation programs. To resolve this dilemma, we propose to control the

search in the space of semantically equivalent queries by a simple heuristic. Our heuristic is to accomodate by means of semantic transformantion, if possible, the most efficient access path to each relation specified in the original query. The result of our semantic query optimization technique is thus a single query that is passed on to a conventional query optimizer.

The remainder of this paper is organized as follows. In Section 2 we present our assumptions regarding the class of queries considered, the underlying storage structure for the database and the cost model employed. Section 3 deals with semantic query equivalence and the different types of semantic transformations. In Section 4 we discuss the selection of target attributes for semantic query optimization. Section 5 presents the concept of constraint strength as the means to control the search in the space of semantically equivalent queries. Section 6 compares and contrasts our technique with related work in semantic query optimization. We conclude in Section 7 with a summary.

2. BASIC CONCEPTS

2.1. SJP Expressions.

The class of SJP expressions consists of queries expressed with the relational operators select, join and project. These operators are defined as follows.

1. SELECT. Let R be a relation on a set of attributes X. Let A be an attribute in X and c a value from the domain of A. Let B and C be attributes in X which are compatible (i.e., have the same data type). Then, r.A theta c and r.B theta r.C, where theta is an arithmetic comparison operator and the tuple variable r ranges over relation R, are referred to as selection conditions on R. A selection predicate, F, on R is a propositional calculus formula consisting of selection conditions connected by the logical operators & (AND), V (OR) and ~(NOT). The function of the select operator is to obtain tuples in a relation that satisfy a selection predicate. Thus

$$R[F] = \{ r \mid R(r) \ \& \ F(r) \}$$

2. JOIN. The theta join of R and S on A of R and B of S permits two relations to be joined into a single relation whose attributes are the union of the attributes of the two relations and whose tuples are those in the Cartesian

product of R and S such that the A component of R is in relation theta with the B component of S. Thus

$$R[A \text{ theta } B]S = \{ r\hat{}s \mid R(r) \& S(s) \& r.A \text{ theta } s.B \}$$

where r^s denotes the tuple t resulting from the concatenation of the tuple r of R with the tuple s of S.

3. PROJECT. Let R be a relation on a set of attributes X. Let Y be a subset of X. The projection of R onto Y is the relation obtained by discarding all the attribute values in tuples of R that do not belong to Y and eliminating duplicates that may result. Thus

$$R[Y] = \{ r[Y] \mid R(r) \}$$

where r[Y] denotes a tuple consisting of only the values for the attributes specified by Y.

We note in passing that a relational query language based on select, join and project operators is incomplete (in the sense of [Codd72]), since SJP expressions cannot express the set operations of union and difference which are required to form a complete set of relational operators.

The general form of an SJP query is given by the relational calculus expression

{ u1...um ¦ E(u1...um) }

where E(u1...um)=(R1(r1) & ... & Rk(rk))

& u1 = ri1.Aj1 &...& um = rim.Ajm & E'

which states that ri is a tuple in the data base relation Ri; that u = <u1, ..., um>, a tuple in the result relation, is composed of m components of the ri's; and that the ri's satisfy the predicate E'. E' is a prositional calculus formula consisting of selection predicates and/or join predicates connected by the logical operator AND (&).

2.2. Storage Organization for Relations.

We assume a record based implementation for relations with indices, binary links and the clustering property for a single relation as the only physical structuring mechanisms [Astrahan76]. It is assumed that the relations in the database are mapped into separate files whose record formats correspond to the attributes of the relations. We assume that each tuple has associated with it a unique tuple identifier (TID) which serves as its address.

A TID consists of two components, a page number and a slot number. The slot number refers to a pointer array that is maintained in the page header and which contains the starting byte position for every record stored in that page. As such, the TID provides a level of indirection that makes it possible to move records within a page.

A key is one or more attributes whose values identify sets of individual records in a file. When the corresponding sets are singleton sets the key is called a primary key; otherwise it is said to be a secondary key. We assume that the records in a file may be stored with no ordering (non-keyed organization) or clustered according to their values for a key (keyed organization). In the non-keyed organization a new tuple is inserted in the first available place in the file. In the keyed organization a new tuple must be inserted in its correct position (as determined by its key value) in the file. In either case deletion of tuples is handled by marking them as deleted.

To insert a tuple in a file organized in key value order the correct position is determined by consulting the clustering index for the file and the record is placed there. If the appropriate page is full, the solution is to llocate a new page, link the full one to it, place half the records from the full page on the new one, and link the new page to the successor of the old one. The new tuple can then be inserted in its correct position.

An index provides content addressability by maintaining a mapping from the values of the indexed key to the addresses of the records with those values. Conceptually, an index may be viewed as a binary relation consisting of pairs whose first component is a value for the key and whose second component is the address of a record with that value. We assume that an index is stored in a file by itself and that it is implemented by a balanced hierarchic structure in the style of B-trees [Bayer72]. The leaf pages contain records whose first component is a key value and whose second component is a variable length list of TIDs of those tuples with that key value. The nonleaf index pages contain records consisting of the address of a lower level page and the highest key value on it. The index records in each page are kept sorted on the value for the key. The TID list stored in each index record at the leaves is sorted on the first component of each TID (i.e., the page number in the file in which the tuple resides). We assume that the leaf pages of an index are threaded together so that it is possible to obtain pointers to all tuples in the relation in the index key value order without referencing upper level pages of the index.

It is important to recognize that when an index key consists of, say, two attributes, the index also serves to provide content addressability for the attribute which is the major sort field of the index file.

To retrieve tuples based on their content one or more indices may be maintained for a relation. An index is called a clustering index if the relation for which the index is defined is organized according to the index key values. Clearly, only one index for a relation may have the clustering property. We assume that all indices for a relation are updated when tuples are deleted, inserted, or modified.

An index allows the selection of related records of the same type. But instances of two record types may also be related by virtue of possessing the same value for a matching attribute. Such an association between records can be provided via links. A 1-to-n link relates each parent record, i.e., records of a first type, to the set of children records, i.e., records of a second type, that share the primary key value of the parent record for the matching attribute. Conceptually, a 1-to-n link may be viewed as a binary relation consisting of pairs whose first component is the address of a parent record and whose second component is the address of a related child record. We assume that binary links between relations are implemented by storing in each parent record the list of TIDs for its related children records.

2.3. Cost Model of the Storage Structure.

The optimization technique to be described is based on the inclusion of the most efficient access path to each relation specified in the query. In order to select amongst the alternatives we will use the following cost equations. The access cost is measured in terms of page transfers and we assume the availability of the following parameters:

PR = the number of pages in the file that holds the tuples of relation R

NR = the number of tuples of relation R

P'I = the number of leaf pages in the index on the attribute I

N'I = the number of distinct values in the index on the attribute I

hI = height of the index file on attribute I

In addition to the above parameters, we assume the availability of information on the number of distinct values appearing in each column of a relation along with the smallest and largest of these values. In the absence of semantic information to the contrary, we assume uniform distribution of values in each column and statistical independence between values in different columns.

The cost equations for the various access operations are given below.

File Scan. The cost to obtain all tuples in a relation by sequentially accessing each page in the file is PR.

Index Access. The cost to obtain pointers to all tuples in a relation satisfying the selection condition C(r.A) by accessing the index on A is

COST= hA + fA * P'A

where fA is the selectivity of the condition; that is, the ratio of the number of distinct values satisfying the selection condition to the number of distinct values for attribute A.

Clustered Record Access. The cost, exclusive of index access cost, to obtain m records sharing the same key value for a file organized in key value order is approximately

COST = PR * (m / NR)

Non-clustered Record Access. The cost to obtain m records uniformly distributed across the pages of a file is [Yao77]

PR *(1 - ((NR -NR/PR,m)/(NR,m)))

where (n,m)denotes the number of ways to select m objects out of n objects.

Record Access Through Links. Given a 1-to-n link from relation R to relation S on the attributes A of R and B of S, the cost to access the related tuples of S for a selected subset of tuples in R can be computed as follows. Let E(NR) be the expected number of tuples of R that satisfy the selection condition on R. Let N'B be the number of distinct values of the join attribute B of S. For every tuple in R, the expected number of matching tuples in S is given by NS/N'B. The expected number of pages of S that must be accessed to retrieve these matching tuples can be derived from the clustered and non-clustered record access formulas given above. So the retrieval cost is E(NR) * (PS / N'B) if S is clustered on B. Otherwise we use E(NR) * (NS / N'B) to approximate the number of page transfers.

3.SEMANTIC QUERY EQUIVALENCE

As we noted in the introduction, a relational database is not merely an arbitrary collection of relations. Rather, it is a collection of relations constrained by rules that apply in the real world application it pertains to model. By the same token, a relational query is not an arbitrary statement in a relational calculus. Rather, it is a statement specifying a meaningful subset of the relationships in the user's view. In this section we explore the ramifications of real world constraints on a

relational database and their exploitation in query optimization.

Since a database is meant to represent some particular application in the real world, it is only natural to expect that at any quiescent point the database state correspond to a legitimate and plausible state in the modeled application. What constitutes a legitimate and plausible state in the real world is, of course, specific to the application and must be specifiable in unambiguous and consistent terms. It is the function of the semantic integrity subsystem [Hammer75] of a relational database management system to provide the means for the specification and maintenance of such real world constraints.

For our purposes, we shall only be interested in semantic integrity constraints that govern quiescent database states and shall ignore those that deal with state transitions. Furthermore we shall limit ourselves to constraints which can be expressed as first order formulae in which each tuple variable is universally quantified over a database relation.

Semantic constraints of the first kind, called <u>domain rules</u>, place a restriction on the domain of an attribute. They are of the form

(all r in R) (r.A theta r.B) or

(all r in R) (r.A theta a)

where A and B are attributes of relation R in the database, a is a constant value from the domain of A, and theta is a comparison operator. The domain rule expresses the constraint that the value for attribute A in every tuple of R must be in relation theta with the value for attribute B (or, with constant value a).

Semantic constraints of the second kind, called dependency rules, are implications involving attributes of the same relation. The general form of dependency rules is

(all r in R)(C1(r.A1) & C2(r.A2) &...& Cn(r.An)-->C(r.B))

where C(r.B) is a condition on attribute B and Ci(r.Ai) (i = 1, ..., n) are conditions on attributes Ai. (The Ai's and B are attributes of relation R.) The dependency rule expresses the constraint that the righthand side condition, C(r.B), will be implied when all the lefthand side conditions, Ci(r.Ai), are satisfied. Conversely, if the righthand side condition, C(r.B), does not hold, then at least one of the conjuncts must be false.

For convenience, dependency rules of the form

(all r in R)(C(r.A)-->C(r.B))

shall be referred to as simple dependency rules. Simple dependency rules, of course, have the same interpretation as the general dependency rules. Specifically, the lefthand side

condition C(r.A) does not hold when the righthand side condition C(r.B) is not satisfiable.

Semantic constraints of the third kind, called production rules, are of the form

(all r in R)(all s in S)(C(r.A1) &...& C(r.An) & r.C theta s.D

& C(s.B1) &...& C(s.Bm) —> C(s.B))

where A1, ..., An are attributes of R, B1, ..., Bm and B are attributes of S; and r.C theta s.D stands for a join condition between R and S. Production rules serve to express inter-relational constraints in a relational database.

These three kinds of semantic constraints provide a rich mechanism for capturing much of the real world constraints that must be addressed in data modeling. Their enforcement by the database management system ensures that at any quiescent point the database is a snapshot of the modeled application and that each and every one of the defined constraints does in fact hold. Consequently, it must also be true that given a consistent database state, a query, and any defined semantic constraint, the results of the original query and the query modified by the conjunction of the semantic constraint to its predicate would be identical. It is therefore possible to employ the existing semantic constraints on a relational database to transform a

given query into a semantically equivalent (i.e., the result of the modified query being identical to that of the original), yet syntactically different query.

An important application for semantic transformation is to determine whether or not a given query is satisfiable. If C(r.A) is a condition on the attribute A of some relation R, the set of elements in the domain of A such that C(r.A) is true will be referred to as the solution range of the condition.

Definition. Let C(r.A) and C'(r.A) be conditions on attribute A. Let S and S' be solution ranges of C(r.A) and C'(r.A), respectively. If S does not intersect S', then we say that the two conditions are contradictory. Otherwise the two conditions are said to be compatible. If S' is a subset of S, we say that C(r.A) is satisfied with respect to C'(r.A)(or C'(r.A) implies C(r.A)), i.e., C(r.A) holds whenever C'(r.A) holds.

A sufficient condition for a query to be unsatisfiable is that a conjunct of the selection predicate associated with some relation specified in the query is unsatisfiable. This suggests the following lemma.

Lemma 1. A query is unsatisfiable if one of the following conditions hold:

1. All the selection conditions in a conjunct of the selection predicate associated with a relation specified in the query are contradictory to the relevant domain rules in the database.

2. All the antecedent conditions in some dependency rule in the database are satisfied with respect to the relevant conjuncts in the query while its consequent condition is contradictory to some conjunct in the query.

3. All the antecedent conditions and the join conditions in some production rule in the database are satisfied with respect to the relevant conjuncts in the query while its consequent condition is contradictory to some conjunct in the query.

Example. Suppose we have the semantic rule:

(all x in EMP)(x.Salary < 20000 ---> x.Job = "programmer")

Then the following query would be unsatisfiable:

{x.Name| EMP(x) & x.Job="secretary" & x.Salary<20000}

Semantic transformations can also be used to add implied conditions to, or eliminate redundant conditions from, the predicate of a given query.

Lemma 2. Suppose r is a dependency or a production rule such that all the antecedant conjuncts in r are satisfied with some conjuncts in the predicate E of query Q (in the case of r being a production rule, the join conditions must also appear in E). Let C(r.B) be the consequent condition of r. Then,

1. E is equivalent to E & C(r.B)

2. E is equivalent to E', where E is of the form E' & C'(r.B) and C(r.B) is implied by C'(r.B)

Lemma 2 can be extended to the case when r is a simple dependency rule and the consequent condition C(r.B) is contradictory to some conjunct in the predicate E. In that case, E is equivalent to

E & ~C(r.A)

where C(r.A) is the antecedant condition of the rule. Note that bound rules of the form (all r in R) (r.A < r.B) can be used to generate the following dependency rules:

(all r in R)(r.A >= k --> r.B >= k)

(all r in R)(r.A > k --> r.B > k)

(all r in R)(r.B <= k --> r.A <= k)

(all r in R)(r.B < k --> r.A < k)

where k is a special character which would match any constant in the query. If the lefthand side condition matches some conjunct (say, r.A > a) in the query, the consequent condition r.B > a could be generated. The same thing applies to the bound rule (all r in R)(r.A theta r.B) when theta is <=, =, >, >=. Thus lemma 2 can also be applied to the case of these bound rules. In the following discussion, we will not mention the point of using bound rules to generate equivalent queries.

Lemma 3. Suppose we have a simple dependency rule of the form:

(all r in R)(C'(r.A1) -->C'(r.B))

Let the conjunctive normal form of the selection predicate F on relation R be:

(C(r.A1) V C(r.A2) V...V C(r.B)) &...& (...)

If C'(r.A1) is implied by C(r.A1) and C(r.B) is implied by C'(r.B), then F is equivalent to

(C(r.A2) V ... V C(r.B)) & ... & (...)

Example. Suppose we have the semantic rule:

(all x in EMP) (x.Salary < 20000 --> x.Job ="programmer")

then the following two queries are equivalent:

{x.Name ¦ EMP(x) & (x.Salary<10000 V x.Job="programmer" V
x.Job="secretary") }

{x.Name¦EMP(x) & (x.Job = "programmer" V x.Job="secretary")}

Production rules can also be used to remove conditions from, or introduce conditions to, the predicate of a given query. However, some equi-join production rules, as the next lemma suggests [Dadashzadeh82], can also serve to eliminate joins.

Lemma 4. Let R(A,B,C) and S(D,E,F) be two relations. If

(all r in R) (all s in S) (r.A=a & r.C=s.D —> s.E=e)

is a semantic constraint for the database, then

(((R[A=a][C=D](s[E=e])))[B]
and
(R[A=a])[B]

are equivalent under the condition that the semantic constraint,

R[C] subset of S[D], is enforced.

Proof.

The first expression is equivalent to

((R[C=D]S)[A=a&E=e])[B]

Suppose T = (R[C=D])[A=a&E=e]. Then we have

{ t | T(t) }
= { r^s | R(r) & S(s) &
r.A = a & s.E = e & r.C = s.D }
T[B] = { t[B] | T(t) }
= { r^s[B] | R(r) & S(s)
& r.A = a & s.E = e & r.C = s.D }

From the semantic constraint, we have

(all r in R)(all s in S)(r.A = a & r.C=s.D —> s.E=e)

Therefore we can derive

T[B] = { r^s[B] | R(r) & S(s)
& r.A = a & r.C = s.D }

But the given condition that R[C] is a subset of S[D] tells us that

(all r in R)(some s in S)(s.D = r.C)

So we have

$$T[B] = \{ r\hat{}s[B] \mid R(r) \;\&\; S(s) \;\&\; r.A = a \;\&\; r.C = s.D \}$$
$$= \{ r\hat{}s[B] \mid R(r) \;\&\; S(s) \;\&\; r.A = a \}$$
$$= \{ r[B] \mid R(r) \;\&\; r.A = a \}$$

Corollary. Let R and S be two relations and

(all r in R)(all s in S)(C(r.A1) & ... & C(r.An)
& r.C=s.D --> C(s.B))

be a semantic constraint in the database. Let F be a selection predicate on R such that C(r.A1), ..., C(r.An) are implied by it. Let F' be a selection condition on S such that it implies C(s.B). Let Y be a subset of the attributes of R. Then

((R[F][C=D](S[F']))[Y]

and

(R[F])[Y]

are equivalent under the condition that the semantic constraint, R[C] subset of S[D], is enforced.

We conclude this section by restating the fundamental concept behind semantic transformation of a relational query.

Theorem. Let SC be the set of the semantic integrity constraints defined for a relational database. Let Q be the query:

Q: { x1...xm | E(x1...xm) }

If E'(x1...xm) can be generated from E(x1...xm) using the semantic integrity constraints in SC, then Q is semantically equivalent to:

Q': { x1...xm | E'(x1...xm) }

i.e., the transformations specified in the above lemmas preserve the correctness of the query.

4.TARGET ATTRIBUTES

We have shown that we can use the semantic rules in the knowledge database to transform a given query into a syntactically different, yet semantically equivalent query. We certainly require that the latter could be as efficient as possible. An obvious way to the solution is to generate the whole set of semantically equivalent queries. Then we can use the cost equations to estimate the number of page transfers needed to execute the individual queries and choose among them the query

which needs the least number of page transfers. This method suffers from obvious drawbacks.

Thus we have to exercise the control of the search space. A necessary condition for deriving a new constraint on an attribute A is that there must be a semantic rule in the knowledge database relating to A (i.e., A appears in some semantic rule). If there is a semantic rule relating to A in the knowledge database, we say that A is semantically related.

Thus semantic query transformation is possible only when at least one constraint of the query is on the semantically related attribute. And we can derive new constraints which are on semantically related attributes. Therefore we should first of all choose from the semantically related attributes our target attributes on which we are going to derive useful constraints. We assume that the DBMS provides us with the information of whether an attribute is semantically related and where the semantic rule with the atttribute as its target is stored. We want to find the semantically equivalent query which can be processed efficiently.

One way of achieving this is to make the selection operation of the individual relation as efficient as possible. This can be attainded in a number of ways.

1. The selection predicate F of relation R consists of a number of conjuncts each one of which is, in turn,

composed of disjuncts. If a conjunct is composed of those disjuncts which are all defined on semantically related attributes, we should try to remove as many disjuncts as possible. In this case, all the attributes appear in the conjunct are our target attributes. Our goal is to make an index irresolvable conjunct index resolvable and finally an index would possibly match the final conjunct. This is a deterministic process and can be done through the application of Lemma 3.

2. The aim of applying Lemma 2 to the selection predicate F of relation R is to provide new access paths to relation R, i.e., a new constraint C(r.B) can be ANDed with F only when attribute B has an index on it and the access path through the index of B is the most efficient one. Thus the semantically related attribute B of R, which has index and its constraint C(r.B) can be further restricted, are our target attribute.

Suppose that we only have one target attribute. To make the derivation of a new constraint on it possible, either the selection condition of R contains a conjunct on a semantically related attribute different from the target attribute or the selection condition of the join partner S of R contains a conjunct on a semantically related attribute.

3. Links between relations are also worthy of consideration. Let R be a relation constrained in the query Q and relation S has a link to R. Suppose that S does not appear in Q. If we could infer a constraint of S and the constraint is defined on the path of easy access (e.g., a clustered index). The introduction of the join operation, under the above conditions, is conducive to the search for the tuples in R. Thus all semantically related attributes of S, which have indices on them, are our target attributes when S does not appear in Q and has a link to some relation R in Q.

If the query contains a join, then the DBMS will spend a great deal of time on the join operation. The nested loop method of computing the join operations is generally incorporated in the DBMS. Consider the join operation

(R[F])[C theta D](S[F'])

By the nested loop method, for every tuple r of R satisfying the constraint condition F, we open a scan on S to retrieve the tuple s which satisfies the join condition and the constraint condition F'. Thus the new tuple r^s is formed. If we can infer a new constraint F" on R from any constraint other than F (say F'), then for every tuple r of R satisfying the constraint F and F", instead of F alone, a new scan on S is opened. This should

be a great saving of execution time.

As is analysed above, we would like to infer new constraints on relation R (or S) if there is a join operation between R and S. Obviously the new constraints on R should be derived from the constraints in Q on relations other than R itself. The new constraints derived from the constraints on R itself will not reduce the number of tuples satisfying the conditions since the data base is in a consistent state. Thus all semantically related attributes of R are our target attributes when some selection conjuncts of S, the join operation partner of R, are defined on semantically related attributes of S.

Last, the cost of doing the join operation is certainly great. In many cases we obviously hope that we could, if possible, reduce the number of join operations. Consider the following example :

SUPPLIER(Sno,Sname,City)

PARTS(Pno,Pname,Size)

SUPPLY(Sno,Pno,Quantity)

If we have the following production rule :

(all x in PARTS)(all y in SUPPLY) (y.Quantity >2000 &

x.Pno = y.Pno --> x.Pname = "A")

When SUPPLY[Pno] is a subset of PARTS[Pno], then obviously the query "find those supplier's names who supply part A in quantity greater than 2000":

{z.Sname| PARTS(x) & SUPPLY(y) & SUPPLIER(z) & x.Pno = y.Pno
& y.Sno = z.Sno & x.Pname = "A" & y.Quantity>2000}

is equivalent to

{z.Sname | SUPPLY(y) & SUPPLIER(z) & y.Sno = z.Sno
& y.Quantity > 2000 }

Generally speaking, we first identify "dangling relations" in the query, i.e., relations that has a single join operation and none of its attributes is the output attribute of the query. For each dangling relation S, check if the selection conjunct C(s.Bi) of S on the relation appears as the consequence of some equi-join production rule. If it appears in some equi-join production rule, check if the conditions of the lemma are satisfiable and eliminate the join operation if every selection conjunct of relation S in the query can be derived from some production rule in the knowledge data base.

Some dependency rules can also be used to eliminate join operations. Let us consider the following example:

S(Sno,Sname,Loc,Pno)

P(Pno,Size,Weight)

And we have the following two different semantic rules

(all s in S)(s.Loc = "A" --> s.Pno < 100)

(all s in S)(s.Loc = "B" --> s.Pno = 5)

Now, what do they mean? The former tells us that the Pno attribute of a tuple in S is less than 100 when the Loc attribute of the tuple in relation S is "A", while the latter shows that the Pno attribute is guaranteed to be 5 when the Loc attribute is "B". Notice that the former does not necessarily mean that the suppliers at location A supply all the parts whose numbers are less than 100. We did not distinquish these two cases in the above discussion because the relation is still there. It does not matter if the field values are "time-varying". But when we try to delete the join operation the distinction is important. We refer to the former as a "nondeterministic" dependency rule and the latter where there is only one value satisfying the consequence of the rule as a "deterministic" dependency rule.

For the convenience of discussion, we consider the case where there is only one constraint C(r.A') of R on some attribute A' other than the join attribute A from which we can possibly infer a new constraint C(r.A) on A. It is helpful to regard the constraint C(r.A') on the relation R as a constraint which finally leads to the set of values of the join attribute A of R. If the A attribute values of R have nothing to do with the A' attribute values of R, or to be specific, the constraint C(r.A') does not suggest any constraint on A attribute, then the A attribute values has to be determined by the time-varying relation of R. Suppose there is a dependency-rule in the knowledge data base from which we can infer a constraint on attribute A. If the rule is "nondeterministic", the join attribute values are still time-varying. This leaves only the case where there exists a dependency-rule and the rule is deterministic. Now, if we can, from the new constraint C(r.A) and the join expression r.A theta s.B, infer a new constraint C(s.B), we can certainly delete the join operation of R and S and insert a new constraint C(s.B) in its stead. The same thing is true with the join attribute B of relation S. If we can derive, from a deterministic dependency-rule, some constraint C(s.B), the join operation of S with R is redundant. The element satisfying C(s.B) must be there no matter how the other side of the join operation is constrained, since the database is in a consistent state.

The interesting thing is that the above join deletion can itself be generalized to any join operation. But, for simplicity, we limit ourselves to the join operation that involves only a dangling relation.

Now, in order to eliminate the join operation with R, R must be a dangling relation and all constraints of R are on the semantically related attributes. When these conditions are met, the attributes of R that are constrained in the query are our target attributes. And the join attribute A of R (and/or B of S) if it is semantically related in the knowledge database is also our target attribute.

5.CONSTRAINT STRENGTH AND SEMANTIC QUERY PREPROCESSOR

Having determined the target attributes in query Q, we try to infer new constraints on them. This could only be done by using semantic rules in the knowledge data base. There are the following cases that need to be considered.

1. The semantic rule r is a dependency rule and all the constraints of the lefthand side of r are satisfied with respect to some conjuncts in Q.

2. Either its lefthand side constraint is satisfied with respect to some disjunct of a conjunct in some selection predicate or if its righthand side of r is contradictory to some disjunct of a conjunct in some selection

predicate in the case of r being a simple dependency rule.

3. The semantic rule r is a production rule. All of its lefthand side constraints are satisfied with respect to some selection conjuncts of R and S, and the join condition r.C theta s.D appears in Q. That is, the rule is applicable to relation R in Q. And, S is not a dangling relation when the join condition of R and S is an equi-join condition.

4. The production rule is applicable to relation R whose join partner S is a dangling relation and the join condition is an equi-join condition.

5. The semantic rule r is an equi-join production rule. All of its lefthand side constraints are satisfied with respect to some selection conjuncts of relation R. There is a link from S to R on matching the values of the join attributes though the join condition does not appear in Q.

The semantic rules which fall into the above category are said to be relevant to query Q. The question of whether a semantic rule is constraint insertion relevant or constraint deletion relevant to Q depends on the target attribute. From the relevant rule r, we can derive a new constraint on some attribute B. The attribute B is called a resultant attribute of r. We certainly require that the resultant attribute B of r are one of the target attributes we have determined.

Definition. Let r be a semantic rule relevant to query Q in the knowledge database. If the resultant attribute of r is a target attribute for Q, then r is called an active rule.

Thus when some rule r in the knowledge data base is active, we can always infer a new constraint on some target attribute. When this new constraint is generated from applying a constraint deletion relevant rule:

(all r in R)(C(r.A) --> C(r.B))

to some disjunct (say, C'(r.A)) in a conjunct and there already exists in the conjunct a disjunct (say, C'(r.B)) which is satisfied with respect to the resultant constraint (say, C(r.B)) of the rule, then the corresponding disjunct (say, C'(r.A)) should be deleted. When the resultant constraint comes from a rule of case 1 and 3, such a constraint does not necessarily cater to our needs, because either the new constraint C'(r.A) on is contradictory to the old constraint C(r.A) on A in Q, or the new constraint C'(r.A) does not provide more information than the old constraint C(r.A). This latter case takes place when C(r.A) itself is satisfied with C'(r.A), i.e., when S' is a subset of S.

Example. Suppose we have the following schemes:

DEPT(Dno,Dname,Location)

EMP(Eno,Ename,Dno,Position)

The query Q is "find the names of those employees who work in NY as accountants". If there is such a rule, then in the knowledge data base that the accountant offices are either located in NY or Boston. The new constraint we can infer from the rule is of no use to us. Note if the relevant rule stipulates that the accountant offices are only located in Boston, then the constraint we can infer on attribute Location of DEPT is contradictory to the constraint on attribute Location of DEPT in Q. Without the execution of the query, we know that the query Q has no solution.

Definition. Let C(r.A) and C'(r.A) be two compatible constraints on attribute A. S and S' are their solution ranges respectively in the domain of A. We say

1.C(r.A) and C'(r.A) have the same constraint strength if S = S';

2.C'(r.A) has more constraint strength than C(r.A) if S n S'(n stands for set intersection) is a proper subset of S.

We postulate that, when we are talking about constraint strength, the constraint on attribute A is T if A is not constrained in the query, i.e., the whole domain of attribute A is the solution range. With the concept of constraint strength, we can determine whether an active semantic rule can generate some needed constraint C'(r.A) on target attribute A.

Our query preprocessor first checks whether query Q is satisfiable by examining all bound rules of the relations that appear in Q. If some bound rules are contradictory to the conjuncts of the selection predicates of relations in Q, then even without executing the query the preprocessor can tell the user that Q has no solution. When Q is satisfiable, the semantic query preprocessor proceeds to determine the set of target attributes. If there are some target attributes, it begins examining the activeness of the semantic rules of the relations of Q which are relevant to Q one by one. The new constraint C'(r.B) should be generated if the current rule is active.

Constraint Deletion Case. Suppose that A is our target attribute. Constraint C'(r.B) has been generated using constraint deletion relevant rule

(all r in R)(C'(r.A) --> C'(r.B))

If C(r.B), which appears in the same conjunct the rule applies, is satisfied with respect to C'(r.B) then the constraint C(r.A) which is satisfied with respect to C'(r.A) should be deleted.

Constraint Insertion Case. Suppose that B is our target attribute. Constraint C'(r.B) has been generated using constraint insertion relevant rule r. There are two cases that need to be discussed.

When r is a dependency rule, the dependency rule is used to provide the most efficient access path to relation R if B is not a join attribute. To solve the efficiency problem, we have the following efficiency assurance conditions:

1. The new constraint C'(r.B) on attribute B should have more constraint strength than the old constraint C(r.B);

2. The access to relation R through the constraint C'(r.B) on attribute B is the most efficient one, i.e., if we use the access path on attribute B, we need the least number of estimated pages to retrieve the qualified tuples of R.

The first condition deals with the efficiency comparison among the constraints on the same attribute. Among the constraints on different attributes of the relation, we use the estimated number of pages to make sure that the access path introduced is the most efficient one. When the efficiency assurance conditions are met, C'(r.B) should be combined with the old constraint C(r.B) to generate C"(r.B) which is then ANDed with the rest of the query expression. Once C" (r.B) fails to provide the most efficient access path to the relation, it should be deleted from the query expression when our semantic query preprocessing ends. The deletion of C"(r.B), in this case, depends on whether it was generated entirely from some semantic rule, i.e., whether there once existed some constraint C(r.B) in the query expression which was used in the generation of C"(r.B).

Now, suppose B is a join attribute and the rule is a deterministic dependency rule. If relation R of which B is an attribute is a dangling relation, C(r.B) should be migrated to the other side of the join operation, i.e., C'(r.B) should be combined with the join condition r.B theta s.D to generate some constraint on the join attribute of S and the join operation with R is unnecessary if this is done. If relation R is not a dangling relation and R has a join operation with the dangling relation S, then the join operation of R with S should be discarded and C'(r.B) is ANDed with the query expression.

When r is a production rule, we require that the newly generated constraint C'(r.B) has more constraint strength than the old constraint C(r.B). If this is true, then C'(r.B) should be combined with the old constraint C(r.B) to generate C"(r.B) which is ANDed with the rest of the query expression.

Equi-join Introduction Case. This happens when the rule is a production rule which is equi-join introduction relevant to Q. As in the case of a dependency rule which is constraint insertion relevant to Q, we require that if the equi-join operation of the new relation S with relation R is to be introduced the access of relation R through the clustered link of S to R should also meet the efficiency assurance conditions. When these conditions are met, the equi-join condition r.C=s.D and the resultant constraint C(s.B) should be ANDed with the query expression. If later, the link access path of S to R fails

to be the most efficient access path to relation R, the join operation of R with S should be discarded.

Equi-join Elimination Case. Suppose that S is a dangling relation and the join condition of S with its partner is an equi-join condition r.C=s.D. The success of eliminating the join operation with S depends on the existence of the semantic rule

(all r in R)(some s in S)(r.C=s.D)

If there is such a rule, then we should check whether the conditions of the Corollary or Lemmas are satisfied. If they are, then the corresponding conjunct should be marked as deletable. The join operation with R is deleted together with all its constraints only when all the constraints of R have have been marked as deletable.

Note that in the above process query Q is unsatisfiable whenever C'(r.B) generated by some deletion-relevant rule is contradictory to some conjunct C(r.B) of the selection predicate F of R in Q.

The above process is repeated once Q has been modified so as to provide more information to the conventional query optimization phase. The process is terminated when no modification has been made to Q.

6. COMPARISONS

The method proposed here considers a superset of the strategies adopted by some other methods [King81]. We shall only give three simple examples which show that the proposed semantic query preprocessor can reduce the I/O costs substantially.

Example. Suppose that we have the following schema:

APPLICANT(Ssno, Jobtitle, Officeno,...)

Assume that the relation has 50,000 records, occupying 5000 pages (10 records per page). There are 100 different jobtitle's. The Jobtitle field is indexed by a two level B-tree and the index (not clustered) occupies 200 pages (250 records per page).

consider the following query Q:

{ a.Ssno | APPLICANT(a) & a.Officeno = 17 }

Suppose that there is the semantic rule:

(all a of APPLICANT)(a.Officeno=17 —>a.Jobtitle='programmer')

we can transform Q into Q'

{ a.Ssno | APPLICANT(a) & a.Officeno = 17
& a.Jobtitle = 'Programmer' }

With the other strategy, this modification is not considered since the index on Jobtitle is not clustered. Lack of this information, sequential scan of the file APPLICANT is needed. The cost is 5000 pages.

But with our strategy, the modification is carried out. The conventional query optimizer can use the index on Jobtitle to access the desired records of APPLICANT. Accessing the index on jobtitle will require 2 random page accesses and 4 sequential page accesses. The selectivity factor for Jobtitle = 'programmer' is 1/100 (500 out 50,000). The total cost is 6 index pages + 500 data pages from APPLICANT which is 506 pages.

Example. Let us proceed to the following query:

```
{ a.Ssno ¦ APPLICANT(a) & ( a.Officeno = 17
                          V a.Jobtitle = 'programmer' ) }
```

Assume that we have the semantic rule:

```
(all a of APPLICANT)(a.Jobtitle = 'programmer'--> a.Officeno=17)
```

Our strategy transforms the query into the following one:

```
{ a.Ssno ¦ APPLICANT(a) & a.Officeno = 17 }
```

But with the other strategy, no modification is made and the

conventional query optimizer uses file scan method to retrieve the qualified records. The cost is 5000 pages. Our strategy leads to the accesses through the index on Officeno and the cost is 506 pages.

Example. Suppose that we have the following schema:

EMP(Ename, Job, Dno,...)

DEPT(Dno, Dname, Loc,...)

Assume that EMP has 2000 records, occupying 100 pages (20 records per page) and DEPT has 200 records which occupies 20 pages (10 records per page). The Job field is indexed by a two level B-tree. There are 100 different Job's and the index occupies 10 pages (200 per page). Consider the following query Q:

```
{ e.Ename | EMP(e) & DEPT(d) & e.Dno = d.Dno
            & d.Loc = 'A' & e.Job = 'secretary' }
```

Suppose that we have the following semantic rule:

```
(all d of DEPT)( d.Loc = 'A'--> d.Dno = 15)
```

With the other method, the rule is not helpful if there is no index on Dno attribute of DEPT. So if we use the nested loop method to process the query, the cost is 423 pages (2+1+20 pages to retrieve the records of EMP plus 20*20 pages to do the join).

Our method transforms the above query into the following form:

```
{ e.Ename | EMP(e) & e.Dno = 15 & e.Job = 'secretary' }
```

The estimated cost of processing the above query is only 23 pages.

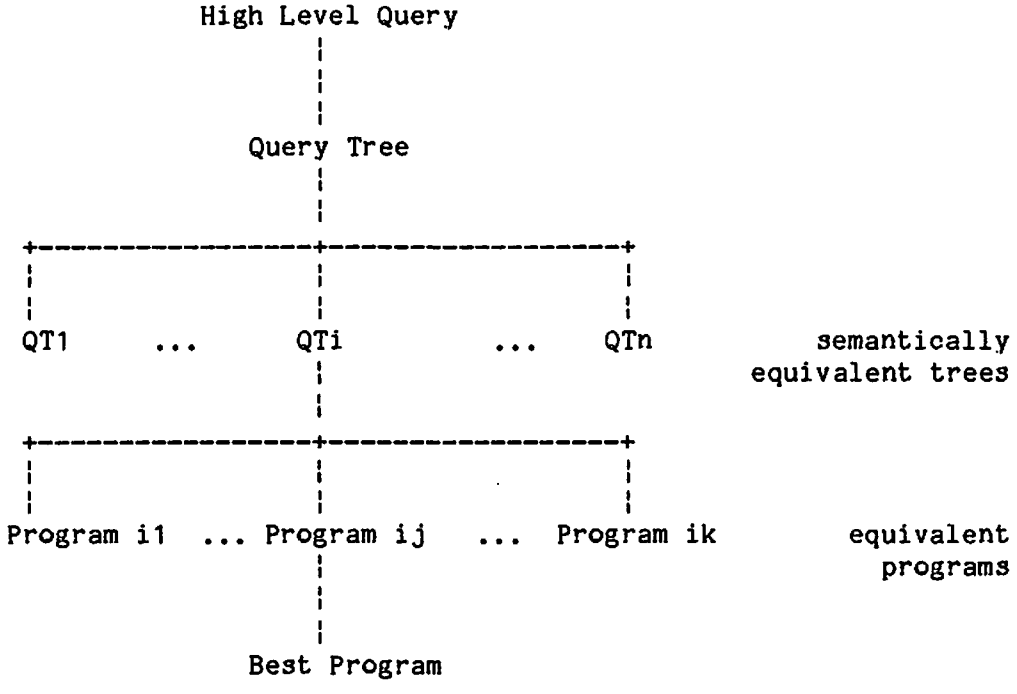
Semantic query optimization method for a specific class of languages similar to IDA [Sagalowicz77] has been proposed and some heuristics are used to generate a set of queries equivalent to the original one and, for each one in the set, a call is made to the testing phase where the cost is estimated. The testing phase is a cost estimator which is derived from [Selinger79]. Thus trees for all execution plans are constructed, costs involved are estimated and the execution plan with the minimum cost is chosen for the specific query which is equivalent to the original one. Finally the query with the minimum cost is chosen from among the set and passed to conventional query optimization for further processing. We know that for every user query Q, there exists a minimum query in the sense of the number of attributes involved. If constraints on at most n additional attributes can be added to this minimum query using the semantic rules in the knowledge database, 2**n calls are necessary to choose the query with the minimum cost.

The cost estimator [Selinger79] works as follows. First, the best way is found to access each single relation for each "interesting" tuple ordering (every join column defines an "interesting" order) and for the unordered case. Next, the best way of joining any relation to these is found, subject to the heuristics for join order (i.e., we only consider join orders which have join predicates). This produces solutions for joining pairs of relations. Then the best way to join sets of three relations is found by consideration of all sets of two relations and joining in each third relation permitted by the join order heuristics, etc. After the complete solutions have been found, the cost estimator chooses the cheapest solution which gives the required order of execution. The probablistic mathod in our approach is in line with the conventional query optimization method we assume. The efficiency assurance conditions in our approach provide the most efficient access paths to respective relations. Thus, the output of the query preprocessor is deterministic in the sense that only one query tree needs to be produced. This query can be directly sent to the conventional query optimization phase for further processing and no efficiency is lost.

7. SUMMARY

We have proposed a query preprocessing method for SJP expressions by incorporating the concept of constraint strength and probablistic method into the semantic query optimization. The problem of eliminating the join operation in SJP expressions is also discussed.

It is to be noted that the semantic query optimization method confirms the following idea [Stemple80] that the general query optimization technique would have the following structure :



The present paper deals with the selection of the right one among the semantically equivalent trees.

It should also be mentioned that the semantic query preprocessor must pass some informantion to the conventional query optimization phase, such as using the link of R to S to execute the join of R and S if R is introduced by the semantic query preprocessor. The information about the independence of constraints on the relations in Q should also be made available to the conventional query optimizer, since some constraints may not be independent.

ACKNOWLEDGEMENTS

It is a pleasure to acknowledge the helpful discussions with Professor David Stemple and Mohammad Dadashzadeh.

REFERENCES

1. Astrahan, M.M. et al "System R: Relational Approach to Database Management", ACM Trans. Database Sys., Vol. 1, no, 2, June 1976.

2. Bayer, R. et al. "Organization and Maintenance of Large Ordered Indexes", Acta Informatica, Vol. 1, 173-189.

3. Brodie, M.L. "Specification and Verification of Data Base Semantic Integrity", Ph.D. thesis, Dept of Computer Science, University of Toronto, Toronto.

4. Codd, E.F. "A Relational Model of Data for Large Shared Data Banks", CACM 13, 6 (June 1970),377-387.

5. Codd, E.F. "Relational Database: A Practical Foundation for Productivity", CACM 25, 2 (February 1982) 109-177.

6. Codd, E.F. "Relational Completeness of Data Base Sublanguage", In " Data base system " (R. Rustin, ed), Prentice Hall, 1972, 33-64.

7. Dadashzadeh, M. Z. Personal communication, 1982.

8. Hall, P.A.V. "Optimization of Single Expressions in a Relational Data Base System", IBM Journal of Research and Development, 20,3(1976) 244-257.

9. Hammer, M. et al. "Knowledge-based Query Processing", Proc. of the Sixth International Conference on Very Large Data Bases, 1980, 137-147.

10. Hammer, M. et al. "Semantic Integrity in a Relational Data Base System", Proc. of the International Conference on Very Large Data Bases, 1975, 25-47.

11. Kim, W. "Relational Database System", Computing Surveys 11:3, 185-210.

12. King,J.J. "Modelling Concepts for Reasoning About Access to Knowledge", Proc. of the Workshop of Data Abstraction, Databases and Conceptual Modelling, 1980, 138-140.

13. King, J.J. "QUIST: A System for Semantic Optimization in Relational Data Bases", Proc. of the Seventh International Conference on Very Large Data Bases, 1981, 510-517

14. Hall, P. A. V., " Optimization of Single Expressions in a Relational Data Base System", IBM Journal of Research and Development, Vol. 20, no. 3, 1976, 244-257.

15. Sagalowicz, D. "IDA: An Intelligent Data Access Program", Proc. Third Intl. Conf. on Very Large Data Bases. 1977, 293-300.

16. Selinger, P.G. et al. "Access Path Selection in a Relational Data Base Management System", Proc. ACM SIGMOD Intl. Conf. on Management of Data 1979, 23-34.

17. Smith, J.M. and Chang, P. Y. T., "Optimising the Performance of a relational Algebra Database Inteface", CACM, Vol. 18, no. 10, Oct., 1976, 568-579.

18. Stemple, D.W. et al. "A General Technique for Optimising Relational Query Execution", CIONS Technical Report, University of Massachusetts, 1980.

19. Wong, E. et al "Decomposition —A Strategy for Query Processing", ACM Trans. Database Sys., 223-241 1976.

20. Yao, S.B. "Approximating Block Accesses in Database Organization", CACM 20, 4 (April 1977) 260-261.