Improved Information Retrieval
using
Semantic Transformation

Barry G.T. Lowden and Jerome Robinson
Department of Computer Science, The University of Essex,
Wivenhoe Park, Colchester, CO4 3SQ, Essex,
United Kingdom
Telephone 0044 1206 872773 Fax 0044 1206 872788 Email lowdb@essex.ac.uk

Abstract:
Semantic query optimisation uses semantic knowledge to transform a query into another form that can be executed in a more efficient manner but still yields the same result as the original query. Commonly this semantic knowledge is in the form of rules which are generated either during the query process itself or are constructed according to defined heuristics. Over a period of time the rule set may, therefore, become very large and the number of semantically equivalent queries which may be derived rises exponentially. The problem is to identify a near optimal alternative query in a time which is minimal and also short relative to the overall query execution time. In this paper we propose a method of measuring the effectiveness of each rule and present a fast algorithm which selects the most cost effective transformations to yield a near optimal alternative query. Experiments carried out, on a large publicly available dataset, show worthwhile savings using the approach.

Keywords: Intelligent Information Systems, Semantic Query Optimisation, Optimization, Query Transformation, Rule Selection.

1. Introduction

Semantic Query Optimisation (SQO) provides an intelligent interface between user and database which uses association rules, derived from the data itself, to generate a set of alternative yet equivalent queries according to given transformation rules. A process of evaluation is then carried out to select, from this alternative query set, one which has a much lower execution cost than the original [5, 19]. SQO differs, therefore, from conventional optimisation in that it uses semantic information rather than the statistical data held within the DBM system catalogue [1, 2, 11].

Initially SQO was limited to ‘user’ provided knowledge eg. integrity constraints and data dependencies [3] though later work focused on the automatic generation of database rules [20, 6] using techniques closely related to Data Mining [12, 21]. However, unlike data mining, the rules used in semantic optimisation are precise and so yield semantically equivalent queries.

0As an example consider an employee relation for which the following rules apply:

(i) *department = 'computing'* $\rightarrow$ *benefit = 'car'*

(ii) *benefit = 'car'* $\wedge$ *salary > 30K* $\rightarrow$ *status = 'executive'*

and the relation is *indexed* on status.

A query of the form '*retrieve all staff in computing who earn more than* 35K' may thus be transformed, using the above rules, into *'retrieve all executives who work in computing and earn more than 35K'.* If executives represent only 5% of the workforce then it may be seen that this semantically equivalent query will execute in around 5% of the time taken by the original since only 'executive' tuples need be checked for the two original conditions.

In general, given a query Q with constraints (C_1, …..C_k), query transformation may be achieved by repeated use of the following reformulation operators:

(a) Addition – given a rule x $\rightarrow$ y, if a subset of constraints in Q implies x, then we can add constraint y to Q.
(b) Deletion – given a rule x $\rightarrow$ y, if a subset of constraints in Q implies x, and y implies C_i, then we can delete C_i from Q.
(c) Refutation – given a rule x $\rightarrow$ y, if a subset of constraints in Q implies x, and y implies $\neg C_i$, then we can assert that Q will return NIL.

Based on these operators, the transformation process intercepts and rewrites the original query using a given rule set. Use is also made of other database information to estimate alternative query costs in order to select the optimum from the alternatives. Current query reformulation algorithms add new conditions to the user's query and then seek to remove all non-beneficial conditions [6].

However, generating and selecting the optimum reformulated query from the set of semantically equivalent queries, derived by alternative rewrite rules in various sequential orders, has traditionally been seen to be a very time-consuming task [17]. This has led to research into improved techniques for rule generation [9, 7, 19, 20], together with ways of limiting the size of rule sets and improving the quality of the rules [4, 13, 10].

In this paper we show how semantic optimisation costs may be reduced to a linear function of the rule set cardinality. Our approach is based on the notion of a `cost ratio', associated with each rule, which is a measure of its effectiveness in reducing the work done to evaluate a query condition. The cost ratio is a simple function of the number of tuple instances defined by the antecedent and consequent conditions of the rule, which are conveniently evaluated when the rule is first derived. Cost ratios are used to generate query modifications, in linear time, yielding a single alternative query which is near optimum.

The structure of our paper is as follows. In the next section we first review the standard reformulation approaches and then introduce our fast transformation and query selection technique in Section 3. This is further illustrated by means of example and then in Section 4 we give the results of applying the method to `real' data including part of a `Public Domain' dataset. Finally in Section 5 we outline our conclusions.

2. QUERY REFORMULATION

Given a user query Q, expressed in a language such as SQL, and a set of association rules R, of the form A $\rightarrow$ B, there is an exponential number of possible ways of applying **n** rules to the query yielding ($2^n - 1$) distinct alternatives. To determine the optimum, each alternative would have to be evaluated using a standard cost estimation function [4] and the cheapest selected. Clearly this is impracticable in a realistic environment as even with the most efficient cost estimation procedures, it may be seen that the time taken to find the optimal alternative could easily exceed that of executing the original query.

A more practical approach is therefore needed and the one normally adopted is based on *best first* query generation. The essence of this strategy is to allocate a given amount of resource (time) in which to find a cheaper alternative to the original query. This allocation will usually be some function of the estimated time to execute the original query f(Q). The process is illustrated in Figure 1.

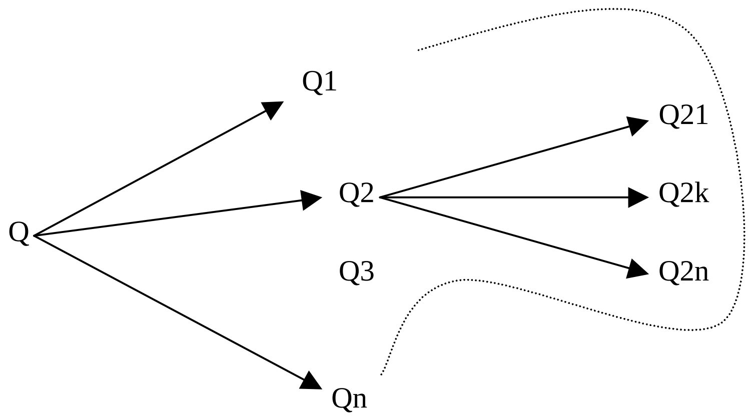

Figure 1. Best First Query Generation

Q is reformulated with respect to each matching rule in the rule set yielding alternative queries $Q_1 \ldots Q_n$. The projected cost of each alternative is evaluated and the cheapest (Q_2 in the illustration) is further reformulated and then eliminated from the search. The process continues with repeated selection, reformulation and elimination until the resource allocation runs out. At this point the cheapest alternative is returned for execution.

The strategy may not necessarily identify the optimum alternative and is heavily dependent on the order in which the rules are applied since this will determine which subset of the full range of alternatives is generated.

Another area of difficulty is that estimating the full costs of constructing these alternatives must take into account both the cost of transformation and query selection. Although the method by [17] is of interest, it is not realistic for large numbers of matching rules.

3. INTRODUCING A LINEAR TRANSFORMATION TECHNIQUE

In order to overcome the weaknesses prevalent in earlier approaches we have developed a transformation technique which runs in linear time. Our method begins by identifying those rules whose antecedent matches a constraint in the original query.

The closure of these matching rules may be simply derived using a graph-based model of the initial rule set [14, 15, 16]. In this model, conditions are nodes, and edges correspond to rules. The rule set therefore defines a graph (by listing its edges). The edge (or rule) A => B means A implies B, where A and B each represent attribute conditions.

Rule inference may be seen as path discovery in the rule graph. Query conditions map to condition nodes in the graph where paths can reveal redundant query conditions and lower cost equivalent conditions.

For example, Fig 2 shows that it is possible to delete condition E from a query with conditions A, E, F. The path from A to E means that testing for condition A is sufficient for both conditions A and E. The pre-process of deriving transitive rules adds the direct rule A => E to the rule set, thereby eliminating the need for path traversal or search at query run-time. The process reduces, therefore, to a simple table lookup of query condition pairs. Cycles denote equivalent conditions, which select the same set of tuples. So G, H or I could replace F in the query.

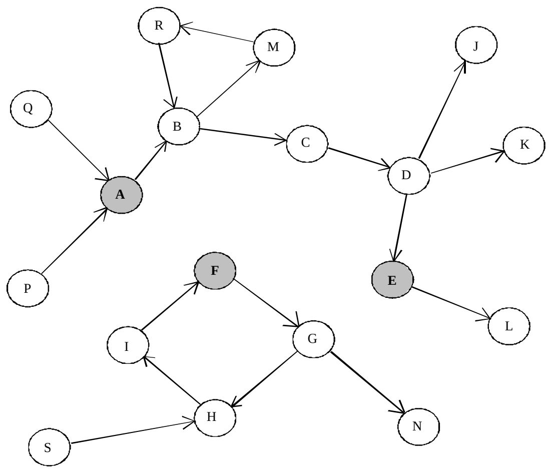

Fig 2: Part of the Rule Graph

A check is first made to determine whether any matching consequent leads to a restriction condition that immediately refutes the query or provides a single-value answer. If so the query process may be terminated without the need to access the database providing a very fast response to the user.

If it is not possible to provide an immediate response then the optimiser proceeds to estimate the costs of evaluating antecedent and consequent conditions of all matching rules in order to generate a *positive* and a *negative* condition list. Conditions in the positive list are added to the original conditions whilst those in the negative list are subtracted.

Given a matching rule of the form (x $\rightarrow$ y), where x is an original condition, the condition lists are generated as follows.

> A: Positive and negative condition lists are set to NULL;
>
> B: If x forms part of a cycle in the rule graph with implied consequents y_i (i = 1–n) where n is the number of implied consequents, then the cheapest $y_i = y_{min}$ is selected. If the evaluation cost of y_{min} is less than that of x, then it is cost effective to replace x with y_{min} since the cycle implies that ($y_{min} \rightarrow$ x). Thus condition y_{min} is added to the positive list and condition x to the negative;
>
> C: Where x and y are not part of a cycle then it is only cost effective to add y to the positive list if this reduces the combined cost of evaluating both x and y, since x will be retained. In this case, therefore, the consequent cost is based on applying both conditions x and y to those tuples identified by y. For example let a matching rule be *black* $\rightarrow$ *dark* where it is not the case that *(dark* $\rightarrow$ *black).* Thus it is only beneficial to add the condition *dark* to the positive list if it brings about a sufficient reduction in the overall number of tuples to be identified by the condition *black* since the latter condition cannot be eliminated. A specific example would be where *dark* was an indexed attribute. In cases where the consequent cost, determined in this way, is greater than that of the antecedent then condition y is added to the negative list;
>
> D: Finally, after removing duplicates, the positive conditions are added to the original query and negative conditions (where originally present) are deleted resulting in a near optimum query, Qo which is semantically equivalent to the original.

It may be seen that the above process generates only one alternative query in linear time rather than all possible alternatives in exponential time. The method is not therefore adversely affected by the size of the original rule set, as in traditional approaches and provides a fast and practical technique for semantic optimisation.

3.1. ESTIMATING CONDITION COSTS

Given that the number of instances, identified by the antecedent and consequent conditions of a rule, together with the total number of the tuples in a table, are available when the rule was initially derived [9], we may use this information to determine whether the rule is cost effective for transforming a specific query.

Let the number of instances of a condition be R, then the approximate number of disk blocks retrieved A for the R instances can be found using the following expression where B is the total number of disk blocks [8].

$$A = B * (1 - (1 - 1/B)^{R})$$

This assumes a random distribution of tuple instances across the relation space and 100% block packing density. If the condition is not indexed then it is necessary to calculate the number of disk blocks to be searched in order to retrieve those A blocks out of the total. Moreover, since there is no information about the location of the A blocks, we assume that C sequential blocks must be searched to retrieve the R instances where:

$$C = \frac{A * (B+1)}{A+1}$$ which for large A, B approximates to B.

Assuming N tuples per block, the number of tuples searched is therefore:

N*A where the selection condition is indexed,
N*C where the selection condition is not indexed.

The search cost will be the (number of tuples searched) * (the evaluation cost of the condition attribute). This latter is typically an implementation defined function of the attribute length and type. In our examples evaluation costs are (for simplicity):

strings = n bytes where n is the string length.
integers = 1 byte.

Assume a rule:

Dcode='ACCT' (30) $\rightarrow$ Dname='Accounting' (40)

where the number of condition instances is shown in brackets and the condition evaluation cost is defined as the number of bytes to be compared * number of condition instances. Assume further that the rule applies to a model relation 'DEPARTMENT' consisting of 100 blocks each consisting of 12 tuples.

Taking the antecedent as an example, we first compute approximately how many disk blocks need to be retrieved for the condition:

$$A = B * (1 - (1 - 1/B)^{R}) \cong 100*(1-(1-1/100)^{30}) = 27$$ (rounded up to whole block)

If the antecedent attribute is not indexed, we determine the expected number of disk blocks which need to be searched to retrieve the A blocks:

$$C = \frac{A*(B+1)}{A+1} \cong 27*(100+1)/28 \cong 98 \text{ (rounded up to whole block)}$$

giving:

number_of_tuples (to be searched) $\cong$ C * N $\cong$ 98 * 12 $\cong$ 1176

and total search cost of the antecedent condition $\cong$ 1176 * 4 = 4704

In the same way we may compute the search cost of the consequent condition since the number of instances is known at the time of rule generation.

3.2. A TRANSFORMATION EXAMPLE

Assume that we are looking for all information in the 'DEPARTMENT' relation where Dname = 'Marketing' and Manager = 'M3'. This query can be represented in SQL as :

Q: select * from DEPARTMENT where Dname = 'Marketing' AND Manager = 'M3';

Comparing the conditions of the given query, the following matching rules are found from the rule graph:

Dname = 'Marketing' (40) → Project > 7 (60)
Dname = 'Marketing' (40) → Project < 12 (90)
Dname = 'Marketing' (40) → Dcode = 'MKTG' (40)
Dname = 'Marketing' (40) → Manager = 'M3' (100)
Manager = 'M3' (100) → Salary > 30K (120)

where, as before, the brackets indicate the number of tuples identified by the condition.

It may be seen that no rule causes a refutation of the given query or may be used to find the answer to the query. As mentioned before, the optimiser enters a loop to estimate the costs of evaluating antecedents and consequents of the matching rules thereby determining whether the consequents are entered to the positive or negative condition lists. The results are shown in Table 1.

Rules	**R1**	**R2**	**R3**	**R4**	**R5**
Antecedent costs	10800	10800	10800	10800	2400
Consequent costs	5520	5910	4800	13200	3600
Condition List selected	+	+	+CYCLE	—	—

Table 1 Rule Costs

Hence the positive condition list generated consists of:

```
        Project > 7;
        Project < 12;
        Dcode = 'MKTG';
```

and the negative condition list is:

```
        Dname = 'Marketing';
        Manager = 'M3';
        Salary > 30K;
```

Finally by adding the positive conditions and subtracting the negative from the original query list, a near optimum query can be constructed as:

```
Qo: select * from  DEPARTMENT
      where (project > 7) AND (project < 12) AND (Dcode = 'MKTG');
```

The original query cost evaluates to 13200 whereas the near optimum alternative cost is a maximum of 3312 (applying all the alternative query conditions to the 552 tuples identified by the indexed condition (project > 7). This may be further reduced, depending on the implementation, if both indexed conditions can be pre-combined eg. in an inverted structure.

It is possible to see from the given examples that our method is straight-forward to apply even where there is a large number of matching rules. In our system, the optimiser selects the most restrictive rules in order to construct the near optimum query. We also minimise the overall effort required since it is not necessary to construct all alternative queries. Our experimental results in the following section show that the method is viable.

4. EXPERIMENTAL RESULTS

In order to measure the effectiveness of our approach, extensive experiments were undertaken on a publicly available dataset the 'General Household Survey Data, GN: 33124, Study number: 3170, Year: 1993-1994' provided by the ESRC Data Archive at the University of Essex. From this dataset was derived a 27266 instance relation 'HOUSEHOLD' encompassing 12 attributes, including three indexed, with a row length of 71 bytes. Total number of blocks was 6139, including pointer storage, in B-tree structure.

A typical experiment was as follows:

(1) A set of 500 sample queries was randomly generated, each query consisting of from 1 to 6 conjunctive terms with value domains drawn from the test relation.

(2) These queries were then run experimentally against the relation and execution times recorded. As a by-product of the execution process, a rule set consisting of 50 rules was constructed using the rule generation and selection techniques described in [8, 15].

(3) The same queries were then re-run using the semantic transformation approach introduced in section 3 of this paper.

Observations, as a result of semantic transformation, were as follows:

a) for all queries in the set, 6% were either refuted or the answer found by the matching rules alone. In the case of refutation the average saving was 99.15% and for direct query answering the saving on average was 99.53%.

b) in a further 8% of queries, semantic transformation introduced one or more indexed attributes leading to savings on average of 83.52% .

c) for the remaining 86% of queries the average saving was 2.13% due to reduced attribute matching time.

5. CONCLUSIONS

In this paper we have shown how it is possible to quantify the effectiveness of rules, used in semantic query optimisation, thus permitting specification of a query transformation sequence which yields a near optimum alternative query in minimum time. Unlike more conventional approaches, our method combines the processes of query transformation and selection in such a way that the costs of optimisation are negligible, compared to query execution costs, even when the rule set is large. Currently our work has focused on conjunctive queries using simple rules composed of single condition antecedents and consequents. Even so the results are encouraging and we are now extending our system to include inter-relation conditions and disjunctions.

REFERENCES

[1] M.W. Blasgen and K.P. Eswaran. "*Storage and access in relational data bases"*, *IBM Systems Journal*, 16(4), 1977, 363-377.

[2] A.F. Cardenas. "*Analysis and performance of inverted data base structures"*, Communications of the ACM, 18(5), May 1975, 253-263.

[3] S. Chakravarthy, J. Grant and J. Minker, "*Logic-based approach to semantic query optimisation"*, ACM on Database Sys., 15(2), 1990, pp. 162-207.

[4] K.C. Chan and A.K.C. Wong, "*A statistical test for extracting classificatory knowledge form databases*", Knowledge Discovery in Databases, 1991, pp. 107-123.

[5] G. Graefe and D. Dewitt, "*The EXODUS optimiser generator*", In Proc. of the 1987 ACM-SIGMOD Conf. on Management of Data, May 1987, pp. 160-171.

[6] J. Han, Y. Cai and N. Cercone, "*Data-driven discovery of quantitative rules in relational databases*", IEEE on Knowledge and Data Eng., 5(1), 1993, pp. 29-40.

[7] C. Hsu and C.A. Knoblock, "*Rule induction for semantic query optimisation*", In Proceedings of the 11th International Conf. on Machine Learning, 1994.

[8] B.G.T. Lowden, *"An Approach to Multikey Sequencing in an equiprobable keyterm retrieval situation*", Proceedings of the 8th Annual International ACM SIGIR Conference on Research

[9] B.G.T. Lowden, J. Robinson and K.Y. Lim, *"A semantic query optimiser using automatic rule derivation"*, Proc. Fifth Annual Workshop on Information Technologies and Systems, Netherlands, December 1995, pp. 68-76.

[10] B.G.T. Lowden and J. Robinson, *'A statistical approach to rule selection in semantic query optimisation'*. Proc. 11th International Symposium on Methodologies for Intelligent Systems, LNCS Springer Verlag, pp330-339, Warsaw, June 1999.

[11] L. F. Mackert and G. M. Lohman, "*R* optimizer validation and performance evaluation for local queries*", ACM-SIGMOD, 1986, pp. 84-95.

[12] H. Mannila, *Methods and problems in data mining*, Proc. International Conference on Database Theory, Delphi, Greece, Springer-Verlag, 1997.

[13] G.Piatetsky-Shapiro and C. Matheus, *"Measuring data dependencies in large databases*", Knowledge Discovery in Databases Workshop, 1993, pp. 162-173.

[14] J. Robinson and B.G.T. Lowden, *Data analysis for query processing*, Proc. 2nd International Symposium on Intelligent Data Analysis, London, 1997.

[15] J. Robinson and B.G.T. Lowden, *'Semantic optimisation and rule graphs'*, Proc. 5th KRDB Workshop, Seattle, WA, 31st May 1998.

[16] J. Robinson and B.G.T. Lowden, *'Attribute-pair range rules'*, Proc. DEXA'98, LNCS Springer Verlag, pp680-691, Vienna, August 1998.

[17] S. Shekhar, J. Srivastava and S. Dutta, "*A formal model of trade-off between optimisation and execution costs in semantic query optimization*", Proceedings of the 14th VLDB Conference, Los Angeles, California, 1988, pp. 457-467.

[18] S. Shekhar, B. Hamidzadeh and A. Kohli. "*Learning transformation rules for semantic query optimisation: a data-driven approach*", IEEE, 1993, pp. 949-964.

[19] S.T. Shenoy and Z.M. Ozsoyoglu, *"Design and implementation of semantic query optimiser*", IEEE Transactions on Knowledge and Data Eng., 1(3), 1989, pp. 344-361.

[20] M. Siegel, E. Sciore and S. Salveter, *"A method for automatic rule derivation to support semantic query optimisation*", ACM on Database Sys., 17(4), 1992, pp. 563-600..

[21] Yu C. and Sun W., *Automatic knowledge acquisition and maintenance for semantic query optimisation*, IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 3, 362-375, 1989.