# A Fast Transformation Method to Semantic Query Optimisation

Ayla Sayli and Barry Lowden
*University of Essex*
*Dept. of Computer Science*
*Wivenhoe Park*
*Colchester, CO4 3SQ*
*Essex, UK*
*saylia@essex.ac.uk, lowdb@essex.ac.uk*

## Abstract

*Semantic query optimisation is a comparatively recent approach for the transformation of a given query into equivalent alternative queries using matching rules in order to select an optimum query based on the costs of executing of these alternative queries. The most important aspect of this optimisation approach is that this resultant query can be processed more efficiently than the original query. This paper describes how a near optimal alternative query may be found in far less time than existing approaches. The method uses the concept of a 'search ratio' associated with each matching rule. The search ratio of a matching rule is based on the cost of the antecedent and consequent conditions of the rule. This cost is related to the number of instances in the database determined by these conditions. This knowledge about the number of instances is available and can be recorded when the rules are first derived. We then compare search ratios of rules to select the most restrictive rules for the construction of a near optimum query. The technique works efficiently regardless of the number of matching rules, since resources are not used to construct all alternative queries. This means that transformation and selection costs are minimised in our system. It is hoped that this method will prove a viable alternative to the expensive optimisation process normally associated with semantic query optimisation.*

## 1. Introduction

Semantic Query Optimisation (SQO) uses rules to transform a query into equivalent alternative queries. According to costs of these alternative queries, one of the queries may be selected as an optimum query that has the same result set but can be processed more cheaply than the original query [3; 7]. In general, there are four main stages in SQO: (1) representing the given query in a query language such as SQL, (2) query optimisation, (3) automatic rule derivation for a given query, and (4) maintaining rules. This last stage is the subject of a forthcoming paper by the authors.

Automatic rule generation has been tackled by many researchers and is the process of deriving association rules from the data itself. Such rules hold for a given state of the database as opposed to say integrity constraints which are true for any state of the database. Approaches may be classified as heuristic_based systems [17], logic_based systems [1], graph_based systems [16] and data_driven systems [4; 15; 5; 9; 12]. Research shows, however, that there is a serious problem in terms of the varying quality and sheer volume of the rules derived. For example, a heuristics based system can be used to learn all possible rules using heuristics cheaply and easily but this can lead to a very large rules set since the rules are derived automatically regardless of their quality. Data_driven systems may be used for learning currently relevant rules but the increasing size of the rule set still remains a problem [2; 6; 11].

The second stage, query optimisation, is less well researched in the literature, the most significant work already having been referenced above. In EXODUS by [3], statistical methods are employed known as the cumulative arithmetic average method and cumulative geometric average method. These are used with a sliding factor, to calculate costs of alternative queries, which is given a constant value equal to 0.5 on the basis that if the same query runs again, the cost should be half using matching rules, where a matching rule is one whose antecedent matches a query condition. However, in reality, this may not be attainable unless special circumstances occur such as a given query is refuted, the answer set is found using a matching rule, or the matching rules contain index attributes. Another issue, often overlooked is that, in fact, the optimisation cost usually comprises the transformation cost together with the selection cost [10; 14; 9]. Clearly we need to consider the combined costs of these processes together with the execution cost of the optimum query when computing the overall savings. The paper by [14] is realistic, in this regard, but is based on a small number of transformation rules providing a limited range of alternative queries. Our method is based on notion of a 'search ratio' associated with each rule in that this number shows how effective this rule is in restricting the given query for a given database. In order to do this, we keep the number of instances associated with the antecedent and consequent conditions of the rule and the total number of the instances of the rule relation as a cost variable when we derive the rule. It is then possible to identify the most effective rules to construct the optimum query according to their search ratios using the knowledge which is kept in the rules set. This means that we avoid the need to construct all possible alternative queries as is the case with traditional approaches.

In Section 2, we first give the optimisation method of the traditional approaches and then present the cost estimation function to show how the search ratio is determined for each matching rule. In section 3, examples of the method are given using a model relation. Finally, in Section 4, results of our experiments using the method are given.

## 2. An Approximation Method to Semantic Query Optimisation

Using knowledge to guide any process in SQO is essential where learning, optimisation, and rule maintenance can be made more intelligent and practical. In our system, we rank the matching rules according to their search ratios. For a rule W→Z, assuming that Wcost is the search cost to find the tuples identified by the antecedent condition W, and Zcost is the search cost to find the tuples identified by the consequent condition Z, a search ratio represents the saving if Z is used instead of W, which can be formalised as (Wcost - Zcost)/ Wcost. That is, this search ratio is used to compare our saving of using Z against W in order to determine a rule's effectiveness when used in query transformation.

### 2.1. Traditional Semantic Query Optimisation Method

In general, SQO takes a given query in a language such as SQL, QUEL, or variant of the algebra or calculus, and adopts one of a number of different approaches to represent the query. Secondly, each Original Condition (OC) from the query is checked for matching rules in a rules set. Thirdly, matching rules are used to transform the given query into alternative queries. Fourthly, all alternative queries are compared according to their costs in order to select the optimum query. If there is no matching rule for an OC, the latter may be used to initiate an automatic rule derivation process. The method is shown in Figure 1 where the search space contains the alternative queries which are syntactically different from each other but semantically equivalent. This means that they can be used to retrieve the same answer set as the original query. In the figure, we do not show alternative queries which use the same rules but in a different order. For example, in the case of using R1 and R2, Qn+1 can be constructed using the orders {R1, R2} or {R2, R1}.

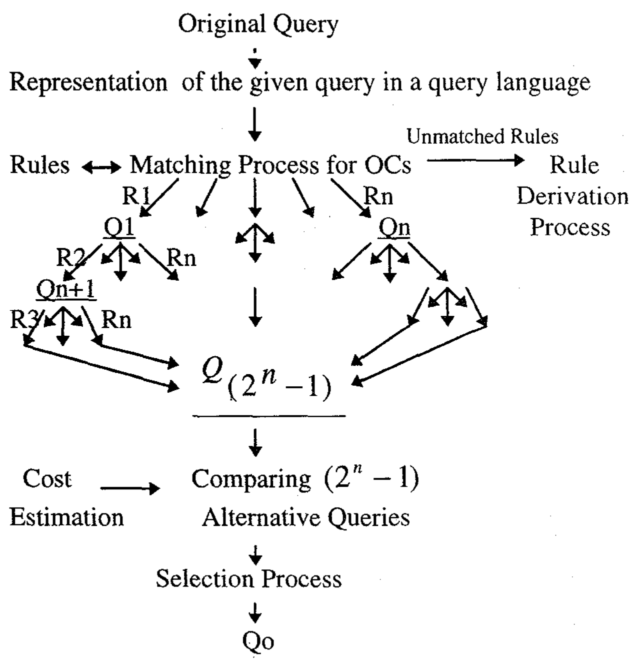

Figure 1. Traditional Semantic Query Optimisation Method

However, problems arise if the number of matching rules increases, since the number of alternative queries also grows making the process expensive and inefficient even considering only the queries which are syntactically different from each other but semantically equivalent to the original query. Moreover assuming that the antecedent conditions of these matching rules are the same as the conditions of the given query and n is the number of distinct antecedent conditions, then the number of the possible alternative queries using the rules in Figure 1 is $(2^n - 1)$. If we consider all possible alternative queries, including these with a simple reordering of the same conditions, then the number can be found using the formula ( $\sum_{x=1}^{x=n} \frac{n!}{(n-x)!}$ ). Another area of difficulty is estimating the costs of these queries which must take into account both the cost of transformation and query selection. Although the method by [14] is of interest, it is not realistic for large numbers of matching rules. Also most research on optimisation techniques tends to be based on statistical information in the system catalogue. For example, INGRES uses 'optimizedb' to analyse table data to estimate needed information (e.g. total number of tuples in a table, number of distinct values of the index attribute of a column, etc). However, there is little research on the time taken to produce these statistics in relation to the optimisation time for a query. Our method provides a practical solution to those problems as given in the following section.

## 2.2 A Fast Transformation Method For Semantic Query Optimisation

The method is illustrated in Figure 2. SQL is chosen as a query language to represent the given query. Next, the conditions of the given query are matched with the antecedent condition of corresponding rules [12]. Moreover we also check the matching rules to determine whether they imply other matching rules. For example, assume a matching rule is found, W->Z, we then can use Z in the matching process as well as W, which may be used to find a rule such as Z->V. This step can be seen as a loop which terminates when all matching rules are found. If there are no matching rules, the condition may be used to derive a new rule to be used with subsequent queries.

A further check is then made to find out if the antecedent condition of any matching rule provides an immediate response to the query, and finally whether there is refutation. In case of such queries whose answers are found with no further need for optimisation, SQO clearly provides major savings.

If the query remains unanswered then, in our approach, the optimiser enters a loop to estimate costs of antecedent and consequent conditions of all matching rules. According to the costs, the matching rules are divided into two groups, (X) and (Y). (X) contains matching rules where the consequent condition is less expensive to evaluate than the antecedent condition. The remainder of the rules are kept in the second group, (Y). We then order the rules in the (X) using their search ratios. A condition list may be generated from the consequents of rules in (X). The rules in (Y) may now be used to eliminate any expensive conditions. The result is a near optimum query, Qo which is semantically equivalent to the original.

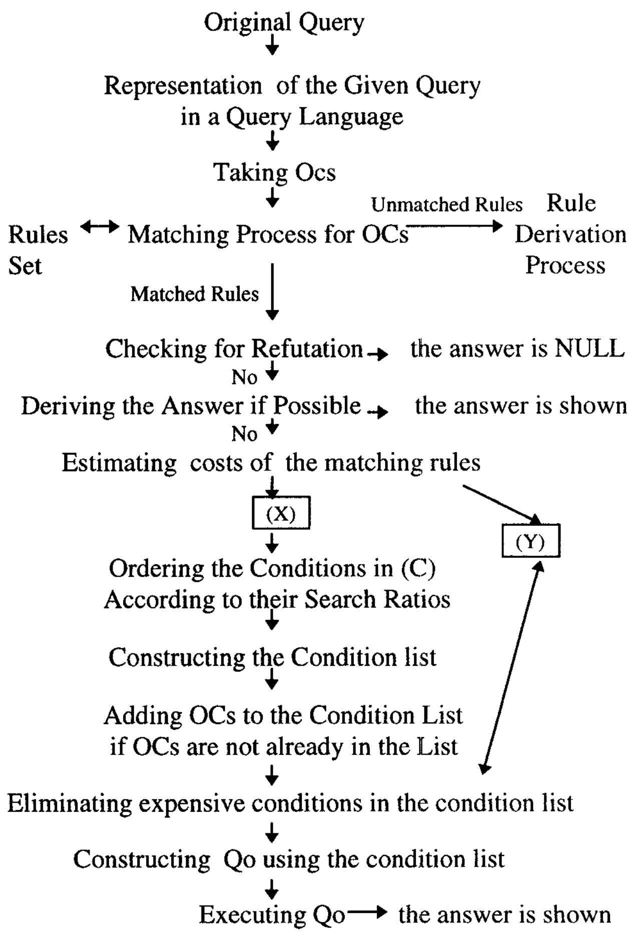
Original Query
Representation of the Given Query in a Query Language
Taking Ocs
Unmatched Rules
Rule Derivation Process
Rules Set
Matching Process for OCs
Matched Rules
Checking for Refutation
the answer is NULL
No
Deriving the Answer if Possible
the answer is shown
No
Estimating costs of the matching rules
(X)
(Y)
Ordering the Conditions in (C) According to their Search Ratios
Constructing the Condition list
Adding OCs to the Condition List if OCs are not already in the List
Eliminating expensive conditions in the condition list
Constructing Qo using the condition list
Executing Qo
the answer is shown

Figure 2. A Fast Transformation Method for Semantic Query Optimisation

Our method is very efficient when there are a large number of rules in the rule set. It is not limited by the number of rules as in traditional approaches [10; 14; 9] since it does not generate all possible alternative queries to select the optimum. It is also very practical and fast because it is not dependent on any specific statistical information from the system catalogue. The transformation algorithm is shown below, and the process of cost estimation, referred to in the algorithm, is described in the next section.

**Linear Query Transformation Algorithm**

Ri : all matching rules ( i:1, 2, ..., n) as (R1, R2, ..., Rn)
The syntax for one of the matching rules, W→Z:
W presents the antecedent condition of the rule,

Z presents the consequent condition of the rule.
Zlength: Length of the consequent attribute value ,
Wlength: Length of the antecedent attribute value
*Zinsnum: Number of instances that can be identified by Z,
*Winsnum: Number of instances that can be identified by W
Zindex: If the consequent attribute of the first rule is an index attribute, it is equal to 1. Otherwise, it is equal to 0 (zero)
Windex: If the antecedent attribute of the first rule is an index attribute, it is equal to 1. Otherwise, it is equal to 0 (zero)

** These values are computed as a by-product of the initial rule generation process. They are therefore provided with the rule.*

Step 1: Estimate costs of antecedent conditions and costs of consequent conditions of all matching rules

```
cost(Windex, Zindex, Winsnum, Zinsnum,
                    Wlength, Zlength, Wcost, Zcost);
```

Step 2: Divide the matching rules into two groups, (X) and (Y)

```
if(Ri->Wcost > Ri->Zcost)
        Ri goes into (X)
else
        Ri goes into (Y)
```

Step 3: Order all rules (i: 1, 2, ..., k) in (X) according to their search ratios

a} Calculate search ratios using
Ri.search_ratio=(Ri->Wcost - Ri->Zcost)/rule->Wcost

b) Order Ri according to Ri.search_ratio

Step 4: Construct a condition list using the ordered rules
Step 5: Add OCs to the condition list if OCs are not already in the list
Step 6: Eliminate any expensive condition in the condition list using (Y)
Step 7: Construct a near optimum query, Qo using the condition list
Step 8: Execute Qo

### 2.2.1 Cost Estimation

In this section, we show how the number of instances identified by the antecedent condition of a matching rule and the consequent condition of the rule can be used to estimate an approximate cost saving using statistical methods. As mentioned before, the number of instances and the total number of the tuples in a table are available when the rule was initially derived [12].

Assuming that the number of instances of a condition is R, the approximate number of disk blocks retrieved (A) for the R instances can be found using Function (1) where B is the total number of disk blocks in the database [8].

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

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 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:

$$C = \frac{A * (B + 1)}{A + 1} \qquad (2)$$

The number of tuples searched is therefore:

$$\text{Number\_of\_tuples} \cong C * N \qquad (3a)$$

where N is the number of tuples per block. However if the consequent attribute is an indexed attribute, the optimiser only searches the number of the instances located in 'A' blocks as follows.

$$\text{Number\_of\_tuples} \cong A * N \qquad (3b)$$

Our cost approximation may then be extended to predict the number of byte comparisons as shown in (4).

$$\text{COMPARISON\_COST} = \text{Number\_of\_tuples} * \text{Length} \qquad (4)$$

where Length is the size of the condition attribute length in bytes. This will depend on the implementation but typically, for INGRES, Table 1 can be used.

Table 1. INGRES data types and their lengths

| INGRES Data Types | Range | Length |
|---|---|---|
| char | - | The length of the attribute value |
| integer1 | -128 to +127 | 1 byte |
| integer2 | -32,768 to + 32,767 | 2 byte |
| integer4 | -2,147,483,648 to + 2,147,483,647 | 4 byte |
| float4 | -10**38 to +10**38 (7 decimal precision) | 4 byte |
| float8 | -10**38 to +10**38 (17decimalprecision) | 8 byte |

Function (4) can be used to evaluate the search ratio for each matching rule since the number of instances is known at the time of rule generation. In the following section, we give some examples to illustrate how the method works in practice using a model database, 'DEPARTMENT'.

## 3 Examples of the Optimisation Method

Our examples are based on a model 'DEPARTMENT' relation which has 240 instances and 5 different attributes (Dcode char(4), Dname char(12), Project integer, Manager char(4), Location char(15)), and assumes initially that the system has 8 rules in the rules set. These rules are shown in Table 2 with their associated number of instances within the 'DEPARTMENT' relation. 'Project' is an indexed attribute of the relation. Row length is 40 bytes and a disk block can hold 12 tuples. Total disk blocks B = 20.

Table 2. Rules Set

| Rule No: W -> Z | Winsnum | Zinsnum |
|---|---|---|
| R1:Dcode='ACCT'→Dname='Accounting' | 30 | 40 |
| R2:Dcode='ACCT'→ Manager = 'A1' | 30 | 80 |
| R3:Dname = 'Marketing'→Project > 7 | 40 | 60 |
| R4:Dname = 'Marketing'→Project < 12 | 40 | 200 |
| R5:Dname='Marketing'→Dcode='MKTG' | 40 | 40 |
| R6:Dcode='MKTG'→Dname='Marketing' | 40 | 40 |
| R7:Dname='Marketing'→Manager='M3' | 40 | 100 |
| R8:Project>7→Location='London' | 60 | 150 |

**Example 1:** Assume that we are looking for all information in the 'DEPARTMENT' relation where Dcode = 'ACCT'. This query can be represented in SQL as :

Q1: select * from DEPARTMENT where Dcode = 'ACCT';

In our system, we match the single query condition against antecedent conditions in our rules set. From Table 2, two rules below can be used to transform the given query:

| | | |
|---|---|---|
| R1:Dcode='ACCT'→Dname='Accounting' | 30 | 40 |
| R2: Dcode='ACCT'→ Manager='A1' | 30 | 80 |

There is no rule to cause refutation of the given query and the answer of the query can not be found using matching rules alone, so the optimiser enters a loop to calculate costs of both the antecedent condition and the consequent condition for each associated rule and then ranks them. For R1, the antecedent condition is W1, the length of the comparison value of the condition is Wlength = 4, the number of instances identified by W1 is Winsnum = 30, and the condition does not contain any indexed attributes thus Windex = 0. The consequent condition is Z1, Zlength = 10, Zinsnum = 40 and Zindex = 0. The optimiser then proceeds to the cost estimation process as follows:

(a) Function (1) is used to compute approximately how many disk blocks need to be retrieved for the antecedent condition:

$$A = B * (1 - (1 - 1/B)^{R})$$
$$\cong 20 * (1 - (1 - 1/20)^{30}) \cong 15.70$$

(b) Since the antecedent attribute is not indexed, we determine the expected number of disk blocks which need to be searched to retrieve the A blocks using Function (2):

$$C = \frac{A * (B + 1)}{A + 1} \cong \frac{15.70 * (20 + 1)}{15.70 + 1} \cong 19.74$$

(c) Using Function (3a), the number of tuples can be found as follows:

$$\text{Number\_of\_tuples} \cong C * N$$
$$\cong 19.74 * 12 \cong 236.91$$

(d) Using Function (4), the number of bytes to be compared for the condition:

$$\text{Wcost} = \text{Number\_of\_tuples} * \text{Length}$$
$$= 236.91 * 4 = 947.66$$

For the consequent condition, the calculation is similar. Using Function 1, the approximate number of blocks retrieved can be found:

$$A = B * (1 - (1 - 1/B)^{R})$$
$$\cong 20 * (1 - (1 - 1/20)^{40}) \cong 17.42$$

Since the consequent attribute is not indexed, we determine the expected number of disk blocks to be searched using Function (2):

$$C = \frac{A * (B + 1)}{A + 1} \cong \frac{17.42 * (20 + 1)}{17.42 + 1} \cong 19.86$$

The number of the tuples is found as follows:

Number_of_tuples ≅ C * N ≅ 19.86 * 12 ≅ 238.326

We then estimate the number of bytes to be compared for the condition:

Zcost = Number_of_tuples * Length
= 238.326 * 10 = 2383.26

The cost of the antecedent condition of the rule is less than the cost of the consequent condition of the rule, so the rule is added into (Y).

For rule, R2, Wcost and Zcost can be found in a similar way to the first rule; Wcost = 947.66, Zcost = 479.6. The cost of the antecedent condition of the rule is higher than the consequent of the rule, so the rule is added into (X). Since a query condition is the antecedent condition of R2 whose consequent is Manager = 'A1', this latter is added into the condition list. At this stage a check is made to eliminate any conditions in the list, which can be matched with the consequents of rules in (Y). Using R1 in (Y), it is not possible to eliminate any rules in the list . The optimisation process is completed with the construction of the near optimum query as below:

Qo: select * from DEPARTMENT
where Manager = 'A1' and Dcode = 'ACCT';

This resultant query is more efficient to execute than the original.

**Example 2:** Assume that we are looking for all information in the 'DEPARTMENT' relation where Dname = 'Marketing'. This query can be represented in SQL as :

Q2: select * from DEPARTMENT
where Dname='Marketing';

Comparing the conditions of the given query, the following matching rules can be found:

| | | |
|---|---|---|
| R3:Dname='Marketing'→Project>7 | 40 | 60 |
| R4:Dname='Marketing'→Project<12 | 40 | 200 |
| R5:Dname='Marketing'→Dcode='MKTG' | 40 | 40 |
| R6:Dcode='MKTG'→Dname='Marketing' | 40 | 40 |
| R7:Dname='Marketing'→Manager='M3' | 40 | 100 |
| R8:Project>7→ Location='London' | 60 | 150 |

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 both antecedents and consequents of the rules. These costs can be seen in the following table where the first row shows costs of antecedent conditions and the second row shows that of consequent conditions.

| Rules | R3 | R4 | R5 | R6 | R7 | R8 |
|---|---|---|---|---|---|---|
| Wcost | 2144.93 | 2144.93 | 2144.93 | 953.30 | 2144.93 | 228.94 |
| Zcost | 228.94 | 239.99 | 953.30 | 2144.93 | 479.86 | 1439.96 |
| Search Ratio | 0.89 | 0.88 | 0.55 | - | 0.77 | - |

R3, R4, R5 and R7 are added into (X). R6 and R8 are added into (Y). We then order all rules in (X) according to their search ratios that is R3, R4, R7, and R5. The condition list is constructed taking the consequent conditions of the ordered group as 'Project > 7', 'Project < 12, 'Manager = 'M3'' and Dcode = 'MKTG'. Then the OC, 'Dname = 'Marketing'' is added to the condition list because it does not exist in (X) as a consequent condition. The optimiser then determines whether there are any rules in (Y) where the consequent condition of the rule is in the condition list. Using R6, it is possible to eliminate the condition 'Dname = 'Marketing'' from the condition list knowing the antecedent condition of the rule is less expensive than the consequent condition and already exists in the list. Finally the near optimum query can be constructed as:

Qo: select * from DEPARTMENT
where Project > 7 and Project < 12 and
Manager = 'M3' and Dcode = 'MKTG';

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 Computational Results

The two relations in our database were: a 5861 instance relation 'STUDENT' provided by the Student Administration Office at Essex University, and a 27266 instance relation 'HOUSEHOLD' which was created from the 'General Household Survey Data, GN: 33124, Study number: 3170, Year: 1993-1994'. Survey data was provided by the ESRC Data Archive at the University. The row length was 43 bytes for 'STUDENT' and 71 bytes for 'HOUSEHOLD'. The total number of blocks was 248 for 'STUDENT' and 6139 for 'HOUSEHOLD' in Btree storage structure respectively. The schemas of the relations are

STUDENT (logname c8, regno int, advisor int, entry int,
year c2, scheme c6, uccacode c6, status c1,
examno int, school c4),

HOUSEHOLD (hserno int, persno, int, region c10, npersons int,
typaccm c10, bedrooms int, centheat c4, ncars int,
ownrent c5, mortgage c4, cost int, loan int).

The machine used was a 33Mhz SPARC-ELC with Ingres files held on a 2GB Fujitsu SCSI running synchronously. Both relations are based on actual data used within the University. Rules in our rules set were derived from the system by [12; 9]. The rule sets for each relation contained 50 rules.

The experiment was done for several thousand queries on both relations which was based on many observations using different featured rules in order to analyse the time saving using SQO. For example, if the rules contain any index attribute or not. From our tests, the following results were observed:

a) For the both relations, if the original query is refuted by any rule, the saving on average was 99.15%. If the answer to the original query was found by one of the matching rules alone, the saving on average was 99.53%.

b) If the rule contained an indexed attribute, the savings on average were up to 86.40% for 'STUDENT' and 83.52% for 'HOUSEHOLD'.

c) In queries for which no indexed condition could be found, the saving on average was 6.39% for the relation 'STUDENT', and 1.94% for the relation 'HOUSEHOLD'. The poorer result from the 'HOUSEHOLD' data was due to the lower number of instances per block compared with the 'STUDENT' relation.

## 5 Conclusions

This paper has described a fast query transformation process in SQO which constructs a near optimum query taking into account all matching rules. The results are encouraging and promise large savings even when the rule set is large since transformation time is in linear function of rule set cardinality. We are now extending our current work in statistics and knowledge discovery to address the issue of complex queries such as join queries and maintaining rules set [13; 18].

**Acknowledgement**

We would like to thank John Ford & Tony Lawson in Computer Science Dept., and Ken Miller & other personnel in the ESRC Data Archive at University of Essex. Thanks also to Prof. Tahir Sisman at Yildiz University for all his support during this research.

**References**

[1] S. Chakravarthy, J. Grant and J. Minker, "Logic-based approach to semantic query optimisation", ACM on Database Systems, Vol 15, No 2, 1990, pp. 162-207.

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

[3] 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.

[4] J. Han, Y. Cai and N. Cercone, "Data-driven discovery of quantitative rules in relational databases", IEEE on Knowledge and Data Eng., Vol 5, no 1, Feb 1993, pp. 29-40.

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

[6] I. F. Imam, R. S. Michalski and L. Kerschberg, "Discovering attribute dependence in database by integrating symbolic learning and statistical analysis tests", Knowledge Discovery in Databases Workshop, 1993, pp. 264-275.

[7] J. J. King, "QUIST: A system for semantic query optimisation in relational databases", In Proceeding of the 7 th VLDB Conference, Sept. 1981, pp. 510-517.

[8] B. G. T. Lowden, "An Approach to Multikey Sequencing in an equiprobable keyterm retrieval situation", Proceedings of the Eighth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1985, pp. 92-96.

[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, 68-76, December 1995, pp. 68-76.

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

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

[12] A. Sayli and B. G. T. Lowden, "The use of statistics in semantic query optimisation", Thirteenth European Meeting on Cybernetics and Systems Research, Vienna, April 1996, pp. 991-996.

[13] M. SCHKOLNICK and P. TIBERIO, "Estimating the cost of updates in a relational database", ACM Trans. Database Systems, 10, 2, June 1985, pp. 163-179.

[14] 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.

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

[16] S.T. Shenoy and Z.M. Ozsoyoglu, "Design and implementation of semantic query optimiser", IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 3, Sept. 1989, pp. 344-361.

[17] M.D. Siegel, E. Sciore and S. Salveter, "A method for automatic rule derivation to support semantic query optimisation", ACM Transactions on Database Systems, Vol. 17, No. 4, Dec. 1992, pp. 563-600.

[18] C. Yu and W. Sun, "Automatic knowledge acquisition and maintenance for semantic query optimisation", IEEE Trans. Knowl. Data Eng., 1, 3, Sept. 1989, pp. 362-375.