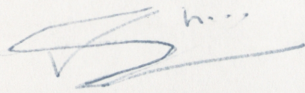


A Blackboard Architecture for a Rule-Based SQL Optimiser

A thesis submitted for the degree of
Master of Science of
The Australian National University

Vikram Sharma
August 1997

I declare that this thesis reports my own original work, that no part of it has been previously accepted or presented for the award of any degree or diploma by any University, and to the best of my knowledge no material previously published or written by another person is included except where due acknowledgment is given.

A handwritten signature in blue ink, appearing to be 'Vikram Sharma', with a long horizontal stroke extending to the right.

Vikram Sharma
Canberra ACT
August 1997

Abstract

In relational database systems the query optimiser plays a critical role in translating a query from its initial form as input by a user into an efficient program which can be executed by the database component which performs the physical retrieval of data. For queries other than the most trivial, there usually exist numerous different possibilities for the sequence in which tables are accessed and the access methods used to retrieve the requested data. It is the role of the optimiser to select a good, and possibly the best, program to execute the query.

In addition to the inherent complexity of the problem, designers of optimisers have to contend with often conflicting requirements such as the need for modularity and extensibility versus the need for efficient execution of the program. Often this leads to designs which compromise on some qualities of the optimiser in order to maximise others. This thesis proposes a model which attempts to address characteristics desirable in a relational query optimiser more completely than contemporary designs.

An architecture for an SQL optimiser which is based on the concept of a blackboard is investigated. The proposed design incorporates a set of rules to perform the transformations necessary to optimise the query. Subsets of these rules are grouped into knowledge sources which operate on the evolving problem solution in an independent manner. A mechanism for the back propagation of the results of optimisations is also incorporated in the design.

The proposed model has been implemented in Aion/DS, a knowledge base development tool. The results of the optimisation of a sample set of queries are examined. The impact of restricting the number of alternatives explored by the optimiser, both on query execution plan quality and optimiser performance, is also investigated.

The thesis concludes with a brief discussion of possible further work including enhancements to the model and automated tuning of the optimiser.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Dr Vicki Peterson, who has patiently guided me throughout this work. Her insight and direction have added greatly to the final outcome of the project.

I would like to thank my wife, Gauri, who has unfailingly supported and encouraged my endeavours. I would also like to express my delight at the arrival of our beautiful daughter, Diya, which provided the catalyst for the completion this work.

1.	INTRODUCTION	1
1.1	BACKGROUND	1
1.2	AREA OF RESEARCH	2
2.	RELATED WORK	3
2.1	BLACKBOARD SYSTEMS	3
2.1.1	<i>Model for Problem Solving</i>	3
2.1.2	<i>The Blackboard</i>	5
2.1.3	<i>Knowledge Sources</i>	6
2.1.4	<i>Control Strategy</i>	8
2.1.5	<i>Conclusions</i>	10
2.2	A MODEL FOR A RULE-BASED SQL OPTIMISER	11
2.2.1	<i>Query Optimisation and Evaluation</i>	12
2.2.2	<i>Source and Target Languages</i>	13
2.2.3	<i>Operators and Functions</i>	14
2.2.4	<i>The Transformation Rules</i>	16
2.2.5	<i>Conclusions</i>	20
2.3	A BLACKBOARD ARCHITECTURE FOR QUERY OPTIMISATION IN OBJECT BASES.....	21
2.3.1	<i>Blackboard Structure</i>	23
2.3.2	<i>Search Strategy</i>	24
2.3.3	<i>Back propagation</i>	25
2.3.4	<i>Conclusions</i>	27
3.	BBQ CONCEPTUAL MODEL.....	28
3.1	OBJECTIVES.....	28
3.2	ARCHITECTURE	28
3.2.1	<i>Transformation Rule Set</i>	30
3.2.2	<i>The Blackboard</i>	30
3.2.3	<i>Cost function and Control Strategy</i>	31
3.2.4	<i>Back Propagation Function</i>	32
4.	BBQ DESIGN	35
4.1	SQL TRANSFORMATION RULES	35
4.2	COST MODEL	37
4.3	SEARCH STRATEGY.....	41
4.4	BACK PROPAGATION OF OPTIMISATION RESULTS.....	42
5.	RESULTS	44
5.1	DATABASE MODEL	44
5.2	RESULTS OF SAMPLE OPTIMISATIONS.....	46
5.3	DISTRIBUTION OF QEP COSTS.....	51
5.4	RESULTS FROM RESTRICTED SEARCHES	56
5.4.1	<i>Limited number of alternatives expanded</i>	57
5.4.2	<i>Increase Search Breadth</i>	59
5.4.3	<i>Increase Search Depth</i>	63
5.4.4	<i>Summary of restricted search runs</i>	64
5.5	CHANGE IN IO TO CPU WEIGHTING	65
5.6	DELTA PROFILES FOR KNOWLEDGE SOURCES	67
5.7	EXAMPLE OF EXTENDING TRANSFORMATION RULE SET	69
5.8	ADVANTAGES OF BLACKBOARD ARCHITECTURE.....	73
5.9	BENEFITS OF SEARCH STRATEGY	74
6.	FUTURE WORK AND EXTENSIONS.....	76
6.1	EXTENSION OF SOURCE LANGUAGE AND TRANSFORMATION RULE SET	76
6.2	SET OF REPRESENTATIVE SQL FOR CALIBRATING BBQ	76

6.3	SELF TUNING OPTIMISER.....	77
6.4	TOOL FOR BENCHMARKING AND TESTING TRANSFORMATION RULES.....	78
6.5	APPLICABILITY TO COMMERCIAL DBMS'	78
7.	CONCLUSIONS.....	80
	APPENDIX A - LISTING OF RULES	82
	APPENDIX B - BBQ INTERNAL DATA STRUCTURES.....	93
	BIBLIOGRAPHY	95

1. INTRODUCTION

1.1 BACKGROUND

In relational database systems the query optimiser plays a critical role in translating a query from its initial form as input by a user into an efficient program which can be executed by the database component which performs the physical retrieval of data. For queries other than the most trivial, there usually exist numerous different possibilities for the sequence in which tables are accessed and the access methods used to retrieve the requested data. In the case of more complex queries, alternative programs may number in the thousands. It is the role of the optimiser to select a good, and possibly the best, program to execute the query. Thus the quality of the optimiser greatly influences the overall performance of a database management system.

As a key component of database management systems, a great deal of research has been devoted to the design of the optimiser. Due to the nature of its problem domain, the optimiser is inherently a very complex piece of software. In addition to the problem complexity, designers of optimisers have to grapple with often conflicting requirements such as the need for modularity and extensibility versus the need for efficient execution of the program. In many cases, this has led to designs which have had to compromise on some aspects of the optimiser in order to maximise the performance of others.

Ideally, the design of an optimiser should attempt to address as many of the following desirable characteristics as possible :

- Early assessment of quality of alternative solutions
- Modularity to facilitate maintenance
- Architecture which supports extension of functionality
- Provision of metrics to aid analysis of performance
- Support of parallel processing

This thesis proposes an architecture for a relational database optimiser which aims to address these characteristics more completely than other contemporary designs.

1.2 AREA OF RESEARCH

An SQL optimiser architecture which is based on the concept of a blackboard, borrowed from the area of artificial intelligence, is investigated. This research draws from several previous works, which are discussed in Chapter 2, and builds upon the concepts presented in those works.

The proposed design incorporates a set of rules to perform the transformations necessary to optimise an SQL query. Subsets of these rules are grouped into knowledge sources which operate on the evolving problem solution in an independent manner. A mechanism for the back propagation of the results of optimisations is also incorporated in the design.

The research aims to investigate the feasibility of the architecture outlined above, construct a software suite to implement the proposed design and examine various characteristics of the model. Specific characteristics to be studied include quality of optimisations, efficiency of the proposed architecture and extensibility of the model. The back propagation of optimisation results as a mechanism for calibrating and tuning knowledge sources is also to be examined.

2. RELATED WORK

This chapter examines research which has been conducted in areas relevant to the topic of this thesis. As part of the study of previous and contemporary work, a number of papers on blackboard systems and optimiser design were consulted. A complete list of these works can be found in the References section.

This chapter is divided into three sections. The first section looks at the concept of blackboard systems and describes two projects which are considered to have originated this model of problem solving. The second section discusses a paper which describes a set transformation rules for the optimisation of relational queries. The third section discusses a paper which employs the blackboard concept coupled with a rule set, derived from the paper presented in the second section, to present a model for query optimisation in object-bases.

2.1 BLACKBOARD SYSTEMS

2.1.1 Model for Problem Solving

An effective method of describing the concept of blackboard systems is by way of analogy to a group of people collectively solving a jigsaw puzzle (Engelmore and Morgan, 1988). The problem domain is the construction of the puzzle from the jigsaw pieces, the group of people working of the puzzle is analogous to a set of knowledge sources solving the problem and the board on which the emerging solution is being constructed is representative of the blackboard.

In the analogy, the pieces of the puzzle are distributed amongst the group of problem solvers. The problem solution commences with each person placing their most promising piece or pieces on the blackboard. As pieces are placed on the blackboard, each group member examines their own pieces and adds new ones that may now fit as a consequence of others having being added. The solution evolves as more and more pieces fit and terminates once all the pieces have been placed.

This model of problem solving provides a number of interesting features and presents the basis of an architecture suitable for a certain class of problems. Some of the points of particular note are :

- No direct communication between the problem solvers is required
- No predetermined sequence is defined for the order in which the problem is solved
- Solution is formed incrementally
- Problem solvers can exhibit opportunistic behaviour
- Distribution of the puzzle pieces amongst group does not greatly affect the problem solution

Many of the concepts of this model of problem solving find their origins in two projects conducted in the 1970's.

The first of these, Hearsay-II, was one of the systems developed at the Carnegie-Mellon University as part of a five year speech recognition project sponsored by the Defence Advanced Research Projects Agency (DARPA). The project commenced in 1971 with three organisations demonstrating systems in 1976. Although Hearsay-II was not the most successful system, it did produce some original software engineering techniques that have general applicability. Hearsay-II was the product of approximately 40 person years of effort and several design iterations.

The second, HASP, was one of the early applications to utilise and extend some of the concepts developed in Hearsay-II. It was developed to identify and track vessels, particularly submarines, using data from concealed hydrophone sensors in the deep ocean. The main feature of the blackboard architecture for HASP was the capability for opportunistic problem solving.

Discussions of these two projects in the sections which follow are extracts from (Engelmore and Morgan, 1988), complete descriptions of the projects can be found in that text.

The blackboard model consists principally of three components, a global solution board containing emerging solution alternatives, a set of knowledge sources which progress the problem solution and a control strategy which determines the sequence of invocation of the knowledge sources.

2.1.2 The Blackboard

The global solution board or blackboard is a structure for storing solution alternatives as the problem is being solved. The solutions on the blackboard are visible to all knowledge sources and cost functions which form part of the control strategy. The blackboard is segmented into levels, each containing partial solutions which are at a similar stage of evolution. While a knowledge source has read access to the entire range of emerging alternatives, it usually operates on alternatives on one level, emitting more developed alternatives to an adjacent “higher” level. In some instances the knowledge source may produce alternatives at the same level as its input.

In the HEARSAY-II project, the blackboard is segmented into levels that correspond to the various stages of speech recognition. Hypotheses at each level have a unique identifier and are tagged with additional information including time within the spoken sentence and credibility ratings. The levels of the blackboard form a hierarchical structure with each higher level aggregating elements of lower levels. A diagram of the architecture of HEARSAY-II is given in the next section.

An important component of the HASP architecture is its model of the current ocean scene known as the Situation Board. This describes the state of the geographical area of interest and provides a reference model for the interpretation of new information, assimilation of new events and generation of expected future events.

The problem of understanding the state of the ocean is organised into a hierarchy of blackboard levels with the highest one corresponding to the Situation Board and the lowest level consisting of sonagram data from ocean sensors. A diagram of these blackboard levels and some the knowledge sources operating between them is presented in the next section.

2.1.3 Knowledge Sources

Knowledge sources are program units which progress the problem solution by generating partial, and ultimately complete, solution alternatives. Each knowledge source has access to all the partial alternatives that have been produced prior to its invocation. Upon invocation a knowledge source will typically use a partial solution at one level to generate one or more alternatives at an adjacent higher level on the blackboard.

In Hearsay-II, the knowledge sources are implemented as independent programs which perform the functions of generating, merging and evaluating hypotheses. Although the nature of knowledge sources varies greatly, due to the differing problem domains of Hearsay-II components, each is represented by a condition-action tuple. The condition specifies the situations in which the knowledge source may be able to contribute to the solution and the action specifies what the contribution is and how this can be integrated into the evolving solution.

The condition part of each knowledge source searches through existing alternatives on the blackboard searching for conditions where it may be appropriate to apply the action part of the knowledge source. In Hearsay-II, each condition program declares a set of primitive conditions in which it may be applicable and is only invoked if changes to the problem solution trigger these conditions. This improves efficiency as it minimises the evaluation of condition programs and changes the architecture from polling to interrupt driven.

The diagram below shows the main components of the architecture of Hearsay-II. Functions implemented by the knowledge sources include extracting acoustic parameters, classifying acoustic segments into phonetic classes, recognising words, parsing phrases and generating and evaluating predictions for undetected words or syllables. Each of these knowledge sources use partial solutions at one level to generate one or more alternatives which are placed on the adjacent higher level. Partial solutions at all levels of the blackboard are accessible by the Blackboard Monitor

knowledge source which interacts with the Focus of Control program. This program directs the Scheduler on the selection of the next knowledge source to be invoked.

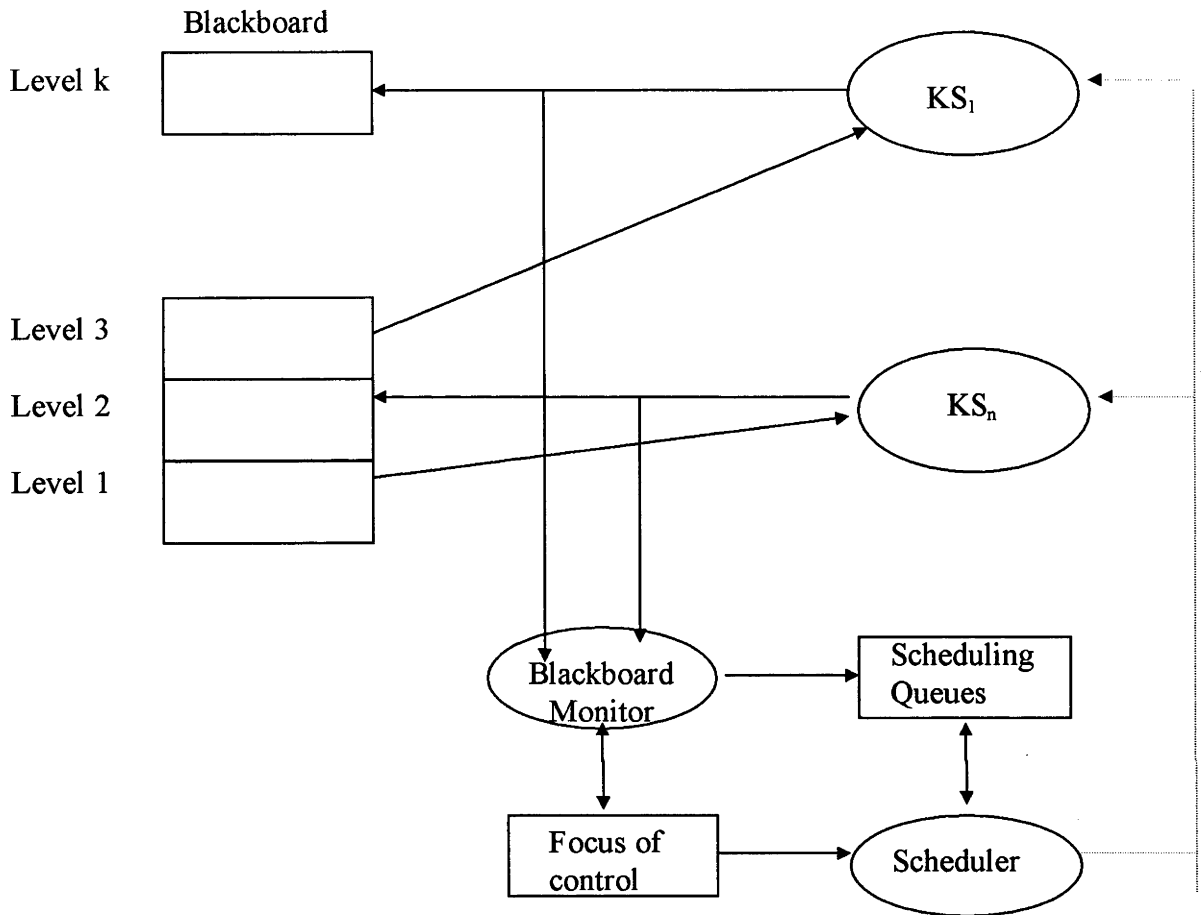


Diagram 1 - Hearsay-II Architecture (Figure 3.3 in (Engelmore and Morgan, 1988))

In the HASP system, alternative generation is opportunistic and is both data-driven and model-driven. Control knowledge sources determine the most appropriate knowledge source to invoke at each step of the problem solution. Modifying the analysis strategy involves changes only to the control knowledge sources.

Diagram 2 illustrates the blackboard levels (on the left) and some of the knowledge sources (on the right) in HASP. Knowledge sources use one or more hypothesis elements at one level to infer hypotheses at other levels, these are shown as links

between the levels in the diagram. Where the knowledge source makes inferences in one direction only these links are represented by directed lines.

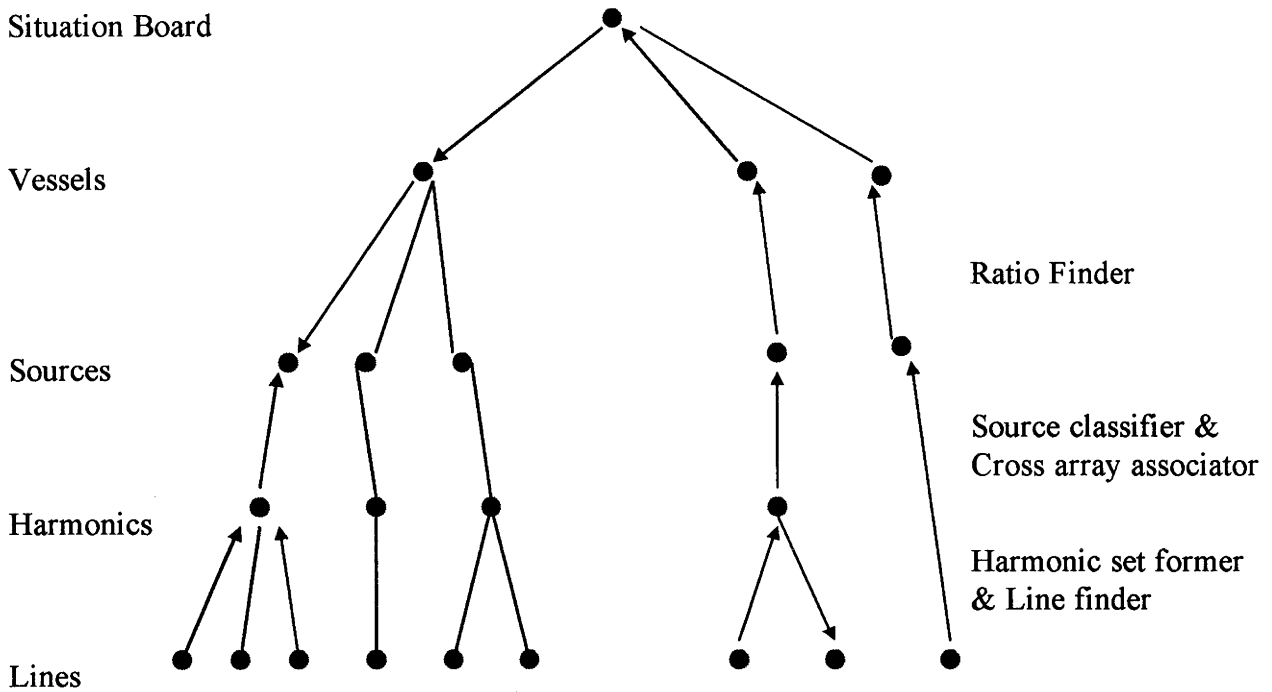


Diagram 2 - HASP blackboard levels and KSs
(Figure 6.2 in (Engelmore and Morgan, 1988))

Specialist knowledge sources are responsible for generating new hypotheses and/or modifying existing ones. Their focus of attention is usually a hypothesis that has recently changed. Although a knowledge source has access to all hypotheses, it normally operates only on hypotheses contained in its input and output levels.

2.1.4 Control Strategy

As there is limited procedural control in most blackboard systems, the control strategy is responsible for the selection of the next knowledge source to be executed. It also selects the partial solution on which this knowledge source should operate. The selection of the knowledge source and partial solution alternative is often predicated on a cost metric which estimates both the cost of operations already incorporated in the partial solution and the cost of operations which are yet to be incorporated. The

architecture of blackboard systems allows segregation of the components which implement the control strategy from the remainder of the program units. Thus changes to the control strategy may be effected with relative ease and impact only the control strategy components.

In Hearsay-II, the sequence of activation of knowledge sources is determined purely by the state of the problem solution as described by the hypotheses on the blackboard. The system exhibits opportunistic behaviour as it is able to invoke the knowledge source that is most likely to be appropriate to each stage of the problem solving process.

This requires an evaluation of three metrics (Engelmore and Morgan, 1988) :

1. the probable effects of invocation of a knowledge source
2. significance of the actions by an analysis of its cooperative and competitive relationships with existing hypotheses
3. the relative value of invoking a knowledge source versus the other potential candidates

Hearsay-II incorporates these metrics in a heuristic scheduler which calculates a priority for all candidate knowledge sources and invokes the knowledge source with the highest priority rating.

The control strategy of HASP is implemented by KS-Activators, which know when to invoke particular Specialist knowledge sources, and the Strategy-KS, which determines the “focus of attention”. One execution cycle consists of the following steps (Engelmore and Morgan, 1988) :

1. Focusing attention on one of : time-dependent activities, verification of hypotheses or one of the hypothesised elements
2. Choosing the most appropriate knowledge source for the focus of attention
3. Invoking the selected knowledge source

KS-Activation knowledge sources perform the task of selecting Specialists according to the kind of problem solving strategy being employed. Thus a model-driven strategy would have a different goal to a data-driven strategy. Two other important factors considered by the control strategy are the efficiency and the accuracy of each Specialist.

The high level Strategy-KS mirrors the problem solving strategy of a human analyst. It determines the accuracy of the CBH and selects the task that will have the greatest impact on the current problem state.

2.1.5 Conclusions

The blackboard model offers an alternative problem solving paradigm and possesses characteristics which make it well suited to certain classes of problem.

Classes of problems to which the blackboard model would be suited according to (Engelmore and Morgan, 1988) include those where :

- Large amounts of signal data are to be analysed
- Heuristics are applied to interpret data
- Problem domain inherently possesses a hierarchical structure
- Opportunistic strategies may be used to advantage

Some of the advantages of this model for problem solving as listed in (Engelmore and Morgan, 1988) are :

- Multiple sources of knowledge allow incorporation of diverse types of knowledge.
- Multiple levels of abstraction in the global blackboard structure allow for representation of the problem at several different levels.
- Knowledge sources can represent knowledge in a consistent format and share partial results.

- Interaction between knowledge sources is limited to the changes that each makes to the data on the blackboard. This means that each knowledge source can be developed independently and without any description of the others allowing for a high degree of modularity.
- Solutions are formed incrementally with lower level hypotheses integrated into larger and more credible composites as part of the problem solving process.
- Opportunistic behaviour exploits the most promising alternative(s) to which the most significant addition can be made.
- Control strategy is flexible with changes to search method (eg. depth-first, breadth-first, left-to-right etc.) requiring modifications only to the control knowledge source.

For certain types of problem however, a blackboard architecture can have significant disadvantages. The calculation of a cost metric for partial solution alternatives may be computationally expensive. The process of selecting the next knowledge source to trigger and the partial alternative to expand imposes an overhead on the search for a solution. And lastly, the need for data structures which are globally visible can lead to complex blackboard structures.

2.2 A MODEL FOR A RULE-BASED SQL OPTIMISER

The query optimiser is the component of a database management system which generates a Query Execution Plan (QEP) to efficiently compute the result of a user-submitted query. The non-trivial task of finding a good QEP has led to sophisticated, however, complex implementations of optimisers. In these implementations, changes or extensions of functionality are often difficult and time-consuming.

This section describes a basis for a modular query optimiser, presented in (Freytag, 1987), which is designed to alleviate some of the problems associated with the

inflexibility of traditional query optimiser implementations. The proposed architecture attempts to clearly separate different aspects of the optimisation process thereby reducing the inter-dependence between components of the optimiser. The basis of the design is a set of transformation rules which convert the user-submitted query into an algebraic QEP. These transformation rules provide an implementation independent description of the steps required to generate the QEP. This design for an optimiser is amenable to change and facilitates extensions to the set of possible QEPs produced.

2.2.1 Query Optimisation and Evaluation

The paper identifies three processes for the conversion of a query from a format input by a user to a program which can be executed by a database management system to retrieve the data. The processes are - validation, optimisation and translation. The validation phase checks for semantic and syntactic correctness, view resolution and possibly checks authorisation prior to generating an internal representation of the query. The optimisation phase uses information about the physical representation of the data to be accessed and available evaluation strategies to generate a query execution plan. The final phase, translation, transforms the query execution plan into a representation suitable for efficient execution. The focus of the paper is the second of these processes, the generation of a query execution plan.

A general non-procedural query representation is selected as the source language and an extended relational algebra is the target language. The rule-based transformations described are central to the optimisation algorithm. The algorithm selected is sufficiently complex to illustrate the power of rule-based description. There are two other important aspects of the optimisation process which are not in the scope of the paper. The first is the selection of a search strategy to define how to search through alternative QEPs, examples of possible search strategies are breadth-first, depth-first and k-step look-ahead. The second is the selection of a cost function to compare alternative QEPs to determine which ones are better than others. This function may be dependant on the cost of using various resources such as CPU time, number of I/O operations and number of messages.

The stated objective of the paper is to present a rule-based description for the generation of QEP alternatives from an initial query specification in an implementation independent manner.

2.2.2 Source and Target Languages

The source language selected for the purposes of the paper allows conjunctive queries which exclude sub-queries and aggregates. This selection is only to limit the scope of the paper and not due to any fundamental limitations of rule-based transformation. To demonstrate this the paper includes a discussion of how the rule set can be readily extended to cater for more complex query formulations.

The query input to the optimisation phase is assumed to have the following form :

```
SELECT    <project_list>
          <select_pred_list>
          <join_pred_list>
          <table_list>
```

The lists respectively describe the projection of the result tuples, predicates applicable to individual tables only, join predicates and tables accessed.

Query 1 as defined in section 5.2, is used as an example to illustrate the effect of the transformations which follow. Represented in the form described above, this query is specified as :

```
SELECT    (PortfolioItem.ASXCode, PortfolioItem.NumOfUnits)
          (Client.ClientName = "Vikram Sharma")
          (PortfolioItem.ClientId = Client.ClientId)
          (Client, PortfolioItem)
```

The target language is an extended form of relational algebra as some of the operations required are not available in traditional algebra. All the operators defined manipulate a

list of tuples which is derived from either a relation referenced by name or the output of another operation.

2.2.3 Operators and Functions

The operators defined do not allow transformation of all SQL queries, however they are sufficient to transform the selected source language. An extension of this set of operators to process a more complex source language is shown to be feasible. Freytag defines the following operators :

(FSCAN <t_pred> rel) - scans a relation while applying the list of predicates in <t_pred>, which may be empty.

(ISCAN <i_pred> index <t_pred> rel) - scans a relations using the index to apply the index predicates <i_pred> before scanning the table and applying the predicates <t_pred>. One or both predicate lists may be empty.

(PROJECT <proj_list> tuple_list) - projects tuples in tuple_list onto the attributes specified in <proj_list>.

(LJOIN <join_pred> list1 list2) - denotes a nested-loop join with list1 being the outer list and list2 the inner.

(MJOIN <join_pred> list1 list2) - denotes a merge-join with list1 being the outer list and list2 the inner.

(SORT <attr_list> tuple_list) - sorts tuple_list according to the order specified by attr_list.

Some additional operators are defined to generate intermediate steps required in the course of the transformation.

(SCAN <sel_pred> rel) - scans a relation without specifying the access path.

(JOIN <join_pred> <scan_list>) - joins an arbitrary number of relations without specifying their order.

(TJOIN <join_pred> list1 list2) - denotes a two-way join without specifying the type of join.

Each step of the optimisation process is described by a transformation or rewrite rule. A transformation rule ($t_1 \rightarrow t_2$) specifies the replacement of t_1 by t_2 . This notation is extended to introduce restricted rules. These are of the form ($t_1 \overset{C}{\rightarrow} t_2$) signifying that t_1 is to be replaced by t_2 if condition C evaluates to true. A restriction on C is that it may only access variables in expressions t_1 and t_2 .

The notation ($\dots t_i \dots$) denotes zero or more subexpressions to the left and right of some t_i .

Restricted rules often use functions in their conditions to determine properties of a relation, predicate or general expression. Functions defined for the transformation rules are :

Ind(I1, R) - determines if I1 is an index on relation R.

T(p) - denotes the set of relation names referenced in p.

A(t, RS) - denotes the set of attributes of the relations in RS which are referenced in t.

O(I) - returns an attribute list determining the ordering of rows retrieved from a relation using index I.

$\Omega(Q)$ - denotes the order in which tuples are retrieved by a QEP Q. This function is defined recursively as follows :

$$\Omega((\text{FSCAN } \langle p \rangle \text{ rel})) = \langle \rangle$$

$$\Omega((\text{ISCAN } \langle ip \rangle \text{ ind } \langle tp \rangle \text{ rel})) = O(\text{ind})$$

$$\Omega((\text{PROJECT } \langle pr \rangle \text{ list})) = \Omega(\text{list})$$

$$\Omega((\text{LJOIN } \langle jp \rangle \text{ list1 list2})) = \Omega(\text{list1})$$

$$\Omega((\text{MJOIN } \langle jp \rangle \text{ list1 list2})) = \Omega(\text{list1})$$

$$\Omega((\text{SORT } \langle a_list \rangle \text{ list})) = \langle a_list \rangle$$

Note : $\langle \rangle$ in the definition of the function applied to the FSCAN operator denotes that no order can be ascertained.

2.2.4 The Transformation Rules

The first step of the optimisation requires a translation of the query from its initial form to an algebraic form for further manipulation.

The following two rules create a list of SCAN operators attached to each relation in the input query.

$$((\text{SELECT } e_1 e_2 (\dots t_1 \dots)) \rightarrow (\text{SELECT } e_1 e_2 (\dots \dots) ((\text{SCAN } () t_1))))$$

$$((\text{SELECT } e_1 e_2 (\dots t_1 \dots) (\dots \dots)) \rightarrow (\text{SELECT } e_1 e_2 (\dots \dots) (\dots (\text{SCAN } () t_1) \dots)))$$

For the example query defined in section 2.2.2, these transformations result in following expression :

```
SELECT      (PortfolioItem.ASXCode, PortfolioItem.NumOfUnits)
            (Client.ClientName = "Vikram Sharma")
            (PortfolioItem.ClientId = Client.ClientId)
            ((SCAN () Client), (SCAN () PortfolioItem))
```

Next the selection predicates are distributed amongst the SCAN expressions depending on the relation name accessed in each predicate.

$$((\text{SELECT } e_1 (\dots p_1 \dots) e_2 (\dots (\text{SCAN}(\dots \dots) t_1) \dots)) \xrightarrow{C1} (\text{SELECT } e_1 (\dots \dots) e_2 (\dots (\text{SCAN}(\dots p_1 \dots) t_1) \dots)))$$
$$C1 = (T(p_1) = \{t_1\})$$

Applying this transformation to the example query produces :

```
SELECT      (PortfolioItem.ASXCode, PortfolioItem.NumOfUnits)
            (PortfolioItem.ClientId = Client.ClientId)
            ((SCAN (Client.ClientName = "Vikram Sharma") Client),
             (SCAN () PortfolioItem))
```

Once the selection and relation lists are empty, an n-way join is created followed by a projection for the resultant expression.

$$((\text{SELECT } e_1 () e_2 () e_3) \rightarrow (\text{PROJECT } e_1 (\text{JOIN } e_2 e_3)))$$

After this transformation the example query takes the following form :

```
(PROJECT (PortfolioItem.ASXCode, PortfolioItem.NumOfUnits)
         (JOIN (PortfolioItem.ClientId = Client.ClientId)
              ((SCAN (Client.ClientName = "Vikram Sharma") Client),
               (SCAN () PortfolioItem))))
```

The second step of the optimisation involves generating alternatives for accessing individual relations, in particular selection predicates which might be evaluated using indexes.

The first transformation below converts a generic scan to a full relation scan without using any indexes. The second applies an index scan if a subset of the predicate list can form a prefix of the attribute list denoting the index order.

$((SCAN\ p_1\ t_1) \rightarrow (FSCAN\ p_1\ t_1))$

$((SCAN\ p_1\ t_1)^{C1} \rightarrow (ISCAN\ p_1' \ I1\ p_1''\ t_1))$

$$C1 = (Ind(I1, t_1) \wedge (p_1 = (p_1' \cup p_1'') \wedge (p_1' \cap p_1'' = \emptyset) \wedge (A(p_1', t_1) \in O(I1))))$$

One possible alternative generated by the above transformations for the example query is :

(PROJECT (PortfolioItem.ASXCode, PortfolioItem.NumOfUnits)
 (JOIN (PortfolioItem.ClientId = Client.ClientId)
 ((ISCAN (Client.ClientName="Vikram Sharma") ClientName () Client),
 (FSCAN () PortfolioItem))))

The next step of the optimisation process involves exploring different join orders among the tables involved as well as choosing join methods.

The first rule selects a relation at random to be the outer expression of a join. The second rule successively creates two-way joins by taking any relation and combining it with the two-way join expression created so far. The final rule discards the n-way join operator when both the predicate and relation lists are empty. Between them, these rules can generate a number of alternative join orders for a given expression.

$((JOIN\ t_1\ (... \ t_2\ ...)) \rightarrow (JOIN\ t_1\ (... \ ...) \ t_2))$

$((JOIN\ (... \ p_1\ ...) \ (... \ t_1\ ...) \ t_2)^{C1} \rightarrow (JOIN\ (... \ ...) \ (... \ ...) \ (TJOIN\ (p_1)\ t_2\ t_1)))$

$$C1 = (T(p_1) \in T(t_1) \cup T(t_2))$$

$((JOIN\ () \ () \ t_1) \rightarrow t_1)$

The rules listed above are not sufficient to completely reduce a list of join predicates. To process cyclic queries and ones in which two relations are joined by

more than one predicate, two further rules which push the remaining predicates into two-way joins are required :

$$((\text{JOIN } (\dots p_1 \dots) t_1 (\text{TJOIN } (\dots) t_2 t_3))^{C1} \rightarrow \\ (\text{JOIN } (\dots \dots) t_1 (\text{TJOIN } (\dots p_1 \dots) t_2 t_3)))$$

$$C1 = (T(p_1) \in T(t_2) \cup T(t_3))$$

$$((\text{TJOIN } (\dots p_1 \dots) (\text{TJOIN } (\dots) t_1 t_2) t_3)^{C2} \rightarrow \\ (\text{TJOIN } (\dots \dots) (\text{TJOIN } (p_1 \dots) t_1 t_2) t_3))$$

$$C2 = (T(p_1) \in T(t_1) \cup T(t_2))$$

One possible alternative generated by the above transformations for the example query is :

```
(PROJECT (PortfolioItem.ASXCode, PortfolioItem.NumOfUnits)
  (TJOIN (PortfolioItem.ClientId = Client.ClientId)
    ((ISCAN (Client.ClientName="Vikram Sharma") ClientName () Client),
      (FSCAN () PortfolioItem))))
```

The final step involves the selection of a join method to implement each two-way join. The paper implements transformations for nested-loop joins and merge joins. The first rule below transforms a two-way join into nested-loop join. The second and third rules generate a merge join, applying the SORT operator to the outer expression if required.

$$((\text{TJOIN } p_1 t_1 t_2) \rightarrow (\text{LJOIN } p_1 t_1 t_2)) \\ ((\text{TJOIN } p_1 t_1 t_2)^{C1} \rightarrow (\text{MJOIN } p_1 t_1 t_2))$$

$$C1 = (A(p_1, T(t_1)) \in \Omega(t_1))$$

$$((\text{TJOIN } p_1 t_1 t_2)^{C2} \rightarrow (\text{MJOIN } p_1 (\text{SORT } A(p_1, T(t_1)) t_1) t_2))$$

$$C2 = (A(p_1, T(t_1)) \notin \Omega(t_1))$$

Finally, in the case of a merge join, a transformation to ensure that the two sets of tuples being merged are in a compatible order is defined. The SORT operator is only applied to the inner expression when the order of tuples of the outer and inner expressions is not the same.

$$((\text{MJOIN } p_1 \ t_1 \ t_2) \xrightarrow{C1} (\text{MJOIN } p_1 \ t_1 \ (\text{SORT } A(p_1, T(t_2)) \ t_2)))$$

$$C1 = ((\Omega(t_2) \notin \Omega(t_1)) \wedge (\Omega(A(p_1, T(t_2))) = \Omega(t_1)))$$

Two possible alternatives generated by the application of the above transformations to the example query are :

```
(PROJECT (PortfolioItem.ASXCode, PortfolioItem.NumOfUnits)
  (LJOIN (PortfolioItem.ClientId = Client.ClientId)
    ((ISCAN (Client.ClientName="Vikram Sharma") ClientName () Client),
      (FSCAN () PortfolioItem))))
```

```
(PROJECT (PortfolioItem.ASXCode, PortfolioItem.NumOfUnits)
  (MJOIN (PortfolioItem.ClientId = Client.ClientId)
    ((SORT (Client.ClientId)
      ISCAN (Client.ClientName = "Vikram Sharma") ClientName () Client),
      (SORT (PortfolioItem.ClientId) FSCAN () PortfolioItem))))
```

2.2.5 Conclusions

The paper presents a set of rules which effect the necessary transformations to convert a query from the input language into a query execution plan of fundamental operations on the relations accessed in the query. Adoption of such an approach in the design of an optimiser allows a clear separation between the components which transform the input into a QEP and those which control the process of QEP generation. Additionally, the set of rules is extensible, facilitating the incorporation of new functionality.

2.3 A BLACKBOARD ARCHITECTURE FOR QUERY OPTIMISATION IN OBJECT BASES

This section describes a model presented in (Kemper et al., 1993) which proposes the adoption of a blackboard architecture for an object-oriented database query optimiser.

The blackboard problem solving model is coupled with a set of rule-based transformations derived from the paper discussed in the previous section (Freytag, 1987) to present a new architecture for an object base query optimiser. The proposed architecture possesses two desirable characteristics - firstly, the design enables the back propagation of optimised queries to allow evolutionary improvement in optimiser performance and secondly, a modified version of A* search can be selectively applied to control the search space.

Query optimisers, whether for relational or object databases, are very complex pieces of software and much research is still devoted to their design. The qualities which are desirable in a query optimiser, as listed in (Kemper et al., 1993), are :

1. **extensibility and adaptability** : the architecture should facilitate design change as new optimisation techniques or index structures are developed.
2. **evolutionary improvability** : it should be possible to tune the optimiser after gathering data from a sequence of queries which have been optimised. The ultimate goal being a self-tuning optimiser which is able to automatically improve the quality of its optimisations based on previous results.
3. **predictability of quality** : as there is often a trade-off between the time used for the optimisation and the quality of the result, it would be useful to be able to predict the quality of the result relative to the time allocated for the optimisation.
4. **graceful degradation under time constraints** : given a time constraint, the quality of the optimised result should degrade gracefully.

5. early assessment of alternatives : the performance of the optimiser to a large extent depends on the number of alternatives generated and typically, heuristics are used to restrict search space. A more flexible approach is to abandon less promising alternatives as early as possible. For this a cost model which estimates the potential quality of each alternative at an early stage in the optimisation process is required.
6. specialisation : the optimiser design should support the incorporation of specialised knowledge to deal with particular parts of the optimisation process or to deal with specific sub-classes of queries.

Numerous optimiser designs have been proposed to try to incorporate some of the above qualities. However it is often the case that in order to maximise some qualities others are neglected. For example, while rule-based systems place emphasis on extensibility, an estimation of the quality of the result in relation to optimisation time allocated becomes difficult.

The paper proposes an architecture which segments the query into building blocks consisting of fundamental operations in order to construct a query execution plan in a well-structured fashion. A cost model is central to the proposed design. As it is not generally obvious which transformation will lead to the optimal solution, alternatives are generated. The alternatives are graded by an expected cost function and this cost has to be improved as each alternative is developed.

The architecture presented tries to address each of the previously listed characteristics desirable of an optimiser. It is based on the blackboard model which facilitates a bottom up building block design and early assessment of alternatives by utilising future cost estimates.

2.3.1. Blackboard Structure

The blackboard is organised in r successive regions $R_0..R_{r-1}$ each containing a set of items which represent alternatives being generated by the optimiser in the search for an optimal query execution plan.

The original query is translated into an internal form and is placed into region R_0 as the only item. A knowledge source KS_i is associated with a pair of successive regions (R_i, R_{i+1}) . Each knowledge source KS_i retrieves items from region R_i and emits, in an order which it determines, one or more alternatives to the region R_{i+1} . There is no restriction on the data which the knowledge source may read, it may come from any region and may include database statistical data, schema definition and indexing information.

Each alternative emitted by a knowledge source has the name of the knowledge source and the next sequence number for that knowledge source added to an identification tag. Thus any item on the blackboard can be uniquely identified and a history of its derivation can be easily determined. This feature is essential for evaluation of quality and calibration of knowledge sources.

The diagram below illustrates blackboard levels, knowledge sources operating between these levels and examples of identification tags of alternatives at each level.

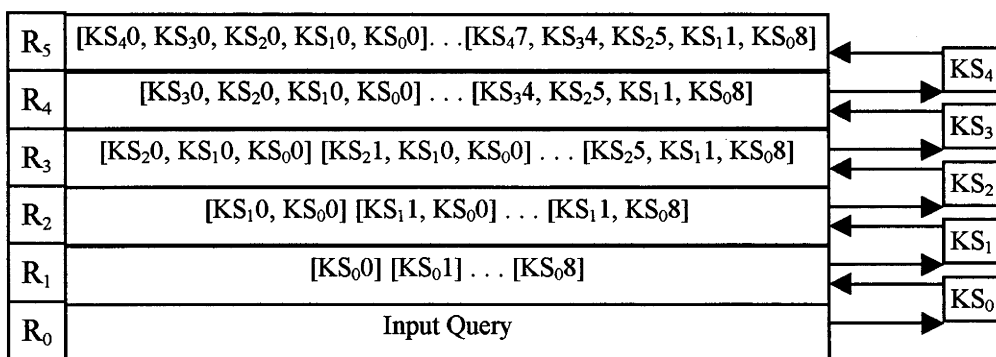


Diagram 3 - Blackboard Architecture (Figure 1 in (Kemper et al., 1993))

2.3.2 Search Strategy

As previously described a building block approach is used for query execution plan generation. Thus successive regions of the blackboard contain more and more complete query execution plan alternatives. One or more complete query execution plans are contained in the final region R_{r-1} .

To avoid an exhaustive and expensive search of all possible alternatives a cost is computed for each item on the blackboard. Two cost functions, $cost_h$ and $cost_f$ have been defined to compute the historical and future costs respectively for an item. $Cost_h$ determines the cost of operations already incorporated into the QEP and $cost_f$ estimates the cost of operations which have yet to be integrated. The sum of these two costs is used to drive an A* search, the knowledge source which is applicable to the item with the lowest total cost is allowed to generate further alternatives for that item.

For A* to operate efficiently, $cost_f$ should represent a close lower bound on future costs. However, for query optimisation a lower bound estimate for future operations is always based on the best case for each operation. Therefore the estimate can be considerably lower than the actual cost of those operations. This could lead A* to degenerate into an almost exhaustive search and lead to unacceptably long optimisation times. To overcome this potential problem, a variation of the A* strategy has been proposed.

One of the characteristics desirable of the knowledge sources is that they emit more promising alternatives early in the optimisation process. To take advantage of this feature A* is modified to periodically and temporarily switch off A* control and process the first few alternatives without any cost control. Under this regime some promising alternatives will progress through successive blackboard regions and possibly to the top-most region where they would represent complete query execution plans. When A* control is resumed items which were generated in intermediate steps are discarded which has the effect of "straightening" the optimisation. This strategy allows the search to process some promising alternatives efficiently without backtracking. A degree of control over the trade off between optimisation time and

quality of the result can be obtained by varying the periodicity A* disablement and the number of alternatives which are fully expanded during this time.

2.3.3 Back propagation

The architecture of the optimiser, in particular the use of an identification tag to trace the derivation history of alternatives, supports the collection of metrics for quantitative evaluation and subsequent calibration of the knowledge sources.

This is achieved by back propagating the results of optimising an extensive set of benchmark queries. For this set of benchmark queries, the optimiser is run under pure A* control except that it is allowed to continue to generate alternatives even after the optimum alternative has been generated. Once the run has completed the top-most region will contain a list of complete query execution plan alternatives for each benchmark query. Due to the nature of the search strategy each list of QEP alternatives is already sorted in order of increasing cost.

To determine the quality of knowledge sources, results from these ordered lists are back propagated to each knowledge source. The quality of a knowledge source is measured in terms of a comparison of the sequence number at which the knowledge source produced its contribution to the QEP alternative versus the position of this QEP alternative in the list of QEPs ordered by running times. Using the results from a representative set of queries, a Top-Rank profile as shown below can be derived for each knowledge source.

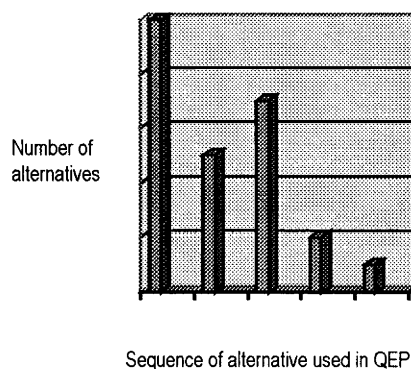


Diagram 4 - Top-Rank Profile

Quantitative analysis of the profiles allows prediction of the average quality of the solution. Statistical functions can be derived fairly easily to compute the probability that the optimal solution will be amongst a certain number of alternatives generated.

Additional quantitative analysis of the profiles enables tuning of individual knowledge sources. The paper gives examples of the different types of delta profiles.

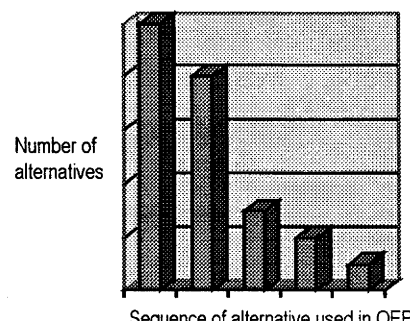
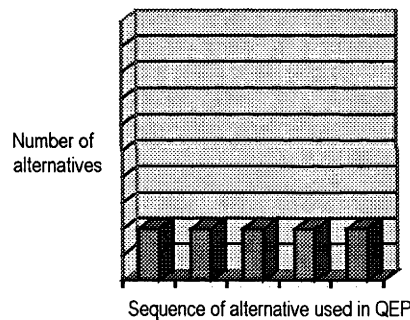
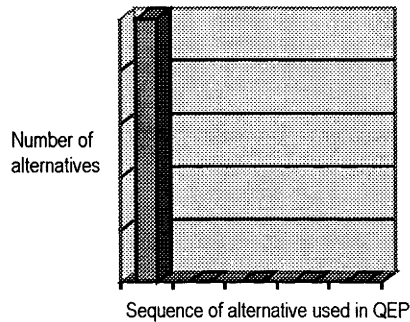


Diagram 5 - Different types of profiles

An ideal profile is depicted at (a) above, no further improvement is possible since the knowledge source always generates the optimal alternative the first time. Profile (b) is

the worst possible and indicates that the knowledge source seems to produce alternatives at random. In practice it is worthwhile trying to achieve the profile shown at (c) where the optimal and semi-optimal alternatives are contained in the first few generated. The ultimate objective of this design is that the optimiser may be able to utilise this information for self-tuning.

2.3.4 Conclusions

A novel architecture for optimiser design has been proposed. It utilises a blackboard structure and knowledge sources which carry out a finite set of algebraic operations to derive a query execution plan. Due to its structured design the optimiser can be continually improved and readily extended. The use of back propagation of optimisation results allows evaluation and calibration of the knowledge sources. This facilitates the identification and possible elimination of weak points in the optimiser.

3. BBQ CONCEPTUAL MODEL

This chapter provides a conceptual view of the architecture of the proposed Blackboard-Based Query optimiser (BBQ). The first section lists objectives of the proposed architecture and the second describes each of the components of the model.

3.1 OBJECTIVES

As discussed in the previous chapter the design of an optimiser has to attempt to satisfy conflicting characteristics. Often, this leads to compromise on some features in order to maximise others. The model presented in this thesis is an attempt to develop an architecture which minimises the compromises required between desirable characteristics. Specific requirements set out for the design are :

- A modular architecture to enhance clarity of design
- The ability to easily add or change optimiser transformation rules to incorporate new query evaluation techniques or database structures
- Efficient performance via the use of a search strategy and a cost model which allows early selection of promising query execution plan (QEP) alternatives
- Collection of a set of metrics on the performance of components of the optimiser to allow calibration and enhancement
- A framework which enables the use of results of past optimisations in order to improve future ones
- A design which lends itself to parallel processing

3.2 ARCHITECTURE

This section describes, at a conceptual level, the components which comprise the design. The major components are : a set of rules which transform a query from its initial form into a query execution plan, a search strategy and cost model to drive the search for solutions, a blackboard structure which contains alternative emerging execution plans and a mechanism for the back propagation of optimisation results.

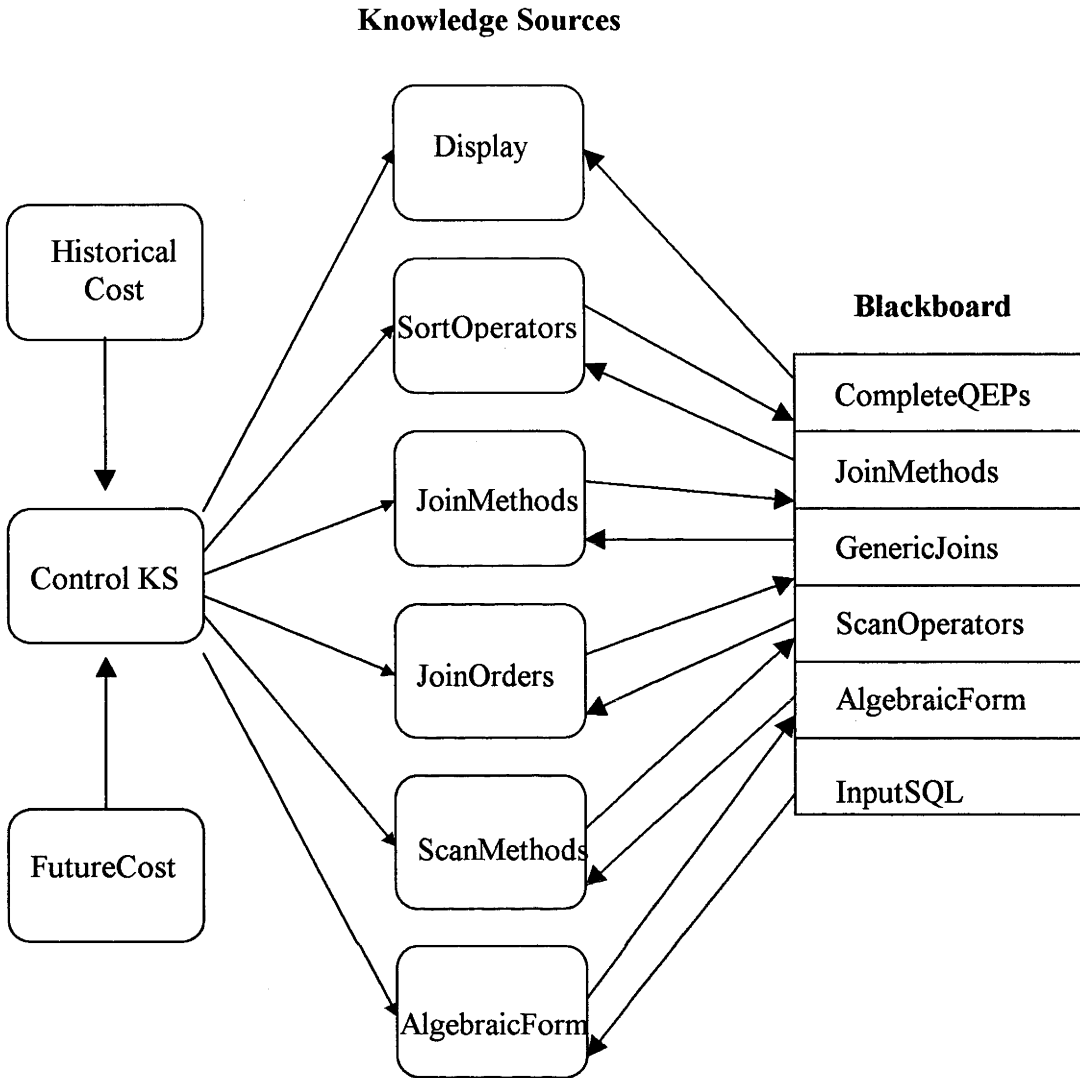


Diagram 6 - BBQ architecture showing interactions between Knowledge Sources and Blackboard levels

Knowledge sources which implement the transformation rules operate on solution alternatives on the blackboard with the highest level of the blackboard ultimately containing one or more complete QEPs. The cost functions and control strategy determine the next knowledge source to trigger and the termination condition of the search. The back propagation function allows an assessment of the performance of knowledge sources and provides metrics on each to facilitate tuning. These components of the model are described in more detail in the sections which follow.

3.2.1 Transformation Rule Set

The transformation rule set translates the query from its initial input form into an ordered sequence of fundamental operations on the relations being accessed. Using a rule-based approach facilitates meeting three of the design objectives : modularity of design, extensibility of function and support of parallel processing.

The rule set used in BBQ has been adapted from the rules described in Section 2.2. The applicability of a rule at any point in the execution sequence is evaluated by examining the blackboard for emerging partial solutions which would satisfy the condition component of the rule. Selection of the rule to trigger and the partial solution to expand are dependent on the control strategy.

The set of rules has been segregated such that each subset of related rules has been assembled into a knowledge source. This segregation of the transformation rules resulted in the following knowledge sources : AlgebraicForm, ScanMethods, JoinOrders, JoinMethods and SortOperators.

3.2.2 The Blackboard

The blackboard is the structure which houses the emerging alternatives as the search for solutions progresses. Some of the concepts presented in (Kemper et al., 1993) have been adapted as the basis for the architecture of the blackboard. This structure is conceptually segregated into six levels, each level containing alternatives which are at differing stages of evolution :

- InputSQL
- AlgebraicForm
- Scan Operations
- Generic Joins
- Join Methods
- Complete QEPs (Sort Operators)

Knowledge sources, which generate alternatives that progress the problem solution, take items at one level as input and emit one or more alternatives to the same or adjacent level. The data structures which implement the blackboard are described at Appendix B.

As discussed in Section 2.1 the use of the blackboard simplifies the problem control strategy and eliminates the need for communication between knowledge sources. This leads to modular architecture which is conceptually clear. The functionality implemented by each knowledge source is readily visible and therefore may be easily changed or enhanced. Also, extension of functionality is facilitated as the effect of the addition of a new knowledge source can be quickly assessed since the impact is limited to its interactions with the blackboard.

3.2.3 Cost function and Control Strategy

The cost function and the control strategy are critical parts of the optimiser which have a direct impact on its performance. The cost function used in BBQ have been adapted from that presented in (Kemper et al., 1993) which is discussed in Chapter 2.

The cost function consists of two components, one representing the historical cost of operations already incorporated into the partial solution and the other an estimate on a lower bound of the cost of operations which are yet to be incorporated.

Calculation of the historical cost component makes use of information on cardinality of tables and selectivity of columns from the database. The cost function attempts to estimate the number of IO operations and the number of CPU operations required to retrieve the data specified by the partial QEP. A weighting factor is applied to the CPU cost which is then summed with the IO cost to derive a composite cost metric.

An estimate of a lower bound on future cost is derived by using a best case scenario on IO cost using selectivity and cardinality data for the relations which yet to be incorporated into the emerging QEP.

The sum of historical and future costs is used by the heuristic of the control algorithm to select the next knowledge source to invoke and to select a partial QEP for expansion. The strategy used to drive the search process is a modified version of A* search. It uses a combination of procedural and A* control to efficiently generate QEP alternatives which are expected to be close to the optimal solution.

The search commences by invoking the knowledge source which converts the query from its initial representation to an internal form which is placed at the second level of the blackboard. Next all possible access paths, full scan and indexed, for the relations in the query are generated. The subsequent three steps are performed iteratively with the number of alternatives expanded under A* control in each step limited by a tunable parameter. The iteration cycle terminates once the lowest cost solution produced in the iteration exceeds the lowest cost solution of all iterations thus far by more than a predefined factor. This termination factor is also tunable.

The steps in the iterative process include generation of n-way joins, generation of two-way joins, selection of join methods and introduction of sort operators.

Each step operates on the lowest cost item on its blackboard input level. Using this item as input it emits all possible alternatives. The expansion of alternatives continues until the number of input items expanded in the step is equal to a control parameter or no more unexpanded items remain. Upon completion of the last step, the termination condition described above is tested to determine whether to cycle through the sequence again or to stop generation of alternatives.

Once the search terminates, QEP alternatives are sorted according to cost and the lowest cost solution is selected.

3.2.4 Back Propagation Function

To enable analysis of the performance of knowledge sources the architecture provides a mechanism to trace the derivation of each alternative QEP. Using this derivation trail the performance of each QEP alternative can be back propagated to knowledge

sources. An assessment of the quality of each knowledge source can be made based on its contribution to the optimal and near optimal solutions. Another useful metric in tuning the knowledge sources is a determination of how early in the sequence of alternatives generated by a knowledge source the components which went on to become part of near optimal QEPs were produced. Delta profiles which are bar-charts graphing this performance metric are discussed later in this section.

Each knowledge source maintains a sequence number which is incremented each time a new component is added to an emerging QEP alternative. All partial and complete QEPs have a derivation trail, represented by a list, associated with them. Each time a knowledge source operates on a partial alternative it adds its unique identifier and the next sequence number for the knowledge source to the derivation trail of the partial QEP. Thus, when a complete QEP is produced the derivation trail contains a history for each component which describes the contributing knowledge source and the position of the component in the sequence of alternatives emitted by the knowledge source.

The optimiser is calibrated by allowing it to produce all possible alternative QEPs for a given input query. The alternatives are then ordered by the execution time for each QEP. From these results a delta profile can be graphed for each knowledge source. A delta profile charts, for each QEP alternative, performance versus the sequence number of the component for the knowledge source being analysed. As shown in Section 2.3, the shape of a delta profile can provide useful information on the performance of a knowledge source.

As already discussed, an ideal profile is represented by Diagram 5 (a) where the knowledge source produces a contribution to the optimal solution the first time for all input queries. Diagram 5 (b) illustrates the worst case where the knowledge source appears to be producing alternatives at random and does not appear to possess any heuristic for grading alternatives. In practical terms, the profile in Diagram 5 (c) is considered worth attaining. For a knowledge source with such a profile, alternatives produced early in the sequence often form part of near optimal solutions.

The delta profile of a knowledge source is a readily comprehensible tool for assessing quality and provides a reasonable basis for performance tuning. Furthermore, the data used to construct a delta profile could be used by an automated tuning mechanism as discussed in Chapter 6.

4. BBQ DESIGN

The previous chapter provided a conceptual overview of each of the major design components of BBQ. This chapter describes the physical implementation of each of these components. BBQ has been implemented using a knowledge base development tool called Aion/DS. In the descriptions which follow certain sections refer specifically to its implementation in this environment.

4.1 SQL TRANSFORMATION RULES

As discussed in Section 3.1, the rule set used by BBQ has been adapted from that described in (Freytag, 1987). The transformation rules have been implemented in Knowledge Definition Language (KDL) which is the programming language of Aion/DS.

Knowledge sources, which are the basic building blocks of the transformation component, are comprised of groups of rules which are logically related in some way. The knowledge sources implementing this component are :

- AlgebraicForm
- ScanMethods
- JoinOrders
- JoinMethods
- SortOperators

A description of the function of each of these knowledge sources follows. A listing of the KDL rules implementing these knowledge sources is given at Appendix A.

The AlgebraicForm knowledge source translates the input query to an internal algebraic form. The input query is assumed to have the following structure :

```
SELECT    <project_list>
          <select_pred_list>
          <join_pred_list>
          <table_list>
```

The internal form is implemented by the creation of instances of object classes which represent each list in the input query. Thus, instances of ProjAttr, SelectPred, JoinPred and Relation are created. SelectPred has attributes SelectAttr, Operator and ConstantValue. JoinPred has attributes JoinAttr1, Operator and JoinAttr2. A listing of the data structures implementing these object classes is given at Appendix B.

The ScanMethods knowledge source generates all possible access paths to the relations in the input query. For each relation, an instance of the class Fscan is created. The selection predicates applicable to this relation are also extracted. For relations which are indexed and an index is applicable, instances of the class Iscan are created. The selection predicates are divided into those applicable to the index and the remainder.

The JoinOrders knowledge source uses the Fscan and Iscan instances created in the preceding step along with JoinPred instances to generate alternative join orders for the query. It initiates the process by creating an instance of the class JoinExpression with the innermost relation being a relation selected from the instances of the Fscan and Iscan. The selection of this relation, as well as the sequence of subsequent expansions, is determined by the cost model and control strategy which are described in the next section.

Once the JoinExpression is initialised the knowledge source progressively generates two-way joins with alternative join orders for the remaining relations. As each two-way join is created, applicable join predicates are selected from the instances of JoinPred and recorded against the join.

The JoinMethods knowledge source takes a completed JoinExpression from the previous step and creates new alternatives with the generic joins replaced with

permutations of merge and nested loop joins. As with the previous knowledge source, the order in which the permutations are generated is dictated by the cost model and control strategy.

As the name suggests, SortOperators inserts sort operators where required into expressions containing merge joins. The outer relation in the merge join is sorted to an order which is compatible with the order of the tuples of the inner expression if it is not already so. The tuples of the inner expression may still require sorting to be in the same order as the outer expression, in this case, a sort operation is introduced to the tuples of the inner expression.

4.2 COST MODEL

The cost model used in BBQ consists of two components, the first calculates a cost for the operations already incorporated in an evolving solution alternative and the second attempts to estimate a close lower bound on the cost of the operations which are yet to be incorporated.

As the cost model is not the central to the focus of this thesis, a model which provides a reasonable approximation of the efficiency of alternative solutions has been used. The algorithms used by BBQ to derive the various cost components are simplified versions of the cost functions described in (Elmasri and Navathe, 1989). In the future these algorithms could be enhanced to provide more accurate cost estimates.

In deriving the cost of operations already incorporated into an alternative, an estimate is made of the number of tuples accessed in each relation and, if applicable, in the indexes. An estimate of the number of CPU operations required by the join and sort operators is also computed. A composite cost is derived as the weighted sum of IO and CPU costs. This IO to CPU weighting factor is tunable and can be varied as appropriate for a specific hardware/software mix. In BBQ this factor has been set to 1000.

The algorithm which estimates the historical IO costs uses information from the database on the cardinality of relations and selectivity of columns (inversely proportional to the number of distinct values) in those relations.

For a full table scan operation, the number of relation tuples accessed is simply equal to the cardinality of the relation. For an index scan operation, the number of tuples accessed is estimated as follows :

Number of tuples accessed = cardinality of relation
* combined selectivity of select predicates
* (1 + number of index accesses)

where number of index accesses is an approximation derived from the
cardinality of the relation and a tunable application constant

In the above calculation it has been assumed that the cost of access to an index page is of the same order of magnitude as access to a data row. While this does not yield a precise cost, it is adequate for the purposes of this thesis. In future work, this calculation could be further refined in future to use information on data and index page sizes to estimate the number of disk pages accessed which is a closer approximation of the number of physical IO operations. For the purposes of this thesis this has not been required.

A simple estimate of the number of tuples selected from each relation is obtained by multiplying the cardinality of the relation by the selectivity factor for each selection predicate applying to the relation. The selectivity factor is determined by the comparison operator and the selectivity of the attribute contained in each selection predicate. It is calculated as follows :

if the comparison operator is '=' then
 selectivity factor = column selectivity
else if '<>' then
 selectivity factor = (1 - column selectivity)
else if '<' or '>' then
 selectivity factor = 0.5

As the source language defined for this project allows only conjunctive queries the combined selectivity factor of all select predicates applicable to a relation is the product of the individual selectivity factors.

An estimate of the cardinality of a join expression is calculated as the product of the number of tuples selected from each relation and the combined selectivity factor of the join predicates. The selectivity factor for a join predicate is determined as follows:

If the join operator is '-' then

selectivity factor = $\max(\text{selectivity of attribute 1, selectivity of attribute 2})$

else if join operator is '<>' then

selectivity factor = $1 \cdot \max(\text{selectivity of attribute 1, selectivity of attribute 2})$

else if join operator is '<' or '>' then

selectivity factor = 0.5

As with selection predicates, the combined selectivity factor of multiple join predicates is the product of the individual factors.

An estimate of CPU costs is based on a calculation of the number of operations required to perform join operations and, in the case of merge joins, sort intermediate results. While more accurate estimates of the number of CPU operations can be obtained through more sophisticated algorithms, the formulas described below are adequate for the purpose of this research.

In calculating join costs it has been assumed that after scanning, both the expressions being joined can be held in memory and therefore no additional disk accesses are required to perform the join. While this assumption is reasonable in the context of this thesis, it may be desirable in future to extend join cost calculations to account for cases, particularly if large intermediate result sets are involved, where additional disk accesses are required to complete the join operation. In that case a tunable parameter representing memory size could be defined to BBQ allowing it to further adapt to specific operating environments.

The number of operations required to perform a loop join is calculated by the following formula :

CPU operations - number of tuples in outer expression
 * number of tuples in inner expression

The same computation for a merge join is :

CPU operations - number of tuples in expression 1
 + number of tuples in expression 2

A reasonable estimate of the number of CPU operations required for a sort is given by:

CPU operations = constant * number of tuples * log(number of tuples)

The constant has been defined as a tunable parameter in BBQ.

An estimate of the cost of operations yet to be incorporated into the partial QEP is based on a best case scenario. A requirement of the search strategy is that the future cost should provide a good lower bound on future costs. The model adopted here uses a very simple heuristic to derive this lower bound. Future research could look to refine this algorithm which would improve efficiency of the search strategy.

The algorithm for future costs uses the best case for the number of tuples accessed in the relation and does not attempt to take into account any CPU costs which may be incurred. Thus for each relation the calculation is as follows :

Number of tuples accessed - Relation cardinality
 * Selectivity factor of select expression

It is acknowledged that the above approximation is simplistic and that a more accurate estimate would be desirable if BBQ were to be developed further.

4.3 SEARCH STRATEGY

The search strategy used in BBQ is a modified version of the A* search incorporating some of the concepts from (Kemper et al., 1993). Main objectives set out for the strategy were that it should enable an early assessment of the quality of alternative partial solutions and should be able to progress some of the more promising ones to completion without exploring numerous search paths.

The strategy utilises a combination of sequential control and non-deterministic generation of alternatives under a modified form of A*. The A* search is driven by the cost model described in the previous section which estimates past and future costs for each partial solution. Flow of control between the knowledge sources under this strategy is shown below.

```
AlgebraicForm
ScansMethods
Loop
    JoinOrders stop when
        number of alternatives expanded = BranchingFactor
    JoinMethods stop when
        number of alternatives expanded = BranchingFactor
    SortOperators
Until (cost of best solution from this iteration > =
    cost of best solution thus far * SearchTerminationFactor)
    or no more alternatives possible
DisplayResults
```

After sequential invocation of AlgebraicForm and ScanMethods, JoinOrders, JoinMethods and SortOperators are iteratively invoked. The two main alternative generating knowledge sources, JoinOrders and JoinMethods, operate under A*. In each, the item with the lowest combined historical and future cost is selected from the list of candidate items for expansion. Once the number of alternatives expanded in an invocation is equal to the tunable parameter BranchingFactor or all alternatives have been expanded, the knowledge source returns control to the Control knowledge source.

At the end of each iteration of the processing cycle the Control knowledge source compares the cost of the best solution produced in iteration against the best solution produced from all iterations thus far. If it exceeds by a factor more than the tunable parameter `SearchTerminationFactor` or all alternatives have been expanded, `DisplayResults` is invoked prior to process termination.

The justification for this termination condition is that the alternatives are produced in an ordered sequence with lower cost alternatives produced early due to the search strategy adopted. Under this regime, even after allowing for the fact that some degree of error will be present in estimates of future costs for emerging solutions, it is reasonable to assume that there is a high probability that a near optimal solution is produced amongst the early alternatives. Quantitative analysis techniques can be applied to determine the probability of obtaining a near optimal QEP amongst a certain number of alternatives produced and this in turn can be used to set an appropriate value for `SearchTerminationFactor`.

It can be seen from the preceding discussion that the performance of the optimiser can be varied considerably by changing the values of the parameters used to drive the search. These parameters provide an effective method of tuning the knowledge sources to enhance optimiser efficiency and provides a basis for the possibility of a self-tuning optimiser which is discussed further Section 6.3.

4.4 BACK PROPAGATION OF OPTIMISATION RESULTS

One of the objectives of the BBQ architecture was to facilitate the collection of data on the performance of knowledge sources. The delta profile of a knowledge source, which can be easily generated using data maintained by BBQ, is a tool which can be used to readily calibrate and tune optimiser components. Thus with the assistance of delta profiles, overall optimiser performance can be improved by the identification and refinement of knowledge sources which are not operating optimally.

For each item on the blackboard, BBQ maintains a derivation tag. This derivation tag is a list containing the sequence of knowledge sources which contributed to formation of the item. Each element in the list comprises two components : a unique string identifying the knowledge source and a number representing the position of this contribution in the sequence of alternatives produced by the knowledge source.

AlgebraicForm 1	ScanMethods 4	JoinOrders 7	JoinMethods 5	SortOperators 3
--------------------	------------------	-----------------	------------------	--------------------

Diagram 8 - Example of a derivation tag

An example of a derivation tag is given in the above diagram. The sequence of knowledge sources which have contributed to the formation of this alternative can be identified from this tag. An indication of how early the contributions were generated is also available.

The optimiser is calibrated by allowing it to generate all possible alternatives for each input query from a representative set. The specification of a set of queries which are representative of a general class of queries could be the subject of further research and is briefly discussed in Section 6.2.

Once all alternatives have been generated for a particular query, they are ordered by cost of execution. The set of such QEP lists produced from all the input queries along with information contained in the derivation tag of each QEP can be used to produce a delta profile for each knowledge source in the optimiser. The delta profiles chart the frequency, at each sequence position in the alternative generation sequence, where the contribution went on to form part of the optimal QEP.

As discussed in Section 2.3, it is desirable to obtain a delta profile similar to Diagram 5 (c) where the knowledge source, at an early stage, generates one or more contributions to near optimal solutions.

5. RESULTS

This chapter presents some results and observations from the implementation of BBQ. It commences by presenting results of optimisations performed on a sample set of queries followed by a discussion of various aspects of the design.

5.1 DATABASE MODEL

A rigorous analysis of the behaviour of BBQ would require compilation of a set of SQL statements which is in some way representative of all or at least a large proportion of queries which may be presented for optimisation. Collection of such a set of SQL is beyond the scope of this research. For the purposes of this project the behaviour of BBQ for a small set of queries against a hypothetical database is studied.

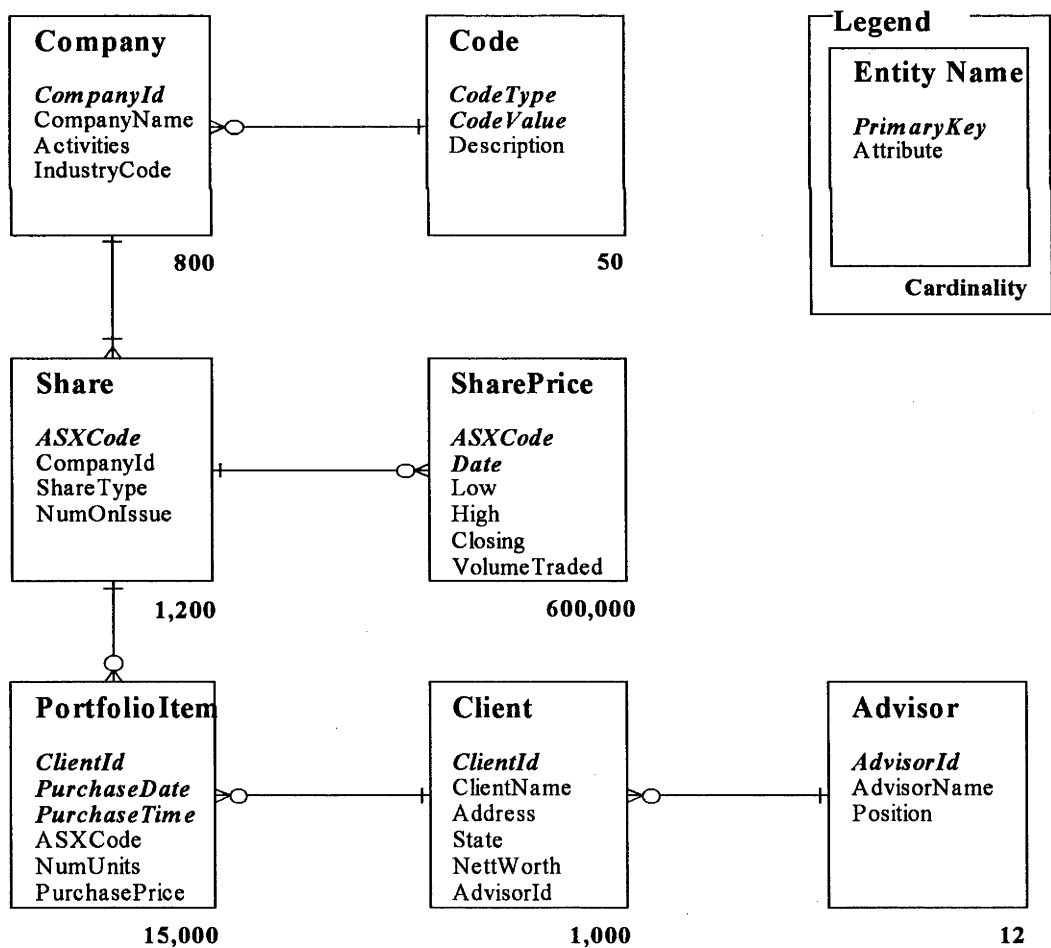


Diagram 9 - Entity-Relationship diagram of hypothetical database

An Entity-Relationship diagram of the hypothetical database is given above. The queries which have been formulated against this database to investigate the behaviour of BBQ are described in the next section.

It is assumed that information on the composition of indexes and statistics, such as cardinality and selectivity, are available from the database. The indexes, cardinalities and selectivities listed below have been used for the query optimisations which are presented in the sections which follow.

<i>Table</i>	<i>Index Name</i>	<i>Column(s)</i>
Client	ClientAdvisorId	AdvisorId
Client	ClientClientId	ClientId
Client	ClientName	ClientName
Code	CodeTypeValue	Type, Value
Company	CompanyName	CompanyName
PortfolioItem	PortfolioASXCode	ASXCode
PortfolioItem	PortfolioClientId	ClientId
Share	ShareASXCode	ASXCode
SharePrice	SharePriceASXCode	ASXCode

Table 1 - Indexes in database

<i>Table</i>	<i>Cardinality</i>
Advisor	12
Client	1000
Code	50
Company	800
PortfolioItem	15000
Share	1200
SharePrice	600000

Table 2 - Cardinality of tables

<i>Table</i>	<i>Column</i>	<i>Selectivity</i>
Advisor	AdvisorId	.0833
	AdvisorName	.1111
	Position	.2500
Client	ClientId	.0833
	ClientName	.0011
	Address	.0011
	State	.1429
	NettWorth	.0015
	AdvisorId	.0833
Code	Type	.2500
	Value	.0500
	Description	.0200
Company	CompanyId	.0013
	Name	.0013
	Activities	.0013
	IndustryCode	.0250
PortfolioItem	ClientId	.0011
	PurchaseDate	.0020
	PurchaseTime	.0001
	ASXCode	.0040
	NumOfUnits	.0001
Share	PurchasePrice	.0001
	ASXCode	.0008
	CompanyId	.0013
	ShareType	.2500
SharePrice	NumOnIssue	.0009
	ASXCode	.0008
	Date	.0020
	Low	.0002
	High	.0003
	Closing	.0002
	VolumeTraded	.0001

Table 3 - Selectivity of columns

5.2 RESULTS OF SAMPLE OPTIMISATIONS

Five queries of varying complexity have been constructed to investigate various characteristics of BBQ. The queries formulated to run against the hypothetical database which was described in the preceding section are listed below.

1. List the portfolio of 'VIKRAM SHARMA'
2. List the portfolio of all clients advised by 'JOHN FRANCIS'

3. List all clients who own more than 1000 BHP preference shares
4. List the closing price and volume traded on 30/3/97 of all stocks in the 'TOURISM AND LEISURE' sector
5. List clients who live in 'ACT' , have a nett worth > \$100,000 and have bought stocks in the 'DIVERSIFIED INDUSTRIALS' sector since 30/6/96

For each of these queries, an SQL statement and the best QEP generated by an exhaustive run of BBQ is presented below. The cost of executing the QEP, as computed by the cost algorithm used in BBQ, is shown along with a dissection of the cost to each QEP step. Where the step cost is less than 1, it is displayed with a precision of one decimal place to provide an indication of its relative cost. The final projection operation is not shown in the QEP listings.

5.2.1 Query 1

```

SELECT      PortfolioItem.PurchaseDate, PortfolioItem.ASXCode,
            Porfolio.NumOfUnits
FROM        Client, PortfolioItem
WHERE       Client.ClientId = PortfolioItem.ClientId
AND         Client.Name = 'Vikram Sharma'
    
```

Query 1		Step
QEP Cost 15018		cost
(Ljoin (PortfolioItem.ClientId=Client.ClientId)		15
(Iscan (Client.Name="Vikram Sharma") ClientName () Client)		2
(Fscan () PortfolioItem)		15,000

An additional transformation commonly implemented by optimisers is the use of predicate(s) in a join expression to perform, if applicable, an index scan on the inner relation. The addition of this transformation as an extension to the set of rules implemented by BBQ is shown in Section 5.7.

5.2.2 Query 2

```

SELECT      Client.Name, PortfolioItem.ASXCode, PortfolioItem.NumOfUnits
FROM        Advisor, Client, PortfolioItem
WHERE       Advisor.AdvisorId = Client.AdvisorId
AND         Client.ClientId = PortfolioItem.ClientId
AND         Advisor.Name = 'John Francis'
    
```

Query 2		Step
QEP Cost 16237		cost
(Mjoin (PortfolioItem.ClientId=Client.ClientId)		15
(Sort (PortfolioItem.ClientId)		208
Fscan () PortfolioItem)		15,000
(Sort (Client.ClientId)		1
(Ljoin (Client.AdvisorId=Advisor.AdvisorId)		1
(Fscan (Advisor.Name="John Francis") Advisor)		12
(Fscan () Client)))		1,000

5.2.3 Query 3

```

SELECT      Client.Name, PortfolioItem.NumOfUnits
FROM        Company, Share, PortfolioItem, Client
WHERE       Company.CompanyId = Share.CompanyId
AND         Share.ASXCode = PortfolioItem.ASXCode
AND         PortfolioItem.ClientId = Client.ClientId
AND         PortfolioItem.NumOfUnits > 1000
AND         Company.Name = 'Broken Hill Proprietary'
AND         Share.Type = 'Preference'
    
```

Query 3		Step
QEP Cost 17224		cost
(Mjoin (PortfolioItem.ClientId=Client.ClientId)		1
(Sort (Client.ClientId)		10
Fscan () Client)		1,000
(Sort (PortfolioItem.ClientId)		0.1
(Ljoin (Share.ASXCode=PortfolioItem.ASXCode)		10
(Fscan (PortfolioItem.NumOfUnits>1000) PortfolioItem)		15,000
(Ljoin (Company.CompanyId=Share.CompanyId)		0.6
(Fscan (Share.Type="Preference") Share)		1,200
(Iscan (Company.Name="Broken Hill Proprietary") CompanyName () Company))))		2

5.2.4 Query 4

```

SELECT    Company.Name, Share.Type, SharePrice.Closing
FROM      Code, Company, Share, SharePrice
WHERE     Code.Value = Company.IndustryCode
AND       Company.CompanyId = Share.CompanyId
AND       Share.ASXCode = SharePrice.ASXCode
AND       Code.Type = 'INDUSTRY'
AND       Code.Description = 'TOURISM AND LEISURE'
AND       SharePrice.Date = '31/3/97'
    
```

Query 4		Step
QEP Cost 602064		cost
(Mjoin (Share.ASXCode=SharePrice.ASXCode)		1
(Sort (SharePrice.ASXCode)		12
Fscan (SharePrice.Date="31/3/97") SharePrice)		600,000
(Sort (Share.ASXCode)		0.1
(Mjoin(Code.Value=Company.IndustryCode)and(Company.CompanyId=Share.CompanyId))		1
(Sort (Company.IndustryCode,Company.CompanyId)		8
Fscan () Company)		800
(Sort (Code.Value,Share.CompanyId)		2
(Ljoin		0.3
(Iscan (Code.Type="INDUSTRY") CodeTypeValue		39
(Code.Description="TOURISM AND LEISURE") Code)		
(Fscan () Share))))		1,200

It is interesting to note that the innermost join in this QEP generates a cartesian product. According to the cost model used, this alternative is very slightly cheaper than one which contains a loop join between Code and Company followed by merge joins with Share and subsequently SharePrice. Although that QEP does not generate any cartesian products, it is slightly more expensive as increased sort costs more than offset cost savings from the innermost join.

5.2.5 Query 5

```

SELECT      Client.Name, Company.Name, Share.Type, PortfolioItem.NumOfUnits
FROM        Code, Company, Share, PortfolioItem, Client
WHERE       Code.IndustryCode = Company.Value
AND         Company.CompanyId= Share.CompanyId
AND         Share.ASXCode = PortfolioItem.ASXCode
AND         PortfolioItem.ClientId = Client.ClientId
AND         Code.Type = 'INDUSTRY'
AND         Code.Description = 'DIVERSIFIED INDUSTRIALS'
AND         PortfolioItem.PurchaseDate > '30/06/96'
AND         Client.State = 'ACT'
AND         Client.NettWorth > 100000
    
```

Owing to the limited capability of the hardware platform on which the QEPs were generated, a subset of approximately 750 alternatives out of a possible 3840 has been generated for this query. The search was restricted by limiting the number of alternatives expanded by each knowledge source, refer to Section 5.4.1 for a discussion on how this is achieved. The relationship between the set of QEPs produced in a restricted search and the set of all possible QEPs from an exhaustive search is touched upon in Section 5.4.5.

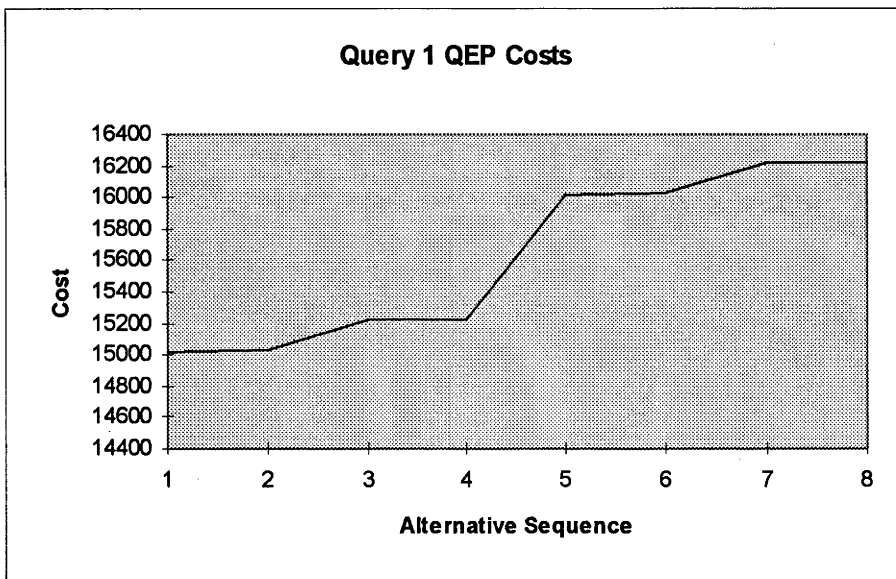
Query 5		Step cost
QEP Cost 18240		
(Mjoin ((Company.IndustryCode=Code.Value) and (Share.CompanyId=Company.CompanyId))		11
(Sort (Company.IndustryCode,Company.CompanyId)		8
Fscan () Company)		800

(Sort (Code.Value,Share.CompanyId)	130
(Mjoin (PortfolioItem.ASXCode=Share.ASXCode))	2
(Sort (Share.ASXCode)	12
Fscan () Share)	1,200
(Sort (PortfolioItem.ASXCode)	10
(Mjoin (Client.ClientId=PortfolioItem.ClientId))	2
(Sort (Client.ClientId)	4
Fscan (Client.NetWorth>100000) Client)	1,000
(Sort (PortfolioItem.ClientId)	20
(Ljoin()	2
(Iscan (Code.Type="INDUSTRY") CodeTypeValue	
(Code.Description="DIVERSIFIED INDUSTRIALS") Code)	39
(Fscan (PortfolioItem.PurchaseDate>"30/06/96") PortfolioItem))))	15,000

5.3 DISTRIBUTION OF QEP COSTS

This section presents graphs showing the behaviour of QEP costs from the results of running BBQ to produce an exhaustive set of solutions for each of the sample queries.

For Query 1, only a small number of alternatives are possible and the cost of alternatives does not vary greatly. It ranges from a minimum of 15,018 to a maximum of 16,223.



The main component of the total QEP cost is the full scan of the PortfolioItem table which has a cost of 15,000. An enhancement to the transformation rule set which allows the join predicate to be used for an index scan on this table is given in Section 5.6. Alternatives 5 - 8 have a higher cost when compared with earlier QEPs due to the index scan on the Client table, which had a cost of 2, being replaced with a full scan which has a cost of 1,000.

Diagram 10 illustrates the identification tags of alternatives, as discussed in Section 4.4, generated at each level of the blackboard at the conclusion of the optimisation for Query 1. Note that the Algebraic Form level contains a solitary alternative as only one algebraic representation of the query is produced and that the list of identification tags at the Complete QEPs level is the same as that at the Join Methods level since the Sort operator is not required for this query. In the diagram, KS₀, KS₁, KS₂, and KS₃ represent the knowledge sources AlgebraicForm, ScanMethods, JoinOrders and JoinMethods respectively.

Level Name

Complete QEPs	[KS ₃ 0, KS ₂ 0, KS ₁ 2, KS ₀ 0] [KS ₃ 1, KS ₂ 2, KS ₁ 0, KS ₀ 0] [KS ₃ 2, KS ₂ 0, KS ₁ 2, KS ₀ 0] [KS ₃ 3, KS ₂ 2, KS ₁ 0, KS ₀ 0] [KS ₃ 4, KS ₂ 1, KS ₁ 2, KS ₀ 0] [KS ₃ 5, KS ₂ 3, KS ₁ 1, KS ₀ 0] [KS ₃ 6, KS ₂ 1, KS ₁ 2, KS ₀ 0] [KS ₃ 7, KS ₂ 3, KS ₁ 1, KS ₀ 0]
Join Methods	[KS ₃ 0, KS ₂ 0, KS ₁ 2, KS ₀ 0] [KS ₃ 1, KS ₂ 2, KS ₁ 0, KS ₀ 0] [KS ₃ 2, KS ₂ 0, KS ₁ 2, KS ₀ 0] [KS ₃ 3, KS ₂ 2, KS ₁ 0, KS ₀ 0] [KS ₃ 4, KS ₂ 1, KS ₁ 2, KS ₀ 0] [KS ₃ 5, KS ₂ 3, KS ₁ 1, KS ₀ 0] [KS ₃ 6, KS ₂ 1, KS ₁ 2, KS ₀ 0] [KS ₃ 7, KS ₂ 3, KS ₁ 1, KS ₀ 0]
Generic Joins	[KS ₂ 0, KS ₁ 2, KS ₀ 0] [KS ₂ 1, KS ₁ 2, KS ₀ 0] [KS ₂ 2, KS ₁ 0, KS ₀ 0] [KS ₂ 3, KS ₁ 1, KS ₀ 0]
Scan Operators	[KS ₁ 0, KS ₀ 0] [KS ₁ 1, KS ₀ 0] [KS ₁ 2, KS ₀ 0]
Algebraic Form	[KS ₀ 0]
Input SQL	Input Query

Diagram 10 - Identification tags of alternatives at each level of blackboard on completion of optimisation of Query 1

The query trees for the best (cost 15,018) and the worst (cost 16,223) plans are shown at Diagram 11 (a) and Diagram 11 (b) respectively.

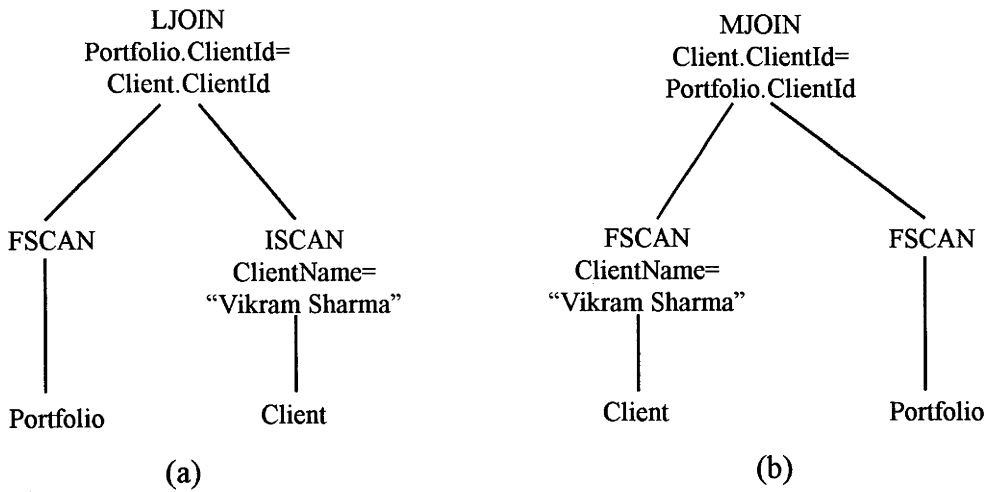
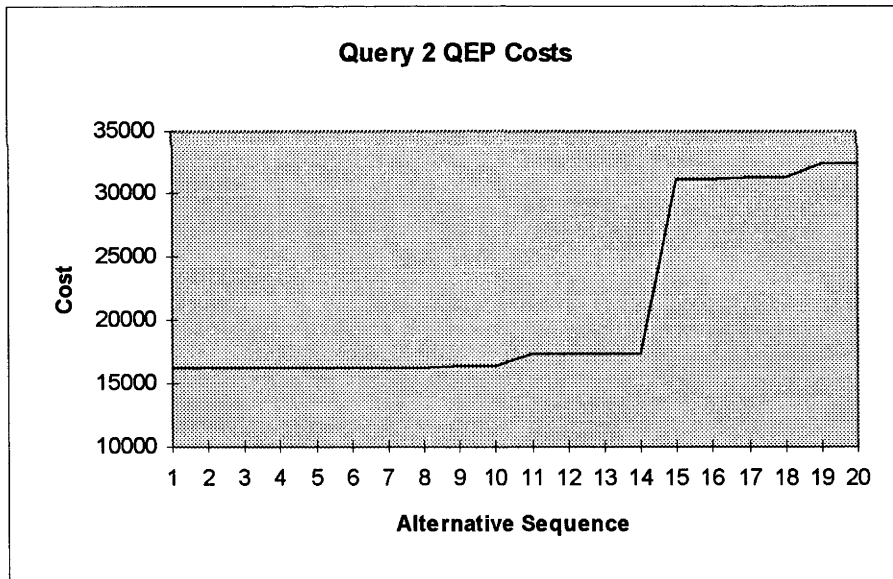


Diagram 11 - Query trees for best and worst plans for Query 1

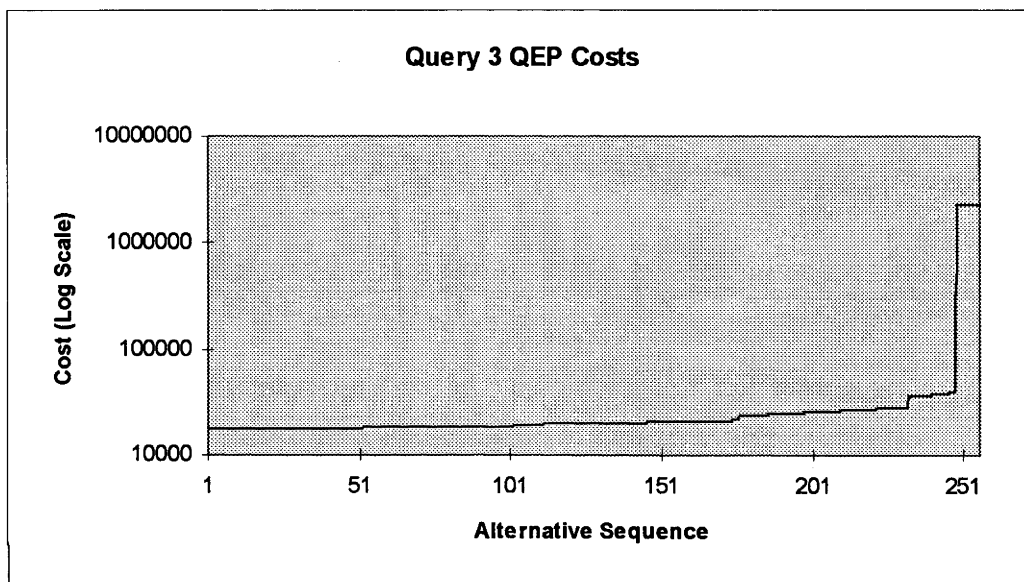
For Query 2, again only a small number of alternatives are possible. The cost does not vary greatly for the first 14 alternatives and but then increases significantly. It ranges from a minimum of 16,237 to a maximum of 32,406.



As with Query 1, a major contributor to the total QEP cost is the full scan of the PortfolioItem table. In alternatives 15 - 20 the significant increase in cost is due to additional CPU costs associated with a loop join between the PortfolioItem and Client tables. The scans on these tables, which have cardinalities of 15,000 and 1,000

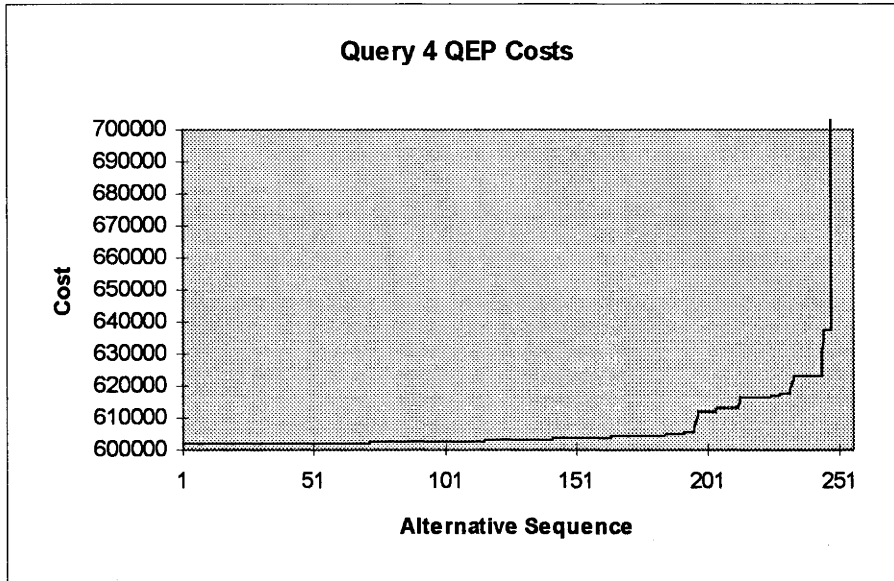
respectively, do not have any associated selection predicates and thus retrieve all rows from each relation, leading to a high computation cost for the loop join.

For Query 3, a total of 256 alternative QEPs are possible. The cost of the first 248 alternatives falls in the range 17,224 to 39,233 with the cost of the last 8 rising steeply to be in the range 2,267,529 to 2,268,608.



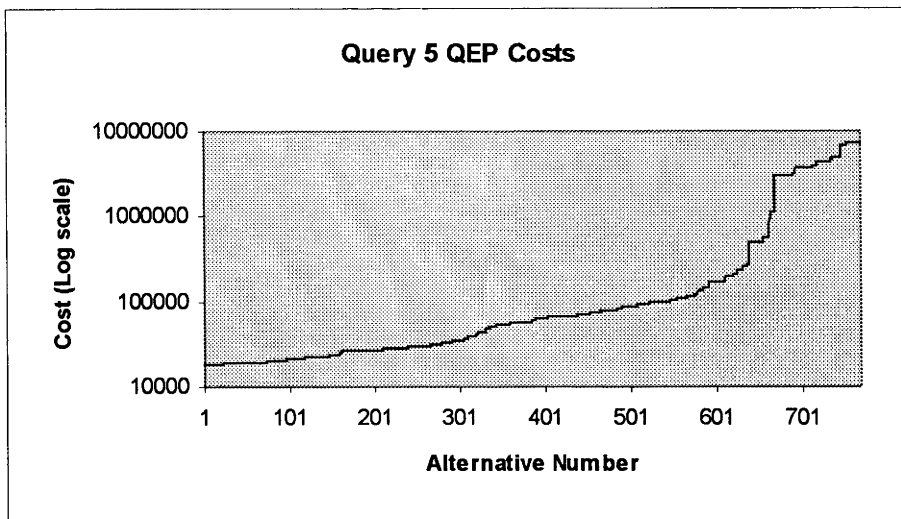
In the early alternatives for this query, the major component of the total cost is the full scan of the PortfolioItem table which has a cost of 15,000. Alternatives with sequence numbers approximately in the range 150 - 248 have a loop join between PortfolioItem and Share or between PortfolioItem and Client as the innermost join. In each of these cases, the number of rows selected from both relations is large leading to join computation costs which contribute significantly to the total cost of the QEP. For the last 8 alternatives, the inner most join is a loop join between Client and Share which produces a cartesian product. This large resultant set of rows is loop joined with PortfolioItem from which a large number of rows are also likely to be retrieved. Thus the approximate cost of performing the second loop join is 2,250,000 which represents over 99% of the cost of these QEPs .

As with Query 3, a total of 256 alternative QEPs are possible for Query 4. The distribution of QEP costs is also similar to Query 3 although the cost curve is flatter. The cost of the first 248 alternatives falls in the range 602,064 to 637,450 with the cost of the last 8 rising steeply to be in the range 1,755,004 to 1,755,186.



The major component of the cost for the first 248 alternatives is the full scan of the SharePrice table which has a cost of 600,000. The cost of the last 8 alternatives rises sharply as these QEPs contain a loop join between the SharePrice and Company tables followed by another loop join between the resultant set of rows and the SharePrice table. The first join is estimated to return 960,000 rows which are then loop joined with an estimated 1,200 rows from SharePrice resulting in a cost of 1,152,000 for the second join and thus adding significantly to the QEP cost.

The cost distribution of Query 5 QEPs is different to that of the other four queries in that it does not have the flat distribution exhibited in the early portion of the cost curves for those queries. The costs range from a minimum of 16,237 to a maximum of 7,262,040.



In the early alternatives for this query the cost of disk access is the major contributor to the total QEP cost. In the case of the optimal QEP, disk accesses account for approximately 98% of the QEP cost. For the remainder of the alternatives there is no significant increase in the cost of disk accesses. The increase in QEP costs is primarily due to degradation in the efficiency of joins leading to an exponential rise in their costs.

5.4 RESULTS FROM RESTRICTED SEARCHES

The search strategy used by BBQ can be adjusted by altering the values of parameters which control both the breadth and depth of search. The weighting of the cost of IO operations relative to CPU may also be changed to reflect a setting which is appropriate to the hardware and system software mix of the operating environment. This capability represents a significant differentiator from traditional optimiser designs and provides a mechanism to improve the quality of QEPs generated and the efficiency of the optimiser. It may also form the basis for a self-tuning optimiser which is briefly discussed in Section 6.3.

The results of experimentation with various parameter settings are presented in this section. The effect of these changes both in terms of the QEP quality and optimiser efficiency is discussed. The number of alternative QEPs generated in a restricted run is approximately proportional to the time taken to perform the optimisation. Therefore, a reasonable measure of the efficiency of the optimiser is the ratio of the number of alternatives produced during a restricted search to the total number possible.

5.4.1 Limited number of alternatives expanded

This section presents the QEPs generated for the sample queries in Section 5.2 when the number of alternatives expanded by each knowledge source is severely restricted. BBQ parameter settings used were : BranchingFactor = 2, SearchTerminationFactor = 1.00 and IO to CPU weighting = 1000. The effect of these settings is to limit each knowledge source to expanding the two most promising alternatives and to terminate the search after one processing cycle. A complete discussion of these parameters was given in Section 4.3.

5.4.1.1 Query 1

The best QEP produced was the same as under an exhaustive search, possibly because the total number of alternatives possible is small. Four out of eight possible QEPs were generated.

5.4.1.2 Query 2

As with Query 1, the best QEP produced was the same as under an exhaustive search, again the total number of alternatives possible is small. Eight out of 20 possible QEPs were generated.

5.4.1.3 Query 3

The best QEP produced was within 1% of the optimum with only eight out of a total of 256 possible QEPs produced. In the case of this query the result may have been aided by the flatness of its cost distribution curve.

Query 3
QEP Cost 17325
(Mjoin((Share.ASXCode=PortfolioItem.ASXCode) and (PortfolioItem.ClientId=Client.ClientId)) (Sort ((PortfolioItem.ASXCode,PortfolioItem.ClientId) Fscan (PortfolioItem.NumOfUnits>1000) PortfolioItem) (Sort (Share.ASXCode,Client.ClientId) (Mjoin (Company.CompanyId=Share.CompanyId) (Sort (Share.CompanyId) Fscan (Share.Type="Preference") Share) (Sort (Company.CompanyId) (Ljoin() (Fscan () Client) (Iscan (Company.Name="Broken Hill Proprietary") CompanyName()Company))))))

5.4.1.4 Query 4

As with Query 3, the best QEP produced was within 1% of the optimum QEP with only eight out of a total of 256 possible QEPs produced. Again, the results for this query may have been aided by the flatness of its cost distribution curve.

Query 4
QEP Cost 603428
(Mjoin((Company.CompanyId=Share.CompanyId) and (Share.ASXCode=SharePrice.ASXCode)) (Sort (Share.CompanyId,Share.ASXCode) Fscan () Share) (Sort (Company.CompanyId,SharePrice.ASXCode) (Ljoin (Code.Value=Company.IndustryCode) (Iscan (Code.Type="INDUSTRY") CodeTypeValue (Code.Description="TOURISM AND LEISURE") Code) (Ljoin() (Fscan (SharePrice.Date="31/3/97") SharePrice) (Fscan () Company))))))

5.4.1.5 Query 5

The best QEP produced was 6% worse than the best QEP available from the more expansive search of Section 5.1.5. This search generated only 16 QEPs compared with approximately 750 for the more expansive search. If we classify near optimal queries as those which are within 5% of the best result obtained previously, this result would not qualify as near optimal. While the restrictions imposed on the optimiser in the case of this complex query appear to be too severe to allow it to produce a result within the above definition of near optimal, it appears to be a reasonable outcome in light of the fact that the number of alternatives generated was only 2% of the number produced by the more expansive search.

Query 5
QEP Cost 19364
(Mjoin ((Share.CompanyId=Company.CompanyId) and (PortfolioItem.ASXCode=Share.ASXCode)) (Sort (Share.CompanyId,Share.ASXCode) Fscan () Share) (Sort (Company.CompanyId,PortfolioItem.ASXCode) (Mjoin (Client.ClientId=PortfolioItem.ClientId) (Sort (PortfolioItem.ClientId) Fscan (PortfolioItem.PurchaseDate>"30/06/96") PortfolioItem) (Sort (Client.ClientId) (Ljoin (Company.IndustryCode=Code.Value) (Iscan (Code.Type="INDUSTRY") CodeTypeValue (Code.Description="DIVERSIFIED INDUSTRIALS") Code) (Ljoin () (Fscan (Client.NetWorth>100000) Client) (Fscan () Company))))))

5.4.2 Increase Search Breadth

The breadth of partial QEP alternatives searched by BBQ, as compared with the preceding section, was increased by setting the parameter BranchingFactor = 3. This change causes each of the knowledge sources under modified A* control to expand a

larger number of partial QEP alternatives. The results produced with this changed setting are presented below.

5.4.2.1 Query 1

The best QEP produced was the same as the optimum solution with six out of a possible eight alternatives generated.

5.4.2.2 Query 2

The best QEP generated is marginally worse than with the previous more restrictive search. It is probable that the increased number of alternatives available caused the cost function and the control strategy to incorrectly ignore a partial QEP which had previously led to a superior solution. This would occur if the total projected cost of a partial alternative previously expanded was greater than that of new competing alternatives. Thus by implication, the function which approximates future costs of partial QEPs is not performing effectively in this case and should be refined to increase the probability that partial QEPs which lead to optimal or near-optimal solutions are explored. 10 out of a possible 20 alternatives were generated.

Query 2
QEP Cost 16238
(Mjoin (PortfolioItem.ClientId=Client.ClientId) (Sort (PortfolioItem.ClientId) Fscan () PortfolioItem) (Sort (Client.ClientId) (Ljoin (Client.AdvisorId=Advisor.AdvisorId) (Fscan () Client) (Fscan (Advisor.Name="John Francis") Advisor)))

5.4.2.3 Query 3

As with Query 2 the result produced is marginally worse than under the previous more restrictive search. The reasons for this are likely to be the same as for the previous query. 12 out of a possible 256 alternatives were generated.

Query 3
QEP Cost 18123
(Mjoin ((Share.ASXCode=PortfolioItem.ASXCode) and (PortfolioItem.ClientId=Client.ClientId)) (Sort (PortfolioItem.ASXCode,PortfolioItem.ClientId) Fscan (PortfolioItem.NumOfUnits>1000)PortfolioItem) (Sort (Share.ASXCode,Client.ClientId) (Mjoin (Company.CompanyId=Share.CompanyId) (Sort (Share.CompanyId) Fscan (Share.Type="Preference") Share) (Sort (Company.CompanyId) (Ljoin() (Fscan () Client) (Fscan (Company.Name="Broken Hill Proprietary") Company))))))

5.4.2.4 Query 4

A QEP superior to that generated with previous optimiser settings was produced. The QEP was within 1% of the optimal solution with only 12 out of a possible 256 QEPs generated.

Query 4
QEP Cost 602065
(Mjoin (Share.ASXCode=SharePrice.ASXCode) (Sort (SharePrice.ASXCode) Fscan (SharePrice.Date="31/3/97") SharePrice) (Sort (Share.ASXCode) (Mjoin(Code.Value=Company.IndustryCode)and(Company.CompanyId=Share.CompanyId)) (Sort (Company.IndustryCode,Company.CompanyId))

```
Fscan () Company)
(Sort (Code.Value,Share.CompanyId)
(Ljoin
(Fscan () Share)
(Iscan (Code.Type="INDUSTRY") CodeTypeValue
(Code.Description="TOURISM AND LEISURE")))))
```

5.4.2.5 Query 5

A QEP superior to that generated with previous optimiser settings was produced. The QEP was within 1% of the optimum with 20 out of a possible 3840 alternatives generated.

Query 5

QEP Cost 18337

```
(Mjoin ((PortfolioItem.ASXCode=Share.ASXCode) and (Client.ClientId=PortfolioItem.ClientId)
(Sort (PortfolioItem.ASXCode,PortfolioItem.ClientId)
Fscan (PortfolioItem.PurchaseDate>"30/06/96") PortfolioItem)
(Sort (Share.ASXCode,Client.ClientId)
(Mjoin (Share.CompanyId=Company.CompanyId)
(Sort (Share.CompanyId)
Fscan () Share)
(Sort (Company.CompanyId)
(Mjoin (Company.IndustryCode=Code.Value)
(Sort (Company.IndustryCode)
Fscan () Company)
(Sort (Code.Value)
(Ljoin()
(Fscan (Client.NetWorth>100000) Client)
(Iscan (Code.Type="INDUSTRY") CodeTypeValue
(Code.Description="DIVERSIFIED INDUSTRIALS") Code)))))
```

Increase of the search breadth parameter yielded two solutions which were marginally worse, two which were superior and one which was the same when compared with the corresponding solutions generated with the previous, more restrictive, BBQ settings. The solutions which were either the same or superior are as expected given that the

number of alternatives under consideration has increased due to the higher search breadth setting. The two solutions which were poorer indicated a shortcoming of the future cost algorithm and suggest that this algorithm should be enhanced to reduce the possibility that promising partial QEPs are ignored.

5.4.3 Increase Search Depth

The depth of QEP alternatives generated by BBQ, as compared to that in Section 5.4.1, was increased by setting the parameter `SearchTerminationFactor = 1.05`. This change causes BBQ to continue generation of QEP alternatives until the cost of the best QEP produced in an iteration exceeds the cost of the best QEP generated thus far by 5%. QEPs produced under this setting for the previously described queries are presented below.

5.4.3.1 Query 1

This setting caused all eight possible QEPs to be generated for this query.

5.4.3.2 Query 2

This setting caused all 20 possible QEPs to be generated for this query.

5.4.3.3 Query 3

The best QEP produced was the same as the optimum solution with 52 out of a possible 256 alternatives generated.

5.4.3.4 Query 4

The best QEP produced was the same as that produced in the previous restricted search and within 1% of the optimum with 16 out of a possible 256 alternatives generated.

5.4.3.5 Query 5

A QEP which is within 3% of the optimal solution was produced with 48 out of a possible 3840 alternatives generated.

Query 5
QEP Cost 18827
(Mjoin ((PortfolioItem.ASXCode=Share.ASXCode) and (Client.ClientId=PortfolioItem.ClientId)) (Sort (PortfolioItem.ASXCode,PortfolioItem.ClientId) Fscan (PortfolioItem.PurchaseDate>"30/06/96") PortfolioItem) (Sort (Share.ASXCode,Client.ClientId) (Mjoin (Share.CompanyId=Company.CompanyId) (Sort (Share.CompanyId) Fscan () Share) (Sort (Company.CompanyId) (Ljoin (Company.IndustryCode=Code.Value) (Iscan (Code.Type="INDUSTRY") CodeTypeValue (Code.Description="DIVERSIFIED INDUSTRIALS")Code) (Ljoin () (Fscan (Client.NetWorth>100000) Client) (Fscan () Company))))))

Increase of the search depth parameter yielded two solutions were superior and three which were the same when compared with the corresponding solutions Section 5.4.1. This results is as would be expected given that the number of alternatives under consideration has been increased with the higher parameter setting.

5.4.4 Summary of restricted search runs

The preceding sections described the results of generating a limited set of QEP alternatives by restricting the number of alternatives expanded by BBQ or by triggering the termination condition for the search once the quality of the QEPs had degraded by a specified factor.

In these runs, the number of alternatives generated varied from less than 1% to 100% of the total number of alternatives possible. As mentioned earlier, the time taken to perform the optimisation is roughly proportional to the number of alternatives produced. Therefore, one measure of the efficiency of the optimiser is the ratio of the number of QEPs required to produce at least one near optimal QEP to the total number of QEPs possible. For the purposes of this thesis a near optimal QEP has been defined as one with a cost which is within 5% of the optimum; although other definitions, for example an upper bound of the highest cost of the best 10% of all QEPs, may be equally valid.

Given this definition of near optimal, each of the restricted searches, with the exception of one which was just outside the 5% bound, produced at least one near optimal QEP. Thus a satisfactory result in terms of QEP quality was achieved with the generation of only a small fraction of the total number possible alternatives. This is one of the five characteristics, listed in Chapter 1, which the BBQ model attempts to achieve. Further benchmarking against a more comprehensive set of queries could be used to establish BBQ parameter settings which achieve near optimal results with the generation of a minimum number of alternatives

5.5 CHANGE IN IO TO CPU WEIGHTING

The relative weighting of IO operations to CPU operations was changed to 500, effectively making CPU operations more expensive relative to IO operations. This change demonstrates the ability of the model to adapt to specific hardware and software environments. The effects of this setting on the optimisation of the previously described queries are presented below.

5.5.1.1 Query 1

The best QEP produced was the same as with the original optimiser settings. However, as expected the cost was different. The cost with the changed setting was 15034.

5.5.1.2 Query 2

The best QEP produced was the same as with the original optimiser settings. However, as expected the cost was different. The cost with the changed setting was 16462.

5.5.1.3 Query 3

The best QEP produced was the same as with the original optimiser settings. However, as expected the cost was different. The cost with the changed setting was 17449.

5.5.1.4 Query 4

The best QEP produced was the same as with the original optimiser settings. However, as expected the cost was different. The cost with the changed setting was 604817.

5.5.1.5 Query 5

A QEP different to that generated using the previous settings was produced.

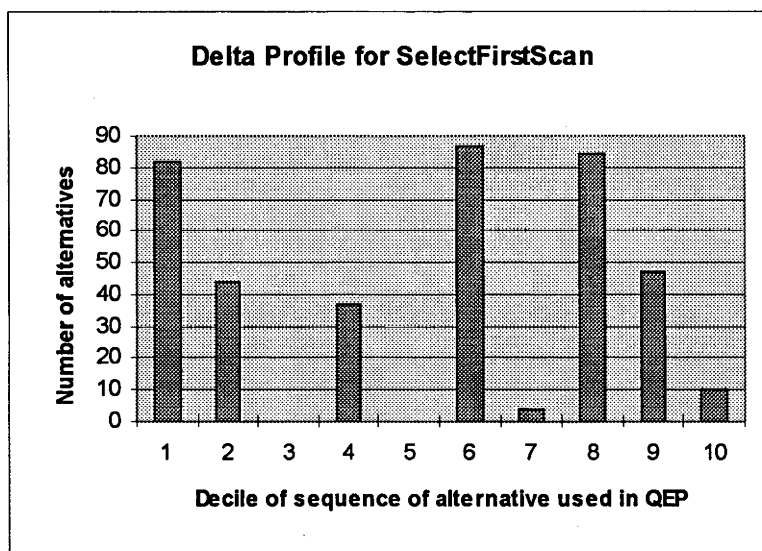
Query 5
QEP Cost 19616
(Mjoin ((PortfolioItem.ASXCode=Share.ASXCode) and (Client.ClientId=PortfolioItem.ClientId)) (Sort (PortfolioItem.ASXCode,PortfolioItem.ClientId) Fscan (PortfolioItem.PurchaseDate>"30/06/96") PortfolioItem) (Sort (Share.ASXCode,Client.ClientId) (Mjoin (Share.CompanyId=Company.CompanyId) (Sort (Share.CompanyId) Fscan () Share) (Sort (Company.CompanyId) (Ljoin (Company.IndustryCode=Code.Value) (Iscan (Code.Type="INDUSTRY") CodeTypeValue (Code.Description="DIVERSIFIED INDUSTRIALS") Code) (Ljoin() (Fscan (Client.NetWorth>100000) Client) (Fscan () Company))))))

For Query 5, the QEP generated was different to the corresponding one produced under the original optimiser settings. This demonstrates the flexibility of the model to select a QEP which takes into account characteristics of the operating environment.

5.6 DELTA PROFILES FOR KNOWLEDGE SOURCES

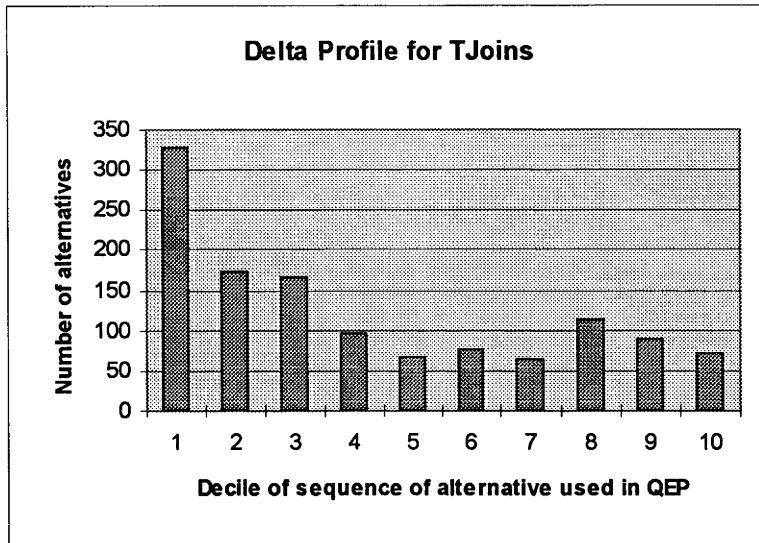
Using results of the optimisations from section 5.2, delta profiles of near optimal solutions for the five sample queries were constructed. For the delta profiles, near optimal QEPs have been defined as those having a cost within 5% of the optimum.

The method of construction of these profiles has been described in Chapter 4 and relies on the derivation tag which is associated with each QEP produced. A variation introduced in the profiles below is to plot the decile of the alternative sequence as opposed to the sequence ordinal. This enhancement allows a more meaningful comparison across a range of queries. Also, the metrics maintained by BBQ allow construction of delta profiles with a granularity finer than knowledge source level. Thus, delta profiles for the main alternative generating transformation rules are presented.

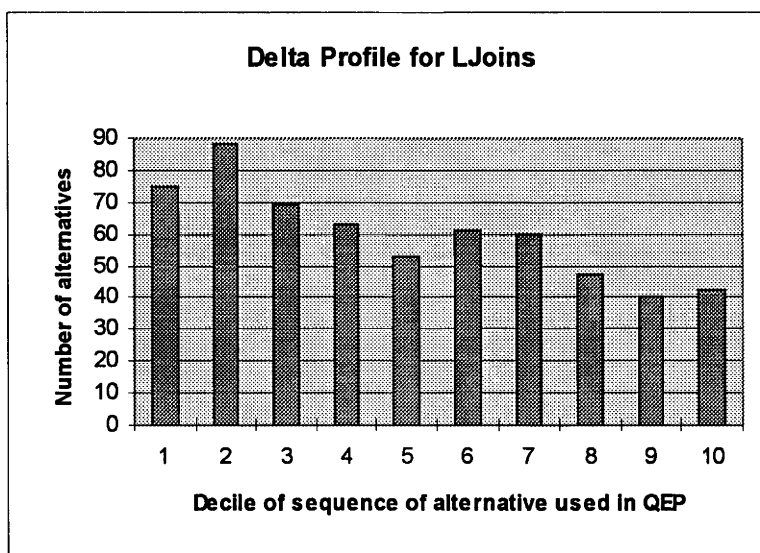


The delta profile for SelectFirstScan has been constructed for comparative purposes only. It shows that this rule does not appear to have any ordering in its contributions to

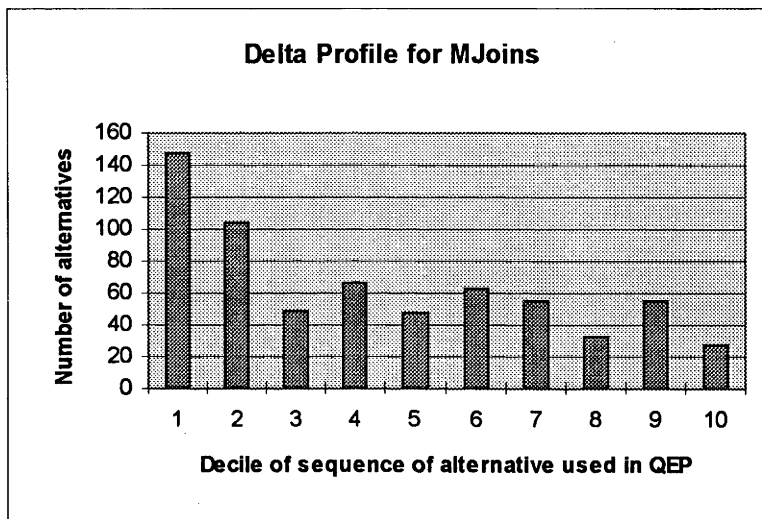
near optimal solutions. This result is as expected given that the rule is allowed to generate all possible first scans without restriction since the cost of this operation relative to the rest of the optimisation is small.



The delta profile for Tjoins is significantly skewed to the lower deciles. This indicates that the rule is generating contributions to near optimal queries early in the sequence of alternatives produced by it. This implies that it should produce a high proportion of alternatives which form part of near optimal solutions even as the number of alternatives it is allowed to generate is restricted.



The delta profile for Ljoins is only slightly biased to lower deciles. This indicates that in restricted searches, the rule should still produce contributions which lead to near optimal solutions, however the probability of not generating near optimal contributions is higher than for the Tjoins rule. This rule is a candidate for improvement and its associated cost function should be examined for possible refinement. The assumptions made by the cost model with regard to loop joins, as discussed in Section 4.2, may need to be re-examined.



In contrast to the delta profile for Ljoins, the delta profile for Mjoins is significantly more biased to low deciles. The rule appears to be performing efficiently and there is a high probability that it will contribute to near optimal query plans in restricted searches.

5.7 EXAMPLE OF EXTENDING TRANSFORMATION RULE SET

One of the primary objectives of the architecture of BBQ is that it should be amenable to the incorporation of new or changed transformation rules. As mentioned in 5.2.1 a common transformation implemented by optimisers is to use join predicates to perform an index scan, if applicable, on the inner relation of a join. This section presents the results of incorporating this transformation into BBQ's rule set.

Implementation of the new transformation required four discrete changes to BBQ :

1. Modification of the data structure containing index scan predicates to allow it to point to join predicates (in addition to select predicates as previously).
2. Addition of three new rules to perform the transformation. Listings of the new rules can be found at the end of Appendix A.
3. Modification of the cost algorithm for index scans.
4. Modification of the Tjoins rule to use the new scans only as inner relations.

No changes were required to either the control strategy or other transformation rules. The total time required to accomplish this extension, including problem analysis, was under three hours. The effect of this enhancement on the QEPs for Query 1 and Query 2, described in Section 5.2, are presented below.

The optimum QEP for Query 1 after the enhancement is :

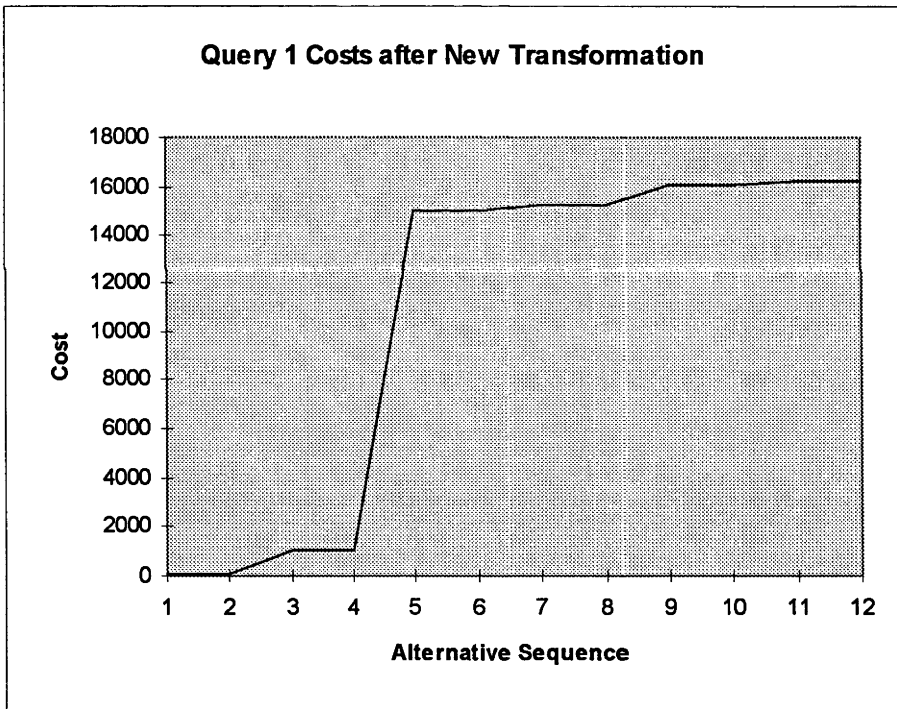
Query 1		Step
QEP Cost 52		cost
Mjoin (PortfolioItem.ClientId=Client.ClientId)		.02
(Sort (Client.ClientId)		.00
Iscan (Client.Name="Vikram Sharma") ClientName () Client)		2
(Iscan (PortfolioItem.ClientId=Client.ClientId) PortfolioItemClientId() PortfolioItem))		50

As expected, the inner relation is now scanned using the index PortfolioClientId and the join predicate. It is interesting to note that the cost of the optimum QEP has improved dramatically from 15,018 previously to 52 after implementation of the new transformation.

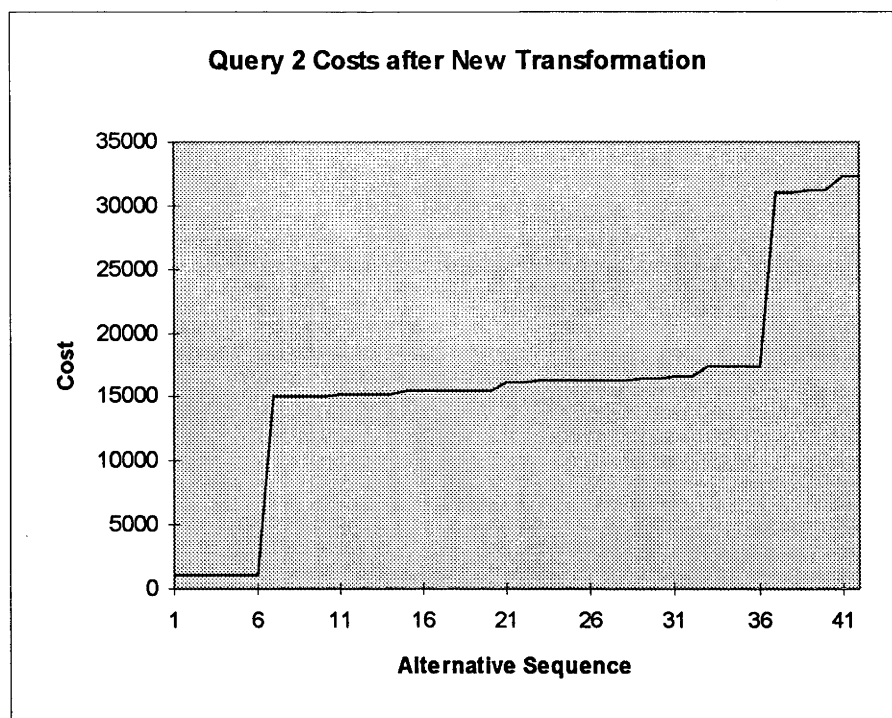
The optimum QEP for Query 2 now is :

Query 2		Step cost
QEP Cost 1073		
Mjoin (Client.AdvisorId=Advisor.AdvisorId)		.02
(Sort (Advisor.AdvisorId)		.00
Fscan (Advisor.Name="John Francis") Advisor)		12
(Sort (Client.AdvisorId)		.07
(Mjoin (PortfolioItem.ClientId=Client.ClientId)		1
(Sort (Client.ClientId)		10
Fscan () Client		1000
(Iscan (PortfolioItem.ClientId=Client.ClientId) PortfolioClientId () PortfolioItem))))		50

Again, the inner relation is now scanned using the index and the cost of the optimum solution has decreased from 16,237 to 1,073 .



The graph above shows the distribution of costs for Query 1 after the new transformation has been introduced. The four additional QEPs have significantly lower costs when compared with the previous optimum QEP. QEPs 1 and 2 are dramatically superior, while QEPs 3 and 4 still represent a significant improvement.



The graph above presents the new distribution of costs for Query 2. The first six QEPs, with a cost range of 1073 to 1081, are significantly superior to best QEP prior to the introduction of the new transformation.

It is desirable that BBQ should still produce some of the new superior QEPs when conducting restricted searches. To examine its performance in such searches, the number of alternatives expanded was severely limited by setting the parameter `BranchingFactor = 2`. This allows each knowledge source to expand only the two most promising alternatives.

With this restriction for Query 1, four out of 12 possible QEPs were generated. The four QEPs generated represented the best four solutions possible from the exhaustive search incorporating the new transformation.

With this restriction for Query 2, eight out of 42 possible QEPs were generated. The costs of the best two alternatives generated in this instance were within 1% of the new optimum QEP cost.

Thus an example of the implementation of an extension to the transformation rule set has been demonstrated. The change was effected with relative ease and resulted in significant improvement in QEP costs for the two sample queries examined. Also, the new rules were able to make useful contributions even when the search space was restricted.

5.8 ADVANTAGES OF BLACKBOARD ARCHITECTURE

A blackboard architecture was selected as the basis of the design of BBQ as it promised to offer a number of advantages over more traditional designs. Following the construction of software based on this design, this section takes a retrospective look at the benefits which were actually delivered.

As a result of storing emerging partial solutions on a common blackboard which is operated upon by a number of knowledge sources, it was possible to segment the design into modules which in some fashion mirrored a natural decomposition of the problem domain. This led to high degree of correlation between the problem domain and its programmatic representation and thus transparency of the algorithms being implemented.

In addition to modularity, the structure of the blackboard allowed knowledge sources to be constructed with minimal interactions/interfaces between them. This provided great flexibility in adding and altering knowledge sources in that it greatly simplified impact analysis.

The design also allows a high degree of decoupling of control logic from problem solving logic. Thus changes to the control strategy could be effected without impacting problem solving logic and vice-versa. This capability is of particular significance as one of the goals of BBQ is to provide the basis for a self-tuning optimiser. It is envisaged that ultimately the optimiser could use the results of a benchmarking process to initially set values for tunable parameters and then continually improve these values as it performs more and more optimisations. This is discussed in greater detail in the next chapter.

The use of a blackboard architecture also supports the control strategy. As all emerging solutions are globally visible and each has an associated cost estimate, comprising a historical and a future component, the process of selection of the next node to expand is greatly facilitated. The architecture also supports the incremental formation of partial solutions and allows flexibility in the granularity of the increments. Thus the knowledge sources can be defined to reflect a level of granularity considered appropriate for the control strategy.

The control strategy was readily able to use the structures provided by the blackboard to focus the search and thus produce complete solution alternatives after expansion of only a limited number of the candidate partial solutions. This allowed the breadth and depth of search to be limited according to certain tunable parameters. This is discussed further in the next section.

The derivation tag mechanism described in preceding chapters enabled the collection of metrics which were used to construct delta profiles graphing the efficiency of knowledge sources. This provided an effective means of identifying specific knowledge sources which could be candidates for improvement.

A final point which should be made is that while the blackboard architecture provided a number of benefits, the data structures required to support its implementation were complex and required a greater degree of effort to design efficiently than may have been the case in other designs.

5.9 BENEFITS OF SEARCH STRATEGY

The search strategy used by BBQ is a modified version of A* as described in preceding chapters. It selectively alternates between pure cost-based control and procedural control. The cost model used estimates both the cost of operations already incorporated and also the cost of operations yet to be incorporated for each partial solution alternative.

The strategy imposes an iterative procedural sequence at the top level but it allows controlled A* at the knowledge source level. This allowed complete QEPs to be generated with the expansion of a limited number of candidate partial solutions. Thus when operating under constraints on the time to perform the optimisation, a set of possible QEPs could be generated at an early stage.

An important consideration in assessing efficiency of BBQ was the quality of these early QEPs. It was desired that at least one near optimal alternative be present in this set. Analysis of QEPs produced presented in a previous section shows this to be case in a significant portion of the sample set. A significant factor in achieving this objective was the use of a cost model containing historical and future components.

As discussed in Chapter 4, it is not the intention of this research to devise a rigorous model for estimating the costs of partial and complete QEP alternatives. It does however attempt to use a model which provides an approximation of these costs and can be used in investigating the proposed optimiser design. A more comprehensive model could be the subject of further research.

The A* and procedural components and the termination condition are controlled by the setting of a number of tunable parameters. The results of variations to these parameters have been presented in a previous section. Changes to their settings allow control over the breadth and depth of search and the trade-off between optimisation time and quality of QEP produced which is one of the qualities desirable in an optimiser. This capability to set these parameters to achieve a reasonable compromise in this trade-off was one of the objectives of this design.

6. FUTURE WORK AND EXTENSIONS

This chapter examines some possibilities for extending the concepts presented in this thesis. In particular, it discusses extension of functionality, collection of a representative set of SQL statements for calibration, automating tuning of the optimiser and feasibility of using BBQ as a tool for benchmarking. It concludes with a discussion on the applicability of the model to commercial database management systems.

6.1 EXTENSION OF SOURCE LANGUAGE AND TRANSFORMATION RULE SET

One of the main criteria driving the design of BBQ is that it should readily support enhancement and extension of functionality. The scope of this thesis limited the source language to a subset of SQL containing conjunctive queries and excluding sub-queries. An area for further research would be to extend the source language to include a more complete set of SQL. However, it should be noted that a significant class of subqueries are logically equivalent to joins. A number of commercial database management systems incorporate transformation rules to translate such subqueries to equivalent join expressions.

The modular structure of the knowledge sources which implement the transformation rules facilitates the task of adding new transformations to process the extended input language. Depending on the scope of the extension, new rules could be added to existing knowledge sources or new ones may be created and the control strategy altered accordingly. The changes to knowledge sources may require modification of existing data classes or the creation of new ones to hold emerging alternatives.

6.2 SET OF REPRESENTATIVE SQL FOR CALIBRATING BBQ

A key area for further research would be the identification of a set of SQL statements which are representative of a broad class of queries. This set could be used to calibrate the performance of BBQ in a particular database and hardware environment. An

investigation of existing benchmark query sets may be an appropriate starting point for this process.

The calibration process would involve running optimisations for the representative set of SQL statements, ordering the resultant QEPs according to execution time and back propagating metrics which have been collected to the knowledge sources. This would enable the tunable parameters to be adjusted to optimum settings.

The identification of such a set of SQL statements has not been examined in this research. It may prove several such sets can be created and the set selected for use in calibration is that with characteristics most similar to queries likely to be input.

6.3 SELF TUNING OPTIMISER

The search strategy used by BBQ can be controlled by adjusting the values of parameters which are part of its architecture. The parameters determine factors such as the number of alternatives expanding in one iteration of a knowledge source and the termination condition for the search. Other parameters include constants which are used by the cost model.

Specific aspects of BBQ's behaviour which can be modified include depth and breadth of the search and the relative weighting of IO operations to CPU operations. By using information from the delta profiles for the knowledge sources these parameters can be progressively refined to improve the performance of the optimiser.

It can be envisaged that a representative set of SQL statements, as discussed in the preceding section, is optimised and delta profiles constructed from the results are used to establish initial parameter settings for BBQ.

Ultimately it is feasible for the results of each optimisation to be back propagated to the knowledge sources which contributed to the optimisation. This could enable each knowledge source to automatically adjust tunable parameters to more optimal values.

In such a scenario, the optimiser would improve performance over time by tuning itself as it generates optimisations.

6.4 TOOL FOR BENCHMARKING AND TESTING TRANSFORMATION RULES

The architecture of BBQ provides a framework within which the performance of sets of transformation rules can be evaluated. Metrics generated by BBQ can be utilised for quantitative analysis. These metrics can also be used to produce delta profiles as described in Section 2.3.

Therefore, it is possible to experiment with alternative sets of rules and benchmark the performance of each set. The impact of changes to database structure or hardware configuration can be investigated. This allows development of efficient transformation rules and permits tuning of the optimiser to maximise query execution performance in a specific operating environment.

While the basis for these types of analyses exists in the proposed design, further research could develop more sophisticated techniques for benchmarking and perhaps extend the types of metrics collected.

6.5 APPLICABILITY TO COMMERCIAL DBMS'

Optimisers which rely purely on a predetermined program sequence and set of rules to generate QEPs do not consistently produce results which can be regarded as near optimal. This has led to implementations where the user can assist the optimiser by providing hints on execution with the query. This however, requires knowledge of the physical structure of the database and still does not provide a satisfactory QEPs for a range of queries.

The design proposed in this thesis should in general produce superior solutions as it searches a number of possible alternatives selecting and then refining a subset on the basis of an estimated cost of execution. Additionally, the search algorithm and cost

model are designed such that near optimal QEPs should occur early in the sequence of alternatives generated.

While the model presented requires significant work to bring to a stage where it can effectively process a complete implementation of SQL, it does possess some desirable characteristics :

- generation of efficient QEPs
- flexibility and extensibility
- control over trade-off between time for optimisation and quality of solution
- ability to improve performance over time

With these advantages to offer, it is possible that this model for optimiser design could benefit commercial database management systems.

7. CONCLUSIONS

An optimiser design which draws upon existing work in the areas of blackboard systems and rule-based optimiser design has been proposed. The model brings draws on concepts developed in previous research works and extends these to present a novel optimiser design. The proposed design was implemented in Aion/DS, a knowledge base development tool.

The model possesses a number of desirable features. It addresses each of the characteristics worth attaining in an optimiser, as listed in the Chapter 1.

The blackboard-based design proposed, comprising a global structure for storing emerging solution alternatives and a set of knowledge sources implementing query transformation rules, is well structured with limited interaction between components. In this model, the heuristics driving the search strategy are segregated from the logic implementing the transformations. This allowed great flexibility both in extending functionality and in changing the strategy used to drive the search for near-optimal QEPs. Also, the use of a set of rules to perform the translation of the input query to a query execution plan adds transparency to the optimisation process.

The search strategy adopted is a modified version of the A* algorithm utilising a cost model incorporating historical and future costs for emerging solution alternatives. The search strategy and cost model enabled early selection of promising alternatives thus enhancing optimiser efficiency and quality of QEPs produced.

A number of tunable parameters which can be used to control the behaviour of the search are defined in the model. These parameters allow tuning of the optimiser to a specific mix of transformation rules, database structure and hardware environment. They also enable control over the trade-off between quality of QEPs produced and time taken to perform the optimisation.

A mechanism which enables the performance of individual components of the optimiser to be quantified has also been described. This is implemented by tagging each solution alternative with a structure showing the knowledge sources which participated in its formation. Metrics derived from these structures were used to construct delta-profiles for the knowledge sources. The delta profiles provided an easily comprehensible format for assessing the quality of knowledge sources and assisted in identifying those which could be improved.

Changes to the search strategy, by altering the settings of BBQ control parameters, were also demonstrated. While changes to control parameters are manual in the present design, further development of the model could lead to automation of this function so that ultimately, the optimiser is able to tune itself. This would lead to an optimiser design where performance improves over time as it learns from the result of previous optimisations.

Since the optimiser is constructed as a set of knowledge sources, each of which is an independent program unit, it could be readily adapted to take advantage of any parallel processing capability. Also, as all the data accessed is globally visible on the blackboard, communication between knowledge sources is minimised which further facilitates parallel operation.

In conclusion, an approach to optimiser construction which possesses several desirable characteristics has been proposed. As distinct from many contemporary optimiser designs, which compromise on certain characteristics to perform better in others, the proposed architecture is able to perform well against a number of criteria without significant trade-offs. This work provides a foundation for an optimiser whose design represents an advance over traditional optimiser architectures.

APPENDIX A - LISTING OF RULES

A listing of the main transformation rules used in BBQ is given below. The language used for implementation is KDL which is part of the Aion/DS development environment. The listing shows the rules grouped by knowledge source.

Knowledge Source : AlgebraicForm

Rule : ConvertSql

```
ifmatch InputSql with
  QueryName = NameOfQueryToOptimise
then
  for RelationList, Idx
    create (Relation with Name=RelationList(Idx))
  end

  for ProjectList, Idx
    create (ProjectAttribute with
      RelationName= ExtractRel(ProjectList(Idx)),
      Attribute= ExtractAttr(ProjectList(Idx)))
  end

  for SelectList, Idx
    create (SelectPredicate with
      RelationName=ExtractRel(ExtractOperand1(SelectList(Idx))),
      Attribute=ExtractAttr(ExtractOperand1(SelectList(Idx))),
      Operator=ExtractOperator(SelectList(Idx)),
      Constant=ExtractOperand2(SelectList(Idx)))
  end

  for JoinList, Idx
    create (JoinPredicate with
      RelationName1=ExtractRel(ExtractOperand1(JoinList(Idx))),
      Attribute1=ExtractAttr(ExtractOperand1(JoinList(Idx))),
      Operator=ExtractOperator(JoinList(Idx)),
      RelationName2=ExtractRel(ExtractOperand2(JoinList(Idx))),
      Attribute2=ExtractAttr(ExtractOperand2(JoinList(Idx))))
  end
end
```

Knowledge Source : JoinMethods

Rule : CalcHistoricalCost

```
ifmatch TjoinExpression with
```



```
Status = 'LMJoinsDone' and
NodeStatus = 'Open'
then

for TjoinExpression.TjoinPtrList

if i > 1
then
  InnerCardinality = TjoinPtrList(i-1)->.TjoinCardinality

  HistoricalCost = HistoricalCost + JoinCost(TjoinPtrList(i)->.ScanCardinality,
                                             InnerCardinality,
                                             ExtractJoinType(TjoinPtrList(i)->))

end
end

FutureCost = 0

if vMinCostThisIteration = 0 or (HistoricalCost < vMinCostThisIteration)
then
  vMinCostThisIteration = HistoricalCost
end

if vMinCostAllAlternatives = 0 or
  (HistoricalCost < vMinCostAllAlternatives)
then
  vMinCostAllAlternatives = HistoricalCost
end
end
```

Rule : CreateLjoins

```
ifmatch TjoinExpression with
  NodeStatus = 'Selected'
  orderby (round(HistoricalCost))
then
  Idx = 0
  for TjoinExpression.TjoinPtrList
    if ExtractJoinType(TjoinPtrList(i)->) = 'Tjoin' and i > 1
    then
      Idx = i
      break
    end
  end
end

if Idx > 0
then
  TjoinPtr = TjoinExpression.TjoinPtrList(Idx)
```

```
TjoinPtr = create (Ljoin with JoinPredicatePtrList =
  TjoinPtr->.JoinPredicatePtrList,
  ScanPtr = TjoinPtr->.ScanPtr,
  ScanCardinality = TjoinPtr->.ScanCardinality,
  TjoinCardinality = TjoinPtr->.TjoinCardinality,
  Selectivity = TjoinPtr->.Selectivity)
TjoinExpressionPtr = create (TjoinExpression with
  TjoinPtrList = TjoinExpression(1).TjoinPtrList,
  RelationNameList = TjoinExpression(1).RelationNameList,
  KSAAlternativeIDList = TjoinExpression(1).KSAAlternativeIDList,
  HistoricalCost = TjoinExpression(1).HistoricalCost,
  NodeStatus = 'Selected')
TjoinExpressionPtr->.TjoinPtrList(Idx) = TjoinPtr

AddAlternativeIDToList(TjoinExpressionPtr,
  'CreateLJoins')
```

```
end
end
```

Rule : CreateMjoins

```
ifmatch TjoinExpression with
  NodeStatus = 'Selected'
  orderby (round(HistoricalCost))
then

  TjoinPosition = 0
  for TjoinExpression.TjoinPtrList
    if ExtractJoinType(TjoinPtrList(i)->)= 'Tjoin' and i > 1
    then
      TjoinPosition = i
      break
    end
  end

  if TjoinPosition > 0
  then
    TjoinPtr = TjoinExpression.TjoinPtrList(TjoinPosition)

    if not (currentvalue(TjoinPtr->.JoinPredicatePtrList) is unknown)
    then
      NewTjoinPtr = create (Mjoin with
        JoinPredicatePtrList = TjoinPtr->.JoinPredicatePtrList,
        ScanPtr = TjoinPtr->.ScanPtr,
        ScanCardinality = TjoinPtr->.ScanCardinality,
        TjoinCardinality = TjoinPtr->.TjoinCardinality,
        Selectivity = TjoinPtr->.Selectivity)
```

```
TjoinExpressionPtr =  
  create (TjoinExpression with  
    RelationNameList = TjoinExpression(1).RelationNameList,  
    TjoinPtrList = TjoinExpression(1).TjoinPtrList,  
    KSAlternativeIDList = TjoinExpression(1).KSAlternativeIDList,  
    HistoricalCost = TjoinExpression(1).HistoricalCost,  
    NodeStatus = 'Selected')
```

```
TjoinExpressionPtr->.TjoinPtrList(TjoinPosition) = NewTjoinPtr
```

```
  AddAlternativeIDToList(TjoinExpressionPtr, 'CreateMJoins')  
end  
end  
end
```

Rule : SelectFirstScan

```
ifmatch Scan with TRUE  
then  
  TjoinPtr = create (Tjoin with ScanPtr = ->Scan)  
  
  TjoinExpressionPtr= create (TjoinExpression with Status='FirstScan')  
  add Scan.RelationName to TjoinExpressionPtr->.RelationNameList  
  add TjoinPtr to TjoinExpressionPtr->.TjoinPtrList  
  
  AddAlternativeIDToList(TjoinExpressionPtr, 'SelectFirstScan')  
end
```

Rule : SelectNodesToExpand

```
ifmatch TjoinExpression with  
  Status = 'TJoinsDone' and  
  NodeStatus = 'Open' and  
  NewNodeCount < KSBranchingFactor('JoinMethods')  
orderby (round(HistoricalCost+FutureCost))  
then  
  NewNodeCount = NewNodeCount + 1  
  NodeStatus = 'Selected'  
end
```

Rule : UpdateNodeStatus

```
ifmatch TjoinExpression with  
  NodeStatus = 'Selected'  
then  
  NodeStatus = 'Closed'  
  
  TjoinFound = FALSE  
  for TjoinPtrList
```

```
if ExtractJoinType(TjoinPtrList(i->)) = 'Tjoin' and i > 1
then
  TjoinFound = TRUE
end
end
```

```
if not TjoinFound
then
  TjoinExpression.Status = 'LMJoinsDone'
  NodeStatus = 'Open'
end
end
```

Knowledge Source : ScanMethods

Rule : AddPredicatesToFscans

```
ifmatch Fscan, SelectPredicate with
  Fscan.RelationName = SelectPredicate.RelationName
then
  add ->SelectPredicate to Fscan.SelectPredPtrList
end
```

Rule : AddPredicatesToIscans1

```
ifmatch
  Iscan, SelectPredicate with
  SelectPredicate.RelationName = Iscan.RelationName
then

  if index(SelectPredicate.Attribute, Iscan.IndexPtr->.AttributeList)
    = size(Iscan.IndexPredPtrList) + 1
  then
    add ->SelectPredicate to Iscan.IndexPredPtrList
  end

end
```

Rule : AddPredicatesToIscans2

```
ifmatch Iscan, SelectPredicate with
  SelectPredicate.RelationName = Iscan.RelationName and
  not (Iscan.IndexPredPtrList includes ->SelectPredicate)
then

  add ->SelectPredicate to Iscan.SelectPredPtrList

end
```

Rule : GenerateFscans

```
ifmatch Relation with TRUE
then
  create (Fscan with RelationName = Relation.Name)
end
```

Rule : GenerateIscans

```
ifmatch Relation, DBIndex, SelectPredicate with
  DBIndex.RelationName = Relation.Name and
  SelectPredicate.RelationName = DBIndex.RelationName and
  SelectPredicate.Attribute = DBIndex.AttributeList(1)
then
  ScanPtr = create (Iscan with RelationName = Relation.Name,
                   IndexPtr = ->DBIndex)

  add ->SelectPredicate to ScanPtr->.IndexPredPtrList
end
```

Knowledge Source : SortOperators

Rule : CalcHistoricalCost

```
ifmatch TjoinExpression with
  Status = 'SortsDone'
then

  for TjoinExpression.TjoinPtrList

    if currentvalue(TjoinPtrList(i)->.InnerSortAttributeList) is not unknown
    then
      HistoricalCost = HistoricalCost + SortCost(TjoinPtrList(i-1)->.TjoinCardinality)
    end

    if currentvalue(TjoinPtrList(i)->.OuterSortAttributeList) is not unknown
    then
      HistoricalCost = HistoricalCost + SortCost(TjoinPtrList(i)->.ScanCardinality)
    end
  end
end
```

Rule : SortInnerExpression

```
ifmatch TjoinExpression with
  Status = 'LMJoinsDone'
then
```

```
for TjoinExpression.TjoinPtrList, MjoinPosition
  if ExtractJoinType(TjoinPtrList (MjoinPosition)->) = 'Mjoin'
  then

    TjoinPtr = TjoinExpression.TjoinPtrList(MjoinPosition)

    TupleOrder(->TjoinExpression, MjoinPosition, MjoinPosition, OuterPredicateList)

    clear(InnerPredicateList)

    for OuterPredicateList, Idx1
      for TjoinPtr->.JoinPredicatePtrList, Idx2
        if TjoinPtr->.JoinPredicatePtrList(Idx2)->.RelationName1 & '!' &
          TjoinPtr->.JoinPredicatePtrList(Idx2)->.Attribute1 = OuterPredicateList(Idx1)
        then
          add TjoinPtr->.JoinPredicatePtrList(Idx2)->.RelationName2 & '!' &
            TjoinPtr->.JoinPredicatePtrList(Idx2)->.Attribute2
            to InnerPredicateList
          break
        else
          if TjoinPtr->.JoinPredicatePtrList(Idx2)->.RelationName2 & '!' &
            TjoinPtr->.JoinPredicatePtrList(Idx2)->.Attribute2 = OuterPredicateList(Idx1)
          then

            add TjoinPtr->.JoinPredicatePtrList (Idx2)->.RelationName1 & '!' &
              TjoinPtr->.JoinPredicatePtrList(Idx2)->.Attribute1
              to InnerPredicateList
            break
          end
        end
      end
    end

    TupleOrder(->TjoinExpression, 1, MjoinPosition-1, TupleOrderList)

    if not IsOrderSame(InnerPredicateList, currentvalue(TupleOrderList), 1)
    then
      TjoinPtr->.InnerSortAttributeList = InnerPredicateList
    end
  end
end

TjoinExpression.Status = 'SortsDone'

end
```

Rule : SortOuterExpression

```
ifmatch TjoinExpression with
  Status = 'LMJoinsDone'
then

  for TjoinExpression.TjoinPtrList, MjoinPosition
  if ExtractJoinType(TjoinPtrList(MjoinPosition)->) = 'Mjoin'
  then

    TjoinPtr = TjoinExpression.TjoinPtrList(MjoinPosition)

    clear(OuterPredicateList)

    for TjoinPtr->.JoinPredicatePtrList
    if TjoinPtr->.ScanPtr->.RelationName
      =TjoinPtr->.JoinPredicatePtrList(i)->.RelationName1
    then
      add TjoinPtr->.JoinPredicatePtrList(i)->.RelationName1 & '!' &
        TjoinPtr->.JoinPredicatePtrList(i)->.Attribute1
      to OuterPredicateList
    else
      add TjoinPtr->.JoinPredicatePtrList(i)->.RelationName2 & '!' &
        TjoinPtr->.JoinPredicatePtrList(i)->.Attribute2
      to OuterPredicateList
    end
  end
end

TupleOrder(->TjoinExpression, MjoinPosition, MjoinPosition, TupleOrderList)

if not IsOrderCompatible(OuterPredicateList, currentvalue(TupleOrderList), 1)
then
  TjoinPtr->.OuterSortAttributeList = OuterPredicateList
end
end
end
end
```

Knowledge Source : JoinOrders

Rule : CalcHistoricalCost

```
ifmatch TjoinExpression with
  Status = 'TjoinsDone' and
  NodeStatus = 'Open'
then

  for TjoinExpression.TjoinPtrList
```

```
TjoinPtrList(i)->.ScanCardinality = ScanCardinality(TjoinPtrList(i)->.ScanPtr)
```

```
if i = 1
```

```
then
```

```
HistoricalCost = ScanCost(TjoinPtrList(i)->.ScanPtr)
```

```
TjoinPtrList(i)->.TjoinCardinality = TjoinPtrList(i)->.ScanCardinality
```

```
else
```

```
TjoinPtrList(i)->.Selectivity =
```

```
JoinSelectivity(currentvalue(TjoinPtrList(i)->.JoinPredicatePtrList))
```

```
TjoinPtrList(i)->.TjoinCardinality =
```

```
TjoinPtrList(i)->.ScanCardinality *
```

```
TjoinPtrList(i-1)->.TjoinCardinality *
```

```
TjoinPtrList(i)->.Selectivity
```

```
HistoricalCost = HistoricalCost + ScanCost(TjoinPtrList(i)->.ScanPtr)
```

```
end
```

```
end
```

```
end
```

Rule : CreateTjoins

```
ifmatch
```

```
Scan, TjoinExpression
```

```
with NodeStatus = 'Selected' and
```

```
not (TjoinExpression.RelationNameList includes Scan.RelationName)
```

```
orderby (round(HistoricalCost+FutureCost))
```

```
then
```

```
JoinPredPtrList = selectall(JoinPredicate with
```

```
(Scan.RelationName = RelationName1 and
```

```
RelationNameList includes RelationName2) or
```

```
(Scan.RelationName = RelationName2 and
```

```
RelationNameList includes RelationName1))
```

```
TjoinPtr = create (Tjoin with ScanPtr= ->Scan,
```

```
JoinPredicatePtrList = JoinPredPtrList)
```

```
TjoinExpressionPtr = create (TjoinExpression
```

```
with TjoinPtrList = TjoinExpression(1).TjoinPtrList,
```

```
RelationNameList = TjoinExpression(1).RelationNameList,
```

```
KSAAlternativeIDList =TjoinExpression(1).KSAAlternativeIDList,
```

```
PreviousHistoricalCost =TjoinExpression(1).PreviousHistoricalCost,
```

```
TjoinIdxAtPreviousCost =TjoinExpression(1).TjoinIdxAtPreviousCost,
```

```
NodeStatus = 'Selected')
```

```
add Scan.RelationName to TjoinExpressionPtr->.RelationNameList
```

```
add TjoinPtr to TjoinExpressionPtr->.TjoinPtrList
```

```
AddAlternativeIDToList(TjoinExpressionPtr, 'CreateTJoins')
```


end

Rule : SelectNodesToExpand

```
ifmatch TjoinExpression with
  Status = 'FirstScan' and
  NodeStatus = 'Open' and
  NewNodeCount < KSBranchingFactor('TJoins')
  orderby (round(HistoricalCost+FutureCost))
then
  NewNodeCount = NewNodeCount + 1
  NodeStatus = 'Selected'
end
```

Rule : UpdateNodeStatus

```
ifmatch TjoinExpression with
  NodeStatus = 'Selected'
then
  NodeStatus = 'Closed'

  if size(TjoinExpression.RelationNameList) = size(selectall(Relation))
  then
    TjoinExpression.Status = 'TJoinsDone'
    NodeStatus = 'Open'
  end
end
```

The following rules was added to the JoinMethods knowledge source to implement the additional transformation which, if applicable, utilises an index along with join predicate(s) to scan the inner relation in a join.

Rule : GenerateIscans2

```
ifmatch
  Relation, DBIndex, JoinPredicate with
  DBIndex.RelationName = Relation.Name and
  ((JoinPredicate.RelationName1 = DBIndex.RelationName and
    JoinPredicate.Attribute1 = DBIndex.AttributeList(1)) or
  (JoinPredicate.RelationName2 = DBIndex.RelationName and
    JoinPredicate.Attribute2 = DBIndex.AttributeList(1))) and
  not exists (Iscan with RelationName = Relation.Name)
then
  ScanPtr = create (Iscan with
    RelationName = Relation.Name,
    IndexPtr = ->DBIndex)
```

```
add ->JoinPredicate to ScanPtr->.IndexJoinPredPtrList
```

```
end
```

Rule : AddPredicatesToIscans3

```
ifmatch
```

```
  Iscan, JoinPredicate with
```

```
  JoinPredicate.RelationName1 = Iscan.RelationName
```

```
then
```

```
  if index(JoinPredicate.Attribute1, Iscan.IndexPtr->.AttributeList)  
    = size(Iscan.IndexPredPtrList) + 1
```

```
  then
```

```
    add ->JoinPredicate to Iscan.IndexJoinPredPtrList
```

```
  end
```

```
end
```

Rule : AddPredicatesToIscans4

```
ifmatch
```

```
  Iscan, JoinPredicate with
```

```
  JoinPredicate.RelationName2 = Iscan.RelationName
```

```
then
```

```
  if index(JoinPredicate.Attribute2, Iscan.IndexPtr->.AttributeList)  
    = size(Iscan.IndexPredPtrList) + 1
```

```
  then
```

```
    add ->JoinPredicate to Iscan.IndexJoinPredPtrList
```

```
  end
```

```
end
```

APPENDIX E - BBO INTERNAL DATA STRUCTURES

BBQ is implemented in Aion/DS which supports object oriented concepts such as object classes and inheritance. A description of the main classes and sub-classes of objects which comprise the blackboard as defined in the implementation is given below. At run time instances of these classes are created to represent solution alternatives.

<i>Class/Subclass Name</i>	<i>Slot Name</i>
DBCardinality	Cardinality
	RelationName
DBIndex	AttributeList
	IndexName
	RelationName
DBSelectivity	Attribute
	RelationName
	Selectivity
InputSql	JoinList
	ProjectList
	QueryName
	RelationList
	SelectList
JoinPredicate	Attribute1
	Attribute2
	Operator
	Relation1
	Relation2
ProjectAttribute	Attribute
	Relation
Relation	Name
Scan	RelationName
	SelectPredPtrList
Fscan	
Iscan	IndexName
	IndexPredPtrList
SelectPredicate	Attribute
	Constant
	Operator

	RelationName
Tjoin	JoinPredicatePtrList
	ScanCardinality
	ScanPtr
	Selectivity
	TjoinCardinality
Ljoin	
Mjoin	InnerSortAttributeList
	OuterSortAttributeList
TjoinExpression	FutureCost
	HistoricalCost
	KSAAlternativeIDList
	NodeStatus
	PreviousHistoricalCost
	RelationNameList
	TjoinIdxAtPreviousCost
	TjoinPtrList

BIBLIOGRAPHY

Adler M R and Simoudis E, Integrating Distributed Expertise, paper submitted to *International Working Conference on Cooperative Knowledge Based Systems*, University of Keele, England Oct 3-5, 1990

Blackboard Technology Group Inc, technical paper by the company - *The Blackboard Problem-Solving Approach*, Amherst, Massachusetts

Bond A H and Gasser L, An Analysis of Problems in Distributed Artificial Intelligence, in *Readings in Distributed Artificial Intelligence*, A H Bond and L Gasser (ed), Morgan Kaufmann, 1988

Corkhill D D, Gallagher K Q and Johnson P M, Achieving Flexibility, Efficiency, and Generality in Blackboard Architectures, in *Readings in Distributed Artificial Intelligence*, A H Bond and L Gasser (ed), Morgan Kaufmann, 1988

Cox B J, *Object Oriented Programming: An Evolutionary Approach*, Addison-Wesley, 1986

Elmasri R and Navathe S B, *Fundamentals of Database Systems*, Addison-Wesley, 1989

Engelmore R and Morgan J (ed), *Blackboard Systems*, Addison-Wesley, 1988

Freytag J C, A Rule-Based View of Query Optimization, in *Proceedings of the ACM SIGMOD Conference on Management of Data*, San Francisco USA, 1987, pp 173-180

Hanson E N, Rule Condition Testing and Action Execution in Ariel, ACM SIGMOD, June 1992, pp 49-58

Hayes-Roth B, A Blackboard Architecture for Control, in *Readings in Distributed Artificial Intelligence*, A H Bond and L Gasser (ed), Morgan Kaufmann, 1988

Hudlicka E and Lesser V, Modeling and Diagnosing Problem-Solving System Behaviour, in *Readings in Distributed Artificial Intelligence*, A H Bond and L Gasser (ed), Morgan Kaufmann, 1988

Kemper A, Moerkotte G and Peithner K, A Blackboard Architecture for Query Optimisation in Object Bases in *Proceedings of the 19th VLDB Conference*, Dublin Ireland, 1993, pp 543-554

Krishnamurthy R, Boral H and Zaniolo C, Optimization of Nonrecursive Queries in *Proceedings of the Twelfth International Conference on Very Large Data Bases*, Kyoto Japan, August 1986, pp 128-137

Leao L V and Talukdar S N, COPS: A System for Constructing Multiple Blackboards, in *Readings in Distributed Artificial Intelligence*, A H Bond and L Gasser (ed), Morgan Kaufmann, 1988

Lochovsky F H, *Knowledge Communication in Intelligent Information Systems*

Nii H P, *Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures*, AI Magazine, Summer 1986, pp 38-53

Pomeroy B and Irving R, *A Blackboard Approach for Diagnosis in Pilot's Associate*, IEEE Expert, August 1990, pp 39-46

Trinzic Corporation, *AionDS Language Reference*, Trinzic Corporation, 1994

Yoshida N and Narazaki S, A Cooperation and Communication Framework for Distributed Problem Solving, in *Proceedings of the Fourteenth Annual International Computer Software Applications Conference*, pp , Chicago 1990