SYSTEM R STYLE JOIN ORDER OPTIMIZATION FOR INTERNET INFORMATION GATHERING

by

Senthil Gnanaprakasam

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

May 2001

SYSTEM R STYLE JOIN ORDER OPTIMIZATION FOR INTERNET INFORMATION GATHERING

by

Senthil Gnanaprakasam

has been approved

May 2001

APPROVED:

____, Chair

____

____

Supervisory Committee

ACCEPTED:

____

Department Chair

____

Dean, Graduate College

ABSTRACT

Internet information gathering is the process of gathering data from sources that include those scattered over the Internet. Query optimization problems for Internet information gathering are different from that of traditional databases due to the lack of knowledge of the behavior of sources and a myriad of binding constraints that exist for many sources over the Web.

Traditional System R style optimizers lose their efficacy when sources are spread across the Internet with high access costs compared to secondary storage media. Such optimizers cannot be used for Internet sources due to various binding restrictions and query capacities.

This research proposes a System R style optimizer that takes binding patterns and restrictions that most Internet sources have. It considers both left and right linear evaluations along with bushy joins. The proposed algorithm assumes full knowledge of statistics and generates a join order accordingly. However, in the absence of full statistics, it degrades gracefully and maintains its improvement over previous algorithms.

To my sister

ACKNOWLEDGEMENTS

I am thankful to my committee chair Dr. Subbarao Kambhampati and committee members Dr Chitta Baral and Dr Susan Urban for guiding, encouraging and motivating me throughout this research work. I also thank Ullas Nambiar and Zaiqing Nie for their ideas, comments and constructive criticism during the course of my research and implementation. I appreciate the Yochan group for the moral support and for sharing my happy and not-so-happy moments.

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

The growing popularity of the Internet[1], especially the World Wide Web[2] has made it a prime vehicle for disseminating information. The number of structured (along with semi-structured ones) information sources is increasing rapidly. Most of these sources have a form-based Web interface and provide the user with information from a traditional database being maintained. Even though each of these sources is structured and supports high-level queries, the interaction with a multitude of sources is like surfing. The user must consider the list of available sources; decide/short-list the ones to access and manually combine the information obtained from such multiple sources.

An Information Gathering system provides a uniform query interface [Lev99a] for a multitude of autonomous heterogeneous data sources. The sources in such an application may be traditional databases, legacy systems or even structured files. The goal of a data integration system is to free the user from having to find the data sources relevant to a query, interact with each source in isolation, and manually combine data from such different sources. To provide a uniform interface, an information gathering system exposes the user to a mediated schema which is a set of virtual relations (they are not

---

[1] internet: (abbreviation for internetwork) A set of computer networks, possibly dissimilar, joined together by means of gateways that handle data transfer and the conversion of messages from the sending network to the protocols used by the receiving network. [AHK+91]

Internet: The collection of networks and gateways that use the TCP/IP suite of protocols. Many internets exist besides the Internet, including many TCP/IP based networks that are not linked to the Internet (the Defense Data Network is a case in point).

[2] WWW aka Web: A global hypertext system that uses the Internet as its transport mechanism. [AHK+91]

The Web is the most popular part of the Internet, while the rest are FTP/Gopher/telnet etc.

stored anywhere)- it is designed manually for a particular application. To be able to answer queries, the system must also contain a set of source descriptions that specify the contents of the source and the attributes contained in it and the corresponding constraints.

Figure 1.1 shows an example of a typical internet information gathering scenario. A machine in the library stores records of *Lost* books and details of each *Borrow* transaction in the secondary storage media. In a different building in the campus, the *Student* information is stored in the Administration building.

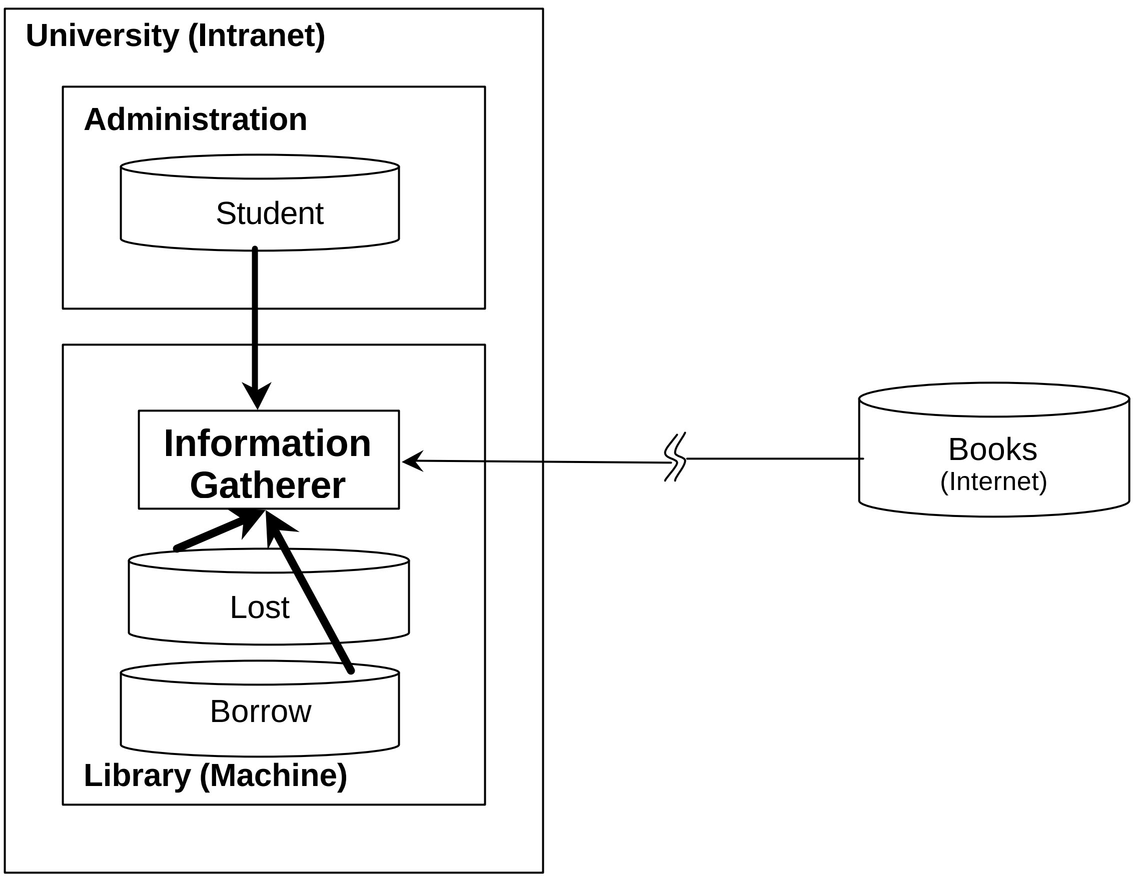

Fig 1.1 Internet information gathering scenario

An intranet connects the two buildings in the university. The library also has an internet link to a *Books* database that could be located tens to thousands of miles away.

While many of the traditional database techniques may be applied to internet information integration, some differences do exist.

- Internet information sources are autonomous - exporting no statistics about themselves, and hence their costs cannot be estimated easily. Access and transfer costs may not be accurately estimated for the *Books* database accessible on the Internet.

- Even when costs are known, Access and transfer times cannot be reliably estimated. Hence even a plan that appears to be optimal might turn out to be sub-optimal if there are unexpected delays in transferring the data from one of the sources.

- Sources have a variety of processing capabilities - while one source might be a form interfaced database another might be an unstructured HTML file. Even form-interfaced databases have additional types of constraints.

1.1. Source constraints

On the Internet, there are a variety of sources that have various constraints associated with them [LKG99].

- An ordinary HTML document needs a wrapper interface so that it may be modeled as a traditional database. For any kind of query posed on such a source, the entire file has to be retrieved regardless of the volume of information required, and extraction of data is done locally.

- Further, some sources have binding constraints that implies that some of the attributes need to be bound to a value. To illustrate, the schema for the *Books* source may be

  Books (isbn^b, title, author, publisher, pages, price)

  indicating that queries posed on this source need to provide a value for *isbn* to be semantically correct.

1.2. Execution optimization

Information gathering consists of three stages

1	Plan generation	Generate a source complete query plan
2	Plan optimization	Optimize the above plan using subsumption, LCW and other such information
3	Plan execution	Execute the optimized query plan for optimality.

This research concentrates on the third phase of information gathering - plan execution. One of the most important execution optimization strategies is to order the source calls in a manner so that the corresponding costs are minimized. Even small differences in sub-optimal partial join orders that constitute the final join multiply very rapidly as the number of source accesses increases.

There exist heuristics as well as systematic methods to find an optimal join order. Bound-is-easier is a heuristic that uses naïve heuristics in the absence of statistics to find the best order. System R [ABC+76] style join order algorithms use statistics to more accurately arrive at the optimal join order. However, they do not take binding constraints into account. This research concentrates on developing a join-ordering algorithm that uses query plans containing source statistics and takes care of binding patterns.

Access costs that represent overhead costs for an internet connection setup (as against disk seek time for traditional databases) are one of the important costs for internet information gathering. Bushy trees lend themselves naturally to internet information gathering by their inherent parallelism to cut down access time when possible by making simultaneous access to sources. For this reason, the search space is expanded to add bushy trees. An experimental setup with simulated sources where various parameters can be changed is used to test the algorithm. Empirical data shows that the algorithm scales as expected and is better than other comparable methods. Empirical data also validates the assumption that searching a larger space of join order trees is paid off with a smaller execution cost.

2. INTERNET INFORMATION GATHERING

Internet information gathering has to deal with different kinds of databases. Figure 2.1 shows an information gatherer than puts data together from different sources- varying from secondary storage media to internet databases. Traditional query optimization algorithms are designed to work with localized databases. The issues related to such an environment is very different from that of an internet one where databases are strewn across the Internet. Databases stored in the same secondary storage device, or in multiple devices accessible directly by the computer have comparatively lower (almost negligible) seek and read times than the setup time to establish a socket connection and transfer data in the case of databases across various networks.

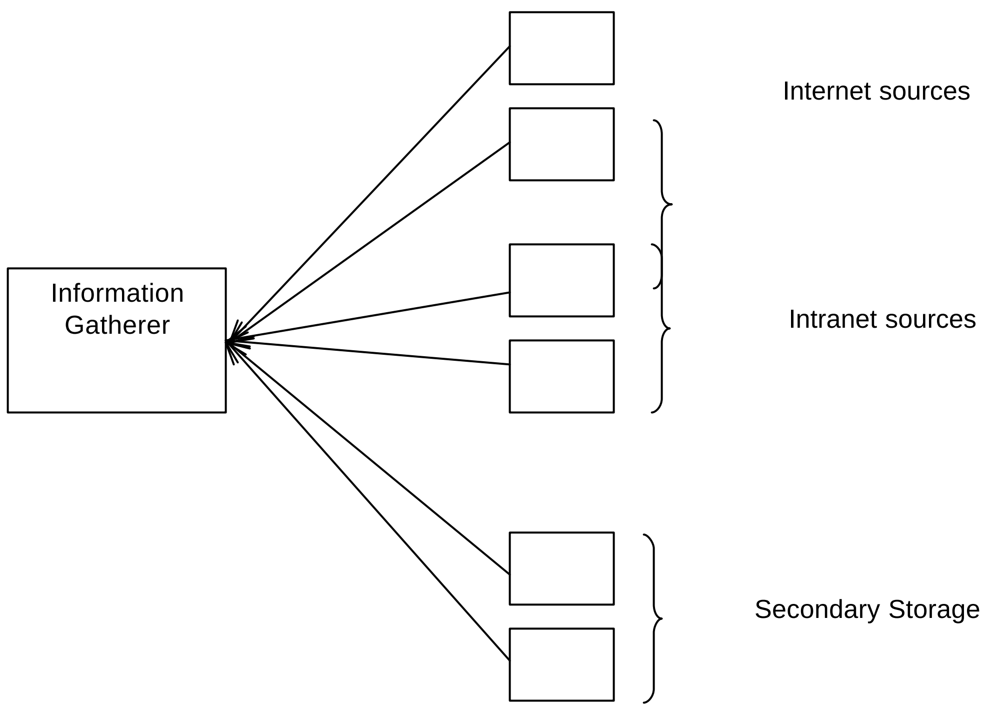

Fig 2.1 Different sources for an internet database.

2.1. Binding patterns

While posing a query to a database, some attributes may be bound to a particular value while some may not [LKG99]. Sometimes, the bound attribute may depend on the availability of data or the query itself. However, quite often in the case of internet databases, attributes may be required to be bound because of the inherent database design.

- The *Books* database requires *isbn* to be bound.

- Student records stored in a university administration may require the *id* of the student to be bound.

On the other hand, some sources may allow a richer set of queries than expected. For example, a source might take multiple bindings (limited disjunctive queries) for a particular attribute. While disregarding such information does not affect the soundness of a query result, optimality of a system is impaired if it doesn't take into account such features of sources. Such limitations and features must be taken into consideration if an optimal query plan is desired.

Examples:

`yellowPages (LastName`^{f}`, FirstName`^{f}`, Zip`^{f}`, Phone`^{b}`)`

`yellowPages (LastName`^{b}`, FirstName`^{f}`, Zip`^{f}`, Phone`^{f}`)`

`books(Title`^{f}`, Author`^{f}`, Price`^{f}`, ISBN`^{b}`, Pages`^{f}`)`

The variations of having some attributes bound and some other free are called binding patterns. In the example given above, `yellowPages` has two possible binding patterns. In the first one `Phone` is bound and denoted by ^{b} while the second one has `LastName` bound

2.2. Access costs

A discriminating feature of internet information gathering is the wide range of access costs of the sources involved. Access time is the overhead associated with getting data from a particular source. Access cost is the unavoidable cost associated with accessing a source without getting any data in return. For this reason, contribution of access costs to the execution cost of a plan must be kept minimal in order to obtain an optimal solution. Furthermore, access costs tend to be nonlinear as they are not proportionate to the number of tuples transferred.

Databases available locally on secondary storage media like hard disks have very small access time in the form of disk seek time. Those present across the intranet linked by high-speed internal networks serving a smaller load have moderate access times. At the far end of the spectrum are those scattered on the Internet where servers are used by a much larger number of people and hence have very high access times.

2.3. Types of internet databases

Databases accessible on the Internet exist in myriad forms. One of the most common and most visible are form interfaced databases. Usually in such cases, fixed queries are written and a web page designed so that it can accept values for various attributes. The result of such a query is used in further transactions. The server site usually limits queries that can be posed to such databases. Consider a book database with the following schema:

Books (ISBN, Title, Author, Publisher, Pages, Price)

A query of the form

```
SELECT * FROM Books WHERE Author="John Doe"
```

is acceptable whereas

```
SELECT * FROM Books WHERE Pages>50
```

is acceptable not even though it is semantically correct. Even though *Pages* is an attribute for *Books*, its value cannot be bound in queries posed on the relation. Such binding patterns are termed *Forbidden Binding Patterns*. For the above *Books* example, `Books (ISBN, Title, Author, Publisher, Price, Pages`^{b}`)` is a forbidden binding pattern.

Let us say that the design of the *Books* relation is such that it requires *ISBN* to be bound. In such a situation, the possible binding patterns are written as

Books (ISBN^b, Title, Author, Publisher, Price, Pages^f)

All binding patterns with the remaining attributes are any combination of *Free* or *Bound* are feasible and not part of Forbidden Binding Patterns.

Some unconventional databases are also present that are in text form. Such databases do not have any query processing capability. If needed to be queried, they have to be transferred in full to the client site and a wrapper program parses the file and converts it into a database for internal use. Text databases usually need customized wrappers and are also inefficient.

2.4. Source statistics

Another important characteristic of internet databases is the lack of statistics regarding their performance and data stored by them [LKG99]. It is not a trivial task to find out the access and transfer times for such databases. Even when known, the fluctuations in network connections can cause the statistics to vary more than that for databases stored in secondary storage media.

Possibly the most common scenario is where nothing is known about an internet database apart from the schema and binding restrictions. Sometimes, partial information may be available. Information that `SELECT * FROM Student WHERE Major=”CS”` returns fewer tuples than `SELECT * FROM Student WHERE Sex=”M”` might be easily available or estimated.

On the other side of the spectrum are databases where all the statistics are known. Corporate intranets are a prime example. Even for those sources for which no statistics are available, probing and other methods can return a very good estimate.

Cost forms an important factor in a query optimization algorithm. A more accurate estimate implies a better-informed decision made by the optimizer. Small errors in source statistics may not affect the final outcome of a join order optimizer as the costs are merely used to rank plans. It is very likely that the same order would have been produced with small changes in source statistics. This indicates the accuracy of the statistics required by most join order optimization algorithms. Small changes in statistics for internet databases are quite natural and common [AH00]. Because a high level of accuracy is not needed, such changes do not usually change the optimality of the solution produced by such algorithms.

3. JOIN ORDERS

The number of permutations of ordering the join operations can provide many equivalent alternatives that provide a sound solution with varying degrees of optimality. Finding an optimal join ordering for a given query is the task of join order optimizers. Selecting the optimal execution strategy for a query is NP-hard in the number of relations [IK84].

Consider the following query

```
SELECT StudentName
FROM Student, Course, Dept
WHERE
      Student.id=Course.takenBy AND
      Course.deptId=Dept.id AND
      Student.Major=Dept.Id
```

The following figure illustrates equivalent join queries all of which are semantically correct and yield the same results. However, depending on the attributes of the sources, the corresponding costs may vary.

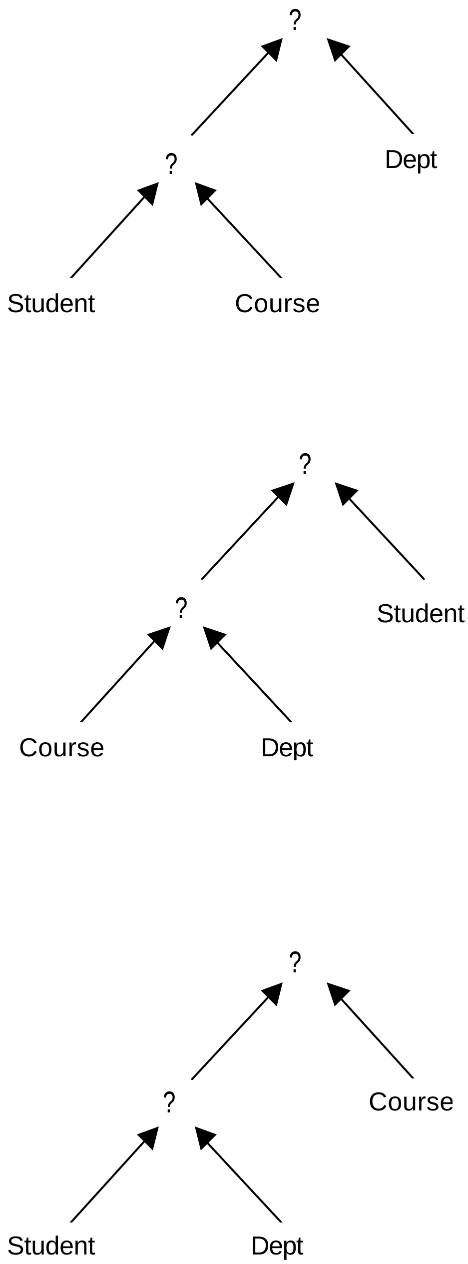

Fig 3.1 Equivalent join trees

3.1. Order affects costs

We will now show how order of a join that is commutative affects join costs. Assume *P* and *Q* are two relations that have 10 & 100 rows each that take part in the join. Limited by binding constraints, a parallel access to both the sources may not be possible. Due to such dependencies, they have to be accessed one at a time and the bindings from the first source are used to query the second. Let a and t be the access and transfer time for the sources. *Access time* refers to the setup (overhead) time incurred in using a source and *transfer time* is the rate at which the source produces data.

P ? Q: Source P is accessed and 10 rows are extracted. For each of these rows, an access must be made to Q to see if there exists a row that might participate in the join, the maximum being 100. Hence the cost can be estimated as

Cost (P ? Q): $a + 10t + 10a + 100t$

Q ? P: Source Q is accessed and 100 rows are extracted. For each of these rows, an access must be made to P to see if there exists a row that might participate in the join, the maximum being 10. Hence the cost can be estimated as

Cost (Q ? P): $a + 100t + 100a + 10t$

The cost difference between the above two alternate approaches is $90a$. It can be seen that P ? Q is a better alternative to Q ? P even though both will produce the same results. There are only 2 alternatives possible for 2 sources. The number of alternatives increases rapidly compared to the number of sources. The corresponding costs vary widely. This underlines the need and necessity for join ordering algorithms.

One of the fundamental assumptions in searching a larger space of possible join orders is that the time spent in searching would be paid off in obtaining a better solution than what would have been obtained searching a subset of the search space. In traditional databases, with mostly localized databases, access and transfer times that constitute a large part of execution cost are negligible. Hence, the difference in execution cost offered by using a more optimal tree is not clearly visible. This is further suggested by the fact that the time spent in searching a larger space might be much more than the time saved by a more optimal query. The increase in processing time eclipses any increase in optimality.

Such differences could be magnified in the Internet information-gathering scenario where access times are relatively much higher than those for traditional ones. Hence, there is a need to search a larger space and come up with an execution plan as optimal as possible. The time saved in access and transfer costs more than makes up for the time spent in searching a larger space.

Query optimization is the process of producing a query execution plan that represents an execution strategy for a given query. The plan so produced minimizes an objective cost function. A query optimizer is usually seen being comprised of three components [OV91]:

1. Search space: Set of alternative execution plans that represent the input query
2. Cost model: Predicts the cost of the given execution plan
3. Search strategy: Explores the search space and selects the best plan, using the cost model

3.2. Search space

Query execution plans are typically abstracted by means of operator trees that define the order in which operations are performed. Join trees whose operators are either join or Cartesian product characterize query optimizers. Permutations of join order have the most important effect on the performance of queries. To avoid investigating a large search space, query optimizers typically restrict the size of the search space under consideration. An important restriction is the shape of the join tree. Though considering only linear trees drastically restricts the search space, bushy trees are useful for information integration because of their inherent parallelism as mentioned in Chapter 1.

3.3. Join tree shapes

Different shapes of join order trees further increase the number of possible join orders. A tree all of whose right nodes are base relations is termed left linear, and the one with all left nodes as base relations is termed right linear. A tree that is neither of the above is termed as a bushy tree. If a particular join tree shape has a feasible possible ordering, it generates the same answers as another. However, they also have different implications with respect to access times and hence the final cost differs.

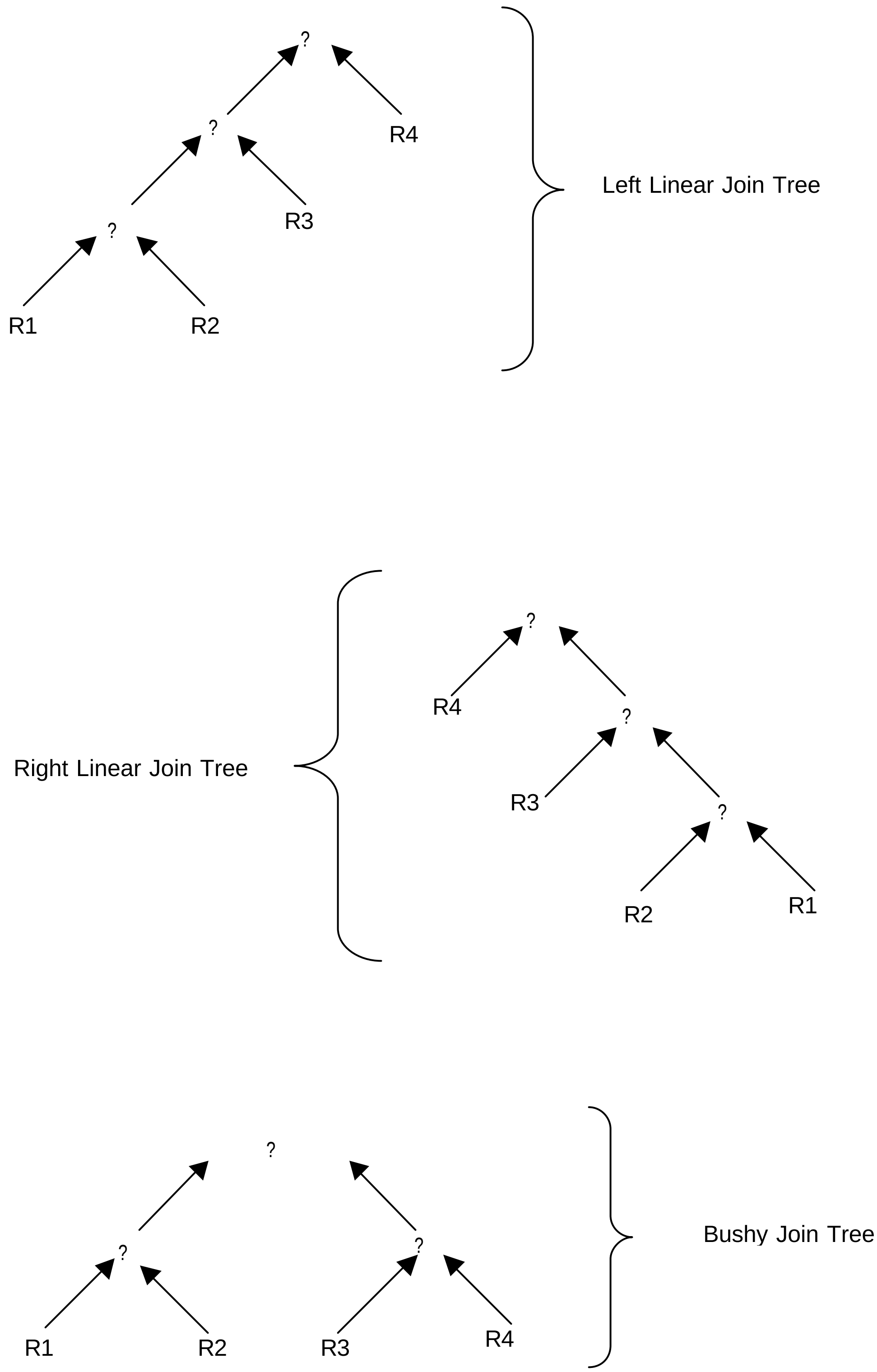

Fig 3.1 Join Tree Shapes

3.4. Size of the search space

Figure 3.1 illustrates various shapes for a join tree. For a left linear tree, there is only a single possible tree shape. Any of the *n* relations can be assigned to each of the leaf nodes in $n!$ ways. On the other hand, there are many possible shapes for a *n*-leaved bushy tree.

The number of possible bushy tree shapes is similar to the classical problem of the number of parenthesizations *P(n)* of *n* multiplications. A sequence of *n* variables can be split between the k^{th} and $(k+1)^{st}$ variables for any $k = 1, 2, \ldots n-1$ and parenthesized recursively.

$$S(n) = \sum_{k=1}^{n-1} S(k)S(n-k) \text{ for } n>1$$[3]

The above recurrence is the sequence of Catalan numbers[4]. Thus the value can be calculated as

$S(n) = C(n-1)$ where

$$C(n) = \frac{1}{n+1}\binom{2n}{n}$$

[3] $P(1) = 1$

[4] Among other things, the Catalan numbers describe the number of ways a polygon with $n+2$ sides can be cut into *n* triangles, the number of ways in which parentheses can be placed in a sequence of numbers to be multiplied, two at a time; the number of rooted, trivalent trees with $n+1$ nodes; and the number of paths of length $2n$ through an *n*-by-*n* grid that do not rise above the main diagonal.

For each of the bushy tree shapes, there are $n!$ ways in which n relations can be assigned to the leaves. The number of possible join orders for bushy trees (which includes left linear trees as well) is given by the formula:

$$P(n+1) = n!\binom{2n}{n}$$

For 8 relations, there are about 17 million join orders, of which about 40 thousand are left linear trees. The brute force method of exhaustive search is a poor strategy for finding an optimal join order. Traditional join order optimizers take a small hit in execution time of the resulting order in return for a large decrease in search space by considering only left linear trees. The degree of sub-optimality is not negligible in the Internet information-gathering scenario dominated by access costs. This is shown by the results of one of the experiments conducted and the size of the search space empirically deduced.

3.5. Motivation for using dynamic programming

The most popular search strategy used by query optimizers is dynamic programming that is a systematic search method – it proceeds by building plans starting from base relations, joining one more relation at each step until complete plans are obtained. Dynamic programming builds all possible plans, breadth first, before it chooses the *best* plan. To reduce cost, partial plans that are not likely to lead to the optimal plan are pruned at the earliest stage.

Dynamic programming is *almost* exhaustive- it searches through all the possible solutions without actually considering all the nodes, and assures that the best of all plans is found and incurs an acceptable optimization cost.

The main objective of my research is to develop and implement a join ordering algorithm based on the existing System R query optimization algorithm that is more suited for Internet information gathering. The algorithm must ensure that binding constraints are taken care of and that it searches bushy trees as well.

My research also includes implementation of a system to test the new algorithm with multiple sources and vary access and transfer times along with the distribution of data and analyze the efficacy of the algorithm with respect to other join ordering algorithms. It is expected that as number of sources increase, the difference between the algorithms will be more pronounced.

4. JOIN ORDER OPTIMIZATION

Present day join order optimization algorithms for Internet information gathering assume sources with no available statistics. This is often true for most sources present on the Internet. However, it is often the case that partial statistics may be available or can be obtained easily. While the exact selectivity indices of tables stored in internet databases may not be known, it is quite reasonable to assume for the relation

```
Books (ISBN, Title, Author, Publisher, Price, Pages)
```

binding *Author* will yield smaller number of tuples than *Publisher.*

4.1. Greedy approach

To utilize this information available to us, we developed a greedy algorithm [KG99] that divides the set of binding patterns feasible for a source into those that make the source generate high traffic- High Traffic Binding Pattern [HTBP] and those that do not. It should be noted that a source might be included in HTBP for one of its binding patterns, not necessarily all. The reason for this is that it's the binding that decides the number of tuples returned as a result, not the source by itself.

The greedy algorithm uses minimal source statistics to order source calls. It attempts to access sources with more feasible access patterns such that they do not belong to a HTBP. The idea behind this approach of dividing the set of sources into two types is that while full statistics might not be available for most internet sources, partial information is usually available and if not, can be estimated in most cases. With each selection of source, the list of available bindings increases and thus the number of feasible access patterns also increases. This ensures the termination of the algorithm. If a feasible non-HTBP is not found for an iteration, the most general binding pattern is chosen. This keeps the algorithm moving.

4.2. System R style optimizer

A System R style optimizer performs a static query optimization based on the exhaustive search of the search space. The input is a list of sources to be called along with their binding patterns resulting from the query rewritings. The output is an execution plan that implements the optimal join tree. The optimizer assigns a cost to every candidate tree and retains the one with the smallest cost. The candidate trees are obtained by permutation of the join orders of the *n* relations of the query using relational algebra rules. The set of alternative strategies is constructed *dynamically* such that only the cheapest one is kept.

The algorithm consists of two steps:

1. The best access method to each individual relation is computed.

2. For each relation *P*, the best join ordering is estimated, where *P* is first accessed using its best single relation access method.

The cheapest ordering becomes the basis for the best execution plan.

4.3. Internet System R (ISR) join ordering algorithm

Most Internet information sources do not expose statistics about themselves. However, more often than not, approximate statistics can be estimated for such sources. For example, Amazon.com has possibly more books under a single *Publisher* than for a given

Author. This implies that binding *Author* will return fewer tuples than from *Publisher*. A slightly different variation of information integration can be for corporate databases that are kept at different locations whose exact statistics are easily available. In such cases, optimization strategies that take advantage of such information will be more optimal than those that don't.

The Internet System R algorithm optimizes source calls using statistics that include the access and transfer cost for each source and the cardinality (value count) for each attribute contained in a particular source. The algorithm uses source descriptions and builds plans from atomic plans (containing a single source) and prunes non-viable and invalid plans thereby cutting down on the search space. In contrast with query optimization for traditional sources, execution is costlier in terms of time compared to the optimization process itself. So it makes more sense to spend more time in coming up with a more optimal plan than passing the cost to the execution phase. My algorithm modifies the traditional algorithm in such a way as to explore bushy trees instead of just left linear trees. While this process increases the search space, the resulting join order would be more optimal and the time spent in searching an expanded space would be paid off by the lowered execution cost.

```
INPUTS
   S [1..m]: Array of all subgoals expanded w.r.t binding patterns;
   Associated data structure along with above which will help calculate
costs;

   Initialize NODE with
      PP = nil; Bindings = {f}; Cost=0.

   IF S has a corresponding BestPlan
      return the corresponding join order
   ENDIF

   REPEAT
      FOR i = 1 TO number of feasible leaf nodes
         FOR j = 1 TO (|Q| choose i) DO
            LET LeftSubGoal = jth element in (|Q| choose i)
            LET RightSubGoal = S - LeftSubGoal
            Recursively call this algorithm with LeftSubGoal and RightSubgoal
            CurPlan = Make a new plan by joining the above resultant plans
            IF it has a lower cost than current BestPlan THEN
               update BestPlan
            ENDIF
         NEXT j
      NEXT i
   UNTIL no child nodes are generated in an entire iteration
   return join order of BestPlan
END.
```

Fig 5.1 Internet System R algorithm

Initially in the algorithm outlined in Figure 5.1, all feasible sources - those that can be called using the available initial bindings are listed as atomic plans. Thus, the first iteration makes sub-plans of size n = 1. The next iteration makes all plans for n = 2 with all possible combinations, ensuring that only feasible join ordered plans are built. Both A ? B and B ? A are considered, thus increasing the search space to include left linear as well as right linear trees. Right linear trees need to be considered in the intermediate stages of the working of the algorithm as bushy trees may contain partial right linear trees. Further, when bushy joins are possible[5], the next iteration of subplans includes those of type (*A* ? *B*) ? (*C* ? *D*) in contrast with the traditional algorithm that would consider (((*A* ? *B*) ? *C*) ? *D*). For each join, the statistics are updated and the corresponding join costs are also calculated. At the final iteration, when all the sources have been taken care of, the search tree consists of plans of size $n = m$ where m is the number of sources given as input to the algorithm for ordering. The plan with the least cost at this iteration represents the optimal join ordering for calling the sources in an information gathering strategy. A detailed pseudo-code follows.

4.4. Internet System R pseudo-code

The program in Figure 5.2 takes as input the list of all subgoals along with statistics about the corresponding access and transfer cost per tuple; cardinality of the relation corresponding to each attribute (required for calculating join sizes); binding restrictions which describe the various bindings require to make a source call. The bindings available from the query are also given as the input.

---

[5] There must be at least 4 nodes to form a bushy tree.

```
GLOBAL

   SubPlans []       : Array of all subgoals with the following associated data
   AvailBindings[]   : Array of available bindings before execution of this plan
   ReqdBindings[]    : Array of variables which require bindings
   Attrib[]          : Array of value counts V for each attribute
   Access            : Access cost
   xfer              : Transfer cost for each tuple
   cost              : Cost of transferring all the tuples
   plan              : Initialized to unary plan

   /* Array of all optimal/best plans for corresponding subgoals
   initialized to all feasible SubPlans*/

   AllBestPlans[]: Array of aggregates containing the following
      Plan: Join tree describing an optimal way of calling the given sources.
      Goals[] : Array of unordered source calls

PROCEDURE OptPlan(Q, QBindings)
   IF ExistsOptimalPlan(Q) THEN BEGIN
      Return(Plan(Q)) /* search for Q and return the corresponding plan */
   END

   initialize BestPlan with
      /* empty array of variables which require bindings */
      AvailBindings := NULL
```

```
    /* empty array of variables which don't require bindings */
    ReqdBindings := NULL
    Access := 0
    xfer := 0
    cost := 8
    Subgoals := Ø
    plan := NULL

// Try all n sized subplans for bushy joins
FOR i := 1 TO |Q|/2 DO BEGIN
    FOR j := 0 TO $\binom{|Q|}{i}$ DO BEGIN
        LeftSubgoal := Combination(Q, i , j)
        RightSubgoal := Q – LeftSubgoal

        PlanLeft := OptPlan (LeftSubgoal,QBindings)
        RightBindings := QBindings U PlanLeft.AvailBindings
        PlanRight := OptPlan (RightSubgoal, RightBindings)

        CurPlan1 := Join (PlanLeft, PlanRight)
        CurPlan2 := Join (PlanRight, PlanLeft)
        IF CurPlan1.cost < BestPlan.cost THEN BEGIN
            BestPlan := CurPlan1
        END
        IF CurPlan2.cost < BestPlan.cost THEN BEGIN
            BestPlan := CurPlan2
        END
    END
```

```
   END
   /* Reuse this for successive levels */
   AddPlan (BestPlan)

   return (BestPlan)

END

PROCEDURE Join (Left, Right)
   initialize a plan New whose cost := 8

   /* Check if feasible w.r.t. binding patterns */
   IF Left.ReqdBindings = Ø AND Right.ReqdBindings ?  Left.AvailBindings) THEN
BEGIN
      New.Plan := Left.Plan ? Right.Plan;
      New.f := Left.f U Right.f U Right.b
      New.b := Ø
      New.subgoals := Left.subgoals U Right.subgoals

      /* Estimate size of join */
      New.size := Left.size * Right.size
      FOR each join attribute ai between Left & Right DO BEGIN
         /* V (R, a) returns the value count for attribute a of relation
         R,that is, the number of distinct values relation R has in attribute
         */
         New.size := New.size / max(V(Left, ai), V(Right, ai) )
      END
```

```
      /* Access and transfer cost from mediator is 0 */
      New.access := 0
      New.transfer := 0
      /* For both Left and Right subplans
         If a subplan is unary, it will have non-zero access and xfer costs
            but cost=0
         If a subplan is non-zero, then its access and xfer costs will be
            zero but its cost will be non-zero
         In other words, in the expression below, one of the parts (on each
            line) will be non zero */

      New.cost := Left.cost + (Left.access +           New.size*Left.transfer)+
                  Right.cost+ (New.size*Right.access + New.size*Right.transfer)
   END
END

PROCEDURE Combination (Goals, n, i)
   /* Return the ith value from all n sized combinations from Q */
END

PROCEDURE IsOptimal (Goals)
   /* Return TRUE if a corresponding plan for Goals exists in AllBestPlans */
END
PROCEDURE Plan (Goals)
   /* Returns the corresponding plan for Goals from AllBestPlans[] if found,
   NULL otherwise */
END
PROCEDURE AddPlan (Plan)
   /* Add Plan to the array of AllBestPlans[] */
```

```
END
```

Fig 5.2 Internet System R pseudo code

A `SubPlan` is made for all possible binding patterns available for a particular source though some of these `SubPlans` may not be feasible. All the feasible `SubPlans` are added to the array of `AllBestPlans` that maintains a mapping of the best plan found for a set of source calls and the corresponding plan that is a join tree.

Given a set of source calls to be ordered, the function `OptPlan` first checks to verify if an optimal plan is present already. If so, it merely returns the corresponding plan from `AllBestPlans`. On the other hand, if an optimal plan is not found, it tries to split the given set of sources and check if an optimal plan exists for a subset of the sources to be ordered. While the traditional System R optimization algorithm considers all subsets of size *n*-1, the proposed algorithm considers all subsets of size *n*-1 through 1 (The loop count `1 TO |Q|/2-1` is to avoid expansions redundant due to symmetry). For each subset thus obtained, all combinations are considered and the function is called recursively for both the subsets. A join is considered for both the right and left linear trees, though in many cases both might not be applicable due to binding restrictions.

Two subplans may be joined if the left `Subplan` is executable using the bindings available currently, and the right `SubPlan` is executable using the bindings available after the execution of the left `SubPlan`. If these criteria are satisfied, joining the left and right subplans makes a plan. The free attributes of the new plan are the union of the free attributes of the left and right `SubPlans` and the bound attributes of the right `SubPlan`. The new plan so obtained is executable by itself, and hence its bound variable set if the null set. This implies that it may participate in joins with other subplans where it occupies the left side. An estimate of the join size is made using a standard heuristic and the access and transfer costs of the new subplan are calculated using a weighted average.

When no more sources to be ordered are left, the procedure returns with a `BestPlan` that contains a bushy tree as its attribute that is an optimal one.

4.5. Modifications to System R style join ordering algorithm

Splitting *n* sources into 1 and n-1 sources is a trivial task when there are no binding restrictions. For *n* sources that have some binding restrictions, the join of the partial plans obtained by the splitting them into 1 and *n*-1 is not always sound. Because of the binding restrictions, new bindings available for the right subplan generated by the left one have to be factored in along with those possibly available from the query itself.

Consider the following relations corresponding to the University information-gathering scenario described earlier.

Books (isbn^b, title, author, publisher, price, pages)

Student (id^b, firstName, lastName)

Borrow (studentId, isbn, dateIssued)

Lost (isbn)

The query to find all the books that are lost by the student is

```
SELECT *

FROM Student, Books, Lost, Borrow

WHERE Borrow.isbn=Lost.isbn AND Books.isbn=Lost.isbn AND Borrow.id=Student.id
```

We need to order the join for *Student* ? *Books* ? *Borrow* ? *Lost* such that the cost incurred is minimal. This query would make the call

```
ISR (Books, Student, Borrow, Lost)
```

One of the many ways this call can be split is `ISR (Student, Borrow, Lost)` ? `ISR (Books)`. Even though `ISR (Books)` does not receive its prerequisite binding from the query, the above call is valid because the left side provides the required bindings.

Modifications to the algorithm also include those to search a larger space of join trees additionally containing bushy and right linear trees. Right linear trees are a non-trivial reversal of join order at each stage - taking care that the required bindings for the right node are satisfied properly. Inclusion of bushy trees in the search space involves addition of subplans instead of atomic sources while the join tree is built. An inherent and further change necessitated is to identify possible areas of symmetry and prune plans that belong to the same equivalence class. Instead of adding one source to the base atomic plan, attempts are made to add combinations of partial plans of varying sizes. This is obtained by a combinatorial generator that internally keeps a combination of sources and provides the set of sources for the next iteration. The call `ISR (Student, Lost)` ? `ISR (Books,` `Borrow)` is an example of how bushy trees are negated by addiction of non-atomic partial plans.

The problem of ensuring that bindings from one side of the join is propagated to the other side is further complicated when n sources are split into *n-k* and *k* sized sets. Of utmost concern is the large number of such possibilities. For each such possible join, binding requirements are tested at the earliest possible instant to prune illegal trees as high up as possible. This helps in effectively cutting down on their child nodes as well.

The call `ISR (Student, Lost)` ? `ISR (Books, Borrow)` is not valid because the bindings required for the left side are not provided by the query and hence is pruned without further expansion.

Annotated query plans are used that contain information about sources including but not limited to access and transfer costs and selectivity indices of the attributes for each of them. Every time a join is made between two sources resulting in a partial plan, this information has to be updated so that the new pseudo source has statistics that accurately reflects a weighted average of the sources it is composed of. The access and transfer costs of the non-atomic source thus formed are 0 because there is no overhead for accessing materialized views. The size of the intermediate join is calculated using the selectivity index of the join attribute. Selectivity index of the join attribute in the newly formed partial plan is the weighted average using the sizes of the participating number of tuples, while those of other attributes remains the same assuming independent attributes and uniform distribution of values.

Consider the following two sources:

Dept (deptId, name, chair, mailCode)

Student (id, firstName, lastName, deptId, year)

Assume the following statistics for *Student* and *Dept* sources:

Statistics	***Dept***	***Student***
Access	100	10
Transfer	10	2
Size	10	50,000
V_{deptId}	10%	5%
Cost	0	0

The statistics for *Dept* ? *Student* are calculated as follows:

Statistics	**Dept ? Student**
Access	0
Transfer	0

Statistics	**Dept ? Student**
Size	$\frac{\text{Size(Student) * Size(Dept)}}{\text{max(V(Student, deptId), V(Dept, deptId)}}$ $= \frac{50000 * 10}{\max(2500,1)}$
Cost	100 + 10 * 10 + 10 * 10 + 50,000 * 2 = 100,300
V_{deptId}	$\frac{10\% * 10 + 5\% * 50{,}000}{10 + 50{,}000} = 5\%$

The modified algorithm also has to take care of binding constraints and consider a different set of atomic and compound sources as and when newer bindings become available from previous joins. This means that the pool of eligible sources is dynamic instead of being static. One of the attributes of an annotated source and the resulting annotated plan is the list of bindings that they provide. As the plan is being built, the associated bindings are updated dynamically. While partitioning the set of sources into different sets, bindings that are provided by the left node to the right have to be accounted for, and also propagated down to the lower nodes.

4.6. An illustrative example

The ISR algorithm is better illustrated with an example. To find all the books that are lost by a student, we need to order the join for *Student* ? *Books* ? *Borrow* ? *Lost*.

To join the four relations and obtain answers to the required query, the algorithm is called as follows:

```
ISR (Student, Books, Borrow, Lost)

    ISR (Student, Books, Borrow) ?  ISR (Lost)
    ISR (Student, Books, Lost) ?  ISR (Borrow)
    ISR (Student, Lost, Borrow) ?  ISR (Books)
    ISR (Books, Lost, Borrow) ?  ISR (Student)
    ISR (Lost) ?  ISR (Student, Books, Borrow)
    ISR (Borrow) ?  ISR (Student, Books, Lost)
    ~~ISR (Books) ?  ISR (Student, Lost, Borrow)~~
    ~~ISR (Student) ?  ISR (Books, Lost, Borrow)~~

    ~~ISR (Student, Books) ?  ISR (Lost Borrow)~~
    ~~ISR (Student, Lost) ?  ISR (Books, Borrow)~~
    ISR (Student, Borrow) ?  ISR (Books, Lost)
    ~~ISR (Lost, Borrow) ?  ISR (Student, Books)~~
    ~~ISR (Books, Borrow) ?  ISR (Student, Lost)~~
    7SR (Books, Lost) ?  ISR (Student, Borrow)
```

The above expansion shows possible join trees that may constitute the search space for the given set of sources. Many join orders are not possible semantically due to binding constraints either at the immediate level or the succeeding level. `ISR (Books) ?  ISR`

`(Student, Lost, Borrow)` is not feasible because of the binding requirements on the *isbn* attribute for Books. Even the ones that are sound at the first iteration level may not be feasible after further expansion. The correct partial join tree for `ISR (Books, Borrow)` ? `ISR (Student, Lost)` is not feasible at successive levels because `ISR (Student, Lost)` does not have any feasible child nodes.

Even though theoretically a large number of join orders are possible when taking bushy trees into consideration, binding constraints cut down on the number drastically. The number is even smaller due to pruning of sub optimal partial join order trees. `ISR (Student Borrow)` ? `ISR (Books Lost)` is a possible candidate for pruning compared to `ISR (Student, Lost, Borrow)` ? `ISR (Books)` because Books is accessed relatively fewer number of times in the later order and is thus likely to be less costly.

It can be seen that there is a large number of child nodes at each stage many of which are still legal but possible sub-optimal. The above partial expansion illustrates this feature. Because access costs form a large portion of execution cost, it is easy to see that one of `ISR (Student, Borrow)` ? `ISR (Books, Lost)` or `ISR (Books, Lost)` ? `ISR (Student, Borrow)` produces the optimal join order. High access cost for Books coupled with the small number of tuples in Lost (compared to Borrow) means that

`(Books ? Lost) ? (Student ? Borrow)`

is likely to be the solution returned by the algorithm.

5. IMPLEMENTATION & EVALUATION

A realistic evaluation of the ISR algorithm would involve usage of form-interfaced web databases and perform tests on them. Though such an evaluation would possibly serve as a good demonstration - not all run time conditions can be tested out. Changing the selectivities of the data might not be possible in case of external web databases and not feasible for internal databases for each run of experiments.

The algorithm has been implemented in Java 2 running on Sun Solaris 5.7 though some of the experiments were also run on a Pentium III 933 MHz PC running Windows 2000. Java was chosen as the language for implementation due to the abundant standard API methods available to prototype a system quickly. Sources were implemented as simulations whose behavior could be set while instantiating. With such sources, it is also possible to model an increasing number of sources without giving much attention to the actual semantics of the joins.

Experiments were run on the system to evaluate the performance of the algorithm. Specifically we attempted to test the following hypotheses:

1. It is reasonable to expect an exponential increase in processing time for ISR. We are more interested in the total cost of join order optimization and empirical tests can show the difference in the costs of the algorithms under scrutiny.

2. Bindings available for some of the attributes decide the number of nodes to be expanded and checked before they are possibly pruned. A set of sources with different placement of bindings was used to empirically show this relationship.

3. As has been mentioned before, not all sources have known statistics. Information gathering often involves sources about which no statistics are known and a scenario where all statistics are known is not common. Graceful degradation of the algorithm due to lack of statistics was also tested for with empirical data.

5.1. Access and transfer times

A fundamental assumption for internet information gathering is that access and transfer times are higher than those of traditional databases. Further, access time dominates transfer time for most sources. Access and transfer times of a source can be calculated though they are not directly known. Get the total time required $t1$ & $t2$ for downloading two different file sizes $s1$ & $s2$. Considering transfer of each byte as a transaction

$$a + t \,.\, s1 = t1$$

$$a + t \,.\, s2 = t2$$

Solving the above two simultaneous linear equations we can calculate the two unknown variables. Further, when we have a set of values, transfer time t is the slope of the graph and access time a is the y-intercept.

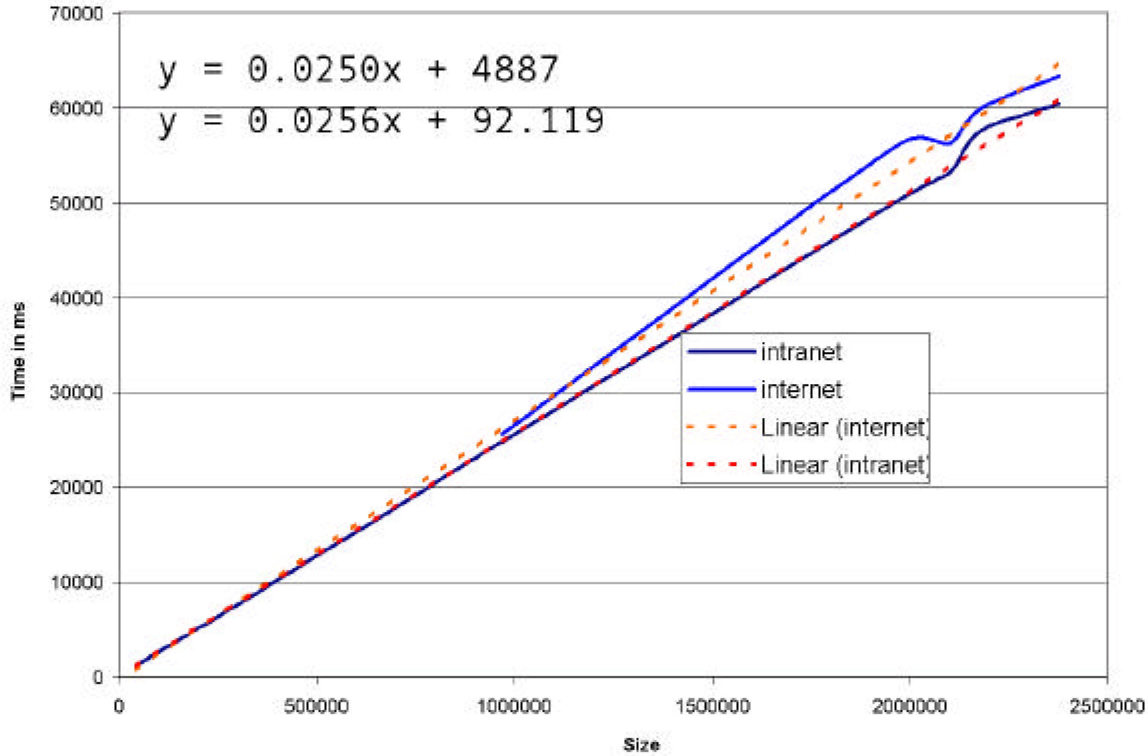

Fig 6.1 Access and transfer times[6]

A Pentium III 933MHz PC running Windows2000 was set up as a local (intranet) source and populated with about 20 files of sizes varying from 50KB[7] to 3MB[8]. The larger files were copies of files downloaded from an Internet source[9]. The Internet connection speed was a T1 line with a typical speed of 500kbps. For each file, the average time taken over 4 runs was recorded. The same was repeated for 6 files from the original source. The

[6] The small undulations towards the top-right corner for larger file sizes is possibly due to the garbage collector mechanism of the Java Virtual Machine.

[7] Files are jpeg encoded images stored at `http://tsangpo.eas.asu.edu/Photos`

[8] Files are movie files stored at `http://tsangpo.eas.asu.edu/Ads`

[9] `http://dvs1.dvlabs.com/adcritic` which is the storage server for `adcritic.com`

graph in Figure 6.1 was plotted to see the relationship between file size and corresponding time taken. Thick solid lines represent the actual readings while the dashed lines represent the trend line for the graph.

From the equation formed, we deduce that:

1. Transfer time is almost constant at 25ms/KB for both sources. This shows that in the absence of any network fluctuations in the 40 minutes that the experiment took to run to completion the transfer time remained almost constant for both the sources

2. Access time for the intranet source is significantly on the lower side at 92ms while that for the external site is about 5 seconds!

This confirms that access costs are indeed more than transfer costs and form a significant portion of the execution cost for a join.

5.2. Increase in planning cost offset by decrease in total cost

The optimizer does not always have to consider many of the large number of possible bushy join orders.

- A particular join may not be possible because the binding requirements of both the participating sources are not met.

- A partial tree might be pruned in the presence of a more optimal shape or order.

- In a realistic scenario, there are very few legitimate bushy trees from the vast original search space of all bushy join trees.

This decreases the penalty associated with searching bushy join orders as well for a more optimal join order than that produced by searching left linear trees alone.

To evaluate the belief that realistic sources would not vastly increase the search space as expected and put a big performance penalty on ISR, 10 sources were created with the ratio of access to transfer time varying from 8 to 512. Each of the sources had varying number of attributes and corresponding selectivity indices chosen from a range of 8% to 64%. Sources were added incrementally to the optimizer and the execution cost of the plan produced was calculated using the given statistics. Planning cost was recorded as a measure of nodes expanded because it is independent of network fluctuations and processor load.

Total cost for each data set was calculated by the weighted addition[10] of the planning and execution costs.

---

[10] A weight of 100 was used which is in tune with current processor and network speeds.

The graph in Fig 6.2 shows that even though the ISR algorithm takes more time in planning due to the increased search space, the execution cost of the plan produced and hence the total costs are significantly lower than those produced by traditional System R. This confirms our hypothesis that ISR has a larger search space than that of traditional System R, but not as large as theoretically possible because of pruning of sub-optimal and illegal plans at their onset. It also shows that in spite of a marginally larger search space for ISR, lower execution cost pays for the slight increase in planning cost.

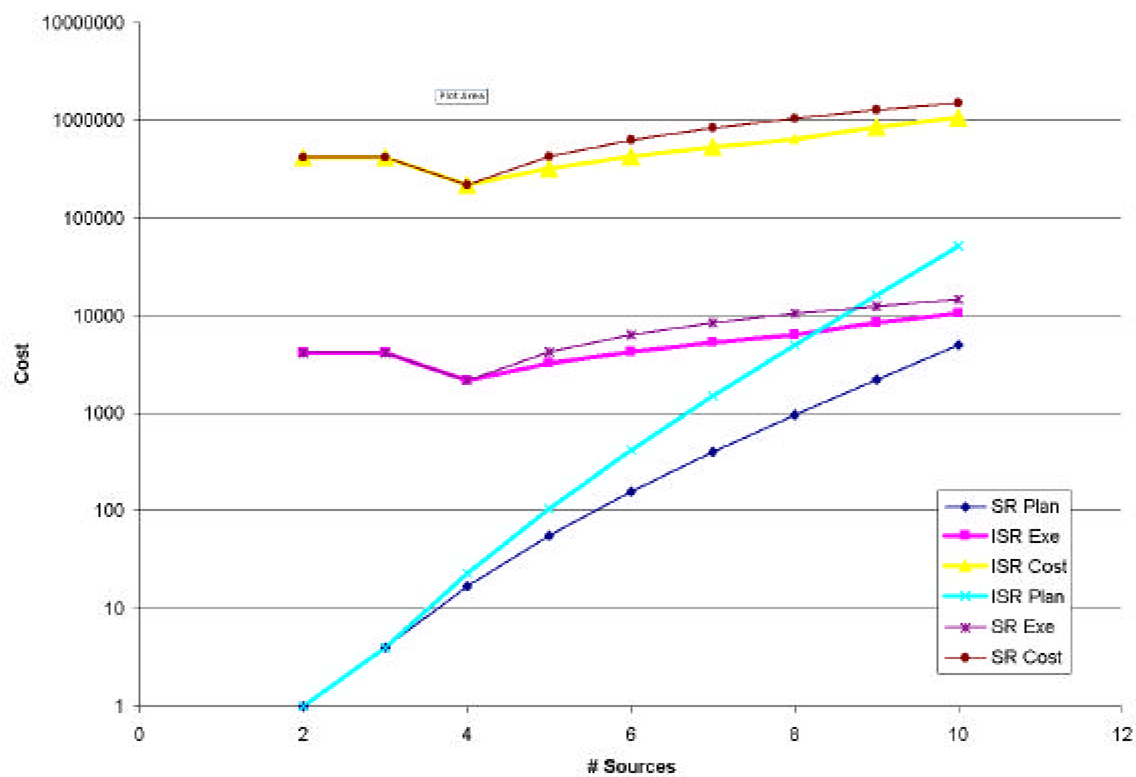

Fig 6.2 Planning costs for System R & Internet System R algorithms

5.3. Position of bound attribute matters

The search space expands as more sources are added and contracts as more binding constraints are added that deem many partial plans illegal. The size of the search space is not the same even within a set of sources for a given number of binding constraints. For a given set of sources and number of binding constraints, size of the search space depends on the interaction between the sources and their attributes. This is shown in the graph in Figure 6.3 where even though there is only one bound variable amongst all sources given to the optimizer, the number of nodes expanded varies. This verifies the hypothesis that number of nodes expanded is a non-trivial function on the number of bound variables.

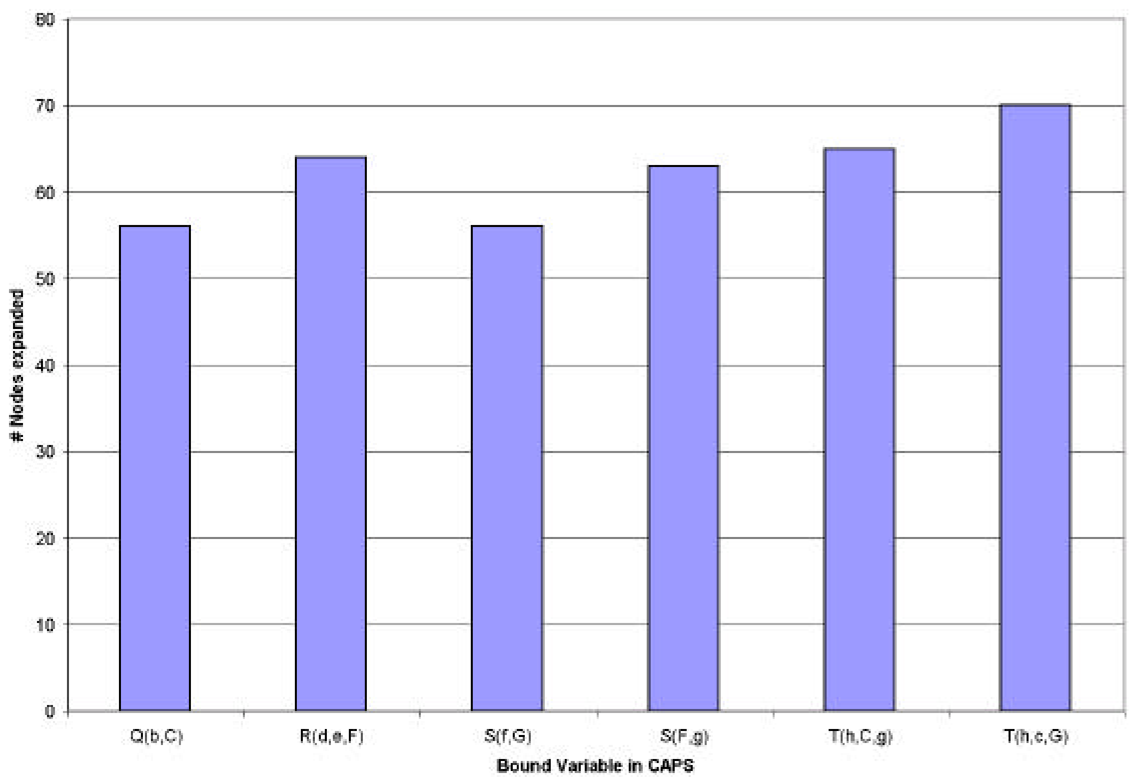

Fig 6.3 Effect of placement of a bound attribute amongst different relations

5.4. Graceful degradation

The Internet System R style join order optimization algorithm proposed in this thesis assumes that all the required statistics are either available or can be easily found using various probing techniques. However, this may not always be possible or feasible. In such scenario, the optimizer has partial statistics and cannot fully estimate the intermediate join sizes and make the correct decisions. To measure the degree of impairment caused by lack of statistics, a set of 4 through 9 sources was given as input to the optimizer. The performance of the optimizer was contrasted with that of the greedy algorithm described in section 5.1. The greedy algorithm does not use any statistics and is run only for sources with no available statistics and was taken as the base for comparison. In the next run, a set of statistics were masked out and the performance degradation was recorded. The algorithm loses out in the absence of any statistics where it degenerates into a pure brute force method of searching and has to go through all possible feasible permutations. The graph in Fig 6.4 plots the improvement of ISR over greedy algorithm as more statistics are given and shows that the optimizer degrades gracefully as less data is made available to it.

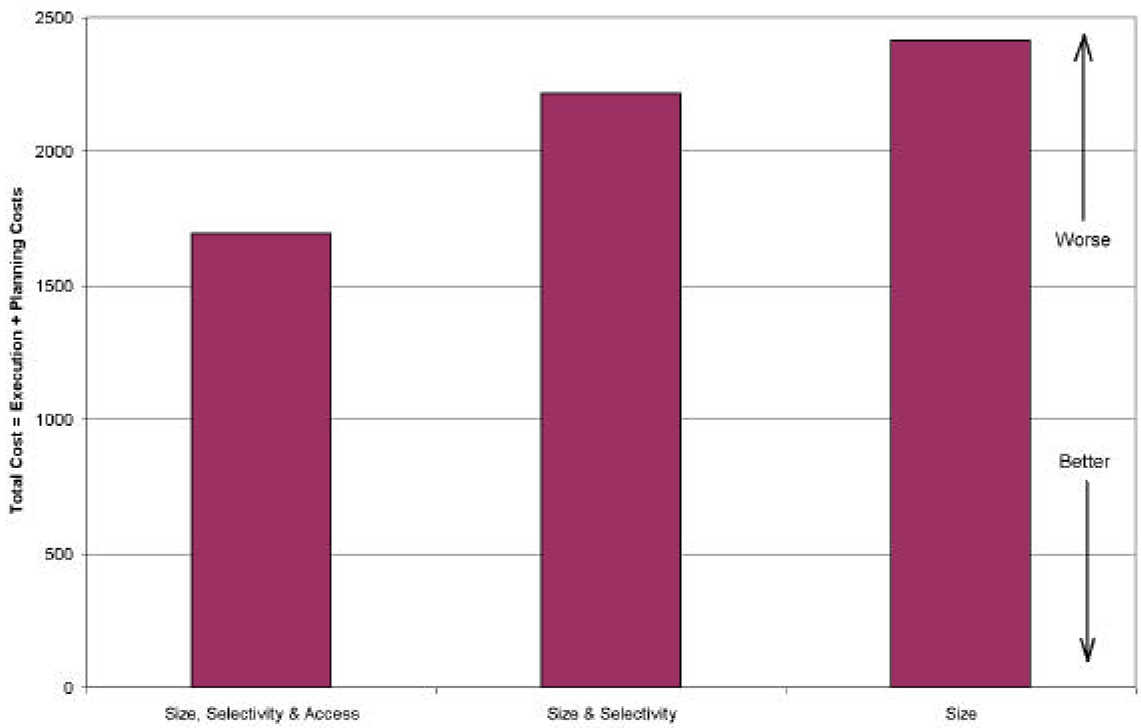

Fig 6.4 Graceful degradation of Internet System R with partial statistics

5.5. Summary

Even with a high speed T1 connection to the Internet, access costs remain an important bottleneck and need to be eliminated as much as possible for an optimal execution. While a large number of solutions are theoretically possible, the search space is not as large because of binding restrictions that eliminate many possible partial plans. Dynamic programming further eliminates partial plans that will result in sub-optimal plans before they can generate full plans. Both these methods only reduce the difference between the planning cost of traditional System R and ISR algorithms – ISR still remains computationally more expensive. The picture changes when execution cost is taken into account. High access costs compared to fast processors result in an overall lower cost and more optimal solution for Internet System R style join optimizer. Planning time is a non-trivial function of the number as well as position of bound attributes. Finally we show that it is not necessary for the new algorithm to have the full array of statistics. Even with partial statistics, it maintains its performance improvement over previous methods.

6. RELATED WORK

The traditional method of ordering sub-goals is to use the "bound-is-easier" assumption that states that sources with more number of corresponding bound attributes tend to return fewer tuples. While such a heuristic is acceptable in the absence of any information about source statistics, it may lead to sub-optimal plans in some cases. This is so because the selectivity of each attribute is not uniform and the number of tuples returned is dependent on this information. For example, a student relation will return fewer tuples if *Major* is bound rather than *Year*. In mediated schemas where each tuple obtained from the first relation is used to query the next source, the number of accesses can increase tremendously if the source returning higher number of tuples is accessed first. In this example, a source that takes a binding on *Year* will produce more tuples than one that binds *Major*.

Florescu, Levy et al in [FLMS99] propose an algorithm similar to the System R style optimizer but there is no explanation of the how the cost metric is arrived at- though it provides a better treatment for the analysis of the search space. As each sub-plan is added to the bag of optimal subplans, a check is made to verify if there exists a plan already - a selection over which would yield the new subplan being added.

The size of the search space is the number of complete query execution plans. This size is more if it includes partial plans as well that may not lead to a complete plan. The bottom up approach used by [FLMS99] considers partial plans as well and has a larger search space. In contrast, my algorithm proceeds top down and partial plans that do not lead to a complete query execution plan are never considered.

To improve response time of the algorithm in [FLMS99], a redundant best-first plan is generated before the System R algorithm runs to completion. This contradicts the verified hypothesis that planning time is not as high as expected due to pruning of illegal and sub-optimal partial plans.

Another set of strategies to handle unpredictable statistics is to push the optimization techniques to the actual execution stage. Some optimizers generate a seemingly optimal plan and use feedback to further modify it with respect to run time behavior. At the other extreme, some optimizers generate a plan that may not necessarily be optimal and perform all the optimization at run-time. The mid-query optimization algorithm by Kabra and DeWitt in [KD98] emphasizes that collection of statistics is a big overhead and must not be done frequently. They identify stages of the execution where statistics should be collected and also use it for dynamic resource allocation. Urhan, Franklin et al. in [UFA98] concentrate on initial delays in their algorithm for cost based query scrambling making the assumption that access costs are much higher than transfer costs. It does not take into account the possible change in source transfer times or selectivities of the resulting data. By making the assumption that the run time environment is almost unpredictable, Avnur and Hellerstein in [AH00] propose a continuous query optimization algorithm that groups sources into eddies (similar to fragments as mentioned by Levy in [Lev99]) and the reordering takes place within those.

7. CONCLUSION AND FUTURE WORK

System R is a popular algorithm for optimization for traditional databases. It falls short for the newer Internet based databases and pseudo databases. The Internet System R join order optimization algorithm presented in this thesis overcomes the shortcomings of the original System R style optimizer so that it can be made applicable to Internet information gathering. Binding patterns pose a problem to evaluating partial join trees as the binding requirements have to be met before calculating a join and estimated when dividing the set. The optimizer has more statistics available to it and it uses them to the best advantage while calculating and estimating intermediate relations and partial joins. Because the statistics so garnered are merely used to order the sources and arrange them in the join tree, it is not prone to slight changes in source statistics. When there is a multitude of sources with varying levels of available statistics, the optimizer degrades gracefully as less data is made available to it.

The algorithm presented addresses one of the open issues in query optimization for internet information gathering. While the algorithm is resilient to small changes to source statistics, the plan produced will be substantially sub-optimal if there are large changes in the source behavior. Some sources may be slower on a particular day and have higher access times than normal. Changes to my algorithm with run time adaptivity built in would produce optimal solutions and be less prone to erratic source behavior. It may also happen that the unavailability of a source be discovered at run time. An approach that combines query planning and selection of sources along with execution optimization can solve this problem by producing alternate solutions at run time.

The presented algorithm penalizes all sources if some of the sources have less available statistics by ignoring any information that is not available for all sources. In the case when varying levels of statistics are available, it is not a trivial task to estimate the unavailable statistics of the remaining sources. Assigning average values for unavailable statistics may not be a good heuristic when very few sources have available statistics. Assigning best or worst values unnecessarily penalizes some of the sources. Such an approach can produce less optimal solution than that possible by using the available data to the best possible extent. More involved heuristics that can take into account the uncommon information available will produce more optimal results.

BIBLIOGRAPHY

[ABCEGGKL MMPTWW76] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade and V. Watson. System R: relational approach to database management. In *ACM Transactions on Database Systems Volume 1, No. 2 June 1976.*

[AH00] R Avnur, J M Hellerstein. Continuous Query Optimization. *Proceedings of ACM SIGMOD-2000 International Conference on Management of Data.*

[AHKMNPPRV WW91] P Aitken, A Himes, C Kinata, W G Madison, R Nelson, W Parker, C Petzold, P Rose, M Vose, B Webster, J Woodcock. Computer Dictionary. *Microsoft Press 1991.*

[CKPS95] S Chaudhuri, R Krishnamurthy, S Potamianos and K Shim. Optimizing Queries with Materialized Views *Proceedings of the 11th International Conference on Data Engineering 1995.*

[FLMS99] D Florescu, A Levy, I Manolescu and D Suciu. Query Optimization in the Presence of Limited Access Patterns. In *Proceedings of ACM SIGMOD-1999 International Conference on Management of Data.*

[IFFLW99] Z G Ives, D Florescu, M Friedman, A Levy and D S Weld. An Adaptive Query Execution System for Data Integration. In *Proceedings of ACM SIGMOD-1999 International Conference on Management of Data*.

[IK84] T Ibaraki, T Kameda**.** On the optimal nesting order for computing *N*-relational joins. In *ACM Transactions on Database Systems, Vol. 9, No.3, September 1984.*

[KD98] N Kabra, D J DeWitt. Efficient Mid-query Re-optimization of Sub-Optimal Query Execution Plans. In *Proceedings of ACM SIGMOD-1998 International Conference on Management of Data.*

[KG99] S Kambhampati and S Gnanaprakasam. Optimizing source-call ordering in information gathering plans. *Proceedings of the IJCAI-99 Workshop on Intelligent Information Integration*.

[Lev99] A Levy Answering queries using views: *a survey submitted for publication 1999*.

[Lev99a] A Levy. Combining Artificial Intelligence and Databases for Data Integration *To appear in a special issue of LNAI: Artificial Intelligence Today: Recent Trends and Developments 1999.*

[LKG99] E Lambrecht, S Kambhampati and S Gnanaprakasam Optimizing Recursive Information Gathering Plans. In *Proceedings of the IJCAI-99*.

[LMSS95] A Y Levy, A O Mendelzon, Y Sagiv, D Srivastava Answering Queries Using Views. *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Jose, CA 1995.*

[OV91] T M Ozsu & P Valduriez. Principles of Distributed Database Systems. *Prentice Hall New Jersey 1991.*

[UFA98] T Urhan, M J Franklin, L Amsaleg. Cost-based Query Scrambling for Initial Delays. In *Proceedings of ACM SIGMOD-1998 International Conference on Management of Data.*

[Ull89] J D Ullman. Principles of Database and Knowledge Base Systems. *Computer Science Press.*

[YLMU99] R Yerneni, C Li, H Garcia-Molina, J Ullman. Computing Capabilities of Mediators. In *Proceedings of ACM SIGMOD-1999 International Conference on Management of Data.*