

DISTRIBUTED QUERY OPTIMIZATION

*A dissertation submitted to the Jawaharlal Nehru University
in partial fulfillment of the requirements
for the award of the degree of*

**MASTER OF TECHNOLOGY
IN
COMPUTER SCIENCE AND TECHNOLOGY**

**BY
ANIL KUMAR GIRI**



**SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067**

JULY 2010

Dedicated

to

My Beloved Daughter

Aney




**SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067**

DECLARATION

This is to certify that the dissertation entitled “**Distributed Query Optimization**” is being submitted to the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Computer Science & Technology**, is a record of bonafide work carried out by me under the supervision of **Dr. T.V. Vijay Kumar**.

The matter embodied in the dissertation has not been submitted in part or full to any University or Institution for the award of any degree or diploma.

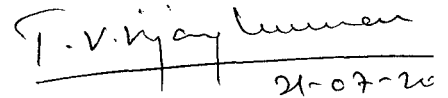

21-07-2010.
Anil Kumar Giri
(Student)



**SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110067**

CERTIFICATE

This is to certify that this dissertation entitled “**Distributed Query Optimization**” submitted by **Mr. Anil Kumar Giri**, to the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, for the award of degree of **Master of Technology in Computer Science & Technology**, is a research work carried out by him under the supervision of **Dr. T. V. Vijay Kumar**.



**Dr. T. V. Vijay Kumar
(Supervisor)**



**Prof. S. Minz
(Dean)**

ACKNOWLEDGEMENT

This is greatest opportunity for me to express my heartiest appreciation to **Dr. T.V. Vijay Kumar** for his valuable and constructive guidance and suggestion during the dissertation work. Sir has helped in every aspect of dissertation with his constructive suggestion. It has been great opportunity for me to learn not only to do dissertation but how to pursue the life. In coming life I will keep myself in touch of sir so that I can learn a lot more.

I would like heartily thanks to my **mother** who has always supported me with her great encouraging word. I would like to thanks my late **father** who has lighten the candle in my heart how to dream and never give up toward any circumstances.

I would like to thanks my senior researcher **Mohammad Haider sir**, **Santosh Gupta sir** and **Ajay Kumar Verma sir**, who's presence in lab encouraged me to complete the dissertation successfully. I would like to thanks my dear friends **Rajesh kumar** and **Kumar Dilip** who has helped me to complete the dissertation and how to keep patience.

I would like to thanks my dear **brother Abhinav Goyal** and dear **Sister Shilpi** who has helped me in my dissertation.


21-07-2010.
Anil Kumar Giri

Table of Contents

Chapter 1: Introduction.....	1-11
1.1 Introduction.....	1
1.2 Distributed Query Processing.....	2
1.3 Distributed Query Optimization.....	4
1.3.1 Search Space.....	5
1.3.2 Search Strategy.....	6
1.4 Aim of the Dissertation.....	9
1.5 Organization of Dissertation.....	11
Chapter 2: Distributed Query Optimization.....	12-44
2.1 Cost Function.....	12
2.2 Iterative Improvement.....	16
2.2.1 An Example.....	18
2.2.1.1 Optimal Query Plan Generation with Query Plan- Constraint.....	18
2.2.1.2 Optimal Query Plan Generation with time Constraint..	21
2.3 Simulated Annealing	24
2.3.1 An Example.....	27
2.3.1.1 Optimal Query Plan Generation with Query Plan- Constraint.....	28
2.3.1.2 Query Plan Generation with time Constraint.....	34
2.4 Experimental Results.....	42
Chapter 3: Conclusion.....	45-46
References.....	47-50

CHAPTER 1

Introduction

1.1 Introduction

In centralized database, data is stored and maintained at a particular node. One of the key problems with the centralized database is the central point failure. This leads to poor reliability of the system. This along with other problems lead to distributed database systems. There are several other reasons why distributed database were developed [CP84]. The first one is organizational and economic reasons, as most of the organizations have a decentralized structure and they may wish to share their data. This can be suitably addressed by designing a distributed database system. Further, a need may arise to integrate existing databases, which may be autonomous and heterogeneous in nature. This problem is addressed by constructing a global schema as in case of federated or multidatabase systems [SL90]. Another reason is that the distributed database may support smooth incremental growth for database with minimal impact on the existing system. Also, the distributed database may result in reduction of the communication overhead by replicating data on different sites. Another important advantage of distributed database

over centralized database is high reliability and availability thereby resulting in system being in operation despite of failure of any site in the network. In other words, it overcome the problem of central point failure.

Distributed database can be defined as a collection of multiple, logically interrelated database distributed over a computer network [VSV10] [CP84] [OV91]. These logically interrelated databases are managed by Distributed Database Management System (DDBMS) [VSV10] [OV91]. A distributed database management system supports the creation and maintenance of distributed database. The DDBMS is of two type namely homogeneous and heterogeneous. The homogeneous DDBMS usually refers to a DDBMS with same DBMS at each site, even if the computers and the operating systems are not the same[CP84][L81]. A heterogeneous DDBMS may have many different types of DBMS being used across sites[CP84][L81].

One of the key issues in distributed databases is the processing of queries posed on it. The queries posed on a distributed database may require processing at different sites of a network. This referred to as distributed query processing is discussed next.

1.2 Distributed Query Processing

A query posed on a distributed database is decomposed and rewritten in a manner so that it can be processed against the component databases. The aim is that the query response time is minimum. To reduce this time and other query processing overheads, a query processing strategy exist that describes the way in which the query is processed. Several technique exist that can be used to process a query, most of them are based on shipping the data from one site to another or itself. To join operation is usually used to process such queries.

The query processing architecture consists of components like parser, query rewrite, query optimization, code generation, query execution engine and supporting components such as

catalog Meta data and Base data [K00]. The query processor takes as input the user query in SQL (or OQL etc.) and translates and optimizes the query to generate a executable query plan. The parser take the query and translate it into internal representation (e.g., a query graph [JW+90][PH+92][K00]. This parse can also be used for centralized database and distributed database. The Query rewrite transforms a query in order to carry out optimization. These transformations may include elimination of redundant predicates, simplifying expressions and the nested queries. One of the key goals of the query rewrite is to make query as much declarative as possible [K00; PH+92], so that a poorly expressed query, though declarative, may force a typical plan optimizer to choose a sub-optimal executions plans. The major goal of query rewrite is therefore the transformation of procedural queries into equivalent and declarative queries. The query optimization phase is concerned with deciding which indices to be used to execute a query and which methods to be used to execute the operation. These operations may be joins, group by etc. If join operation is used then the order in which it should be performed so that result can be retrieved in the shortest time. By query optimizer it is also decided that how much memory is required to execute the query. Query optimizer decides the site in which the operation is to be executed. For this, query optimizer enumerates alternative plan so that best plan can be chosen. This best plan is chosen on the basis of some cost estimation model. Mostly all commercial query optimizer uses dynamic programming to enumerate plans efficiently[K00]. The fourth phase is plan that defines the way in which to execute a query. It is seen that almost all the database systems represent a plan in the form of a tree. The nodes of the plan are operators, and every operator carries out one particular operation like join, group by etc. The nodes of a plan are annotated, indicating where the operator is to be carried out. In the tree, it is decided the site in which the join operation is performed[K00]. The code generation phase is concerned with transforming the plan into

executable plan. In System R, for instance, this transformation involves the generation of an assembler-like code to evaluate expression and predicates efficiently [K00][LW79]. The Query Execution Engine provides generic implementations for every operator and are usually based on an iterator model [K00][G93] where operators are implemented as iterators and all iterators have the same interface. As a result, any two iterators can be plugged together and thus any plan can be executed. The catalog is used to store all the information regarding parser, query rewrite and query optimizer. It stores the schema of the database like definition of tables, views, user defined types, function, integrity constraints, partition schema etc. Catalog stores the physical information, which is the location of copies of partition of tables, and the information about indices and statistics which helps to calculate the cost of a plan. In distributed database catalog is replicated and stored at different sites so that the communication cost of data transfer from one site to another site is reduced. This can be dealt by distributed query optimization.

1.3 Distributed Query Optimization

In distributed query optimization, an optimizer has basically three components namely the search space, search strategy and the cost model [AAO05]. The search space is defined as the number of alternative plans for a user query. These plans are equivalent with respect to producing the same results. They differ on the execution order of the operation and the way they are implemented. The search strategy is concerned with exploring the search space to find the best execution plan. The cost model is used to predict the cost of each plan. The cost model consist different type of elements [VSV10][OV91] like the secondary storage cost, memory storage cost, computation cost, communication cost[VSV10] [AAO05] [EN00].

1.3.1 Search Space

As we know that the search space is the set of all possible query execution plans. These query execution plans produce the same result. The number of ways in which the query plans can be generated is an NP hard problem because of the possibility of large number of ordering of joins [AAO05] [IK84]. The solution of the plan is described by a query tree for executing the join expression and every point of search has cost associated with it. The cost function maps the query tree to their respective costs. Tree can be a binary tree in which base relations form the leaves and joins operations works as inner nodes of a tree. The edge of tree reflects the flow of data in the tree. The join is commutative and associative in nature. As the number of relations increases, the number of query plan also increases in an exponential order [SMK97]. To search for a best query plans amongst every query execution plan is a time consuming task as number of relations required for processing a distributed query may be high. This in turn may lead higher cost. This can be addressed by generating plans that minimizes the cost [AAO05].

There are different type of search space are like left deep and right deep tree [AAO05]. In Left deep tree, the tree is consist of all queries where the inner relation is a base relation. There are $n!$ Ways to allocate n base relations to the tree leaves [AAO05]. Left deep tree provides much smaller space in which it can find the optimal solution [SA+79]. On the other hand, the right deep tree is concerned with a kind of tree in which the inner relation is base relation and it is right oriented. It is like left deep tree but in right orientation. Hash join being asymmetric is used to distinguish between left deep and right deep tree. It also known that hash based join is most efficient for equi-join [DG92]. Zigzag tree [ZZB93] is generally used in distributed and parallel databases. This tree is intermediate format between left deep tree and right deep tree. Zigzag tree gives better performance when there

is limited memory space and temporary relations are not staged to disk[ZB93]. Bushy tree is also a kind of search space in which search space allows join nodes where both operands are composite. There is no base relation involved in it and therefore the solution in search space are not restricted. So bushy tree includes left deep as well as other special tree shapes as subsets without having restriction of left or right shape [AAO05] thereby resulting in the number of solution being higher than the cardinality of the left deep space. Bushy tree is more appropriate for exploiting independent parallelism [AAO05]. It is also suitable for parallel machine if the relations are partitioned on disjoint homes where home is defined as a set of nodes stored at particular place [CYW96][KG99][SMK97].

1.3.2 Search Strategy

Search strategy is defines an enumeration strategy using an enumeration algorithm [AAO05]. It aims to find the plans in an efficient and cost effective manner. Dynamic programming forms the base in most of the search strategy. There are two approaches to solve the search strategy problem. First one is deterministic approach, which builds plans on base relations and join one or more relation at each step till complete plan is obtained. The partial plans that may not lead to optimal plans is pruned. This result in reduction in the optimization cost. The plan which is economical is retained till the complete construction of the plan.

The next approach which is known as randomized strategy. In randomized strategy, focus is on finding the optimal solution around some particular point [IW87] [SG88]. Though the randomized strategy does not guarantee optimal solution, it does not cost much for optimizing the plan in terms of memory and time consumption [LV+94]. The randomized strategy starts with some randomly selected plan and then tries to find the neighbor plan and compare the cost of plan with neighbor plan. If the neighbor plans cost

is less than the starting plan then the neighbor plan is taken as a solution plan and again it starts with taking the neighbor plan. This goes on till it finds a plan which has no neighbour having less cost, for a pre-defined number of neighbours. The most important advantage of random strategy is that it has constant space overhead [KK00].

Several deterministic optimizations algorithm and techniques exist of which dynamic programming is one of the key techniques. Dynamic programming works very well if all queries are in standard query language, moderately complex [KK00]. Dynamic programming, however, does not work well for complex query plans. Dynamic programming works in bottom-up way [AAO05] by first generating the access plan for every table involved in the query. The access plan may consist of one or two operators resulting in different access plans for a table. Next, it considers all possible ways to join tables. These may be two way join plans by using the access plans of the tables as building blocks and calling the join plan function to build a join plan. This may further lead to three way join plans. Next, it generates the four way join plan by considering all combination of two way join plans and all combination of three way join plans with an access plan. In this way dynamic programming continues to produce five ways, six ways join plan and so on up to n-way join plans. Lastly, the n-way join plans are finalized by the finalize plan function in order to arrive at complete plans for the query [KK00]. The dynamic programming discards or prunes inferior building blocks after every step using prune plane function. Dynamic programming enumerate the two way join plan by considering only those plans whose cost is economical to other alternative plans. For example, suppose there are two join plan $A \triangleright \triangleleft B$ and $B \triangleright \triangleleft A$. It will take either $A \triangleright \triangleleft B$ or $B \triangleright \triangleleft A$ whosever cost is cheaper. The plan with lesser cost is retained and considered as building block for three way plan, four way and so on up to n way join plan. Now on the basis of these cheaper cost join, dynamic programming would produce the three way join like

$(C \bowtie B) \bowtie A$ and $(B \bowtie A) \bowtie C$. Now again it looks for a plan with cheaper cost and in this way it goes on and finally finds the access plan with minimum cost. One important property of dynamic programming algorithm is that the query optimizer can be extended easily. Since in distributed database, table is replicated on different sites, extend the dynamic programming requires a little bit change like the access plan function must generate different access plans for every site at which the table is replicated [KK00] [CS94] [CS96]. The join plan function must generate different join plans specifying that a join can be carried out at the site at which the outer table is produced, at the site at which the inner table is produced, and at all other interesting and participating sites. Here interesting site is defined as the site containing the query results. In this way dynamic programming can be used in distributed query optimization. The time complexity of the dynamic programming in distributed database system is $O(s^3 \cdot 3^n)$ and space complexity of dynamic programming in distributed database is $O(s \cdot 2^n + s^3)$ [KK00].

Another query optimization technique is greedy algorithm [P74] [S89] [SYT93]. It has three phases. Greedy algorithm also works in the bottom up way. This algorithm has the same access plan, join plan and finalize plan function in order to generate plans. Next, the greedy algorithm carries out a very simple and rigorous selection of the join order by using a plan evaluation function to select the next best join. The quality of plans produced by the greedy algorithm strongly depends on the plan evaluation function. Though this algorithm runs much faster than dynamic programming, it may produce worse plans [SMK97]. The time complexity of the greedy algorithms is $O(n^3)$ in a centralized system and $O(n^3 \cdot s^3)$ in a distributed database system. The space complexity of the greedy algorithm is $O(n)$ in a centralized system and $O(n \cdot s)$ in a distributed system [KK00].

The early work in the field of iterative dynamic programming [SY98] where dynamic programming is applied iteratively [KK00] [AAO05]. It means dynamic programming is

applied many times in optimizing the query so that solution would be optimal. It is the combination of dynamic programming and greedy algorithm. The advantage of iterative dynamic programming is that it produces the best plans when dynamic programming is not viable due to its high complexity. Secondly variants of iterative dynamic programming are adaptive and produce where dynamic programming turns out to be not viable. Third, all iterative dynamic programming algorithm can be very easily integrated into an existing optimizer based on dynamic programming. The motive behind iterative dynamic programming is to find the optimal solution by combining the dynamic programming and greedy algorithm.

The randomized strategy is concerned with random move in the search space [AAO05]. These moves are defined by certain rules. This move defines the edges between two solutions. For example, during random move one query plan is selected and the cost of selected query plan is computed by some cost function and then again random move is done to select the next query plan and the cost of both query plans is compared if the cost of second query is less then the second query plan is considered as optimal and again random move is performed. This process goes on till some stopping criteria are fulfilled or if there is no applicable move exists. Actually the second query plan selected is considered as a neighbor of the first one. The neighbor can be selected using the uphill move or downhill move depending on the cost function. If the cost of neighbor is lower, the neighbor is selected as per the downhill move [IK90]. If the cost of neighbor is higher, the neighbor is selected as per the uphill move [IK90].

1.4 Aim of the Dissertation

In distributed database, the data is replicated on different sites and available for access for user queries. The user query posed on distributed database is processed at different sites.

The major query processing cost parameters are CPU, I/O and communication cost [KYY82][OV91] of which communication cost is key to query processing. The communication cost is concerned with transferring the data from one site to another site [HY79]. The transmission of data alongwith the local query processing constitutes the distribution query processing strategy. This strategy is also referred to as distributed query processing [K00] [YC84]. In distributed environment, most of the queries on the distributed relational database require access to relations from multiple sites for their processing. It is possible that a relation may be present on different sites. This may lead possibility of multiple query plans for a query. If the number of sites participating in answering the query increases than number of query plans also increases exponentially [IK90]. So the query optimizer need to explore the search space to determine the best query plans amongst all possible query plans. Further, the replication of data on different sites may further lead to increase in the possible query plans. This leads to large search space of query plans from which the best query plan is required to be generated. Optimal query plan selection is a combinatorial optimization problem [JK84] and is not feasible [IK90]. The dissertation aims to generate optimal query plans for a user query using iterative improvement and simulated annealing. The query plan generation discussed in this dissertation uses the query processing cost heuristic defined in [VSV10] where query plans containing information in lesser number of sites are preferred over other query plans. Among the query plans involving same number of sites, the query plan having higher concentration of information is preferred over the other. The iterative improvement and simulated annealing algorithm is adapted to this query plan generation problem and used to generate optimal query plans.

1.5 Organization of the Dissertation

The dissertation is organized as follows. The generation of optimal query processing plan using iterative improvement and simulated annealing is discussed in Chapter 2. Chapter 3 is the conclusion.

CHAPTER 2

Distributed Query Optimization

The iterative improvement and simulated annealing are based on randomized strategy. These techniques have been used to provide solution to a distributed optimization problem. The distributed optimization problem dealt in this dissertation is concerned with generating optimal query processing plans for a given query. The heuristic defined in [VSV10] has been used based on which optimal query plans are generated. In [VSV10], the query plan generation problem has been addressed using genetic algorithms. In this dissertation, iterative improvement and simulated annealing use the same heuristic to generate the optimal query plans for a given user query. The cost function based on the heuristic, defined in [VSV10], is discussed next.

2.1 Cost function

As we know that data is distributed across the sites in the distributed form. When the user poses query in SQL then the data required to process the query need may reside at disparate sites. The efficiency of query processing depends on how close the required data

resides on the distributed sites [VSV10]. As it is known that query processing will be more efficient when the required data is very close to each other i.e. it resides at the same site. On the other hand, if the data required to answer user query resides in disparate sites, then the query processing efficiency would be low as it would require joining data from each site to generate the result. The closeness is defined in terms of number of sites participating to answer the query. If the number of sites are less than query processing will be more efficient and if the number of sites participating is more than the query processing will be less efficient. To understand the closeness, which determines the efficiency of query, let us consider the relation and site matrix shown in Fig. 2.1.

Relation	Sites			
R1	1	6	7	9
R2	1	6	8	2
R3	1	5	8	4
R4	1	3	4	2

Fig. 2.1

From Fig. 2.1, it can be observed that relation R1 is on site 1, 6, 7 and 9. The relation R2 is on the sites 1, 6, 8, and 2. The relation R3 is on the sites 1, 5, 8 and 4. The relation R4 is on the sites 1, 3, 4 and 2.

Suppose the user query UQ in SQL is as given below

```

SELECT      R1.B, R2.C
FROM        R1, R2, R3, R4
WHERE       R1.A= R3.A and R2.D= R4.D

```

In the above query, it can be seen that there are four relations participating in the FROM clause. Suppose the query optimizer is required to generate a query plan that contains these relations are at the same time optimal. It can be seen that the relation R1 is on sites 1, 6, 7 and 9, relation R2 is on the sites 1, 6, 8, and 2, relation R3 is on the sites 1, 5, 8 and 4 and relation R4 is on the sites 1, 3, 4 and 2. The aim is to generate query plans that minimize the cost of query processing.

For a query, a query plan denotes the sites for the relations in the SQL query. For example, the query plan given below implies that for the user query given above, R1 is at site 6, R2 is at site 8, R3 is at site 4 and R4 is at site 2. The query plan is considered valid if the sites given in the plan contain the corresponding relations. The query plan given below is a valid query plan for UQ.

6	8	4	2
---	---	---	---

It can be seen from Fig. 2.1 that relation R1 is in four sites and relation R2 is in four site, relation R3 is in four site in the same way R4 is in four site. There are in all $4*4*4*4=256$ valid query plans for UQ. If the number of sites containing the relations, which are in the From clause of SQL, increases than the number of valid plans would also increase. The aim is to generate optimal query plans among all possible query plans for UQ. Let us consider the valid query plans given in Fig. 2.2.

	Query Plans			
QP1	1	1	1	1
QP2	1	1	1	3
QP3	1	1	4	4
QP4	7	8	1	1
QP5	9	2	5	3

Fig. 2.2

QP1 have all the four relations in the same site i.e. 1. This is the most optimal query plan as all the relations are in the same site. In QP2, the first three relations are in site 1 and relation 4 is site 3. It means there are two sites participating. Also in QP3, there are two sites participating where relation R1 and relation R2 are on site 1 and relation R3 and relation R4 is on site 4. So in QP2 and QP3 two sites are participating now the one which has higher concentration of relation among them would be considered optimal. QP2 will be considered as optimal because the concentration of relation on a site is more as

compared that of QP3, as in case of VQ2 site 1 is having three relation and whereas in VQ3 site 1 is having only two relation. So the concentration of VQ2 is more than VQ3.

This heuristic is defined by a cost function in [VSV10]. The cost function is given below

$$QPC = \sum_{i=1}^M \frac{S_i}{N} \left(1 - \frac{S_i}{N}\right).$$

Where QPC is the Query Processing Cost

M is the number of sites accessed by the query plan

S_i is the number of times i^{th} site is used in the query plan

N is the number of relation accessed by the query.

The query processing cost is computed for the query plans given in Fig. 2.2. These are given in Fig. 2.3.

	Query Plans	QPC				
QP1	<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	$4/4(1-4/4)=0$
1	1	1	1			
QP2	<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>3</td></tr></table>	1	1	1	3	$3/4(1-3/4)+1/4(1-1/4)=6/16$
1	1	1	3			
QP3	<table border="1"><tr><td>1</td><td>1</td><td>4</td><td>4</td></tr></table>	1	1	4	4	$2/4(1-2/4)+2/4(1-2/4)=8/16$
1	1	4	4			
QP4	<table border="1"><tr><td>7</td><td>8</td><td>1</td><td>1</td></tr></table>	7	8	1	1	$2/4(1-2/4)+1/4(1-1/4)+1/4(1-1/4)=10/16$
7	8	1	1			
QP5	<table border="1"><tr><td>9</td><td>2</td><td>5</td><td>3</td></tr></table>	9	2	5	3	$1/4(1-1/4)+1/4(1-1/4)+1/4(1-1/4)+1/4(1-1/4)=12/16$
9	2	5	3			

Fig. 2.3

From Fig. 2.3, it is noted that QP1 has the minimum QPC followed by QP2, QP3, QP4 and QP5 has the maximum QPC. The aim is to generate query plans with lower QPC.

The iterative improvement and simulated annealing techniques are used to generate such optimal query plans. The iterative improvement technique is discussed in the next section followed by simulated annealing in the subsequent section.

2.2 Iterative Improvement

The general iterative improvement algorithm, given in [IK90], is shown in Fig. 2.4. In the iterative improvement algorithm there are two loops. The inner loop has been used for local optimization. The Local optimization starts with a random state selected from search space and improves the solution by repeatedly accepting the downhill moves until it reaches a local minimum. The iterative improvement repeats these local optimization until the stopping condition is met. Once the stopping condition is met it returns the local minimum with minimum cost associated with state. For infinite time, there is a probability 1 that Iterative improvement would attain global minimum [IK90][NSS86]. Attaining the global minimum depends on the cost function and its neighbor function [IK90].

```
Procedure II ()
{
  minS=S0
  while not (stopping condition) do
  {
    S=random state,
    while not (local minimum(S)) do
    {
      S'=random state in neighbors(S),
      If cost(S') < cost(S) then S=S';
    }
    If cost(S) < cost(minS) then minS=S,
  }
  return (minS),
}
```

Fig. 2.4

This generic iterative improvement algorithm is adapted to query plan generation problem. The Iterative improvement algorithm for query plan generation is shown in Fig. 2.5. This algorithm takes site-relation matrix and relation participating in the FROM clause of the SQL query as input and produces optimal query plan as output.

```

Input: Site relation matrix and relations participating in from clause of SQL Query
Output: Optimal plan with minimum cost.
Method:
    minQP = IntMax
    WHILE not (Stopping condition) do
        QP= random query plan
        WHILE NOT (local minimum(QP)) do
            QP'= random query plan in neighbors(S)
            IF cost(QP') < Cost (QP) THEN QP= QP'
        END WHILE
        IFcost(QP) < Cost (minQP) then minS= QP
    END WHILE
    Return minQP.

```

Fig. 2.5

The stopping condition in the algorithm can be defined by either the Query Plan Constraint or the Time Constraint. The working of algorithm starts with taking random query plan for a user query. The inner while loop is used for local optimization, which starts with the random query plan and improves the solution by repeatedly accepting random downhill moves until it reaches a local minimum. This local optimization is repeated until the stopping condition is met. Here the stopping condition can be the query plan constraint, which is nothing but number of local optimization, or the time constraint, time allowed to find the local minima. When the stopping condition is met than minQP is returned which is the minimum cost of a valid query plan which would be the optimal plan. Next parameter in the iterative improvement is the local minimum of a plan QP. Here the local minimum is r-local minimum [IK90]. The r-local minimum is defined in terms of n randomly chosen neighbor of a query plan and tested with repetition, where n is the actual number of neighbors of QP. Actually r-local minimum can be defined in this way also where a random query plan is chosen from the search space and n neighbor of this query plan are found whose cost should not be less than that of the chosen query plan. During testing if neighbor's cost of the query plan is less than the chosen query plan then the neighbor query plan become the new query plan which will be further tested for local minima. The iterative improvement algorithm considers neighbor of a query plan in its

computation of the optimal query plan. The neighbor of a query plan is determined by changing atleast and atmost one site in the query plan by a valid site. For example the neighbor of the query plan QP4 [7, 8, 1, 1] is [7, 8, 5, 1] where site of the third relation is changed from 1 to 5.

2.2.1 An Example

Consider the relation/site matrix shown in Fig. 2.6.

Relation	Site				
	S1	S2	S3	S4	S5
R1	1	1	0	0	1
R2	1	0	1	1	0
R3	1	1	0	0	1
R4	0	0	0	1	1
R5	0	1	1	0	0

Fig. 2.6

According to Fig. 2.6, relation R1 is at site S1, S2 and S5, relation R2 is at site S1, S3 and S4, relation R3 is at site S1, S2 and S5, relation R4 is at site S4, S5 and relation R5 is at site S2 and S3. Let the user query in SQL have R1, R3 and R4 in the FROM clause.

2.2.1.1 Optimal Query Plan Generation with Query Plan Constraint

Let the input parameters be

1. The stopping condition is number of query plans: 5.
2. For r-local minimum the value of n neighbor is taken n=20.
3. The initial value of minQP= ∞ .

Now iterative improvement start with initial value of minQP= ∞ . The stopping condition is 5 query plans i.e. 5 times r-local minimum condition should be satisfied. Let the random valid query plan QP chosen be as given below:

R1	R3	R4
1	1	4

Here R1 is at site 1, R3 is at site 1 and R4 is at site 4. This can be seen from the relation/site table. The cost of the query plan can be computed using the cost function as follows

Here Number of sites accessed i.e. $M=2$.

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S1 is used is 2 and number of times site S4 is used is 1

Then

$$QPC = \frac{2}{3} \left(1 - \frac{2}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.444$$

The cost of the chosen random query plan QP is 0.444.

Next, there is WHILE loop in which there is a condition for finding r-local minimum, which is taken as $n=20$ neighbor of query plan QP, whose cost of all 20 query plans should be greater than QP. Let us consider a neighbor query plan of QP as given below. The neighbor query plan has site for relation R4 changed from 4 to 5.

R1	R3	R4
1	1	5

Here

Number of sites accessed i.e. $M=2$.

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S1 is used is 2 and number of times site S5 is used is 1.

Then

$$QPC = \frac{2}{3} \left(1 - \frac{2}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.444$$

The cost of new valid neighbor query plan QP' is 0.444. Since the cost of QP' is not less than that of QP, so QP remains as the minimum cost query plan. The r-local minimum is

not found yet, so again there is need to find the neighbor of QP. Let another neighbor query plan of QP be as given below where site for relation R3 is changed from 1 to 4.

R1	R3	R4
1	2	4

Here

Number of sites accessed i.e. $M=3$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S1 is used is 1, the number of times site S4 is used is 1 and number of times site S2 is used is 1

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

The cost of QP' is 0.666, which is not less than that of QP and therefore QP remains as the minimum cost query plan. QP is still not the local minima and therefore another neighbor of QP is found and is given below where the site for relation R1 is changed from 1 to 2.

R1	R3	R4
2	1	4

Here

Number of sites accessed i.e. $M=3$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S1 is used is 1, the number of times site S4 is used is 1 and the number of times site S2 is used is 1

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

The cost of QP' is 0.666. This cost is not less than that of QP and therefore QP remains as the minimum cost query plan. This will continue in the similar way till we find query plan QP whose 20 randomly generated neighboring query plans have cost not less than QP i.e. the corresponding QP is identified as the local minima. In similar manner local minimas are identified till the algorithm identifies 5 local minimas. Among them, the query plan with minimal cost value is considered the optimal query plan. The following optimal query plan can be generated by the iterative improvement algorithm.

R1	R3	R4
5	5	5

The above query plan has cost 0 and is the minimal cost that any query plan can achieve for a given user query.

2.2.1.2 Optimal Query Plan Generation with Time Constraint

Let the input parameters be

1. The stopping condition is time = 2 seconds
2. For r-local minimum the value of n neighbor is taken n=20.
3. The initial value of minQP= ∞ .

Let the randomly generated query plan QP be as given below

R1	R3	R4
1	5	4

Here

Number of sites accessed i.e. $M=3$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S1 is used is 1, the number of times site S5 is used is 1 and the number of times site S4 is used is 1



TH-19209

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

The QPC of query plan QP is 0.666.

The next step is to find the neighbor query plan for QP. Let the neighbor query plan QP' for QP is as given below where the site for relation R1 is changed from 1 to 5.

R1	R3	R4
5	5	4

Here

Number of sites accessed i.e. M=2

Number of relation accessed by the query plan i.e. N=3

Number of times site S5 is used is 2 and number of times site S4 is used is 1

Then

$$QPC = \frac{2}{3} \left(1 - \frac{2}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.444$$

The QPC of QP' is 0.444. Since the cost of QP' is lower than that of QP, QP' becomes the new QP. The stopping condition of 2 seconds does not hold true and therefore neighbor query plan QP' of QP is found. Let that be as given below

R1	R3	R4
5	2	4

Here

Number of sites accessed i.e. M=3

Number of relation accessed by the query plan i.e. N=3

Number of times site S5 is used is 1, number of times site S2 is used is 1 and number of times site S4 is used is 1.

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

The QPC of QP' is 0.666. Since the cost of QP' is not less than that of QP and therefore QP remains as the minimum cost query plan. Since the stopping condition is not met yet, the neighbor of QP i.e. QP' is determined and let that be as given below

R1	R3	R4
2	5	4

Here

Number of sites accessed i.e. M=3

Number of relation accessed by the query plan i.e. N=3

Number of times site S2 is used is 1, number of times site S5 is used is 1 and number of times site S4 is used is 1.

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

The QPC of QP' is 0.666, which is not less than that of QP and therefore QP remains as the minimum cost query plan. The stopping condition is not met and therefore neighbor of QP i.e. QP' is determined and is as given below

R1	R3	R4
5	1	4

Here

Number of sites accessed i.e. M=3

Number of relation accessed by the query plan i.e. N=3

Number of times site S5 is used is 1, number of times site S1 is used is 1 and the number of times site S4 is used is 1

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

The QPC of QP' is 0.666, which is not less than the cost of QP and therefore QP remains as the minimum cost query plan. This continues till the stopping condition of 2 seconds is met. If the algorithm is able to find local minima, the corresponding query plan becomes the minimal query plan for the user query. For example, the algorithm may generate the query plan as given below which has QPC as 0. This query plan can be considered an optimal query plan for the user query.

R1	R3	R4
5	5	5

Actually the iterative improvement is dependent on the random generation of the query plans and therefore may not guarantee optimal query plan always. In the above example time constraint is the main factor to generate the optimal query plan. If the time is longer then there is a possibility to generate the optimal query plan.

The problem with the iterative improvement is that it gets stuck in the high cost local minima. Due to this problem, iterative improvement may not give better results always. To address this problem, simulated annealing is used. Query plan generation using Simulated Annealing is discussed next.

2.3 Simulated Annealing

The general simulated annealing algorithm, given in [IK90], is shown in Fig. 2.7. Simulated annealing accepts uphill moves with some probability. It tries to avoid the problem of being caught in local minima. In simulated annealing algorithm the inner loop has been used to control the probability of accepting the uphill moves. It also does the

local optimization. The probability is controlled by the parameter temperature which is fixed for each stage in the inner loop. Here stage is defined as inner loop of simulated annealing. For each stage, temperature is reduced by reduction function. In each stage uphill moves is determined by temperature and the difference of the cost of new state and the original state. The inner loop stops when it reaches equilibrium. The temperature is reduced by some reducing function. The outer loop is stopped when the state is considered as frozen. In simulated annealing algorithm it is considered that it will attain the global minimum of state when the temperature is approaching zero [IK90][RV85]. The overall performance of the simulated annealing is dependent on the cost function and the neighbor function used in it [IK90].

```

Procedure SA()
{
  S=S0,
  T=T0,
  minS=S,
  WHILE NOT (frozen) do
  {
    WHILE NOT ( equilibrium) DO
    {
      S'=random state in neighbors(S),
      ΔC=cost (S') – cost (S),
      IF (ΔC ≤ 0) THEN S= S'

      IF (ΔC ≥ 0) THEN S= S' with probability  $e^{\frac{-\Delta c}{T}}$ 
      IF cost (S) < cost (minS) THEN minS=S,
    }
    T=reduce (T)
  }
  RETURN (minS),
}

```

Fig. 2.7

This generic simulated annealing algorithm is adapted to query plan generation problem. The simulated annealing algorithm for query plan generation is shown in Fig. 2.8. This algorithm takes site-relation matrix, relation participating in the FROM clause of the SQL query and the reduction factor as input and produces optimal query plan as output.

```

Input: Site relation matrix, relations participating in from clause of SQL Query, reduction factor
Output: Optimal plan with minimum cost.
Method:
  QP=QP0
  T=T0
  minQP=QP.
  WHILE NOT (stopping condition) DO
    WHILE NOT (inner stopping condition) DO
      QP'=random query plan in neighbor of QP.
      c=cost(QP')-cost(QP)
      IF (c≤0) THEN QP=QP'
      IF (c>0) THEN QP=QP' with probability  $e^{\frac{-c}{T}}$ 
      IF cost(QP) < cost (minQP) THEN minQP=QP.
    END WHILE
  T= reduce the value of T by some defined function.
  END WHILE
  RETURN minQP

```

Fig. 2.8

As discussed above that problem with the iterative improvement gets stuck in the local minimum of the search space. This is due to the fact that iterative improvement performs only downhill moves and thus gets stuck at the local minimum of the search space. This problem has been addressed by simulated annealing, which also performs uphill moves with some probability. The probability depends on the temperature. The temperature is used here to avoid the circumstances of being caught in the local minimum of the search space.

In the algorithm given in Fig. 2.8, there are two while loops. The inner while loop is used for stage, which is used to accept uphill moves with some probability. This probability is dependent on the temperature of the stage, which is dependent on the cost of the query plan. The stage is a kind of situation in which once downhill move has reached and it has stuck in the local minimum than it is required to take it out from that point. That is the reason why concept of uphill moves is taken from the simulated annealing with some probability. The probability $e^{\frac{-c}{T}}$ is constituted by two factor c and T. The c is the difference of the cost of the query plan QP and its neighbor query plan QP'.

The parameter specific to this algorithm for finding the optimal query plan are as follows:

1. The initial state QP_0 which is randomly selected query plan
2. The temperature $T=T_0$, which is a function of Cost where say $T_0=2000*\text{cost}(QP_0)$. i.e. 2000 times the initial cost of the randomly chosen query plan.
3. The outer loop stopping condition that is also known as frozen condition is
 $T > 1$ AND minQP is unchanged at least in 10 stages.
4. The inner loop condition is $8 \times (\text{no of relation participating in from clause of SQL query posed } - 1) \times (\text{no of relation participating in from clause of SQL query posed } - 2)$.
5. The next state is randomly selected neighbor of query plan.
6. The temperature reduction is - Temperature new = 0.95* Temperature old.

This algorithm will be explained with two stopping condition namely the query plan constraint, which is nothing but number of local optimization, or the time constraint, time allowed to find the local minima.

2.3.1 An Example

Consider the relation/site matrix shown in Fig. 2.9

Relation	Site				
	S1	S2	S3	S4	S5
R1	1	1	0	0	1
R2	1	0	1	1	0
R3	1	1	0	0	1
R4	0	0	0	1	1
R5	0	1	1	0	0

Fig. 2.9

According to Fig. 2.9, relation R1 is at site S1, S2 and S5, relation R2 is at site S1, S3 and S4, relation R3 is at site S1, S2 and S5, relation R4 is at site S4, S5 and relation R5 is at site S2 and S3. Let the user query in SQL have R1, R3 and R4 in the FROM clause.

2.3.1.1 Optimal Query Plan Generation with Query Plan Constraint

The input to the algorithm is the site relation matrix and the relation in the FROM clause of the query. The stopping condition is number of query plans: 10, The number of neighbors for r-local minimum i.e. n=20 and the reduction factor is 0.95 of the old temperature. As per the Site Relation table, shown in Fig. 2.9, relation R1 is at site S1, S2 and S5. Relation R2 is at site S1, S3 and S4. Relation R3 is at site S1, S2 and S5. Relation R4 is at site S4, S5. Relation R5 is at site S2 and S3. Let us consider a SQL query with relation R1, R3 and R4 in its FROM clause. Let the random query plan QP generated be as given below

R1	R3	R4
1	5	4

Here

Number of sites accessed i.e. M=3

Number of relation accessed by the query plan i.e. N=3

Number of times site S1 is used is 1, the number of times site S5 is used is 1 and the number of times site S4 is used is 1.

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

Let $T_0 = 2000 * \text{cost}(QP) = 2000 * 0.666 = 1332.0$

$T = T_0, \text{minQP} = QP$

If the outer loop terminating condition is

$T > 1$ AND minQP is unchanged in at least 10 stages.

If the inner loop terminating condition is

$8 \times (\text{no of relation participating in from clause of SQL query posed} - 1) \times (\text{no of relation participating in from clause of SQL query posed} - 2)$.

For the example the inner loop would execute $8*(3-1)*(3-2) = 8*2*1=16$ times to achieve the equilibrium. The inner loop starts by selecting a random neighbor query plan for QP.

Let the neighbor query plan QP' of QP be as given below

R1	R3	R4
5	5	4

Here

Number of sites accessed i.e. $M=2$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S5 is used is 2 and number of times site S4 is used is 1.

Then

$$QPC = \frac{2}{3} \left(1 - \frac{2}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.444$$

The cost difference of query plan QP' and QP is

$$C = 0.444 - 0.666 = -0.22.$$

Since the cost is less than equal to the zero, $QP = QP'$

Now, $\text{cost}(QP)$ is less than $\text{cost}(\text{minQP})$ therefore $\text{minQP} = QP$

Now the inner loop would execute $16-1=15$ times. Let the next randomly generated neighbor query plan QP' of QP be as given below

R1	R3	R4
5	2	4

Here

Number of sites accessed i.e. $M=3$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S5 is used is 1, number of times site S3 is used is 1 and the number of times site S4 is used is 1.

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

The QPC of QP' is 0.666 and the cost difference is

$$C = 0.666 - 0.444 = 0.222$$

Since the difference of the cost is greater than zero, there is a need to find the probability so that the decision can be made to replace QP by QP'

$$d = (\text{cost}(\text{QP}) - \text{cost}(\text{QP}')) * 2000 / T = (0.666 - 0.444) * 2000 / 1332.0 = 0.33$$

$$\text{prob} = e^{-d} = 2.71^{-0.33} = 0.718948$$

On the basis of this probability, it is found that QP = QP'.

Further, cost(QP) i.e. 0.666 is not less than cost(minQP) i.e. 0.444, therefore minQP value would be retained.

Now the inner loop would execute 15-1=14 times. Let the next randomly generated neighbor be as given below

R1	R3	R4
5	2	5

Here

Number of sites accessed i.e. M=2

Number of relation accessed by the query plan i.e. N=3

Number of times site S5 is used is 2 and number of times site S2 is used is 1

Then

$$QPC = \frac{2}{3} \left(1 - \frac{2}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.444$$

The cost difference c = 0.444 - 0.666 = -0.222.

Since cost difference is less than 0, QP = QP'

Now cost(QP) is same as cost(minQP), so minQP is retained. Now the inner loop has to execute $14-1=13$ times. Suppose, the inner loop further runs for 13 neighbor query plans and it is found that the minQP is

R1	R3	R4
5	5	4

Let a random query plan for finding the another local minima is as given below

R1	R3	R4
1	2	4

Here

Number of sites accessed i.e. $M=3$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S1 is used is 1, number of times site S2 is used is 1 and the number of times site S4 is used is 1.

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

The cost(QP)=0.666. The value of $T = T * \text{Reduction Factor}$

$$= 1332.0 * 0.95 = 1265.40.$$

Again inner loop will execute 16 times as in the case discussed above

Let us consider a random query plan as given below

R1	R3	R4
5	2	4

Here

Number of sites accessed i.e. $M=3$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S5 is used is 1, number of times site S2 is used is 1 and the number of times site S4 is used is 1.

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

The cost difference $c = \text{cost}(QP') - \text{cost}(QP)$

$$= 0.666 - 0.666 = 0.000$$

Therefore, $QP = QP'$ and minQP remains unchanged. In the next iteration, let the random neighbor query plan QP' be as given below

R1	R3	R4
5	5	4

Here

Number of sites accessed i.e. $M=2$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S5 is used is 2 and number of times site S4 is used is 1

Then

$$QPC = \frac{2}{3} \left(1 - \frac{2}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.444$$

The cost difference $c = \text{cost}(QP') - \text{cost}(QP)$

$$= 0.444 - 0.666 = -0.22.$$

Since cost difference is less than 0, $QP = QP'$.

Now $\text{cost}(QP) < \text{cost}(\text{minQP})$ and therefore $\text{minQP} = QP$.

In the next iteration, consider a neighbor query plan QP' given below

R1	R3	R4
5	5	5

Here

Number of sites accessed i.e. $M=1$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S5 is used is 3.

Then

$$QPC = \frac{3}{3} \left(1 - \frac{3}{3}\right) = 0.000$$

The cost difference $c = \text{cost}(QP') - \text{cost}(QP)$

$$= 0.000 - 0.444 = -0.444$$

Thus, $QP = QP'$

Now $\text{cost}(QP) < \text{cost}(\min QP)$ so $\min QP = QP$.

In the next iteration, let the randomly generated neighbor plan be as given below

R1	R3	R4
5	1	5

Here

Number of sites accessed i.e. $M=2$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S5 is used is 2, number of times site S1 is used is 1

Then

$$QPC = \frac{2}{3} \left(1 - \frac{2}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.444$$

The cost difference $c = \text{cost}(QP') - \text{cost}(QP)$

$$= 0.444 - 0.000 = 0.444.$$

Since the difference of the cost is greater than zero, there is a need to find the probability

so that the decision can be made whether to accept QP' as QP .

$$d = (\text{cost}(QP) - \text{cost}(QP')) * 2000 / T = (0.444 - 0.000) * 2000 / 1332.0 = 0.70$$

$$\text{prob} = e^{-d} = 2.71^{-0.70} = 0.4966$$

Based on the probability, suppose QP' as QP is not accepted.

The difference of the cost and temperature decides QP. In the above example it is seen that QP is same as it was before and the same is applied to the minQP. Again, it would start with the selection of the neighbor query plan of QP and their cost is compared. In the similar manner as above the computations are performed. Finally say for this particular example, the following optimal query plan is achieved

R1	R3	R4
5	5	5

2.3.1.1 Optimal Query Plan Generation with Time Constraint

The input to the algorithm is the site relation matrix, shown in Fig. 2.9, and the relation in the FROM clause of the query. The stopping condition is time = 2 seconds, the number of neighbors for r-local minimum i.e. n=20 and the reduction factor is 0.95 of the old temperature. Let us consider a SQL query with relation R1, R3 and R4 in its FROM clause. Let the random query plan QP generated be as given below

R1	R3	R4
1	5	4

Here

Number of sites accessed i.e. M=3

Number of relation accessed by the query plan i.e. N=3

Number of times site S1 is used is 1, number of times site S5 is used is 1 and the number of times site S4 is used is 1

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

Then T = 2000*cost(QP)=2000*0.666=1332.0

minQP = QP

Let us consider a neighbor query plan as given below

R1	R3	R4
5	5	4

Here

Number of sites accessed i.e. $M=2$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S5 is used is 2 and number of times site S4 is used is 1

Then

$$QPC = \frac{2}{3} \left(1 - \frac{2}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.444$$

The cost difference $c = \text{cost}(QP') - \text{cost}(QP)$

$$= 0.444 - 0.666 = -0.22.$$

Since the difference in the cost is less than or equal to zero, $QP = QP'$.

Now $\text{cost}(QP) < \text{cost}(\min QP)$ so $\min QP = QP$.

In the next iteration, let the neighbor query be as given below

R1	R3	R4
5	2	4

Here

Number of sites accessed i.e. $M=3$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S5 is used is 1, number of times site S2 is used is 1 and the number of times site S4 is used is 1.

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

The cost difference $c = \text{cost}(QP') - \text{cost}(QP)$

$$= 0.666 - 0.444 = 0.222$$

Since the difference of the cost is greater than zero, there is a need to find the probability so that the decision can be made whether to accept QP' as QP.

$$d = (\text{cost}(\text{QP}) - \text{cost}(\text{QP}')) * 2000 / T = (0.666 - 0.444) * 2000 / 1332.0 = 0.33$$

$$\text{prob} = e^{-d} = 2.71^{-0.33} = 0.718948$$

Based on the probability, suppose QP' is accepted as QP

Now cost(QP) i.e. 0.666 is not less than cost(minQP) i.e. 0.444, so minQP would remain unchanged.

In the next iteration, let the neighbor query plan be as given below

R1	R3	R4
5	2	5

Here

Number of sites accessed i.e. M=2

Number of relation accessed by the query plan i.e. N=3

Number of times site S5 is used is 2 and number of times site S2 is used is 1.

Then

$$\text{QPC} = \frac{2}{3} \left(1 - \frac{2}{3} \right) + \frac{1}{3} \left(1 - \frac{1}{3} \right) = 0.444$$

The cost difference $c = \text{cost}(\text{QP}') - \text{cost}(\text{QP})$

$$= 0.444 - 0.666 = -0.222.$$

Since the cost difference is less than 0, $\text{QP} = \text{QP}'$

Now cost(QP) i.e. 0.444 is same as cost(minQP) i.e. 0.444, minQP remains unchanged.

This continues till the inner loop is executed in all 16 times and let on its completion the minQP is

R1	R3	R4
5	2	5

Let a random query plan for finding the another local minima is as given below

R1	R3	R4
1	2	4

Here

Number of sites accessed i.e. $M=3$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S1 is used is 1, number of times site S2 is used is 1 and the number of times site S4 is used is 1

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

The cost(QP)=0.666.

Lets the random neighbor query plan QP' of QP be as given below

R1	R3	R4
5	2	4

Here

Number of sites accessed i.e. $M=3$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S5 is used is 1, number of times site S2 is used is 1 and the number of times site S4 is used is 1

Then

$$QPC = \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) + \frac{1}{3} \left(1 - \frac{1}{3}\right) = 0.666$$

The cost difference $c = \text{cost}(QP') - \text{cost}(QP)$

$$= 0.666 - 0.666 = 0.000$$

Since the difference of the cost is less than or equal to zero, $QP=QP'$ and $\min QP$ remains unchanged. In the next iteration, let us consider a neighbor query plan QP' of QP .

R1	R3	R4
5	5	4

Here

Number of sites accessed i.e. $M=2$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site $S5$ is used is 2 and the number of times site $S4$ is used is 1

Then

$$QPC = \frac{2}{3} \left(1 - \frac{2}{3} \right) + \frac{1}{3} \left(1 - \frac{1}{3} \right) = 0.444$$

The cost difference $c = \text{cost}(QP') - \text{cost}(QP)$

$$= 0.444 - 0.666 = -0.222$$

Since the cost difference is less than zero, $QP=QP'$.

Now cost (QP) i.e. 0.444 is less than cost ($\min QP$) i.e. 0.666 and therefore $\min QP=QP$.

Let the neighbor query plan QP' of QP be as given below

R1	R3	R4
5	5	5

Here

Number of sites accessed i.e. $M=1$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site $S5$ is used is 3

Then

$$QPC = \frac{3}{3} \left(1 - \frac{3}{3} \right) = 0.000$$

The cost difference $c = \text{cost}(QP') - \text{cost}(QP)$

$$= 0.000 - 0.222 = -0.222$$

Since the cost is less than or equal to the zero, $QP=QP'$

Now cost (QP) is less than cost (minQP) i.e. 0.222 and therefore $\min QP=QP$

Let the next neighbor query plan be as given below

R1	R3	R4
5	1	5

Here

Number of sites accessed i.e. $M=2$

Number of relation accessed by the query plan i.e. $N=3$

Number of times site S5 is used is 2

Number of times site S1 is used is 1.

Then

$$QPC = \frac{2}{3} \left(1 - \frac{2}{3} \right) + \frac{1}{3} \left(1 - \frac{1}{3} \right) = 0.444$$

The cost difference $c = \text{cost}(QP') - \text{cost}(QP)$

$$= 0.444 - 0.000 = 0.444.$$

Since the difference of the cost is greater than zero, there is a need to find the probability

so that the decision can be made whether to accept the QP' as QP

$$d = (\text{cost}(QP) - \text{cost}(QP')) * 2000 / T = 0.444 - 0.000 * 2000 / 1332.0 = 0.70$$

$$\text{prob} = e^{-d} = 2.71^{-0.70} = 0.4966$$

On the basis of this probability, QP' is accepted as QP

Now, $\text{cost}(QP)$ i.e. 0.444 is not less than $\text{cost}(\min QP)$ i.e. 0, $\min QP$ remains unchanged

In the similar manner, QP' and $\min QP$ are computed. Finally, say the optimal query plan generated is as given below

R1	R3	R4
5	5	5

The simulated annealing in comparison to the iterative improvement algorithm is likely to generate better query plans as it performs uphill moves with some probability along with the downhill moves. This enables it not to get stuck in the local minima. Also it enables exploring the entire search space for local minimas.

In order to study the quality of query plans generated by Iterative Improvement and Simulated Annealing, these algorithms were implemented in C++ programming language. The results based on the experiments are discussed next.

2.4 Experimental Results

Iterative Improvement and Simulated annealing are compared on two parameters namely the number of query plans generated and the average QPC achieved by these generated query plans. These comparisons are made by varying the time from 1 to 9 seconds with the step of 1 second. The experiments are performed for a use query in SQL containing four relations, six relations, eight relations and ten relations given there were 20 relations and 20 sites. The graphs for four relations, six relations, eight relations and ten relations on parameters number of query plans generated and the average QPC are shown in Fig 2.10-2.11, Fig. 2.12-2.13, Fig. 2.14-2.15 and Fig. 2.16-2.17 respectively.

The results of these experiments show that iterative improvement is able to generate significantly large number of query plans in comparison to simulated annealing. This implies that the number of local-minimas attained is more in the case of iterative improvement. Further, the simulated annealing, in comparison to iterative improvement, is able to generate better query plans. This may be due to the fact that simulated annealing explores the search space with both down-hill and up-hill moves thereby able to generate query plans with lesser value of QPC.

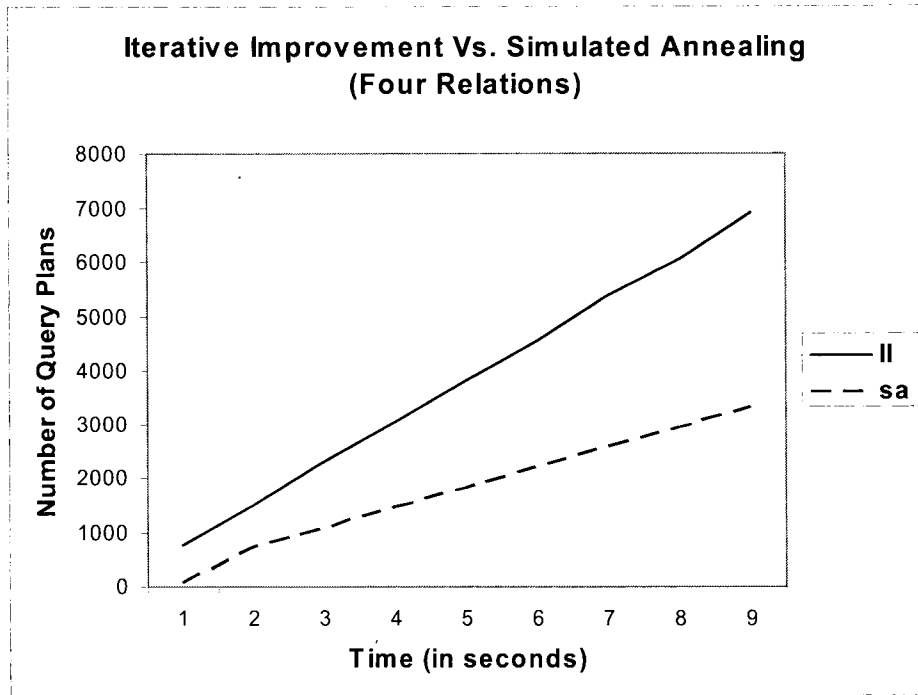


Fig. 2.10

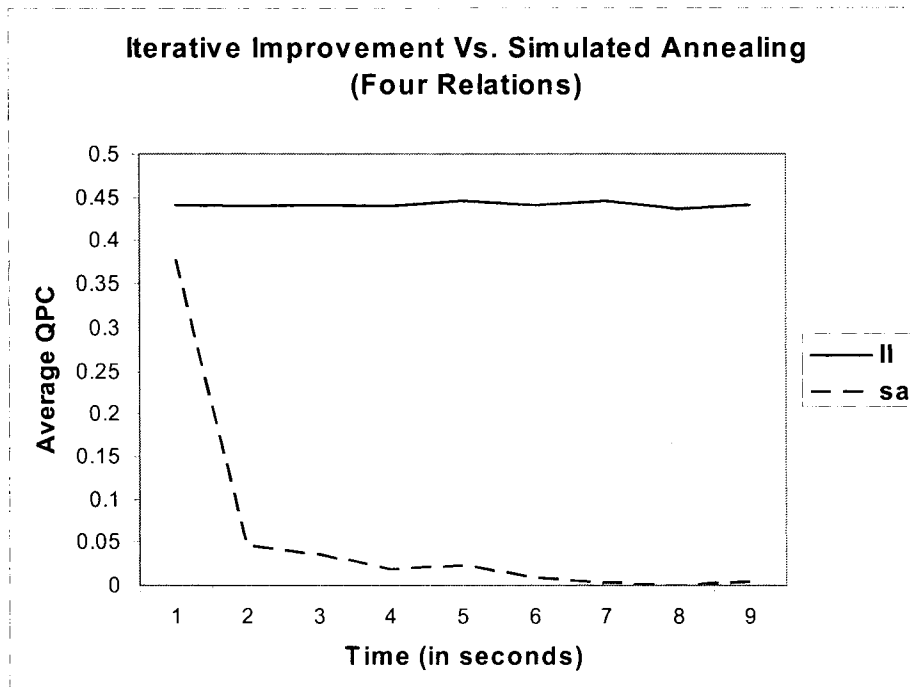


Fig. 2.11

**Iterative Improvement Vs. Simulated Annealing
(Six Relations)**

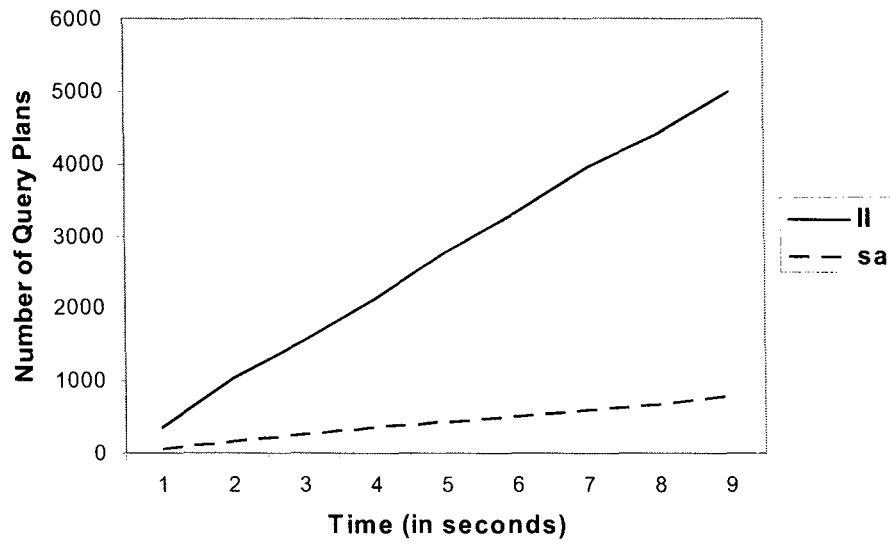


Fig. 2.12

**Iterative Improvement Vs. Simulated Annealing
(Six Relations)**

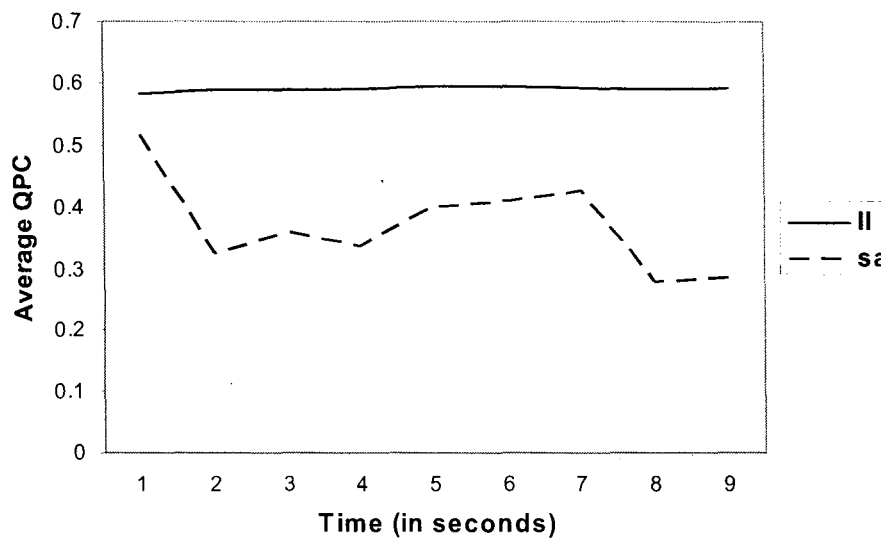


Fig. 2.13

**Iterative Improvement Vs. Simulated Annealing
(Eight Relations)**

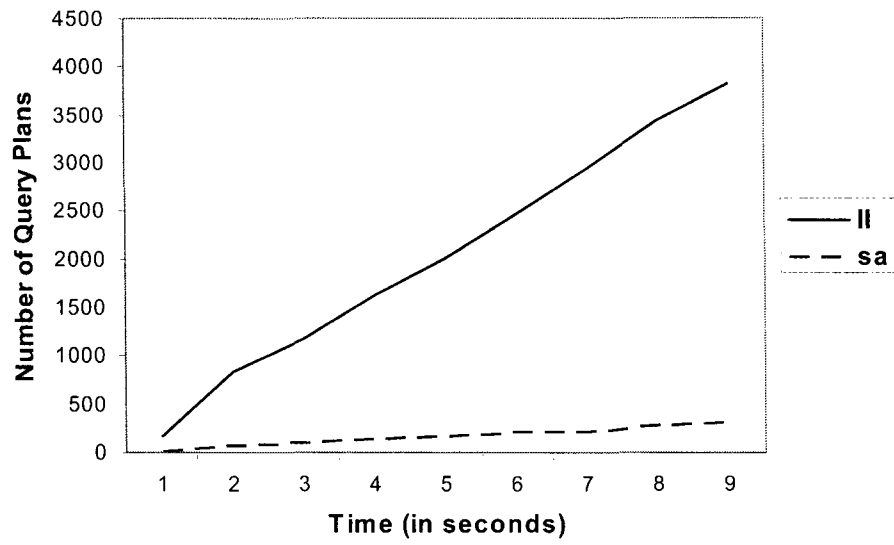


Fig. 2.14

**Iterative Improvement Vs. Simulated Annealing
(Eight Relations)**

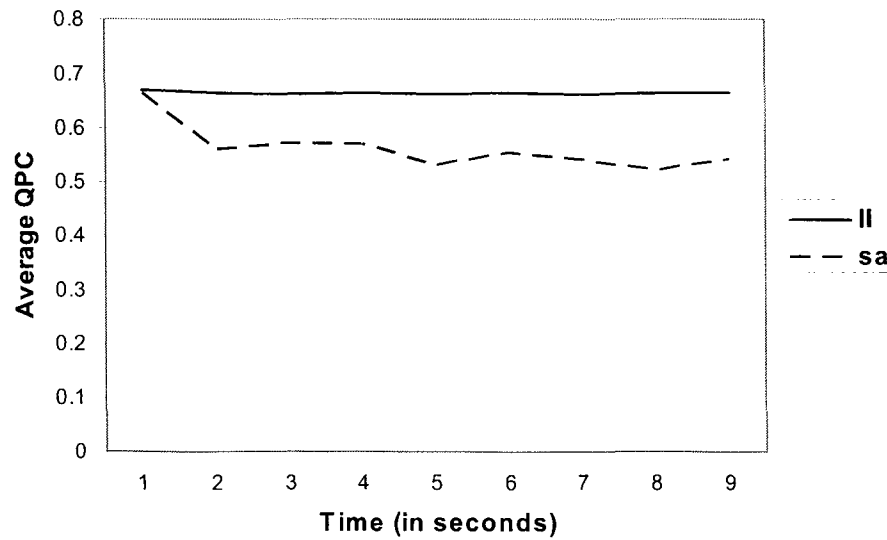


Fig. 2.15

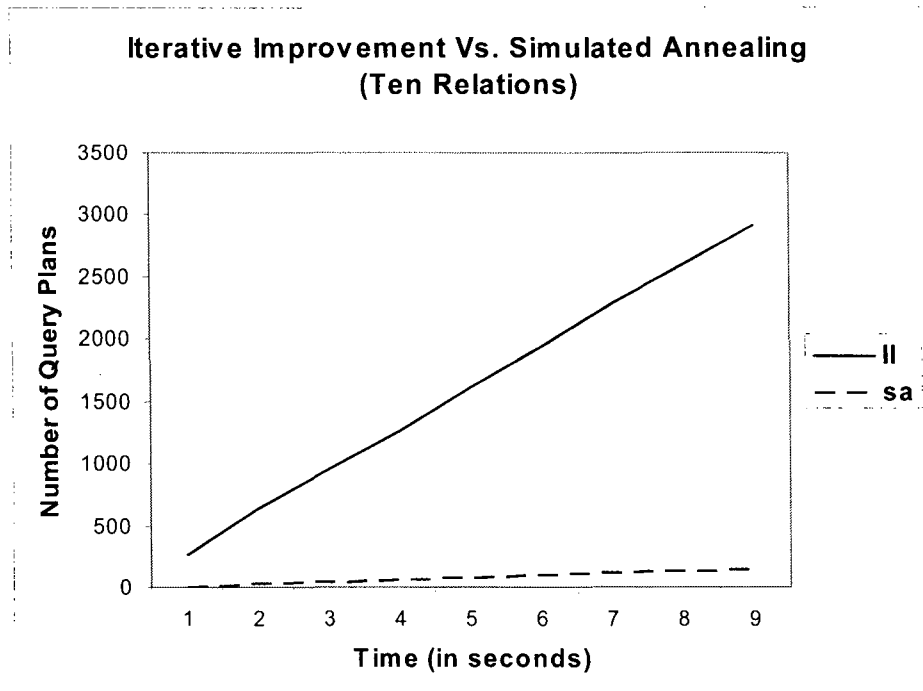


Fig. 2.16

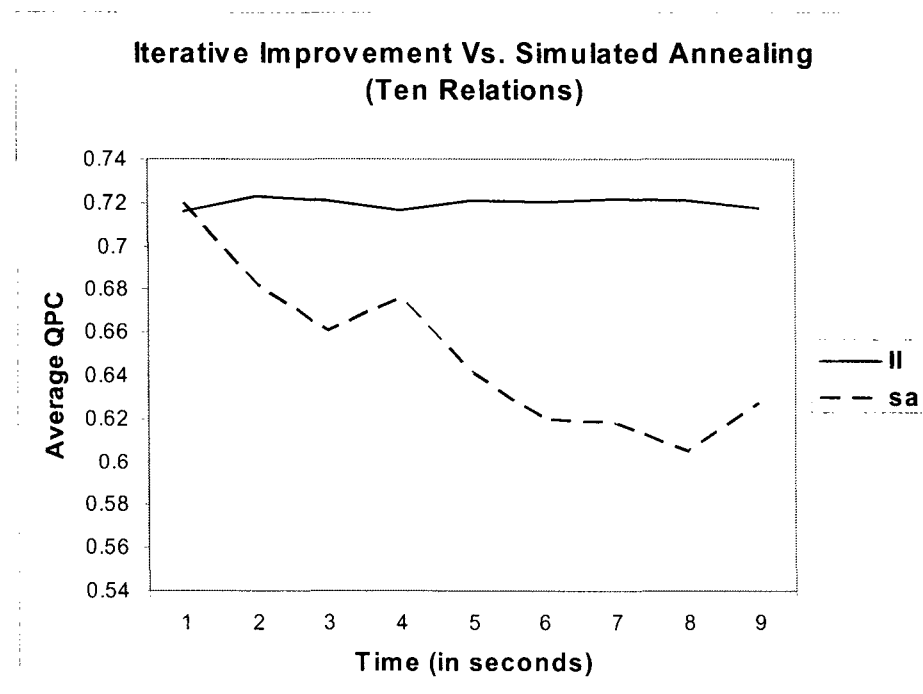


Fig. 2.17

CHAPTER 3

Conclusion

A query posed over a distributed database may get processed against disparate data sources distributed over a network. Each of these sources may contain data relevant to the query. The aim of distributed database system is to provide efficient query processing strategy for the given query. In distributed database scenario, multiple copies of the same data may reside at different sources. As a result, there can be multiple query strategy for a given query and finding an optimal query processing strategy is a combinatorial optimization problem. In this dissertation, an approach is presented that is able generate optimal query processing plans for a given user query. The approach uses iterative improvement and simulated annealing algorithms to determine optimal query plans for a given query. The approach uses the cost heuristic defined in [VSV10].

First, the iterative improvement algorithm is used to determine the optimal query plans. The algorithm finds query plans as per r-local minimas from which the query plan with minimum cost is considered optimal query plan. The problem with the iterative improvement is that it performs only downhill moves and thus gets stuck at the local minimum of the search space. Due to this problem, iterative improvement may not give

better results always. To address this problem, simulated annealing was used to generate optimal query plans. The simulated annealing performs uphill moves with some probability along with the downhill moves thereby avoiding getting stuck at the local minima. As a result, it generates better query plans for a given query.

Further, the experimental based comparison of iterative improvement and simulated annealing shows that though iterative improvement is able to generate more query plans per unit time, simulated annealing is able to generate better quality query plans.

References

- [AAO05] Aljanaby A., Abuelrub E. and Odeh M., 2005, "A Survey of Distributed Query Optimization" The International Arab Journal of information Technology, Vol.2
- [CP84] Ceri S. and Pelagati G., 1984 "Distributed Database: Principles and System," McGraw Hill.
- [CS94] Chaudhuri S. and Shim K. 1994. "Including group-by in query optimization," in proceedings of the 20th International conference on very Large Data Base (VLDB'94, Santiago, Chile, Sept.), VLDB Endowment, Berkeley, CA pp. 354-366.
- [CS96] Chaudhuri S. and Shim K. 1996. "Optimization of queries with user-defined predicates," in proceedings of the 22th International conference on very Large Data Base (VLDB'96, Bombay, India, Sept.) pp.87-98.
- [CYW96] Chen M. S., Yu P. S., and Wu K. L., 1996. "Optimization of Parallel Execution for Multi-join Queries," IEEE Transaction on knowledge and Data Engineering, vol. 3.
- [DG92] DeWitt D. J. and Gray J. 1992. "Parallel Database Systems: The Future of High Performance Database Systems," Communication of ACM vol.35, no. 6, pp. 85-98.
- [EN00] Elmasri R. and Navathe S. B., 2000, "Fundamentals of Database Systems" Reading MA, Addison –Wesley.

- [G93] GRAEFE, G. 1993. Query Evaluation technique for large databases. ACM Computing Surveys 25, 2(June),pp. 73-170
- [HY79] Hevner A.R.and Yao S.B., 1979. "Query processing in Distributed database System," IEEE Transaction on Software Engineering SE-5, pp. 177-187.
- [IK84] Ibraraki T. and kameda T., 1984. "optimal nesting for computing N-Relational joins," ACM Transactions on Database Systems, vol. 9, no. 3, pp.482-502.
- [IK90] Ioannidis Y.E. and Kang Y. C., "Randomized Algorithm for Optimizing Large Join Queries", 1990. In proceeding of the ACM SIGMOD Conference on Management of Data, Atlantic City, USA, pp. 312-321.
- [IW87] Ioannidis Y. and Wong E. 1987. "Query optimization by simulated annealing," In proceeding of the ACM SIGMOD Conference on Management of Data (San Francisco, CA, May), ACM , Newyork, NY. Pp. 9-22.
- [JK84] Jarke, M. and Koch, J. 1984. " Query optimization in database Systems," ACM Computing Surveys, Volume 16, no.2, pp. 111-152.
- [JW+90] Jenq, B., Woelk, D., Kim, W. and Lee, W., 1990. Query processing in distributed ORION. In proceedings of the International Conference on Extended Database Technology (EDBT) (Venice, Italy, March), pp. 169-187
- [K00] KOSSMANN, D.2000. "The State of the Art in Distributed Query Processing". ACM Computing Surveys, Vol.32, No. 4, pp. 422-469.
- [KG99] Kremer M. and Gryz J., 1999. "A Survey of Query optimization in Parallel Databases," Technical Report, CS-04-1999, York University, Canada.
- [KK00] Kossmann, D. and Konrad, S., 2000. Iterative Dynamic Programming: A new Class of Query Optimization Algorithm. ACM Transaction on Database Systems, Vol.25, No.1.pp.43-82

- [KYY82] Kambayashi Y, Yoshikawa M., and Yajima S., 1982. "Query processing for distributed Database using Generalized Semi-joins"
- [L81] Lorin H. 1981. "Aspects of Distributed Computer Systems," Wiley .
- [LV+94] Lanzelotte R. S. G., Valduriez P., and Zait M., and Ziane M., 1994. Industrial Strength Parallel Query Optimization : Issues and Lessons ", Information System, vol.19, no.4, pp. 311-330.
- [LW79] Lorie, R.A. and Wade, B.W. 1979. The Compilation of a High Level Data Language. IBM Research Report RJ2598
- [NSS86] Nahar S., Sahni S. and Shragowitz, 1986. "simulated annealing and combinatorial optimization," In proceeding of the 23rd Design Automata conference. pp. 293-299.
- [OV91] Ozsu M.T. and Valduriez P., 1991 " Principles of Distributed Database System", 2nd edition, Prentice Hall, Englewood Cliffs, N.J.,
- [P74] PALERMO, F.P. 1974. A Database Search Problem. In Information System COINS IV, J.T. Tou, Ed. Plenum Press, New York, NY, pp. 67-101.
- [PH+92] Pirahesh, H., Hellerstein, J., and Hasan, W. 1992. Extensible / rule based query rewrite optimization in starburst. In Proceeding of the ACM SIGMOD Conference on Management of Data (San Diego, CA, June), pp.39-48.
- [RV85] Romeo, F. and Vincentelli, S., 1985. "Probabilistic Hill Climbing Algorithms Properties and Applications, Proceedings of IEEE Conference on VLSI, pp. 393-417
- [S89] Swami, A. 1989. Optimization of large join queries : Combining heuristics and combinational techniques. In proceeding of the ACM Conference on Management of the Data(SIGMOD, 89, Portland, OR, May), ACM Press, New York, NY, pp.367-376.
- [SA+79] Selinger P. G., Astrahan M.M., Chamberlin D. D., Lorie R. A., and price T. G., 1979. "Access path selection in A Relational Database Management System", In

proceeding of the ACM SIGMOD Conference on management of Data, Boston, USA, pp.23-34.

[SG88] Swami A. and Gupta A. 1988."Optimization of large join queries," In proceeding of the ACM SIGMOD Conference on Management of Data (SIGDOM' 88, Chicago, IL , June 1-3), H. Boral and P. A. Larson Eds. ACM press, New York, NY, pp. 8-17.

[SL90] Sheth, A. and Larson, J.A., 1990, Federated database systems for managing distributed, heterogeneous, and autonomous databases, ACM Computing Surveys (CSUR), Volume 22 , Issue 3 (September 1990), pp. 183 - 236

[SMK97] Steinbrunn, M., Moerkotte, G., and Kemper, A. 1997. Heuristic and randomized optimization for the join ordering problem.VLDB J. 6, 3(Aug.), pp. 191-208.

[SY98] Shekita E. and Young H. 1998. "Iterative Dynamic Programming," IBM Technical Report.

[SYT93] Shekita, E., Young, H. and Tan, K. L. 1993. Multi optimization for symmetric Multiprocessors. In proceedings of the conference on very large databases(VLDB ,93, Dublin, Ireland, Aug.). pp. 479-492.

[VSV10] Vijay Kumar T.V., Singh, V., Verma, A.K., 2010. "Generating Distributed Query Processing Plans using Genetic Algorithm", In the proceedings of the International Conference on Data Storage and Data Engineering (DSDE 2010), Bangalore, February 9-10, 2010, pp. 173-177, 2010

[YC84] Yu C. T. and. Chang C. C, 1984. "Distributed Query Processing" ACM Computing Surveys. Volume 16, no.4. pp. 399-433.

[ZZB93] Ziane M., Zait M., and Borle-Salanet P., 1993. "Parallel Query Processing with Zigzag Trees", VLDB Journal, Vol. 2, no. 3 pp. 277-301.