Query Optimization: An Intelligent Hybrid Approach using Cuckoo and Tabu Search

Mukul Joshi, Department of Computer Science and Information System, Birla Institute of Technology & Science, Pilani Rajasthan, India
Praveen Ranjan Srivastava, Department of Computer Science and Information System, Birla Institute of Technology & Science, Pilani Rajasthan, India

ABSTRACT

Query optimization is an important aspect in designing database management systems, aimed to find an optimal query execution plan so that overall time of query execution is minimized. Multi join query ordering (MJQO) is an integral part of query optimizer. This paper aims to propose a solution for MJQO problem, which is an NP complete problem. This paper proposes a heuristic based algorithm as a solution of MJQO problem. The proposed algorithm is a combination of two basic search algorithms, cuckoo and tabu search. Simulation shows some exciting results in favour of the proposed algorithm and concludes that proposed algorithm can solve MJQO problem in less amount of time than the existing methods.

Keywords: Cuckoo Search, Levy Flight, Multi Join Query Ordering (MJQO), Query Execution Plan (QEP), Query Optimization, Tabu Search

1. INTRODUCTION

The amount of data is increasing rapidly in today’s world. A database management system (DBMS) plays a big role in storage management and maintenance of data in an efficient manner. Query language is an effective tool, which provides an interface to the user to store and access that data. In past few decades, SQL has emerged as a standard query language (VidyaBanu & Nagaveni, 2012; Rahman, 2010; Chaudhuri, 1998). Two components; query optimizer and query execution engine (Chaudhuri, 1998) do query evaluation. Query optimizer decides in which order to carry out operations in a query, using the fact that traditional relational algebra operators can be executed in a variety of Order (Badia, 2005). Many different combinations of sub queries can be used to evaluate a query. Though the combinations and cost of evaluation are different but every combination is evaluated to the same result. These combinations are called access plans or query execution plans (QEP) (Matysiak, 1995). The job of the query optimizer is to select the optimal (i.e. minimum cost) query execution plan amongst them; this problem is called query optimization problem (Matysiak, 1995). Query optimizer generates many alternative query execution plans for selecting the optimal query plan and estimates

DOI: 10.4018/jiit.2013010103
the execution cost of each of them to choose the QEP having lowest cost. Optimal query plan selected by query optimizer is forwarded to query execution engine which is responsible for execution of query. Query execution engine uses the QEP which is forwarded by query optimizer. Query optimizer is the most critical step in query evaluation; it decides the execution time and the space complexity of query. Query optimization is itself very complex and expensive; its computational complexity is determined by the number of alternatives for QEPs that must be evaluated before deciding the best query execution plan (Matysiak, 1995). The alternative planes grow exponentially with the increase in number of relations involved in a query. In past three decades this problem is addressed in many ways (Jarke & Koch, 1984; Swami & Gupta, 1988; Horng, Kao, & Liu, 1994; Matysiak, 1995; Steinbrunn, Moerkotte, & Kemper, 1997).

The join operator (Ribeiro, Ribeiro, & Lanzelotte, 1997) relates two tables through their common attributes. Evaluation of a join operation requires the matching of all tuples of relations according to their join attributes (Ribeiro, Ribeiro, & Lanzelotte, 1997). Cosar, Lim, & Srivastava (1995) shows reordering helps to improve the performance of multi query optimization algorithms. So, by reordering the join, query optimizer can lower the cost of execution of the query has join operator in between several tables. First task for a query optimizer is to decide the order of joins, which is called a multi join query optimization, or ordering problem. The multi join ordering is a combinatorial optimization problem (Dong & Liang, 2007) and if the number of input relations and joins are not fixed it is an NP hard problem (Zhou, 2007).

In traditional databases, the total number of relations in multi join queries is usually less than 10 which can be handled by dynamic programming approaches effectively (Li, Liu, Dong, & Gu, 2008). Nowadays the complexity of this problem increases due to the generation of complex multi join queries in some modern applications, such as knowledge base systems, decision support systems, expert systems, On-Line Analytical Processing (OLAP) and data mining etc. Sometimes, the generated query has more than 100 tables in a join (Li, Liu, Dong, & Gu, 2008).

Increase in the number of tables in join query also increases the number of alternative execution planes which makes the query optimizer’s task tougher. Traditional methods are not able to solve this optimization problem effectively because of the increased size of data and the large number of tables (Li, Liu, Dong, & Gu, 2008). Deterministic algorithms, greedy algorithms and heuristic algorithm based approaches have tried to approximate the optimum solution but their performance is not up to the mark (Steinbrunn, Moerkotte, & Kemper, 1997). This problem is then tried with genetic approaches and randomized approaches, such as tabu search, ant colony, bee colony etc. which gave better performance (Kadkhodaei & Mahmoudi, 2011) but betterment in performance with improvement in quality of solution is still required. This paper proposes a new algorithm; which uses cuckoo search (Yang & Deb, 2009) algorithm combined with tabu search (Glover, 1989) algorithm to find out a better solution of this problem. The main objective of this paper is to provide an effective solution for MJQO problem. MJQO is an integrated part of query optimizer. The query optimizer generates a QEP which takes lesser possible time to execute. If we succeed in our objective to provide an effective solution for MJQO than it will deliver two things: first, if it is able to produce good quality of solution than the execution time of QEP will reduce; and second, if it takes less time to calculate this solution, the overall evaluation time of multi join query minimizes. This will be helpful for those applications where multi join queries are heavily used (i.e. OLAP, data warehouse, data mining, etc.).

The organization of this paper is as follow. Next Section reviews the Multi join ordering problem. The related work is explored in Section 3. In contrast Section 4 presents the proposed algorithm and consequently Section 5 covers the experimental results and comparison of differ-
ent algorithms. Finally section 6 concludes the paper and also covers suggestions for the future.

2. PROBLEM STATEMENT

When a user inputs a query, it is first analyzed by parser for syntax errors, if there is no error it is then transformed into standard format i.e. a query graph (Kadkhodaei & Mahmoudi, 2011). Next, query optimizer takes this query graph as input and prepares different query execution plans for that query and selects an optimal query execution plan amongst them. This optimal query plan is forwarded to query execution engine which evaluates it and returns the query result. This overall process is shown in Figure 1.

QEP selected by optimizer (i.e. quality of optimal solution) should be of minimum cost because its optimality decides the time taken by execution engine to execute this query. Time taken by the query optimizer to choose the optimal QEP should be minimal because it is also a part of overall query execution time. There are many steps involved in query optimization, one of them is multi join ordering problem (Kadkhodaei & Mahmoudi, 2011). If query consists of multiple joins among different relations and we reorder the joins then the resultant query (i.e. of different join order) has different cost but result will be the same. So, by reordering we can find out the join order which can be evaluated in minimal time with the same result. Finding such order among all possible orders is called multiple join query optimization problem (Kadkhodaei & Mahmoudi, 2011).

Multi join query optimization process consists of two steps; logical optimization and physical optimization (Dong & Liang, 2007). Input query is converted from high level declarative language to query graph which is as input for logical query optimizer. In query graph (Steinbrunn, Moerkotte & Kemper, 1997), base relations are represented by node. Two nodes are connected by an edge which represents the common attributes among the two relations represented by these two nodes. Two nodes are connected in query graph if and only if these two edges share some common attributes. So, it can be concluded that join operation is possible between these two. This ‘Avoid Cartesian Product’ constraint reduces the search space (Li, Liu, Dong & Gu, 2008), and is discussed later in this section. Logical optimizer generates join processing tree for a query graph (Dong & Liang, 2007). Join processing tree represents the order of execution of join operation. In join processing tree, internal nodes represent the result of join operation of its left and right child. Where the leaf nodes represent the base relations. Join processing trees are shown in Figure 2. The join operator follows two algebraic rules; associative rule and commutative rule. So there can be more than one equivalent
join processing tree possible for a query graph (Dong & Liang, 2007).

For each join processing tree physical optimizer produces several operator trees by selecting a physical operator for a join operator (Dong & Liang, 2007). In operator trees internal node is a physical operator i.e. an algorithm executes the join operator. Finally, the cost of each operator tree is estimated and the operator tree with lowest cost is selected as an optimal QEP. If it is assumed that all join operations are implemented by same physical method, than multi join optimization problem is simplified as finding the optimal join order which makes the cost lowest (Li, Liu, Dong, & Gu, 2008).

The solution space of the MJQO problem is the set of all possible join processing trees (i.e. Query Execution Plans) for a query graph. The goal is to find out the minimal cost join ordering tree in the mentioned solution space. The optimization of join ordering query problem needs (Dong & Liang, 2007):

1. Solution sub space. The above mentioned solution space is very large and can be reduced to small search space. Search space reduction is described after in this section.
2. A cost model that can assess the cost of generated QEPs accurately.
3. Search algorithm, which can efficiently and effectively search the optimal QEP amongst all possible QEPs.

For any query graph there can be three possible join processing trees viz. left deep tree, right deep tree and bushy tree (Kadkhodaei & Mahmoudi, 2011). Five relations called R1, R2, R3, R4 and R5 are in a multiple join query Q. Figure 2 shows three possible join processing trees; a left deep tree (a), a bushy tree (b) and a right deep tree (c) of query Q. It categorized the search space further into three subspaces. The left deep tree can be considered as the subspace for MJQO problem. Left join processing tree can take the full advantage of index (Li, Liu, Dong & Gu, 2008). Generally, with n number of relations, total \( \binom{2(n-1)}{n-1} (n-1)! \) join processing trees are possible out of which only \( n! \) deep trees are possible (Zhou, 2007).

The search space can be reduced further by using ‘Avoid Cartesian Product’ constraint (Li, Liu, Dong, & Gu, 2008). As mentioned before, in a query graph an edge between two nodes (relations) is possible if and only if they have some common attributes i.e. join operation is possible between them. If an edge is absent between two nodes in a query graph, then we can remove those join processing trees from the search space which shows these two nodes are the children of the same node. The multiple join query Q which includes five relations; R1, R2, R3, R4, R5 and attributes’ association (According to database catalog) among these is represented by query graph shown in Figure 3.
Two possible left deep join processing trees for query Q, T1 and T2 are shown in Figure 4. Tree T1 is valid according to query graph shown in Figure 3 while T2 is not a valid join processing tree due to the “Avoid Cartesian Product” constraint. So, Tree T2 can be removed from the search space of query Q.

For estimating the cost of QEPs, the proposed approach uses the simple cost model, which was used by Dong & Liang (2007); Li, Liu, Dong & Gu (2008); Kadkhodaei & Mahmoudi (2011) and Alamery, Faraahi, Javadi & Nourossana (2010). Proposed approach can also be applicable to other cost models.

The simple cost model assumes that the attribute values are uniformly distributed and the cost of the join processing tree (i.e. QEP) is estimated by adding the size of the relations which are obtained as intermediate results. The required parameters are: \( n(r) \): number of tuples in relation \( r \); \( V(A, r) \): numbers of distinct values of attribute \( A \) in relation \( r \). The formula to calculate the cost of a join processing tree is (Dong & Liang, 2007).

\[
\text{Cost} = \sum_{i=1}^{n-1} n(t_i) \tag{1}
\]

Here \( n(t_i) \) is the number of tuples in relation \( t_i \) and \( t_i \) is an internal node of the join processing tree which represents an intermediate relation obtained after joining two relations that are the immediate children of that node in processing tree. So estimated cost of QEP is the submission of size of all intermediate results (i.e. the total extra space needed to calculate the final result).

If relations, \( r \) and \( s \) are left and right children of an inner node \( t \) and the group of common attributes in relations \( r \) and \( s \) is represented by \( C \), then the size of inner node (i.e. resultant relation of join operation between \( r \) and \( s \)) is (Dong & Liang, 2007).

\[
n(t) = \frac{n(r) \times n(s)}{\prod_{c \in C} \max(V(c, r), V(c, s))} \tag{2}
\]
\( n(t) \) is the size of the resultant relation of join operation of two relations \( r \) and \( s \); which is equal to the number of rows having similar values of attribute common in both relations, \( r \) and \( s \). It is obtained by dividing the Cartesian product of relations \( r \) and \( s \) by number of rows having distinct values of common attribute. In equation (2) \( n(r) \times n(s) \) is the Cartesian product of relation \( r \) and \( s \), which represents all combinations over common attributes. 

\[
\prod_{c_j \in c} \max(V(c_j, r), V(c_j, s))
\]

Calculates multiplication of maximum distinct values of each common attribute \( (c_j) \) in \( r \) and \( s \). Division of these two gives the total number of rows in the resultant relation of join operation between \( r \) and \( s \).

The value of function \( V(A, t) \) which is used in equation (2) can be calculated by equation (3) (Dong & Liang, 2007).

\[
V(A, t) = \begin{cases} 
V(A, r) & A \in r - s \\
V(A, s) & A \in s - r \\
\min\{V(A, r), V(A, s)\} & A \in r, A \in s 
\end{cases}
\]

\( V(A, t) \) is the number of distinct values of attributes \( A \) that appear in the relation \( t \).

This model mainly considers the intermediate space needed to execute a query. In multi join queries intermediate space is very important because it’s the space that decides the time to process that intermediate result. If number of rows in intermediate resultant relation are more we require more time to evaluate this result in next step but if its size is small require less time. The intermediate space is directly proportional to execution time of query. So if we can estimate the size of intermediate results, we can easily select the better QEP.

Equation (2) and (3) are used to compute the size (number of tuples) and number of distinct values for attributes of the inner node (intermediate resultant relation). The cost of join processing tree can be calculated by summing the cost of all intermediate nodes by using equation (1). So the cost estimating of a join tree consumes much computation time.

3. RELATED WORK

Many different approaches have been tried to solve MJQO problem in optimal time. Some of solutions are studied in Steinbrunn, Moerkotte & Kemper (1997) which is classifies it into four categories; deterministic algorithms, randomized algorithms, genetic algorithms and hybrid algorithms.

1. **Deterministic Algorithms**: Every algorithm that comes under this class computes step by step solution in deterministic way either by exhaustive search or by using some heuristics. These algorithm lags in the quality of solution and exhaustive search takes more time. For improving the quality of solution heuristics should be accurate. These algorithms calculate the solution step by step using local search strategy which degrades the quality of solution in some cases.

2. **Randomized Algorithms**: Algorithms come under this class have completely different approaches. Solution space is represented as a graph. Solutions in solution space are represented by the nodes of graph and if we can transform from one solution to another by performing exactly one move then these two solutions are connected by an edge. Each of the algorithms performs a random walk according to some predefined rules. This walk terminates when no more applicable moves exist or time limit is exceeded. The result of the algorithm (i.e. optimal solution) is the minimal solution encountered so far. There is no guarantee of obtaining minimal solution. Sometimes probability of getting minimal solution decreases due to inappropriate rules for random walk. These kinds of algorithms depend highly on the problem modeling.
3. **Genetic Algorithm:** These kinds of algorithms are inspired by biological evaluation. These algorithms use randomization with genetic evolution. The main idea behind this approach is to start with a random population and generate offspring by random crossover and mutation. The best member of population survives for the subsequent iterations. The quality of solution is decided by cost function. These algorithms terminate when no further improvement is there. The best member existing in the last generation is the solution. Though these algorithms succeed on the basis of speed, in some cases the result of these solution is very far from optimal solution.

4. **Intelligent Algorithm:** Also called Meta-heuristic approach. These approaches are inspired by nature. Ant colony optimization, bee colony optimization, intelligent water drops algorithms (IWD), cuckoo search algorithm are some examples of intelligent approaches. A lot of work has been done in the area of metaheuristics (Trafali & Suat, 2002) in past few years that also affect MJQO solution approaches. Some of best solutions have been proposed, which are based on metaheuristics approaches viz. ant colony algorithm (Li, Liu, Dong & Gu, 2008), bee colony algorithm (Alamery, Faraahi, Javadi & Nourossana, 2010). The time taken by these approaches to calculate the solution is lowest among all known approaches.

5. **Hybrid Algorithms:** Hybrid approaches are the combination of two or more approaches. The basic idea behind hybrid approach is, one algorithm can never be satisfy all the constraint of an optimization problem so we can use more than one algorithm to satisfy all the constraints and overlap each other’s shortcomings. Many hybrid approaches has proposed for MJQO problem. A new hybrid approach (Kadkhodaei & Mahmoudi, 2011) uses genetic algorithm and ant colony algorithms; does not improve the execution time but improves the quality of solution. This paper also proposing the technique which is the combination of two techniques; cuckoo search (an intelligent approach) and tabu search.

All five shows different capabilities to solve this optimization task. Computational speed of heuristic algorithms is very high, but these algorithms provide evaluation plans, that are far from the optimum in many cases (Steinbrunn, Moerkotte & Kemper, 1997). Though randomized and genetic algorithms provides much better results but in longer running time (Steinbrunn, Moerkotte & Kemper, 1997).

Ibaraki & Kameda (1984) proposed the ‘IK-KBZ’ algorithm (‘A’ algorithm) to solve this problem and comes under the category of deterministic algorithms. It is the initial solution for MJQO problem. Swami & Iyer (1993) has proposed the ‘AB’ algorithm, a solution which combined the randomization with IK-KBZ (Ibaraki & Kameda, 1984) algorithm. It also includes a relatively inexpensive post processing step that allows cartesian products and uses a neighborhood search heuristic. This solution has polynomial space and time complexity. The main problem with this approach is its weak cost model. Its cost model is insensitive to the row size. Horng, Kao & Liu (1994) proposed a genetic algorithm based solution for query optimization problem. This solution uses greedy approach for crossover function which restricted the performance of this solution. Solution proposed by Matysiak (1995) adopted tabu search approach to optimize join queries but its performance varies with size of the query (for some queries it become worse). Ribeiro, Ribeiro & Lanzelotte (1997) also proposed tabu search based solution for distributed databases. It keeps track of hole set of visited solution, so needs hash based data structure to save the history of the search. This strategy is effective for small and medium sized queries but its performance drastically decreases for large size queries. Khan, Ponusamy, McLeod, & Shahabi (2003) developed an adaptive probe-based optimization technique in the context of an Internet-based distributed
database environment. This technique is helpful for optimizing the query response time in an environment where server performance and network traffic is unpredictable. This may result the selection of an expensive query plan when used in a static environment.

Dong & Liang (2007) and Bayir, Toroslu & Cosar (2007) proposed genetic based solutions. Dong & Liang (2007) uses genetic algorithm as randomized optimization algorithm, so there efficiency depends on the nature of the problem (i.e. query model). Bayir, Toroslu & Cosar (2007) presents evolutionary techniques for solving MJQO problem. It is better for small size of queries. The cost model used for this solution is not very effective while cost model used in the approach proposed by Dong & Liang (2007) is sensitive to the row size and intermediate records.

Zhou (2007) and Kadkhodaei & Mahmoudi (2011) proposed approaches are based on hybrid approach. Zhou (2007) used heuristics with genetic algorithm. Though it is very effective solution but very difficult to implement. It uses local search which affects quality of solution. Cost model used for this solution is mainly suited for memory resident databases. There is also a scope of improvement in objective function used in this approach. Kadkhodaei & Mahmoudi (2011) combines ant colony optimization with genetic algorithm. The advantage of genetic algorithm is escaping from local minimums, but convergence speed toward optimum answer is low; in contrast, ant colony algorithm is of high convergence speed, but its disadvantage is blockage in most part of algorithm in local minimums (comparing to genetic algorithm). It is very difficult to set the required parameters of this method.

Ant colony algorithm (Li, Liu, Dong & Gu, 2008) and bee colony algorithm (Alamery, Faraahi, Javadi & Nouroosani, 2010) based approaches are the best known approaches till now. The main focus of these approaches is on the convergence time and not the quality of solution. Ant colony based approach tried to improve the quality of solution but little success has been found while bee colony based approach didn’t address the quality of solution at all. On the other hand the hybrid approach (Kadkhodaei & Mahmoudi, 2011), a combination of ant colony and genetic algorithm improved the quality of solution but was unable to improve the convergence time. So there is a need of an algorithm which can provide improvement in both directions, convergence time as well as quality of solution.

This paper proposes a new solution that is the combination of two approaches cuckoo search algorithm and tabu search algorithm. This approach reduces the computational time to calculate the solution as well as provides the assurance to improve the quality of solution.

3.1. Cuckoo Search
(Yang & Deb, 2009)

The Cuckoo search algorithm, which is a population based scholastic global search algorithm, discovered by Yang and Deb (Yang & Deb, 2009) is inspired by breeding strategy of cuckoo species (Civicioglu & Besdok, 2011). This algorithm is based on following three assumptions;

1. Each cuckoo lays one egg at a time, and chooses a nest randomly to dump this egg;
2. The best nests with high quality of eggs (solutions) will carry over to the next generations.

The number of available host nests is fixed, and a host can discover an egg dumped by cuckoo with a probability pa ∈ [0, 1]. Then host bird can either throw the discovered egg away or abandon that nest so as to build a completely new nest.

For simplicity, last assumption can be approximated by a fraction pa of the n nests being replaced by new nests (with new random solutions at new locations). Levy fly is preferred to generate the new solution (i.e. cuckoo egg) for better results (Yang & Deb, 2009). Levy fly provides a way to perform a random walk
where random step length is drowning from levy distribution. Levy flight is inspired by flight characteristic behavior of many birds and insects, leading to scale free search pattern. These kinds of behaviors have been applied to Srivastava, Singh, Kumhar & Jain(2012); Srivastava, Reddy, Reddy, Ramaraju & Nath(2012) and in many more optimal searches and show its promising capability.

3.2. Tabu Search (Glover, 1989)

Tabu search, developed by Fred Glover in 1970 (Glover, 1989), is the search strategy where memory and search history is its integrated part. It is an intensive local search algorithm, which uses the memory (search history) to avoid the potential cycling of local optimal solutions. A tabu list is maintained, which contains all recently visited solutions to avoid moving again and again to these solutions. This strategy saves time by avoiding the repetition of previous moves and prevents it from being stuck at local minima. Ferm & Zomaya (2010), Hirsch (2013) and others have used this behavior of tabu search and show optimistic results.

MJQO is an optimization problem which is modeled as a search algorithm where the optimal QEP has been searched from all possible QEPs. Heuristic techniques are effective techniques to solve such kind of applications. Cuckoo Search technique is superior to all other existing heuristic techniques (Yang & Deb, 2009). Proposed solution for MJQO problem uses cuckoo search algorithm to find optimal QEP to improve the result of previous existing solutions of this problem. Cuckoo search improves the running time. In some cases cuckoo search algorithm gets stuck at local minima and is not able to give best possible optimal solution. Proposed approach uses tabu search to identify such situations and levy flight to come out of this situation. It improves the quality of solution. As mentioned above, both parameters, execution speed as well as quality of solution are equally important in MJQO and should be improved. Proposed solution addresses both the parameters.

4. PROPOSED APPROACH

Cuckoo search is based on the breeding strategy of cuckoo species. In this approach initially some eggs are there. After validating the eggs’ quality some eggs are abandoned and some new eggs will replace these abandoned eggs whose quality is better than those abandoned eggs. By repeating these steps final outcome will be the best quality of eggs. In proposed algorithm each possible QEP acts as an egg. Initially some random QEP (initial cuckoo eggs) are generated and then follows cuckoo’s breeding behavior. It will generate new QEP (i.e. similar to laying a new egg) and compare the newly generated QEP with any randomly selected QEP from initial set. If the quality of newly generated QEP is better to randomly selected QEP then randomly selected QEP will be abandoned and newly generated will replace it. If we repeat this cuckoo behavior after some iterations we will get a set of best QEP among all possible QEPs of input query.

According to proposed approach when a query graph comes to query optimizer, it will randomly generate the set of initial QEPs (i.e. cuckoo eggs) called S and after it, a new QEP called e will be generated using a generator function. Permutation function P1, used as generator function. P1 is a permutation function which generates different permutations of relations in a query graph by reordering the position of relations in the query graph. Main concern in newly generated QEP by P1 is that it should fulfill all the constraints (i.e. Avoid Cartesian Product etc.). After generating a new QEP, one QEP say r is randomly selected from S. We compare the cost of e and r (i.e. similar to comparing eggs), if the cost of e is less than the cost of r; then r is replaced with e (i.e. r is abandoned). After this step minimal cost QEP is selected from set S. If the minimal solution is optimal than this procedure is stopped here otherwise it will repeat the steps of generating the new QEPs until the maximum number of iterations is completed. Flow diagram of this approach is shown in Figure 5.
This approach works fine but there would be some situations where it could be stuck in local minima. To identify such situations tabu search approach is added with proposed solution; it maintains a tabu list of recently generated, n number of QEPs and a tabu variable which takes the count of n consecutive generation of new QEPs which are not able to replace
the randomly selected QEP from set S. If in some condition it gets in local minima then with the help of tabu variable and tabu list, it can be easily identified. After identifying these situations, levy flight is used to come out from these situations i.e. local minima. Levy flight is nothing but it’s a process of random generation of new QEP, here the permutation function P2 is used for it. P2 is same as P1 but the only difference is it will generate permutation after more position shuffles meaning degree of changes in position is more. Proposed solution with tabu search and levy flight is shown in Figure 6.

Pseudo code of this approach is given in Figure 7. In this algorithm the choice of initial eggs n may be different for different cases but it should not be very large. Large value of n will increase the time complexity because if n is large we need more time to search the optimal solution in S i.e. the set of n solutions. The value of n must be taken in such a way that searching in S will not become time consuming. We are using two permutation functions P1 and P2 for generating new join processing tree (i.e. QEP). P1 permutation generates a new tree from an existing tree by exchanging two nodes of the tree while in P2 the more than two nodes are exchanged. Number of exchanges in P1 and P2 can be varied and depends on implementation. The only condition for it is that P1 and P2 should not become complex and should be as global as possible(means able to exchange every node exists in processing tree). Size of tabu list, n1 and maximum value of tabu variable, n2 also varies and the variation in n1 and n2 depends on implementation.

A simulation described in next section, has been done to compare this proposed approach with other existing approaches.

5. EXPERIMENTAL RESULTS

To illustrate the effect of this solution an experiment has been performed with intel i3 processor+2 GB RAM+windows7 operating system. We generated database of 50 relations where each relation has relation cardinality in [1000, 10000]. The relation cardinality is the number of tuples in a relation. Then a test data has been generated which consists of 100 queries. It is categorized into ten sets of queries of different sizes (i.e. number of relations in a query is of 5, 10, 15, 20, 25, 30, 35, 40, 45, 50). Every query made with an independent set of relations. These relations have been selected randomly with probability 0.2 form the set of 50 relations. Then every set of queries is executed twice with proposed solution and Ant colony solution (Li, Liu, Dong & Gu, 2008).

As discussed in section 3, ant colony algorithm for MJQO (Li, Liu, Dong & Gu, 2008) is the best existing approach for solving MJQO problem (Kadkhodaei & Mahmoudi, 2011). Although hybrid approach (Kadkhodaei & Mahmoudi, 2011)using Ant colony and genetic algorithm, has been proposed as an improvement over ant colony algorithm approach, but this hybrid approach succeeded only to improve the quality of solution while execution time became slower. After this, Bee colony approach (Alamery, Farahai, Javadi & Nourossana, 2010) has been proposed but it provides slight improvement in execution time while quality of solution is not addressed by it. The proposed approach improves both and to analyze it we compared this with ant colony algorithm. If it will show an effective improvement over ant colony approach, we can conclude that the proposed algorithm would become the best solution among all existing solutions.

The values of parameters used in Ant Colony solution (Li, Liu, Dong & Gu, 2008) are as follow: α = 1, β = 3, q0 = 0.02, and ρ = 0.9, ant number m = 10. And parameters used in proposed approach are: number of initial eggs n = 3, size of tabu list n1 = 4 and n2 = 2. These parameters can vary according to the implementation. Here values are taken for the sake of simplicity.

In Figure 8, X-axis represents the number of relations corresponding to a particular query and Y-axis defines the values as ratio of query execution cost i.e. (query execution cost with
Figure 6. Proposed approach with tabu search

- Set of initial eggs (S) randomly generate QEPs
- Set tabu variable \( T_b = 0 \) and tabu list = empty

- Generate a new egg (e) i.e. a new QEP

- If query Plan is valid according to query

- Select an egg (QEP) randomly from set S say (r)

- If cost(e) < cost(r)
  - Replace r with e and add e to tabu list
  - Tb++

- Generate QEP(e) using Levy flight and add to set S

- If (Tb = oo) or tabu list has n similar
  - Select QEP of minimal cost say m

- If (m is minimal) or (maximum iterations)
  - Exit with result m as optimal QEP

- If no
Ant colony solution)/(query execution cost with proposed approach). Query execution cost means time taken by query execution engine to execute a QEP. This QEP is the output of query optimizer, which comes after applying the optimization algorithm. This means if optimization algorithm is more effective than query execution cost should be lesser i.e. quality of solution is better. If proposed solution is better than ant colony than QEP which is the resultant of proposed solution should have less execution cost and the ratio of query execution cost should always be more than 1. The graph shown in Figure 8 clearly shows that the ratio of query execution cost is always greater than 1. This shows that the quality of solution of proposed approach is always better than the quality of solution of ant colony optimization algorithm.

In Figure 9 the number of relations corresponding to a particular query Q are taken in X-axis and time to generate the optimal solution, minimal cost join order tree (i.e. QEP) is taken in Y-axis. This graph clearly shows that proposed algorithm takes less time to calculate the optimal join processing tree than that of Ant Colony solution.

After analyzing the results of experiment this can be concluded that the proposed approach in this paper is more effective and efficient than Ant colony solution which is the best known solution till now. Proposed approach calculates optimal solution faster than best known solution and also provides better quality of solution. Proposed approach is able to improve both the parameters speed and quality which are addressed before.
6. CONCLUSION

MJQO is an interesting research problem in the field of database. Experimental results show that the proposed approach finds optimal solution more effectively than ant colony algorithm, which has fastest convergence rate among all known solutions for MJQO. So we can conclude that proposed algorithm is better than all existing solutions of this problem. This approach can be helpful in those applications where multi join queries are frequently used. It reduces the response time of query processor in such applications.

The proposed solution is a combination of two algorithms cuckoo search and tabu search. Cuckoo search has high convergence speed towards the solution but it has a disadvantage of blocking in local minimum which is taken care of tabu search and levy flight. The proposed solution shows that the combination of two basic algorithms is able to overcome the weakness of each other. The success of this algorithm has broadened the scope of cuckoo search algorithm and hybrid algorithms. In future, we can use such combinations for other research problems too.

This paper opens up two scopes for future. First, the proposed approach is only applicable for normal joins; it can be modified for other kind of join operations and other operations in multiple query problems. Second, it also opens the scope of hybrid and intelligent approaches. There are many optimization problems that can

---

Figure 8. Ratio of query execution cost

![Figure 8](image)

Figure 9. Comparisons of execution time

![Figure 9](image)
be solved more effectively by the use of intelligent algorithms.

REFERENCES


Mukul Joshi is pursuing Master of Engineering (Computer Science) from Computer Science and Information Systems Department at the Birla Institute of Technology and Science (BITS) Pilani, Pilani Campus Rajasthan, India. His research interest is Combinatorial Algorithms, Randomized Algorithms and Distributed Operating Systems.

Praveen Ranjan Srivastava is working with the Software Engineering and Testing Research Group in the Computer Science and Information Systems Department at the Birla Institute of Technology and Science (BITS) Pilani, Pilani Campus Rajasthan, India. Dr Praveen Ranjan Srivastava has received BSc in Mathematics and Master of Computer Application (MCA) from VBSPU, Jaunpur. His M.Tech Specialization in Software Engineering from MNNIT, Allahabad. He earned his PhD from the Computer Science Department at BITS Pilani, Pilani Campus. He is currently doing research in the area of software Testing and advanced soft computing techniques. His research areas are software testing, quality assurance, quality attributes ranking, testing effort, software release, test data generation, Test effort estimation, agent-oriented software testing, and metaheuristic approaches. He has published more than 100 research papers in various leading international journals and conferences in the area of software engineering and testing. He has been actively involved in reviewing various research papers submitted in his field to different leading journals and various international and national level conferences. He received various funds from different agencies like Microsoft, IBM, Google, DST, CSIR, etc.