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
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-
Fuzzy Rough Support Vector Machine for Data Classification
[www.igi-global.com/article/fuzzy-rough-support-vector-machine-for-data-classification/151534?camid=4v1a](www.igi-global.com/article/fuzzy-rough-support-vector-machine-for-data-classification/151534?camid=4v1a)

Facial Expression Recognition for HCI Applications
[www.igi-global.com/chapter/facial-expression-recognition-hci-applications/10312?camid=4v1a](www.igi-global.com/chapter/facial-expression-recognition-hci-applications/10312?camid=4v1a)