Automata Theory based Approach to the Join Ordering Problem in Relational Database Systems

Miguel Rodríguez[1], Daladier Jabba[1], Elias Niño[1], Carlos Ardila[1] and Yi-Cheng Tu[2]
[1]Department of Systems Engineering, Universidad del Norte, KM5 via Puerto Colombia, Barranquilla, Colombia
[2]Department of Computer Science and Engineering, University of South Florida, Tampa, U.S.A.

Keywords: Automata Theory, Query Optimization, Join Ordering Problem.

Abstract: The join query optimization problem has been widely addressed in relational database management systems (RDBMS). The problem consists of finding a join order that minimizes the time required to execute a query. Many strategies have been implemented to solve this problem including deterministic algorithms, randomized algorithms, meta-heuristic algorithms and hybrid approaches. Such methodologies deeply depend on the correct configuration of various input parameters. In this paper, a meta-heuristic approach based on the automata theory will be adapted to solve the join-ordering problem. The proposed method requires a single input parameter that facilitates its usage respect to those previously described in the literature. The algorithm was embedded into PostgreSQL and compared with the genetic competitor using the most resent TPC-DS benchmark. The proposed method is supported by experimental results achieving up to 30% faster response time than GEQO in different queries.

1 INTRODUCTION

Since the early days of RDBMS the problem of finding a join order to minimize the execution time of a query has been approached. (Chaudhuri, 1998) defined large join queries as relational algebra queries with *N* join operations involving *N+1* relations when *N* is greater or equals to 10. Consecutively the Large Join Query Optimization Problem (LJQOP) was formally addressed as finding a Query Execution Plan (QEP) with a minimum cost for a large join query.

The LJQOP have been widely addressed and many methods have been developed to solve it. Randomized algorithms such as iterative improvement and simulated annealing, evolutionary algorithms such as genetic algorithms and Meta heuristics such as ant colony optimization are some common strategies used in the solution of the problem.

The solution space of a LJQOP consists of all query trees that answers the query. There are three types of query trees that can result from the solution space: left deep, bushy and right deep. An extended discussion about types of query trees is given in (Ioannidis and Kang, 1991).

The construction of a LJQOP solution space is theoretically possible for a small number of relations. When *N* increases substantially, finding the optimal join order is considered an NP-hard problem and thus deterministic algorithms cannot find a solution easily.

Systems holding workloads from applications such as decision support systems and business intelligence require the ability of joining more than 10 relations easily. In this paper, a Meta heuristic approach based on the automata theory that has been effectively used in the solution of the Traveling Salesman Problem (TSP) will be presented and its application in the solution of the join ordering problem will be discussed. Finally the automata based query optimizer proposed in this work will be tested using the most recent decision support benchmark TPC-DS.

The remaining parts of this work will be distributed as follows. Previous work on solving the LJQOP is discussed in section two. The proposed methodology will be explained in section three. The experimental design and setup used to test the algorithm is going to be exposed in section four. A discussion about the results obtained by the algorithm and a comparison analysis between the proposed method and the PostgreSQL genetic optimizer module is showed in section five. Finally

Rodríguez M., Jabba D., Niño E., Ardila C. and Tu Y..
Automata Theory based Approach to the Join Ordering Problem in Relational Database Systems.
DOI: 10.5220/0004433802570265
In *Proceedings of the 2nd International Conference on Data Technologies and Applications* (DATA-2013), pages 257-265
ISBN: 978-989-8565-67-9
Copyright © 2013 SCITEPRESS (Science and Technology Publications, Lda.)

conclusions and future work are presented.

2 RELATED WORK

The join ordering problem has been approached in different ways among the years. A literature review presented in (Steinbrunn et al., 1997) provides detailed information on different approaches to the solution of the problem and classifies them in four groups. The first one corresponds to deterministic algorithms such as dynamic programming and minimum selectivity algorithm. The second group, randomized algorithms, includes simulated annealing, iterative improvement, two-phase optimization and random sampling. The third group consists of genetic algorithms, which encode the solutions and then uses selection, crossover and mutation algorithms. Finally the fourth group is compound of hybrid methods.

Three of the most popular approximate solutions to the join-ordering problem are simulated annealing, genetic algorithms and ant colony optimization.

2.1 Simulated Annealing

The annealing process in physics consists of obtaining low energy states of a solid element being heated. Simulated Annealing takes advantage of the Metropolis algorithm used to study equilibrium properties in the microscopically analysis of solids. Specifically the Metropolis algorithm generates a sequence of states for a solid object. Given an element in state i with energy E_i a new element in state j is produced, if the difference between energies is below cero, the new state is automatically accepted; otherwise its acceptance will depend on certain probability based on the temperature the system is exposed to and a physic constant known as Boltzmann constant k_b. Similarly, the simulated annealing algorithm constructs solutions to combinatorial problems linking solution-generation alternatives and an acceptance criterion. The states of the system can be matched to solutions of the combinatorial problem, and in the same way the cost function of the optimization problem can be seen as the energy cost of the annealing system. Therefore the simulated annealing algorithm starts exposing the system to high temperatures and thus accepting solutions that do not improve previous solutions. By terms of a cooling factor, the temperature starts lowering until it reaches zero where solutions that do not improve its parents are not accepted.

The calculation of the acceptance probability of the simulated annealing algorithm is adopted from the Metropolis algorithm and corresponds to the following equation.

$$P_T = \begin{cases} 1, & if\ f(i) \leq f(j) \\ e^{\frac{f(i)-f(j)}{T}}, & if\ f(j) > f(i) \end{cases} \tag{2.1}$$

Different implementations of the simulated annealing algorithm have been used to solve the join ordering problem using different cooling schemas, initial solutions, and solution generation mechanisms.

The implementation in (Ioannidis and Wong, 1987) proposed the use of simulated annealing to solve the recursive query optimization problem. The initial state S_0 was chosen using semi-naïve evaluation methods and the initial temperature T_0 was chosen as twice the cost of the initial state. The termination criterion of the algorithm is composed of two parts: the temperature must be below 1 and the final state must remain the same for four consecutive stages. The generation mechanism is based on a transition probability matrix $R: S_A \times S_A \rightarrow [0,1]$ where each neighbor of the current state has the same probability to be chosen as the next state.

$$R(s,s') = \begin{cases} \frac{1}{|N_A(s)|} & if\ s' \in N_A(s) \\ 0 & otherwise \end{cases} \tag{2.2}$$

Finally authors suggest the use of two different cooling schedules in their implementation. They propose the use of the following equation to control the temperature of the system.

$$T_{new} = \alpha(T_{old})T_{old} \tag{2.3}$$

The function α returns values between 0 and 1. The first strategy proposed consists of keeping α a constant value of 0.95 and the second one consists of modifying the value of α according to Table 1.

Table 1: Factor to reduce temperature.

$T_0/T \leq$	α
2	0.80
4	0.85
8	0.90
∞	0.95

A second approach to query optimization by simulated annealing is proposed in (Swami and Gupta, 1988) where two implementations of simulated annealing are compared to several other algorithms including perturbation walk, Quasi-random sampling, local optimization and iterative improvement. The proposed simulated annealing implementation uses an interesting generation mechanism that combines two different strategies. The first strategy is swapping which consists of selecting two positions in the vector and interchanges its values and the second strategy is 3-cycle, which consists of randomly selecting 3 elements of the actual state and shift them one position to the right in a circle. In order to select which strategy is used to generate the new solution at a given iteration, variable $\alpha \in [0,1]$ that represents the frequency of swap selection and thus $1 - \alpha$ that represents 3-cycle selection is used.

2.2 Genetic Algorithms

The author of (Holland, 1992) explains that *"Most organisms evolve by means of two primary process: natural selection and sexual reproduction. The first determines which members of the population survive to reproduce, and the second ensures mixing and recombination among the genes of their offspring"*. Genetic algorithms use the same procedure to seek an optimal solution to an optimization problem by selecting the most fitted solutions of the problem and combining them to create new generations.

A genetic algorithm heavily depends on the performance of its three basic operations: selection, recombination and mutation.

In general the selection scheme describes how to extract individuals from the current generation to create new elements to be evaluated in the next generation by creating a mating pool. Consequently the elements selected in the present generation must be "good" enough to be the parents of the new generation.

The crossover operation is what makes genetic algorithms different from other randomized methods. Based on the natural reproduction process where parts of the genes of both parents are combined to form new individuals, the crossover function uses two individuals selected from the mating pool and combines them to create new individuals. Several methods to crossover have been designed and some of the most popular are one-point crossover, two-point crossover, cycle crossover and uniform crossover.

The mutation operation adds an additional change to the new individuals of the population to prevent the generation of uniform populations and getting trapped in local optima. The mutation operation in binary encoded genetic algorithms can be easily implemented by selecting a random bit from the encoded string and change its value by using the negation operation. Even though the mutation operation is essential for the genetic algorithm to work properly, it must be used carefully.

Genetic algorithms have also been used to solve the query optimization problem as alternative to randomized algorithms. The genetic algorithm implemented by the authors of (Bennett et al., 1991) is the first known genetic algorithm used to approach the query optimization problem. The authors adapted a genetic algorithm used to solve the assembly line balancing problem focusing on finding an appropriate encoding schema and crossover operation to solve the query optimization problem. The author of (Muntes-Mulero et al., 2006) proposed the Carquinyoli Genetic Optimizer (CGO) which uses a tree fashioned codification for the algorithm to represent solutions, the crossover operation randomly selects two members of the current population and examines each tree's operations and stores them in a list, then a sub tree from each parent is selected and an offspring is generated by combining a sub tree from one parent and the ordered list of operations from the other, the same procedure is applied to the other sub tree and operation list. Five different mutation strategies were used, swap, change scan, change join, join reordering and random sub tree. The selection strategy used by GCO is a simple elitist algorithm.

Finally the commercial database system PostgreSQL is equipped with GEQO, a genetic optimizer that activates when the number of tables involved in a query exceeds 10. GEQO is based on the steady state genetic algorithm GENITOR that presents two main differences compared to traditional genetic algorithms, the explicit use of ranking and the genotype reproduction in an individual basis.

2.3 Ant Colony Optimization

The optimization method based on ant colonies consists of three procedures: ConstructAntsSolution, UpdatePheromones and DeamonActions. The first method manages the construction of solutions by single ants using pheromone trails and heuristic information. The second method uses the solution constructed by the ant and update pheromone trail accordingly increasing the amount of pheromones or reducing the amount of pheromones due evaporation. The third method includes actions that cannot be performed by single ants like local optimization procedures or additional pheromone increases.

Lately new algorithms to solve the query optimization problem have been proposed based on the ACO theory. Different approaches to query optimization using ant algorithms have been developed and also combined with genetic algorithms to increase the accuracy of the solutions found. In this section different approaches to query optimization assembled on ant colony optimization algorithms are studied. Specifically three different approaches will me mentioned: the first known ACO algorithm to be applied to the query optimization problem (LI et al., 2008), a best-worst ACO combined with a genetic algorithm to solve the query optimization problem (Zhou et al., 2009) and finally another type of combination between ACO and GA to approach the join ordering problem is presented in (Kadkhodaei and Mahmoudi, 2011).

3 DSQO: DETERMINISTIC SWAPPING QUERY OPTIMIZATION

The method proposed as a novel algorithm to solve the traveling sales man problem in (Niño et al., 2010) takes advantage of the automata theory to construct a path to find a global optimal solution to the problem. Specifically, a special type of deterministic finite automaton is constructed to model the solution space of the combinatorial problem and a transition function is designed to allow the navigation around neighbor answers. The exchange deterministic algorithm (EDA) was used to browse the structure to rapidly converge to an optimal solution.

3.1 Query Optimization based on the Automata Theory

A deterministic finite automaton of swapping, DFAS, is a kind of DFA that allows the modeling of the set of feasible solutions of combinatorial problems where the order of the elements is relevant and no repetitions are permitted. A DFAS is formally defined in (NIÑO and ARDILA, 2009) as a 7-tuple.

$$M = (Q, \Sigma, \delta, q_0, F, X_0, f) \tag{3.1}$$

Where Q represents the set of all feasible solutions to the problem, Σ is the input alphabet and represents the set of all possible exchanges between two elements of the answer. The author proved that the number of elements of Σ is given by the following equation

$$|\Sigma| = \frac{n * (n-1)}{2} \tag{3.2}$$

$\delta: Q \times \Sigma \rightarrow$ Q, is the transition function and takes the node $q_i \in Q$ and swap the elements in the positions indicated by the element of the alphabet, q_0 is the initial state and is given by an initial solution to the problem, F is the set of final states, X_0 is the input vector containing the initial order of elements corresponding to the state q_0, f is the objective function of the combinatorial problem that evaluates the given order X_k in the node q_k.

For instance the following example is given to understand the construction of a DFAS. Given the objective function of a combinatorial optimization problem and an input vector.

$$f(\bar{X}) = 0.3x_1 + 0.2x_2 + 0.1x_3 \tag{3.3}$$

$$\bar{X}_0 = (1,2,3) \tag{3.4}$$

The alphabet contains six elements, consecutively by the application of the definition.

$$\Sigma = \{(1,2), (1,3), (2,3)\} \tag{3.5}$$

The transition function is constructed by labeling the state q_0 after the input vector $\bar{X}_0$; the swap operation is applied to q_0 for every element of the alphabet and every new vector $\bar{X}_i$ constitutes a new state q_i that is included in the DFAS; the process is repeated until every node q_i has been evaluated. Table 2 shows the transition function for the given example

Table 2: DFAS transition function example.

$\delta(q_0,(1,2)) = (2,1,3)$ $X_1 = q_1$	$\delta(q_0,(1,3)) = (3,2,1)$ $X_2 = q_2$	$\delta(q_0,(2,3)) = (1,3,2)$ $X_3 = q_3$
$\delta(q_1,(1,2)) = (1,2,3)$ $X_0 = q_0$	$\delta(q_1,(1,3)) = (3,1,2)$ $X_4 = q_4$	$\delta(q_1,(2,3)) = (2,3,1)$ $X_5 = q_5$
$\delta(q_2,(1,2)) = (2,3,1)$ $X_5 = q_5$	$\delta(q_2,(1,3)) = (1,2,3)$ $X_0 = q_0$	$\delta(q_2,(2,3)) = (3,1,2)$ $X_4 = q_4$
$\delta(q_3,(1,2)) = (3,1,2)$ $X_4 = q_4$	$\delta(q_3,(1,3)) = (2,3,1)$ $X_5 = q_5$	$\delta(q_3,(2,3)) = (1,2,3)$ $X_0 = q_0$
$\delta(q_4,(1,2)) = (1,2,3)$ $X_3 = q_3$	$\delta(q_4,(1,3)) = (2,1,3)$ $X_1 = q_1$	$\delta(q_4,(2,3)) = (3,1,2)$ $X_2 = q_2$
$\delta(q_5,(1,2)) = (1,2,3)$ $X_3 = q_3$	$\delta(q_5,(1,3)) = (2,1,3)$ $X_1 = q_1$	$\delta(q_5,(2,3)) = (3,1,2)$ $X_2 = q_2$

3.2 The Exchange Deterministic Algorithm (EDA)

EDA is a simple algorithm proposed in (Niño et al., 2010) that describes a strategy to browse a DFAS structure to find global optimal solutions to combinatorial problems that allows finding the state q_i that contains the global optimal solution of the problem in polynomial time by only exploring the necessary states minimizing the use of computer memory.

Taking into consideration the characteristics mentioned above the following algorithm was proposed. Where σ is the current state, $\bar{X}_\sigma$ is the vector associated to the current state and $f(\bar{X}_\sigma)$ is the value of evaluating the current state´s vector in the objective function.

```
Step 1. $\sigma = q_0$
Step 2. $\varphi = f(\bar{X}_\sigma)$ and $\theta = empty$
Step 3. $\forall a_i \in \Sigma$, evaluate $\gamma_i = f(\delta(\sigma, a_i))$ and if
        $\gamma_i < \varphi$, let $\varphi = f(\delta(\sigma, a_i))$ and make
        $\theta = a_i$
Step 4. If $\theta = empty$, $\sigma$ is a global optimum.
        Otherwise $\sigma = \delta(\sigma, \theta)$ and loop back to
        step 2.
```

It is easy to observe that the proposed search strategy does not require the construction of the complete DFAS structure at once. Instead, it only constructs the required areas of the solution space as the neighbors are chosen following the objective function improvement. This strategy is expected to save computer memory because it compares states one by one and rapidly discards portions of the solution space that do not improve the objective function.

3.3 DSQO: Deterministic Swapping Query Optimization

A DFAS structure can be constructed to represent the solution space of the join ordering problem because in fact, it is a combinatorial problem where the order of the elements is relevant and no repetitions are allowed.

Following the definition, a DFAS to solve the query optimization problem is the following 7-tuple.

$$M = (Q, \Sigma, \delta, q_0, F, X_0, f(X_i)) \qquad (3.6)$$

Where Q represents the set of all possible join orders, Σ represents all possible exchanges between two tables in the left deep join query three, δ is the transition function from one query plan to another with a symbol of Σ, q_0, is a random element of Q selected as the initial state of the automaton, F, is the same set as Q because every plan in Q represents a solution, X_0, is the vector that contains the order in q_0 and $f(X_i)$, is the objective function that estimates the cost of executing a given X_i plan.

The solution space that a DFAS can represent is reduced to all possible left deep trees and thus no bushy tree strategy can be directly explored by this method. To illustrate how to construct a DFAS modeling the join-ordering problem, Figure 1 shows the transition diagram of the DFAS corresponding to the following query with three tables to join.

```
SELECT   *
FROM tab1, tab2, tab3
WHERE    tab1.fkt2 = tab2.pk AND
        tab2.fkt3 = tab3.pk
```

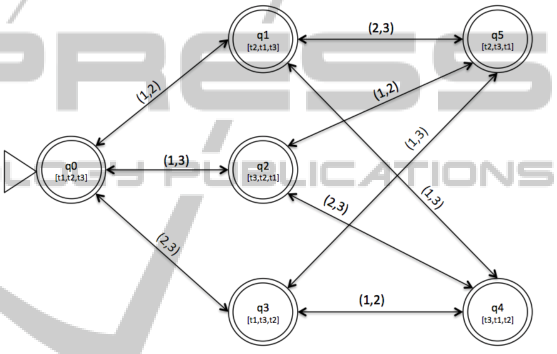

Figure 1: DFAS transition diagram from the example query.

The transition diagram shows how the solution space of the join ordering problem is represented. Each node of the graph contains a vector with a join ordering in the left deep strategy of the following form:

$$[t1, t2, t3] \rightarrow ((t1 \bowtie t2) \bowtie t3) \qquad (3.7)$$

Where the first two elements from left to right are joined first, and then the intermediate table is joined with the next element in the vector, creating another intermediate table. The process is repeated until there are no more elements to join and the last intermediate table contains the expected result.

EDA was proposed as a method to navigate the DFAS structure without building the complete solution space, by the exploration of the neighborhood of a given state and an objective function improvement rule. Even though EDA is capable of finding global optimal solutions to combinatorial problems effectively, it was mainly designed to find optimal solutions to the traveling salesman problem. Despite the similarities between the TSP and the join-ordering problem, it is necessary to adjust the algorithm to perform as well in the solution of the query optimization problem, which is the targeted problem of this work.

The main reason EDA is not efficient in the solution of the join-ordering problem is that the objective function used in the optimization procedure is yet an estimate of the real cost of using a specific join order. Therefore, minimal improvements towards a better solution may not be worth the effort of a new iteration of the algorithm. Another important reason EDA is not effective when applied to the query optimization problem is the lack of representation of the wider solution space, which includes bushy query trees. A slower convergence of the algorithm is caused because it encounters a reasonable number of solutions with Cartesian products in the optimization process of the left-deep only solution space.

In order to improve the convergence speed of the query optimization based on automata theory module, the DSQO algorithm was designed. There were two main improvements made to the original EDA algorithm: an objective function improvement criterion was added in order to avoid unnecessary optimization efforts and a heuristic was included to transform Cartesian product left-deep plans into feasible bushy tree query plans. The heuristic used was taken from the genetic implementation in PostgreSQL.

The objective function improvement criterion added to the algorithm is used to stop the optimization process when the significance of the new optimum found is not relevant in the solution of the problem. An input parameter $\tau \in [0,1]$, which can be seen as a threshold value, is required by DSQO to evaluate if the new solution found in the iteration improves the current solution by a certain percentage given by τ using the following equation.

$$\frac{\varphi - \gamma_i}{\varphi} < \tau \tag{3.8}$$

Where γ_i is the new solution and φ is the current best solution found. The τ parameter can be considered as an attempt to provide the DBA with a tool to adjust the accuracy of the optimization process based on how deep to look into the search space.

The second strategy included in DSQO is the use of a heuristic to avoid Cartesian products in the left-deep solution space and thus expanding it. The heuristic is taken from the PostgreSQL genetic algorithm implementation and works as follows. The given order X_i is evaluated from left to right. At first every relation is seen as a new branch in the query tree and at the beginning, the first relation constitutes the only branch of the tree. Then, every new branch is tried to merge with an existing one or otherwise it is added as a single relation branch to the query tree. Finally, any existing branch in the tree is forced merge to construct a complete query tree.

Comparing DSQO with other meta-heuristic methods applied in query optimization such as simulated annealing, genetic algorithms and ant colony optimization, the proposed method presents a notorious advantage: it only requires a single input parameter. As mentioned before, even though well known meta-heuristics have been successfully applied to the query optimization problem, configuring the right input parameters is a delicate task. With only one parameter to configure the DSQO takes an enormous advantage against its competitors.

4 EXPERIMENTAL SETUP AND DESIGN

The implementation of the DSQO module was built and tested on a Mac mini server running Mac OS X 10.6 operating system. It was equipped with a 2.66 GHz Intel 2 Duo processor, 4GB of DDR3 memory @1067 MHz and a 500GB − 7200 SATA hard disk.

During the execution of tests the machine was not connected to a network to avoid sharing resources with network related tasks, also all queries were run locally in the machine to avoid network-related delay times while measuring response times.

Finally, the machine was setup to stop any other resource demanding service with the purpose of allowing the higher amount of resources to the database system.

The query optimization module based on the automata theory proposed in this work was implemented using the C programming language and compiled using GCC for Mac OSX. It was integrated into the code of the PostgreSQL 9.1 database system within its query optimization module along with GEQO.

The test scenarios were planned based on TPC-DS (POESS, NAMBIAR and WALRATH, 2007), a decision support benchmark that supports a retail product supplier system. The benchmark consists of a database schema with 25 tables, a data population tool, test queries, and a data maintenance model.

The data population tool provided was used to produce a database instance using a 1GB scale factor and 100 test queries were produced as the specification of the benchmark indicates. However, the proposed queries included in the benchmark usually evaluate less than 10 joins with few exceptions.

With the intention of testing the DSQO module with large join queries two scenarios were designed. The first scenario consists of four queries, one for each fact table in the schema, that joins each referenced dimension resulting in queries with 8, 9, 13 and 15 joins, which result still small to test the module but are particularly interesting because they retrieve high amounts of data. The following 8-join query was constructed using the *Store Sales* fact table.

```
SELECT
  date_dim.d_date,
  time_dim.t_time,
  customer.c_first_name,
  customer.c_last_name,
  item.i_product_name,
  cd.cd_gender,
  store.s_store_name,
  customer_address.ca_county,
  hd.hd_dep_count
FROM
  public.store_sales,
  public.date_dim,
  public.time_dim,
  public.item,
  public.customer,
  public.customer_demographics cd,
  public.household_demographics hd,
  public.customer_address,
  public.store
WHERE
  store_sales.ss_sold_date_sk =
  date_dim.d_date_sk AND
  store_sales.ss_sold_time_sk =
  time_dim.t_time_sk AND
  store_sales.ss_item_sk =
  item.i_item_sk AND
  store_sales.ss_customer_sk =
  customer.c_customer_sk AND
  store_sales.ss_cdemo_sk =
  cd.cd_demo_sk AND
  store_sales.ss_hdemo_sk =
  hd.hd_demo_sk AND
  store_sales.ss_addr_sk =
  customer_address.ca_address_sk AND
  store_sales.ss_store_sk =
  store.s_store_sk;
```

To overcome the absence of large join queries, 5 more queries were designed taking advantage of the snowflake design of the database schema. The *Web Returns* fact table was used and its different dimension tables where joined to obtain queries with 15, 20, 25 and 30 join operations.

Finally, each query was run 10 times to obtain an average of the execution time and optimization time. The PostgreSQL GEQO module, the only commercial generic query optimizer, was setup with its standard parameters and the DSQO module was tested using different values for τ specifically 0.02, 0.05, 0.10 and 0.15.

5 COMPARATIVE ANALYSIS

This section will be divided into two subsections comparing the results obtained for each scenario independently. The first subsection shows the results of the star schema experiments and the second subsection displays the results of the snowflake schema experiments. Figures present the average response time and optimization time of each query using the GEQO implementation, included in the PostgreSQL 9.1.2 distribution, and the optimization strategy proposed in this work. The quality of the solution found by each algorithm was measured in terms of plan cost and is also showed using figures that facilitate the analysis.

5.1 Snowflake Scenario

Query execution time results of the tests under the snowflake scenario are shown and also compared to the results obtained by GEQO in figure 2. The graph shows that with small number of join operations both optimizers perform similarly, but when the number of join operations increases DSQO tend to perform better.

A maximum improvement of 13.72% was achieved by DSQO over GEQO solving the 30-join query. Also, the DSQO presented almost a 20% improvement in the plan cost for the same query while, smaller queries obtained similar plan costs with GEQO presenting minimal differences not exceeding 0.5%.

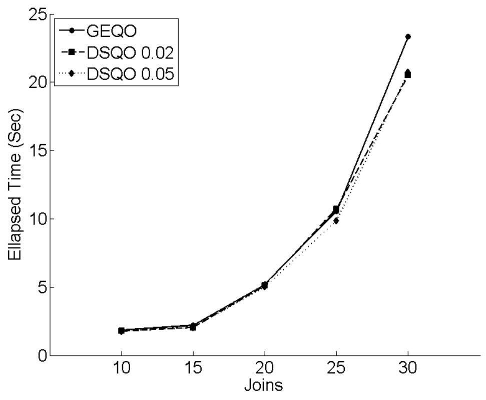

Figure 2: Snowflake query execution time comparison.

In general, optimization time was considerably smaller for DSQO for up to 25 joins. On the other hand, in the 30 join case the optimization time was larger, taking almost 50% more time. The overall query execution time for the 30-join query was smaller due to the quality of the plan that was found, specifically 17.45% better. Figure 3 shows a comparison graph between DSQO with different values of τ and GEQO.

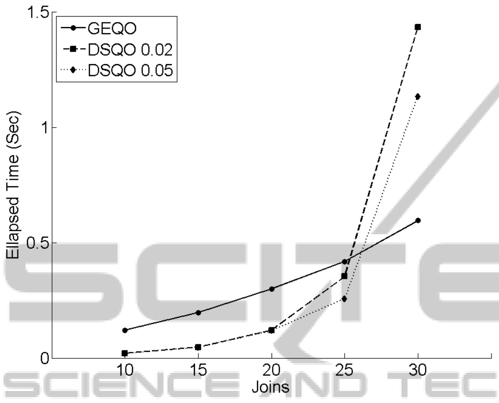

Figure 3: Snowflake query optimization time comparison.

5.2 Star Scenario

The star scenario tests results are shown in figure 4. Query execution times of the proposed algorithm and GEQO are shown for each query in this workload. As the results obtained by DSQO with values of τ of 0.05, 0.10 and 0.15 presented minimal differences, only results with τ values of 0.02 and 0.05 are shown.

The graph indicates that the *Store_sales* fact table presented the least improvement, which is explained by the fact that it is the smallest table. On the other hand, the highest improvement was obtained optimizing the *Web_returns* fact table where a 30% improvement was achieved. The improvement obtained by DSQO in the quality of the plan was not substantial and the highest improvement obtained was 15%.

Similarly as in the query execution time test, results with minimal differences were found between values of τ of 0.10 and 0.15, thus the results presented will omit tests with $\tau = 0.15$.

In general, optimization time spent by DSQO was smaller than GEQO, which is explained by the fact that the maximum amount of joins included in the workload was 15. Figure 5 shows the convergence speed of both algorithms. Even though the proposed algorithm is faster, the quality of the answer found by GEQO is slightly better.

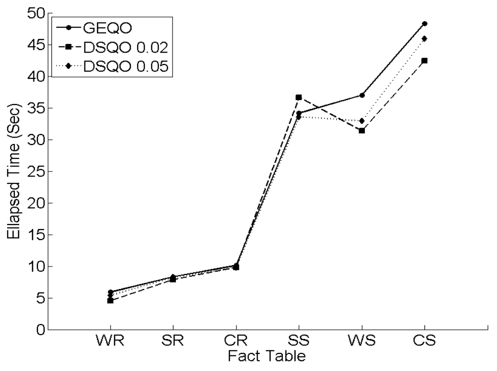

Figure 4: Star query execution time comparison.

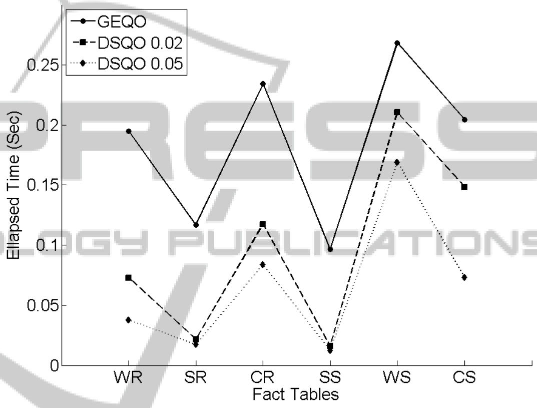

Figure 5: Star query optimization time comparison.

6 CONCLUSIONS AND FUTURE WORK

A new algorithm based on the automata theory was introduced in this work to find global optimal solutions to the join ordering problem in relational database systems. Internal analyses were performed to the EDA algorithm to understand its performance in the solution of the query optimization problem, which derived in the design of the DSQO algorithm. Different types of metrics such as query execution time, optimization time and plan cost were measured to evaluate the efficiency of the proposed query optimizer. DSQO was tested and empirically compared to the genetic algorithm implementation GEQO in PostgreSQL, which is the only commercial Meta heuristic optimizer available. Results show that the utilization of the proposed methodology to solve queries in star and snowflake environments decreases up to 30% the response time of the database system.

Even thought the proposed methodology outperformed the only existing commercial Meta heuristic optimizer, the proposed methodology can be further improved by combining it with other heuristics to obtain better initial solutions. Another future implementation should experiment with different operations other than swapping that may offer the reduction of the exploration of the solution space. Finally, the optimization methodology should be tested under different databases and types of queries, such as those supporting scientific data.

ACKNOWLEDGEMENTS

The research described in this paper was funded by Colciencias and Universidad del Norte, Barranquilla, through research grant approved in the Joven Investigador Proposal 2011.

Yi-Cheng Tu is supported by Grant IIS-1117699 from the US National Science Foundation (NSF).

REFERENCES

Bennett, K., Ferris, M. C. and Ioannidis, Y. E., 1991. *A genetic algorithm for database query optimization.* Computer Sciences Department, University of Wisconsin, Center for Parallel Optimization.

Chaudhuri, S., 1998. An overview of query optimization in relational systems, *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* 1998, ACM, pp. 34-43.

Holland, J. H., 1992. Genetic algorithms. *Scientific American,* 267(1), pp. 66-72.

Ioannidis, Y. E. and Wong, E., 1987. *Query optimization by simulated annealing.* ACM.

Ioannidis, Y. E. and Kang, Y. C., 1991. Left-deep vs. bushy trees: an analysis of strategy spaces and its implications for query optimization, *Proceedings of the 1991 ACM SIGMOD international conference on Management of data* 1991, ACM, pp. 168-177.

Kadkhodaei, H. and Mahmoudi, F., 2011. A combination method for join ordering problem in relational databases using genetic algorithm and ant colony, *Granular Computing (GrC), 2011 IEEE International Conference on* 2011, IEEE, pp. 312-317.

Li, N., Liu, Y., Dong, Y. and Gu, J., 2008. Application of Ant Colony Optimization Algorithm to Multi-Join Query Optimization. *Advances in Computation and Intelligence,* pp. 189-197.

Muntes-Mulero, V., Zuzarte, C. and Markl, V., 2006. An inside analysis of a genetic-programming based optimizer, *Database Engineering and Applications Symposium, 2006. IDEAS'06. 10th International* 2006, IEEE, pp. 249-255.

Niño, E. D. and Ardila, C. J., 2009. Algorithm based on finite automata for obtaining global optimum combinatorial problems. *Ingeniería y Desarrollo,* (25), pp. 99-114.

Niño, E. D., Ardila, C. J., Jabba, D. and Donoso, Y., 2010. A novel Algorithm based on Deterministic Finite Automaton for solving the mono-objective Symmetric Traveling Salesman Problem. *International Journal of Artificial Intelligence,* 5(A10), pp. 101-108.

Poess, M., Nambiar, R. O. and Walrath, D., 2007. Why you should run TPC-DS: a workload analysis, *Proceedings of the 33rd international conference on Very large data bases* 2007, VLDB Endowment, pp. 1138-1149.

Steinbrunn, M., Moerkotte, G. and Kemper, A., 1997. Heuristic and randomized optimization for the join ordering problem. *the VLDB Journal,* 6(3), pp. 191-208.

Swami, A. and Gupta, A., 1988. *Optimization of large join queries.* ACM.

Zhou, Y., Wan, W. and Liu, J., 2009. Multi-joint query optimization of database based on the integration of best-worst Ant Algorithm and Genetic Algorithm, *Wireless Mobile and Computing (CCWMC 2009), IET International Communication Conference on* 2009, IET, pp. 543-546.